

Третий том известной монографии одного из крупнейших американских специалистов по программированию Д. Кнута (первый том вышел в издательстве "Мир" в 1976 г., второй—в 1977 г.) состоит из двух частей: "Сортировка" и "Поиск". В них подробно исследуются различные алгоритмы внутренней и внешней сортировки, изучаются методы поиска информации в таблицах на основе сравнения или преобразования ключей, даются оценки эффективности предлагаемых алгоритмов. Книга снабжена большим количеством задач и примеров разной степени трудности, существенно дополняющих основной текст.

От других руководств по программированию книга выгодно отличается строгостью изложения и широким применением математического аппарата. Вместе с тем она доступна студентам первого курса. Знакомство с двумя первыми томами желательно, но не обязательно. Каждый, кто хочет научиться квалифицированно программировать, найдет в ней много полезного.

Рассчитана на широкий круг программистов.

ПРЕДИСЛОВИЕ РЕДАКТОРОВ ПЕРЕВОДА

Д. Э. Кнут хорошо знаком советскому читателю по переводам двух первых томов его обширной монографии "Искусство программирования для ЭВМ" и не нуждается в аттестации. Настоящая книга представляет собой третий том и посвящена алгоритмам сортировки и поиска информации.

Исторически зарождение методов машинной сортировки можно отнести еще к прошлому столетию, и за столь длительное время многие специалисты успели испробовать свои силы в этой области. Написано немало отчетов, статей, монографий. И даже в этих условиях книга Д. Кнута стала событием. По существу это энциклопедия, в которой можно найти любую справку, касающуюся алгоритмов, методов их оценок, истории вопроса и нерешенных проблем.

Нет нужды говорить о важности самой области. Практически сортировка и поиск в той или иной мере присутствуют во всех приложениях; в частности, при обработке больших объемов данных эффективность именно этих операций определяет эффективность, а иногда и работоспособность всей системы. Поэтому, как справедливо отмечает автор, книга адресована не только системным программистам, занимающимся разработкой программ сортировки и поиска информации. Можно сказать, что достаточно четкие представления об этой области нужны при решении любой задачи на ЭВМ как обязательные элементы искусства программирования.

Кроме теоретической и практической ценности, книга имеет большое методическое значение. Многие авторы и преподаватели смогут извлечь из нее новые и полезные сведения не только по существу рассматриваемых вопросов, но и по способу их изложения. Автору мастерски удается "расслоить" весь материал таким образом, что книгу можно использовать практически на любом уровне знакомства с предметом и при различной общей математической подготовленности читателя.

Перевод выполнен по изданию 1973 г. (первая редакция) с внесением многих (около 700) исправлений и добавлений, любезно предоставленных автором. Разделы с 5.1 по 5.3.2 переведены Н. И. Вьюковой; разделы с 5.3.3 по 5.5 и предисловие — А. Б. Ходулевым; главу 6 перевел В. А. Галатенко.

Ю. М. Баяковский
В. С. Штаркман

ПРЕДИСЛОВИЕ

Кулинария стала искусством, высокой наукой;
повара теперь — благородные люди.

Тит Ливий, *Ab Urbe Condita*, XXXIX.vi
(Роберт Бертон, *Anatomy of Melancholy*, 1.2.2.2)¹

Материал этой книги логически продолжает материал по информационным структурам, изложенный в гл. 2, поскольку здесь к уже рассмотренным концепциям структур добавляется понятие линейно упорядоченных данных. Подзаголовок "Сортировка и поиск" может привести к мысли, что эта книга предназначена лишь для системных программистов, занимающихся построением универсальных программ сортировки или связанных с выборкой информации. Однако в действительности предмет сортировки и поиска дает нам прекрасную основу для обсуждения широкого класса важных общих вопросов:

Как находить хорошие алгоритмы?

Как улучшать данные алгоритмы и программы?

Как исследовать эффективность алгоритмов математически?

Как разумно выбрать один из нескольких алгоритмов для решения конкретной задачи?

В каком смысле можно доказать, что некоторые алгоритмы являются "наилучшими из возможных"?

Как теория вычислений согласуется с практическими соображениями?

Как эффективно использовать различные виды внешней памяти — ленты, барабаны, диски — для больших баз данных?

Я думаю, что на самом деле в контексте сортировки и поиска встречается практически *любой* важный аспект программирования.

Настоящий том состоит из гл. 5 и 6 монографии. В гл. 5 рассматривается сортировка (упорядочение); это очень большая тема, она разбита на две главные части — внутреннюю и внешнюю сортировку. В эту главу входят также дополнительные разделы, развивающие вспомогательную теорию перестановок (§ 5.1) и теорию оптимальных алгоритмов сортировки (§ 5.3). В гл. 6 мы имеем дело с поиском определенного элемента в таблице или файле; содержимое этой главы подразделяется на

¹ Роберт Бертон (1577–1640) — английский ученый, писатель и теолог. *Прим. Перев.*

методы последовательного поиска, методы поиска со сравнением ключей, поиска с использованием свойств цифр, поиска с помощью "хеширования"; затем рассматривается более сложная задача выборки по вторичным ключам. Обе главы поразительно тесно переплетаются между собой, между их предметами имеются близкие аналогии. В дополнение к гл. 2 рассматриваются два важных вида информационных структур, а именно приоритетные очереди (п. 5.2.3) и линейные списки, представляемые посредством сбалансированных деревьев (п. 6.2.3).

Читателю, не знакомому с первым томом этой монографии, рекомендуется обращаться к указателю обозначений (приложение В), так как некоторые из встречающихся в книге обозначений не являются общепринятыми.

Эта книга без большей части математического материала была использована мной в качестве учебника по второму курсу лекций "Структуры данных" для студентов младших и средних курсов. Математические части этой книги, особенно § 5.1, п. 5.2.2, § 6.3 и 6.4, могли бы составить учебник по анализу алгоритмов для студентов средних и старших курсов. Кроме того, на основе п. 4.3.3, 4.6.3, 4.6.4, § 5.3 и п. 5.4.4 можно построить курс лекций "Сложность вычислений" для старшекурсников.

Быстрое развитие информатики и вычислительных наук задержало выход в свет этой книги почти на три года, поскольку очень многие аспекты сортировки и поиска подвергались детальной разработке. Я очень благодарен Национальному научному фонду, Отделению военно-морских исследований, Институту обороны, фирмам IBM и Norges Almementenskapelige Forskningsrad за постоянную поддержку моих исследований.

В подготовке этого тома к печати мне оказали помощь многие лица, особенно Эдвард А. Бендер, Кларк Э. Крайн, Дэвид Э. Фергюсон, Роберт У. Флойд, Рональд Л. Грэхем, Леонидас Гюйба, Джон Хопкрофт, Ричард М. Карп, Гэри Д. Кнотт, Рудольф А. Крутар, Шень Линь, Воган Р. Пратт, Стефан О. Райе, Ричард П; Стэнли, Я. А. ван дер Пул и Джон У. Ренч мл., а также студенты Стэнфорда и Беркли, которым пришлось искать ошибки в рукописи.

Осло, Норвегия,

Д. Э. Кнут сентябрь 1972

Писатель пользуется известными привилегиями, в благодетельности которых, надеюсь, нет никаких оснований сомневаться. Так, встретив у меня непонятное место, читатель должен предположить, что под ним кроется нечто весьма полезное и глубокомысленное.²

(Джонатан Свифт, "Сказка бочки", предисловие, 1704)

ЗАМЕЧАНИЯ ОБ УПРАЖНЕНИЯХ

Упражнения, помещенные в книгах настоящей серии, предназначены как для самостоятельной проработки, так и для семинарских занятий. Трудно, если не невозможно изучить предмет, только читая теорию и не применяя полученную информацию для решения специальных задач и тем самым не заставляя себя обдумывать то, что было прочитано. Кроме того, мы лучше всего заучиваем то, что сами открываем для себя. Поэтому упражнения образуют важную часть данной работы; были предприняты определенные попытки, чтобы отобрать упражнения, в которых бы содержалось как можно больше информации и которые было бы интересно решать.

Во многих книгах легкие упражнения даются вперемешку с исключительно трудными. Зачастую это очень неудобно, так как перед тем, как приступать к решению задачи, читатель обязательно должен представлять себе, сколько времени уйдет у него на это решение (иначе он может разве только просмотреть все задачи). Классическим примером здесь является книга Ричарда Беллмана "Динамическое программирование"; это важная пионерская работа, в которой в конце каждой главы под рубрикой "Упражнения и исследовательские проблемы" дается целый ряд задач, где наряду с глубокими еще нерешенными проблемами встречаются исключительно тривиальные вопросы. Говорят, что однажды кто-то спросил д-ра Беллмана, как отличить упражнения от исследовательских проблем, и тот ответил: "Если вы можете решить задачу, это — упражнение; в противном случае это — проблема".

Можно привести много доводов в пользу того, что в книге типа этой должны быть как исследовательские проблемы, так и очень простые упражнения, и для того чтобы читателю не приходилось

² Перевод А. А. Франковского. — Прим. Перев.

ломать голову над тем, какая задача легкая, а какая трудная, мы ввели "оценки", которые указывают степень трудности каждого упражнения. Эти оценки имеют следующее значение:

Оценка Объяснение

- 00 Чрезвычайно легкое упражнение, на которое можно ответить сразу же, если понят материал текста, и которое почти всегда можно решить "в уме".
- 10 Простая задача, которая заставляет задуматься над прочитанным материалом, но не представляет никаких особых трудностей. На решение такой задачи требуется не больше одной минуты; в процессе решения могут понадобиться карандаш и бумага.
- 20 Задача средней трудности, позволяющая проверить, насколько хорошо понят текст. На то чтобы дать исчерпывающий ответ, требуется примерно 15–20 минут.
- 30 Задача умеренной трудности и/или сложности, для удовлетворительного решения которой требуется больше двух часов.
- 40 Очень трудная или трудоемкая задача, которую, вероятно, следует включить в план практических занятий. Предполагается, что студент может решить такую задачу, но для этого ему потребуется значительный отрезок времени; задача решается нетривиальным образом.
- 50 Исследовательская проблема, которая (насколько это было известно автору в момент написания) еще не получила удовлетворительного решения. Если читатель найдет решение этой задачи, его настоятельно просят опубликовать его; кроме того, автор данной книги будет очень признателен, если ему сообщат решение как можно быстрее (при условии, что оно правильно).

Интерполируя по этой "логарифмической" шкале, можно прикинуть, что означает любая промежуточная оценка. Например, оценка 17 говорит о том, что данное упражнение чуть легче, чем упражнение средней трудности. Задача с оценкой 50, если она будет решена каким-либо читателем, в следующих изданиях данной книги может иметь уже оценку 45.

Автор честно старался давать объективные оценки, но тому, кто составляет задачи, трудно предвидеть, насколько трудными эти задачи окажутся для кого-то другого; к тому же у каждого человека существует определенный тип задач, которые он решает быстрее. Надеюсь, что выставленные мной оценки дают правильное представление о степени трудности задач, но в общем их нужно воспринимать как ориентировочные, а не абсолютные.

Эта книга написана для читателей самых разных степеней математической подготовки и искусственности, поэтому некоторые упражнения предназначены только для читателей с математическим уклоном. Если в каком-либо упражнении математические понятия или результаты используются более широко, чем это необходимо для тех, кого в первую очередь интересует программирование алгоритмов, то перед оценкой такого упражнения ставится буква "М". Если для решения упражнения требуется знание высшей математики в большем объеме, чем это дано в настоящей книге, то ставятся буквы "ВМ". Пометка "ВМ" отнюдь не является свидетельством того, что данное упражнение трудное.

Перед некоторыми упражнениями стоит стрелка ">"; это означает, что данное упражнение особенно поучительно и его рекомендуется обязательно выполнить. Само собой разумеется, никто не ожидает, что читатель (или студент) будет решать все задачи, потому-то наиболее полезные из них и выделены. Это совсем не значит, что другие задачи не стоит решать! Каждый читатель должен по крайней мере попытаться решить все задачи с оценкой 10 и ниже; стрелки же помогут выбрать, какие задачи с более высокими оценками следует решить в первую очередь.

К большинству упражнений приведены ответы; они помещены в специальном разделе в конце книги. Пользуйтесь ими мудро; в ответ смотрите только после того, как вы приложили достаточно усилий, чтобы решить задачу самостоятельно, или же если для решения данной задачи у вас нет времени. Если получен собственный ответ, либо если вы действительно пытались решить задачу, только в этом случае ответ, помещенный в книге, будет поучительным и полезным. Как правило, ответы к задачам излагаются очень кратко, схематично, так как предполагается, что читатель уже честно пытался решить задачу собственными силами. Иногда в приведенном решении дается меньше информации, чем спрашивалось, чаще — наоборот. Вполне возможно, что полученный вами ответ окажется лучше ответа, помещенного в книге, или вы найдете ошибку в этом ответе; в таком случае автор был бы очень обязан, если бы вы как можно скорее подробно сообщили ему об этом. В последующих изданиях настоящей книги будет помещено уже исправленное решение вместе с именем его автора.

Сводка условных обозначений

- > Рекомендуется
- М С математическим уклоном
- ВМ Требуется знания "высшей математики"

- 00 Требуется немедленного ответа
- 10 Простое (на одну минуту)
- 20 Средней трудности (на четверть часа)
- 30 Повышенной трудности.
- 40 Для "матпрактикума"
- 50 Исследовательская проблема

Упражнения

- >1. [00] Что означает пометка "M20"?
2. [10] Какое значение для читателя имеют упражнения, помещаемые в учебниках?
3. [M50] Докажите, что если n —целое число, $n > 2$, то уравнение $x^n + y^n = z^n$ неразрешимо в целых положительных числах x, y, z .

5. Сортировка

Нет дела более трудного по замыслу, более сомнительного по успеху, более опасного при осуществлении, чем вводить новые порядки.

Никколо Макьявелли, "Государь" (1513)

"Но мы не успеем, просмотреть все номера автомобилей",—возразил Дрейк. "А нам и не нужно этого делать, Пол. Мы просто расположим их по порядку и поищем одинаковые".

Перри Мейсон¹. Из "The Case of Angry Mourner" (1951)

Сортировка деревьев с использованием ЭВМ.

При таком новом, "машинном подходе" к изучению природы вы получите возможность быстро распознавать более 260 различных деревьев США, Аляски, Канады, включая пальмы, деревья пустынь и прочую экзотику. Чтобы определить породу дерева, достаточно просто вставить спицу.

Каталог "Edmund Scientific Company" (1964)

В этой главе мы изучим вопрос, который часто возникает в программировании: перерасположение элементов в возрастающем или убывающем порядке. Представьте, насколько трудно было бы пользоваться словарем, если бы слова в нем не располагались в алфавитном порядке. Точно так же от порядка, в котором хранятся элементы в памяти ЭВМ, во многом зависит скорость и простота алгоритмов, предназначенных для их обработки.

Хотя в словарях слово "сортировка" (sorting) определяется как "распределение, отбор по сортам; деление на категории, сорта, разряды", программисты традиционно используют это слово в гораздо более узком смысле, обозначая им сортировку предметов в возрастающем или убывающем порядке. Этот процесс, пожалуй, следовало бы назвать не сортировкой, а *упорядочением* (ordering), но использование этого слова привело бы к путанице из-за перегруженности значениями слова "порядок". Рассмотрим, например, следующее предложение: "Так как только два наших лентопротяжных устройства в порядке, меня призвали к порядку и обязали в срочном порядке заказать еще несколько устройств, чтобы можно было упорядочивать данные разного порядка на несколько порядков быстрее². В математической терминологии это слово также изобилует значениями (порядок группы, порядок перестановки, порядок точки ветвления, отношение порядка и т. п.). Итак, слово "порядок" приводит к хаосу.

¹ Перри Мейсон—герой серии детективных романов популярного американского писателя Эрла Стенли Гарднера.— *Прим. перев.*

² В оригинале "Since only two of our tape drives were in working order I was ordered to order more tape units in short order, in order to order the data several orders of magnitude faster".— *Прим. перев.*

В качестве обозначения для процесса упорядочения предлагалось также слово "ранжирование"³, но оно во многих случаях, по-видимому, не вполне отражает суть дела, особенно если присутствуют равные элементы, и, кроме того, иногда не согласуется с другими терминами. Конечно, слово "сортировка" и само имеет довольно много значений⁴, но оно прочно вошло в программистский жаргон. Поэтому мы без дальнейших извинений будем использовать слово "сортировка" в узком смысле "сортировка по порядку".

Вот некоторые из наиболее важных применений сортировки:

- а) Решение задачи "группировки", когда нужно собрать вместе все элементы с одинаковым значением некоторого признака. Допустим, имеется 10000 элементов, расположенных в случайном порядке, причем значения многих из них равны; и предположим, нам нужно переупорядочить файл так, чтобы элементы с равными значениями занимали соседние позиции в файле. Это, по существу, задача "сортировки" в широком смысле слова, и она легко может быть решена путем сортировки файла в узком смысле слова, а именно расположением элементов в неубывающем порядке $v_1 \leq v_2 \leq \dots \leq v_{10000}$. Эффективностью, которая может быть достигнута в этой процедуре, и объясняется изменение первоначального смысла слова "сортировка".
- б) Если два или более файла отсортировать в одном и том же порядке, то можно отыскать в них все общие элементы за один последовательный просмотр всех файлов, без возвратов. Это тот самый принцип, которым воспользовался Перри Мейсон для раскрытия дела об убийстве (см. эпиграфы к этой главе). Оказывается, что, как правило, гораздо экономнее просматривать список последовательно, а не перескакивая с места на место случайным образом, если только список не настолько мал, что он целиком помещается в оперативной памяти. Сортировка позволяет использовать последовательный доступ к большим файлам в качестве приемлемой замены прямой адресации.
- с) Как мы увидим в гл. 6, сортировка помогает и при поиске, с ее помощью можно сделать выдачи ЭВМ более удобными для человеческого восприятия. В самом деле, листинг (напечатанный машиной документ), отсортированный в алфавитном порядке, зачастую выглядит весьма внушительно, даже если соответствующие числовые данные были рассчитаны неверно.

Хотя сортировка традиционно и большей частью использовалась для обработки коммерческих данных, на самом деле она является инструментом, полезным в самых разных ситуациях, и поэтому о нем не следует забывать. В упр. 2.3.2–17 мы обсудили ее применение для упрощения алгебраических формул. Упражнения, приведенные ниже, иллюстрируют разнообразие типичных применений сортировки.

Одной из первых крупных систем программного обеспечения, продемонстрировавших богатые возможности сортировки, был компилятор *Larc Scientific Compiler*, разработанный фирмой *Computer Sciences Corporation* в 1960 г. В этом оптимизирующем компиляторе для расширенного ФОРТРАНа сортировка использовалась весьма интенсивно, так что различные алгоритмы компиляции работали с относящимися к ним частями исходной программы, расположенными в удобной последовательности. При первом просмотре осуществлялся лексический анализ, т. е. выделение в исходной программе лексических единиц (лексем), каждая из которых соответствует либо идентификатору (имени переменной), либо константе, либо оператору и т. д. Каждая лексема получала несколько порядковых номеров. В результате сортировки по именам и соответствующим порядковым номерам все использования данного идентификатора оказывались собранными вместе. "Определяющие вхождения", специфицирующие идентификатор как имя функции, параметр или многомерную переменную, получали меньшие номера, поэтому они оказывались первыми в последовательности лексем, отвечающих этому идентификатору. Тем самым облегчалась проверка правильности употребления идентификаторов, распределение памяти с учетом деклараций эквивалентности и т. д. Собранный таким образом информация о каждом идентификаторе присоединялась к соответствующей лексеме. Поэтому не было необходимости хранить в оперативной памяти "таблицу символов", содержащую сведения об идентификаторах. После такой обработки лексемы снова сортировались по другому порядковому номеру; в результате в программе, по существу, восстанавливался первоначальный порядок, если не считать того, что арифметические выражения оказывались записанными в более удобной, "польской префиксной" форме. Сортировка использовалась и на последующих фазах компиляции—для облегчения оптимизации циклов, включения в листинг сообщений об ошибках и т. д. Коротче говоря, компилятор был устроен так, что всю обработку файлов, хранящихся на барабанах, фактически мож-

³ В оригинале "sequencing".—Прим. перев.

⁴ Это в большей степени относится к английскому слову "sorting". Здесь автор приводят пример: "He was sort of out sorts after sorting that sort of data". (Он был как будто не в духе после сортировки такого сорта денных), который в русском переводе не столь выразителен.—Прим. перев.

но было вести последовательно. Поэтому-то данные и снабжались такими порядковыми номерами, которые позволяли упорядочивать эти данные различными удобными способами.

Другое, более очевидное применение сортировки возникает при редактировании файлов, где каждая строка снабжена ключом. Пока пользователь вносит с клавиатуры изменения и добавления, необязательно держать весь файл в оперативной памяти. Все изменяемые строки можно позднее отсортировать (а они и так обычно в основном упорядочены) и слить с исходным файлом. Это дает возможность разумно использовать память в режиме мультипрограммирования. [Ср. с С. С. Foster, *Comp. J.*, 11 (1968), 134–137].

Поставщики вычислительных машин считают, что в среднем более 25% машинного времени систематически тратится на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Из этой статистики можно заключить, что либо (i) сортировка имеет много важных применений, либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки. По-видимому, каждое из трех предположений содержит долю истины. Во всяком случае ясно, что сортировка заслуживает серьезного изучения с точки зрения ее практического использования.

Но даже если бы сортировка была почти бесполезна, нашлась бы масса других причин заняться ею! Изобретательные алгоритмы сортировки говорят о том, что она и сама по себе интересна как объект исследования. В этой области существует множество увлекательных нерешенных задач наряду с весьма немногими уже решенными.

Рассматривая вопрос в более широком плане, мы обнаружим, что алгоритмы сортировки представляют собой интересный частный пример того, как следует подходить к решению проблем программирования вообще. Мы познакомимся со многими важными принципами манипулирования со структурами данных и проследим за эволюцией различных методов сортировки, причем читателю часто будет предоставляться возможность самому "открывать" те же идеи, как будто бы до него никто с подобными задачами не сталкивался. Обобщение этих частных методов позволит нам в значительной степени овладеть теми способами мышления, которые помогут создавать добротные алгоритмы для решения других проблем, связанных с ЭВМ.

Методы сортировки служат великолепной иллюстрацией идей *анализа алгоритмов*, т. е. идей, позволяющих оценивать рабочие характеристики алгоритмов, а значит, разумно выбирать среди, казалось бы, равноценных методов. Читатели, имеющие склонность к математике, найдут в этой главе немало способов оценки скорости работы алгоритмов и методов решения сложных рекуррентных соотношений. С другой стороны, изложение построено так, что читатели, не имеющие такой склонности, могут безболезненно пропускать выкладки.

Прежде чем двигаться дальше, необходимо более четко определить задачу и ввести соответствующую терминологию. Пусть надо упорядочить N элементов

$$R_1, R_2, \dots, R_N.$$

Назовем их *записями*, а всю совокупность N записей назовем *файлом*. Каждая запись R_j имеет *ключ* K_j , который и управляет процессом сортировки. Помимо ключа, запись может содержать дополнительную, "сопутствующую информацию", которая не влияет на сортировку, но всегда остается в этой записи.

Отношение порядка " $<$ " на множестве ключей вводится таким образом, чтобы для любых трех значений ключей a, b, c выполнялись следующие условия:

- i) справедливо одно и только одно из соотношений $a < b, a = b, b < a$ (закон трихотомии);
- ii) если $a < b$ и $b < c$, то $a < c$ (закон транзитивности).

Эти два свойства определяют математическое понятие *линейного упорядочения*, называемого еще *совершенным упорядочением*. Любое множество с отношением " $<$ ", удовлетворяющим свойствам (i) и (ii), поддается сортировке большинством методов, описанных в этой главе, хотя некоторые из них годятся только для числовых и буквенных ключей с обычным отношением порядка.

Задача сортировки — найти такую перестановку записей $p(1) p(2) \dots p(N)$, после которой ключи расположились бы в неубывающем порядке:

$$K_{p(1)} \leq K_{p(2)} \leq \dots \leq K_{p(n)}. \quad (1)$$

Сортировка называется *устойчивой*, если она удовлетворяет дополнительному условию, что записи с одинаковыми ключами остаются в прежнем порядке, т. е.

$$p(i) < p(j), \quad \text{если } K_{p(i)} = K_{p(j)} \text{ и } i < j. \quad (2)$$

В ряде случаев может потребоваться физически перемещать записи в памяти так, чтобы их ключи были упорядочены; в других случаях достаточно создать вспомогательную таблицу, которая некоторым образом описывает перестановку и обеспечивает доступ к записям в соответствии с порядком их ключей.

Некоторые методы сортировки предполагают существование величин " ∞ " и " $-\infty$ " или одной из них. Величина " ∞ " считается больше, а величина " $-\infty$ " меньше любого ключа:

$$-\infty < K_j < \infty, \quad 1 \leq j \leq N. \quad (3)$$

Эти величины используются в качестве искусственных ключей, а также как граничные признаки. Равенство в (3), вообще говоря, исключено. Если же оно тем не менее допускается, алгоритмы можно модифицировать так, чтобы они все-таки работали, хотя нередко при этом их изящество и эффективность отчасти утрачиваются.

Обычно сортировку подразделяют на два класса: *внутреннюю*, когда все записи хранятся в быстрой оперативной памяти, и *внешнюю*, когда все они там не помещаются. При внутренней сортировке имеются более гибкие возможности для построения структур данных и доступа к ним, внешняя же показывает, как поступать в условиях сильно ограниченного доступа.

Достаточно хороший общий алгоритм затрачивает на сортировку N записей время порядка $N \log N$; при этом требуется около $\log N$ "проходов" по данным. Как мы увидим в п. 5.3.1, это минимальное время. Так, если удвоить число записей, то и время при прочих равных условиях возрастет немногим более чем вдвое. (На самом деле, когда N стремится к ∞ , время растет как $N(\log N)^2$, если все ключи различны, так как и размеры ключей увеличиваются с ростом N ; но практически N всегда остается ограниченным.)

Упражнения

(ПЕРВАЯ ЧАСТЬ)

- [M20] Докажите, что из законов трихотомии и транзитивности вытекает *единственность* перестановки $p(1), p(2), \dots, p(N)$, если сортировка устойчива.
- [21] Пусть каждая запись R . некоторого файла имеет *два* ключа: "большой ключ" K_j и "малый ключ" k_j , причем оба множества ключей линейно упорядочены. Тогда можно обычным способом ввести "лексикографический порядок" на множестве пар ключей (K, k) :

$$(K_i, k_i) < (K_j, k_j), \quad \text{если } K_i < K_j \text{ или если } K_i = K_j \text{ и } k_i < k_j.$$

Некто (назовем его мистер A) отсортировал этот файл сначала по большим ключам, получив n групп записей с одинаковыми большими ключами в каждой группе:

$$K_{p(1)} = \dots = K_{p(i_1)} < K_{p(i_1+1)} = \dots = K_{p(i_2)} < \dots < K_{p(i_{n-1}+1)} = \dots = K_{p(i_n)},$$

где $i_n = N$. Затем каждую из n групп $R_{p(i_{j-1}+1)}, \dots, R_{p(i_j)}$ он отсортировал по малым ключам.

Тот же исходный файл взял мистер B и отсортировал его сначала по малым ключам, а потом получившийся файл отсортировал по большим ключам.

Взяв тот же исходный файл, мистер C отсортировал его один раз, пользуясь лексикографическим порядком между парами ключей (K_j, k_j) .

Получилось. ли у всех троих одно и то же?

- [M25] Пусть на множестве K_1, \dots, K_N определено отношение $<$, для которого закон трихотомии выполняется, а закон транзитивности — *нет*. Докажите, что и в этом случае возможна устойчивая сортировка записей, т. е. такая, что выполняются условия (1) и (2); на самом деле существуют по крайней мере три расположения записей, удовлетворяющих этим условиям!
- [15] Мистер Тупица (программист) захотел узнать, находится ли в ячейке A машины MIX число, большее числа из ячейки B , меньшее или же равное ему. Он написал

LDA A

SUB B

а потом проверил, какой результат получился в rA : положительное число, отрицательное или нуль. Какую серьезную ошибку он допустил и как должен был поступить?

5. [17] Напишите MIX-подпрограмму для сравнения ключей, занимающих несколько слов, исходя из следующих условий:

Вызов: JMP COMPARE
 Состояние при входе: rI1 = n; CONTENTS(A + k) = a_k, CONTENTS(B + k) = b_k при 1 ≤ k ≤ n; предполагается, что n ≥ 1.
 Состояние при выходе: CI = +1, если (a_n, ..., a₁) > (b_n, ..., b₁);
 CI = 0, если (a_n, ..., a₁) = (b_n, ..., b₁);
 CI = -1, если (a_n, ..., a₁) < (b_n, ..., b₁);
 rX и rI1, возможно, изменились.

Здесь отношение (a_n, ..., a₁) < (b_n, ..., b₁) обозначает лексикографическое упорядочение слева направо, т. е. существует индекс j, такой, что a_k = b_k при n ≥ k > j, но a_j < b_j.

6. [30] В ячейках A и B содержатся соответственно числа a и b. Покажите, что можно написать MIX-программу, которая бы вычисляла min(a, b) и записывала результат в ячейку C, не пользуясь командами перехода. (Предостережение: поскольку арифметическое переполнение невозможно обнаружить без команд перехода, разумно так построить программу, чтобы переполнение не могло возникнуть ни при каких значениях a и b)
7. [M27] Какова вероятность того, что после сортировки в неубывающем порядке N независимых равномерно распределенных на отрезке [0, 1] случайных величин r-е от начала число окажется ≤ x?

Упражнения

(ВТОРАЯ ЧАСТЬ)

В каждом из этих упражнений поставлена задача, с которой может столкнуться программист. Предложите "хорошее" решение задачи, предполагая, что имеется сравнительно небольшая оперативная память и около полудюжины лентопротяжных устройств (этого количества достаточно для сортировки).

- [75] Имеется лента, на которой записан миллион слов данных. Как определить, сколько на этой ленте различных слов?
- [18] Вообразите себя в роли Управления внутренних доходов Министерства финансов США. Вы получаете миллионы "информационных" карточек от организаций о том, сколько денег они выплачивали различным лицам, и миллионы "налоговых" карточек от различных лиц об их доходах. Как бы вы стали отыскивать людей, которые сообщили не обо всех своих доходах?
- [M25](Транспонирование матрицы.) Имеется магнитная лента, содержащая миллион слов, которые представляют собой элементы 1000 × 1000-матрицы, записанные по строкам: a_{1,1} a_{1,2} ... a_{1,1000} a_{2,1} ... a_{2,1000} ... a_{1000,1000}. Ваша задача — получить ленту, на которой элементы этой матрицы были бы записаны по столбцам: a_{1,1} a_{2,1} ... a_{1000,1} a_{1,2} ... a_{1000,2} ... a_{1000,1000} (Постарайтесь сделать не более десяти просмотров данных.)
- [M26] В вашем распоряжении довольно большой файл из N слов. Как бы вы его "перетасовали" случайным образом?
- [24] В некоем университете работает около 1000 преподавателей и имеется 500 комитетов. Считается, что каждый преподаватель является членом по крайней мере двух комитетов. Вам нужно подготовить с помощью машины удобочитаемые списки членов всех комитетов. Вы располагаете колодой из приблизительно 1500 перфокарт, сложенных произвольным образом и содержащих следующую информацию:

Членские карточки: колонка 1 — пробел; колонки 2–18 — фамилия с последующими пробелами; колонки 19–20 — инициалы; колонки 21–23 — номер первого Комитета; колонки 24–26 — номер второго комитета; ...; колонки 78–80 — номер двадцатого комитета (если нужно) или пробелы.

Комитетские карточки: колонка 1 — "*" ; колонки 2–77 — название комитета; колонки 78–80 — номер комитета. Как вы должны действовать? (Опишите свой метод достаточно подробно.)

- [20] Вы работаете с двумя вычислительными системами, в которых по-разному упорядочены литеры (буквы и цифры). Как заставить первую ЭВМ сортировать файлы с буквенно-цифровой информацией, предназначенные для использования на второй ЭВМ?
- [18] Имеется довольно большой список людей, родившихся в США, с указанием штата, в котором они родились. Как подсчитать число людей, родившихся в каждом штате? (Предполагается, что ни один человек не указан в списке более одного раза.)
- [20] Чтобы облегчить внесение изменений в большие программы, написанные на ФОРТРАНе, вы хотите написать программу, выпечатающую таблицу "перекрестных ссылок". Входными данными для нее служит программа на ФОРТРАНе, а в результате получается листинг исходной

программы, снабженный указателем всех случаев употребления каждого идентификатора (т. е. имени) в программе. Как написать такую программу?

9. [33] (Сортировка каталожных карточек.) Способы составления алфавитных каталогов в разных библиотеках несколько отличаются друг от друга. В следующем "алфавитном" списке содержатся рекомендации, взятые из правил регистрации и хранения каталожных карточек Американской библиотечной ассоциации (Чикаго, 1942):

Текст карточки	Замечания
R. Accademia nazionale dei Lincei, Rome	В названиях иностранных (кроме британских) учреждений слово "royalty" (королевский) игнорируется
1812; ein historischer roman.	Achtzehnhundert zwölf
Bibliothèque d'histoire révolutionnaire.	Во французском тексте апостроф рассматривается как пробел
Bibliothèque des curiosités.	Надстрочные знаки игнорируются
Brown, Mrs. J. Crosby	Указание положения (Mrs.) игнорируется
Brown, John	Фамилии с датами следуют за фамилиями без дат...
Brown, John, mathematician	... которые упорядочиваются по
Brown, John, of Boston	описательным словам
Brown, John, 1715–1766	Одинаковые фамилии упорядочиваются по датам рождения
BROWN, JOHN, 1715–1766	Работы "о нем" идут после его работ
Brown, John, d. 1811	Иногда год рождения определяется приблизительно
Brown, Dr. John, 1810–1882	но Указание положения (Dr.) игнорируется
Brown-Williams, Reginald Makepeace	Дефис рассматривается как пробел
Brown America.	Названия книг идут после составных фамилий
Brown & Dallison's Nevada directory.	& в английском тексте превращается в "and"
Brownjohn, Alan	
Den', Vladimir Ėduardovich, 1867–	Апостроф в именах игнорируется
The den.	Артикль в начале текста игнорируется
Den lieben süßen mädeln.	... если существительное стоит в именительном падеже
Dix, Morgan, 1827–1908	Фамилии идут раньше других слов
1812 ouverture.	Dix-huit cent douze
Le XIXe siècle français.	Dix-neuvième
The 1847 issue of U. S. stamps.	Eighteen forty-seven
1812 overture.	Eighteen twelve
I am a mathematician,	(by Norbert Weiner)
IBM journal of research and development.	Аббревиатуры рассматриваются как ряд однобуквенных слов
ha-I ha-eḥad.	Артикль в начале текста игнорируется
Ia; a love story.	Знаки препинания в названиях игнорируются
International Business Machines Corporation	
al-Khuwārizmī, Muḥammad ibn Mūsā, fl. 813–846	Начальное "al-" в арабских именах игнорируется
Labour; a magazine for all workers.	Заменяется на "Labor"
Labor research association	
Labour, see Labor	Ссылка на другую карточку в картотеке
McCall's cookbook	Апостроф в английском тексте игнорируется
McCarthy, John, 1927–	Mc = Mac
Machine-independent computer programming.	Дефис рассматривается как пробел
MacMahon, Maj. Percy Alexander, 1854–1929	Указание положения (Maj) игнорируется

Mrs. Dalloway.	”Mrs.” = ”Mistress”
Mistress of mistresses.	
Royal society of London	
St. Petersburger Zeitung.	”St.” = ”Saint” даже в немецком тексте
Saint-Saëns, Camille, 1835–1921	Дефис рассматривается как пробел
Ste. Anne des Monts, Quebec	Sainte
Seminumerical algorithms.	
Uncle Tom’s cabin.	
U.S. Bureau of the census.	”U.S.” = ”United States”
Vandermonde, Alexander Théophile, 1735–1796	
Van Valkenburg, Mac Elwyn, 1921–	Пробел после префикса в фамилиях игнорируется
Von Neumann, John, 1903–1957	
The whole art of legerdemain.	
Who’s afraid of Virginia Woolf?	Апостроф в английском тексте игнорируется
Wijngaarden, Adriaan van, 1916–	Фамилия никогда не начинается с малой буквы

(У большинства из этих правил есть исключения; кроме того, существует много других правил, которые здесь не упомянуты.)

Предположим, вам пришлось сортировать большое количество таких карточек с помощью вычислительной машины и впоследствии обслуживать огромную картотеку, причем у вас нет возможности изменить уже сложившиеся порядки заполнения карточек. Как бы вы организовали информацию, чтобы упростить операции включения новых карточек и сортировки?

10. [M21](*Дискретные логарифмы.*) Пусть известно, что p —(довольно большое) простое число, а a — первообразный корень по модулю p . Следовательно, для любого b в диапазоне $1 \leq b < p$ существует единственное n , такое, что $a^n \bmod p = b$, $1 \leq n < p$. Как по заданному b найти n менее чем за $O(n)$ шагов? [Указание. Пусть $m = \lceil \sqrt{p} \rceil$. Попытайтесь решить уравнение $a^{mn_1} \equiv ba^{-n_2} \pmod{p}$ при $0 \leq n_1, n_2 < m$.]
11. [M25] (Э. Т. Паркер.) Эйлер выдвинул предположение, что уравнение

$$u^6 + v^6 + w^6 + x^6 + y^6 = z^6$$

не имеет решений (за исключением тривиальных) среди целых неотрицательных чисел u, v, w, x, y, z , когда по крайней мере четыре переменные равны нулю. Помимо этого, он предполагал, что уравнение

$$x_1^n + \dots + x_{n-1}^n = x_n^n$$

не имеет нетривиальных решений при $n \geq 3$, но это предположение было опровергнуто: с помощью вычислительной машины найдено тождество $27^5 + 84^5 + 110^5 + 133^5 = 144^5$; см. Л. Дж. Лэндер, Т. Р. Паркин и Дж. Л. Селфридж, *Math. Comp.*, **21** (1967), 446–459. Придумайте, как можно было бы использовать сортировку для поиска примеров, опровергающих предположение Эйлера при $n = 6$.

- >12. [24] Файл содержит большое количество 30-разрядных двоичных слов: x_1, \dots, x_N . Придумайте хороший способ нахождения в нем всех *дополнительных* пар (x_i, x_j) . (Два слова называются дополнительными, если второе содержит 0 во всех разрядах, в которых были 1 в первом слове, и наоборот; таким образом, они дополнительные тогда и только тогда, когда их сумма равна $(11 \dots 1)_2$, если они рассматриваются как двоичные числа.)
- >13. [25] Имеется файл, содержащий 1000 30-разрядных двоичных слов x_1, \dots, x_{1000} . Как бы вы стали составлять список всех пар (x_i, x_j) , таких, что x_i отличается от x_j не более чем в двух разрядах?
14. [22] Как бы вы поступили при отыскании всех пятибуквенных анаграмм, таких, как **CARET, CARTE, CATER, CRATE, REACT, TRACE; CRUEL, LUCRE, ULCER; DOWRY, ROWDY, WORDY?** [Если бы вы, скажем, захотели узнать, существуют ли в английском языке наборы из десяти или более анаграмм, кроме замечательной серии:

APERS, ASPER, PARES, PARSE, PEARS, PRASE, RAPES, REAPS, SPARE, SPEAR,

к которой можно добавить еще французское слово **APRES.**]

15. [M28] Пусть даны описания весьма большого числа направленных графов. Каким путем можно сгруппировать *изоморфные* графы? (Два направленных графа называются изоморфными, если

существует взаимно однозначное соответствие между их вершинами и взаимно однозначное соответствие между их дугами, причем эти соответствия сохраняют инцидентность вершин и дуг.)

16. [30] В некоторой группе из 4096 человек каждый имеет около 100 знакомых. Был составлен список всех пар людей, знакомых между собой. (Это отношение симметрично, т. е. если x знаком с y , то и y знаком с x . Поэтому список содержит примерно 200000 пар.) Придумайте алгоритм, который по заданному k выдавал бы все *клики* из k человек. (Клика — это группа людей, в которой все между собой знакомы.) Предполагается, что слишком больших клик не бывает.
17. [30] Три миллиона человек с различными именами были уложены рядом непрерывной цепочкой от Нью-Йорка до Калифорнии. Каждому из них дали листок бумаги, на котором он написал свое имя и имя своего ближайшего западного соседа. Человек, находившийся в самой западной точке цепи, не понял, что ему делать, и выкинул свой листок. Остальные 2999999 листков собрали в большую корзину и отправили в Национальный архив, в Вашингтон, округ Колумбия. Там содержимое корзины тщательно перетасовали и записали на магнитные ленты.

Специалист по теории информации определил, что имеется достаточно информации для восстановления списка людей в исходном порядке. Специалист по программированию нашел способ сделать это менее чем за 1000 просмотров лент с данными, используя лишь последовательный доступ к файлам на лентах и небольшое количество памяти с произвольным доступом. Как ему это удалось?

[Другими словами, как, имея расположенные произвольным образом пары (x_i, x_{i+1}) , $1 \leq i < N$, где все x_i различны, получить последовательность x_1, x_2, \dots, x_N , применяя лишь методы последовательной обработки данных, пригодные для работы с магнитными лентами? Это задача сортировки в случае, когда трудно определить, какой из двух ключей предшествует другому; мы уже поднимали этот вопрос в упр. 2.2.3–25.]

5.1. * КОМБИНАТОРНЫЕ СВОЙСТВА ПЕРЕСТАНОВОК

Перестановкой конечного множества называется некоторое расположение его элементов в ряд. Перестановки особенно важны при изучении алгоритмов сортировки, так как они служат для представления неупорядоченных исходных данных. Чтобы исследовать эффективность различных методов сортировки, нужно уметь подсчитывать число перестановок, которые вынуждают повторять некоторый шаг алгоритма определенное число раз.

Мы, конечно, уже не раз встречались с перестановками в гл. 1, 2 и 3. Например, в п. 1.2.5 обсуждались два основных теоретических метода построения $n!$ перестановок из n объектов; в п. 1.3.3 проанализированы некоторые алгоритмы, связанные с циклической структурой и мультипликативными свойствами перестановок; в п. 3.3.2 изучены их отрезки монотонности. Цель настоящего параграфа — изучить некоторые другие свойства перестановок и рассмотреть общий случай, когда допускается наличие одинаковых элементов. Попутно мы узнаем многое о комбинаторной математике.

Свойства перестановок настолько красивы, что представляют и самостоятельный интерес. Удобно будет дать их систематическое изложение в одном месте, а не разбрасывать материал по всей главе. Читателям, не имеющим склонности к математике, а также тем, кто жаждет поскорее добраться до самих методов сортировки, рекомендуется перейти сразу к § 5.2, потому что настоящий параграф непосредственного отношения к сортировке почти не имеет.

5.1.1. *Инверсии

Пусть $a_1 a_2 \dots a_n$ — перестановка множества $\{1, 2, \dots, n\}$. Если $i < j$, а $a_i > a_j$, то пара (a_i, a_j) называется инверсией перестановки; например, перестановка 3 1 4 2 имеет три инверсии: (3, 1), (3, 2) и (4, 2). Каждая инверсия — это пара элементов, "нарушающих порядок"; следовательно, единственная перестановка, не содержащая инверсий, — это отсортированная перестановка $1 2 \dots n$. Такая связь с сортировкой и есть главная причина нашего интереса к инверсиям, хотя это понятие уже использовалось нами при анализе алгоритма динамического распределения памяти (см. упр. 2.2.2–9).

Понятие инверсии ввел Г. Крамер в 1750 г. [Intr. à l'Analyse des Lignes Courbes algébriques (Geneva, 1750), 657–659; ср. с Томас Мюир, Theory of Determinants, 1 (1906), 11–14] в связи со своим замечательным правилом решения линейных уравнений. В сущности, он определил детерминант $n \times n$ -матрицы следующим образом:

$$\det \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} = \sum (-1)^{I(a_1 a_2 \dots a_n)} x_{1a_1} x_{2a_2} \dots x_{na_n},$$

где сумма берется по всем перестановкам $a_1 a_2 \dots a_n$, а $I(a_1 a_2 \dots a_n)$ — число инверсий в перестановке.

Таблицей инверсий перестановки $a_1 a_2 \dots a_n$ называется последовательность чисел $b_1 b_2 \dots b_n$, где b_j — число элементов, больших j и расположенных левее j . (Другими словами, b_j — число инверсий, у

которых второй элемент равен j .) Например, таблицей инверсий перестановки

$$5\ 9\ 1\ 8\ 2\ 6\ 4\ 7\ 3 \quad (1)$$

будет

$$2\ 3\ 6\ 4\ 0\ 2\ 2\ 1\ 0, \quad (2)$$

поскольку 5 и 9 расположены левее 1; 5, 9, 8—левее 2 и т.д., всего 20 инверсий. По определению

$$0 \leq b_1 \leq n-1, 0 \leq b_2 \leq n-2, \dots, 0 \leq b_{n-1} \leq 1, b_n = 0. \quad (3)$$

Пожалуй, наиболее важный факт, касающийся перестановок, и установленный Маршаллом Холлом, это то, что *таблица инверсий единственным образом определяет соответствующую перестановку*. [См. Proc. Symp. Applied Math., 6 (American Math. Society, 1956), 203.] Из любой таблицы инверсий $b_1\ b_2\ \dots\ b_n$, удовлетворяющей условиям (3), можно однозначно восстановить перестановку, которая порождает данную таблицу, путем последовательного определения относительного расположения элементов $n, n-1, \dots, 1$ (в этом порядке). Например, перестановку, соответствующую (2), можно построить следующим образом: выпишем число 9; так как $b_8 = 1$, то 8 стоит правее 9. Поскольку $b_7 = 2$, то 7 стоит правее 8 и 9. Так как $b_6 = 2$, то 6 стоит правее двух уже выписанных нами чисел; таким образом, имеем

$$9\ 8\ 6\ 7.$$

Припишем теперь 5 слева, потому что $b_5 = 0$; помещаем 4 вслед за четырьмя из уже записанных чисел, 3—после шести выписанных чисел (т. е. в правый конец) и получаем

$$5\ 9\ 8\ 6\ 4\ 7\ 3.$$

Вставив аналогичным образом 2 и 1, придем к (1).

Такое соответствие важно, потому что часто можно заменить задачу, сформулированную в терминах перестановок, эквивалентной ей задачей, сформулированной в терминах таблиц инверсий, которая, возможно, решается проще. Рассмотрим, например, самый простой вопрос: сколько существует перестановок множества $\{1, 2, \dots, n\}$? Ответ должен быть равен числу всевозможных таблиц инверсий, а их легко пересчитать, так как b_1 можно выбрать n различными способами, b_2 можно независимо от b_1 выбрать $n-1$ способами, \dots , b_n —одним способом; итого $n(n-1)\dots 1 = n!$ различных таблиц инверсий. Таблицы инверсий пересчитать легче, потому что b независимы, в то время как a должны быть все различны.

В п. 1:2.10 мы исследовали задачу о числе локальных максимумов перестановки, если читать ее справа налево; иными словами, требовалось подсчитать количество элементов, каждый из которых больше любого из следующих после него. (Например, правосторонние максимумы в (1)—это 9, 8, 7 и 3.) Оно равно количеству индексов j , таких, что $b_j = n-j$. Так как b_1 принимает значение $n-1$ с вероятностью $1/n$, b_2 (независимо) принимает значение $n-2$ с вероятностью $1/(n-1)$ и т.д., то из рассмотрения инверсий ясно, что среднее число правосторонних максимумов равно

$$\frac{1}{n} + \frac{1}{n-1} + \dots + 1 = H_n.$$

Аналогичным способом легко получить и соответствующую производящую функцию.

Другие применения таблиц инверсий встретятся далее в этой главе в связи с конкретными алгоритмами сортировки.

Ясно, что если поменять местами *соседние* элементы перестановки, то общее число инверсий увеличится или уменьшится на единицу. На рис. 1 показаны 24 перестановки множества $\{1, 2, 3, 4\}$; линиями соединены перестановки, отличающиеся друг от друга положением двух соседних элементов; двигаясь вдоль линии вниз, мы увеличиваем число инверсий на единицу. Следовательно, число инверсий в перестановке n равно длине нисходящего пути из 1 2 3 4 в n на рис. 1; все такие пути должны иметь одинаковую длину.

Заметим, что эту диаграмму можно рассматривать как трехмерное твердое тело—”усеченный октаэдр”, имеющий 8 шестиугольных и 6 квадратных граней. Это один из равномерных многогранников, которые обсуждал Архимед (см. упр. 10).

Picture: Рис. 1. Усеченный октаэдр, на котором показано изменение числа инверсий, когда меняются местами два соседних элемента перестановки;

Не следует путать "инверсии" перестановок с обратными перестановками. Вспомним, что перестановку можно записывать в виде двух строк

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a_1 & a_2 & a_3 & \dots & a_n \end{pmatrix}; \tag{4}$$

обратной к этой перестановке называется перестановка a'_1, a'_2, \dots, a'_n , которая получается, если в (4) поменять местами строки, а затем упорядочить столбцы в возрастающем порядке по верхним элементам:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ 1 & 2 & 3 & \dots & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a'_1 & a'_2 & a'_3 & \dots & a'_n \end{pmatrix}; \tag{5}$$

Например, обратной к перестановке 5 9 1 8 2 6 4 7 3 будет перестановка 3 5 9 7 1 6 8 4 2, так как

$$\begin{pmatrix} 5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2 \end{pmatrix}.$$

Можно дать другое определение обратной перестановки: $a'_j = k$ тогда и только тогда, когда $a_k = j$.

Обратную перестановку впервые ввёл Х. А. Роте [в К.Ф. Hindenburg(ed.), Sammlung combinatorisch-analytischer Abhandlungen, 2 (Leipzig, 1800), 263–305]; он заметил интересную связь между обратными перестановками и инверсиями: *обратная перестановка содержит ровно столько же инверсий, сколько исходная*. Роте дал не самое простое доказательство этого факта, но оно поучительно и притом довольно красиво. Построим таблицу размера $n \times n$ и поставим точки в j -й клетке i -й строки, если $a_i = j$. После этого расставим крестики во всех клетках, снизу от которых (в том же столбце) и справа (в той же строке) есть точки. Например, для 5 9 1 8 2 6 4 7 3 диаграмма будет такой:

Picture: ()

Количество крестиков равно числу инверсий, так как нетрудно видеть, что b_j равно числу крестиков в j -м столбце. Если мы теперь транспонируем диаграмму (поменяв ролями строки и столбцы), то получим диаграмму для обратной по отношению к исходной перестановке; значит, число крестиков (число инверсий) одинаково в обоих случаях. Роте использовал этот факт для доказательства того, что детерминант матрицы не меняется при транспонировании.

Для анализа некоторых алгоритмов сортировки необходимо знать число перестановок n элементов, содержащих ровно k инверсий. Обозначим это число через $I_n(k)$; в табл. 1 приведены первые несколько значений этой функции.

Таблица 1

n	Перестановки с k инверсиями											
	$I_n(0)$	$I_n(1)$	$I_n(2)$	$I_n(3)$	$I_n(4)$	$I_n(5)$	$I_n(6)$	$I_n(7)$	$I_n(8)$	$I_n(9)$	$I_n(10)$	$I_n(11)$
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	1	2	2	1	0	0	0	0	0	0	0	0
4	1	3	5	6	5	3	1	0	0	0	0	0
5	1	4	9	15	20	22	20	15	9	4	1	0
6	1	5	14	29	49	71	90	101	101	90	71	49

Из рассмотрения таблицы инверсий $b_1 b_2 \dots b_n$ ясно, что $I_k(0) = 1$, $I_n(1) = n - 1$ и что выполняется свойство симметрии:

$$I_n \left(\binom{n}{2} - k \right) = I_n(k) \tag{6}$$

Далее, так как значения b можно выбирать независимо друг от друга, то нетрудно видеть, что производящая функция

$$G_n(z) = I_n(0) + I_n(1)z + I_n(2)z^2 + \dots \tag{7}$$

удовлетворяет соотношению $G_n(z) = (1 + z + \dots + z^{n-1})G_{n-1}(z)$; следовательно, она имеет довольно, простой вид

$$(1 + z + \dots + z^{n-1}) \dots (1 + z)(1) = (1 - z^n) \dots (l - z^2)(l - z)/(l - z)^n. \tag{8}$$

С помощью этой производящей функции можно легко продолжить табл. 1 и убедиться, что числа, расположенные под ступенчатой линией в таблице, удовлетворяют соотношению

$$I_n(k) = I_n(k-1) + I_{n-1}(k) \quad \text{при } k < n. \quad (9)$$

(Для чисел *над* ступенчатой линией это соотношение *не* выполняется.) Более сложные рассуждения (см. упр. 14) показывают, что на самом деле имеют место формулы

$$\begin{aligned} I_n(2) &= \binom{n}{2} - 1, n \geq 2; \\ I_n(3) &= \binom{n+1}{3} - \binom{n}{1}, n \geq 3; \\ I_n(4) &= \binom{n+2}{4} - \binom{n+1}{2}, n \geq 4; \\ I_n(5) &= \binom{n+3}{5} - \binom{n+2}{3} + 1, n \geq 5. \end{aligned}$$

Общая формула для $I_n(k)$ содержит около $1.6\sqrt{k}$ слагаемых:

$$\begin{aligned} I_n(k) &= \binom{n+k-2}{k} - \binom{n+k-3}{k-2} + \binom{n+k-6}{k-5} + \binom{n+k-8}{k-7} - \dots \\ &+ (-1)^j \left(\binom{n+k-u_j-1}{k-u_j} + \binom{n+k-u_j-j-1}{k-u_j-j} \right) + \dots, \quad n \geq k, \end{aligned} \quad (10)$$

где $u_j = (3j^2 - j)/2$ — так называемое ”пятиугольное число”.

Разделив $G_n(z)$ на $n!$, получим производящую функцию $g_n(z)$ распределения вероятностей числа инверсий в случайной перестановке n элементов. Она равна произведению

$$g_n(z) = h_1(z)h_2(z) \dots h_n(z), \quad (11)$$

где $h_k(z) = (1+z+z^2+\dots+z^{k-1})/k$ — производящая функция равномерного распределения случайной величины, принимающей целые неотрицательные значения, меньшие k . Отсюда

$$\begin{aligned} \text{mean}(g_n) &= \text{mean}(h_1) + \text{mean}(h_2) + \dots + \text{mean}(h_n) = \\ &= 0 + \frac{1}{2} + \dots + \frac{n-1}{2} = \frac{n(n-1)}{4}; \end{aligned} \quad (12)$$

$$\begin{aligned} \text{var}(g_n) &= \text{var}(h_1) + \text{var}(h_2) + \dots + \text{var}(h_n) = \\ &= 0 + \frac{1}{4} + \dots + \frac{n^2-1}{12} = \frac{n(2n+5)(n-1)}{72} \end{aligned} \quad (13)$$

Таким образом, среднее число инверсий довольно велико — около $\frac{1}{4}n^2$; стандартное отклонение также весьма велико — около $\frac{1}{6}n^{3/2}$.

В качестве интересного завершения изучения инверсий рассмотрим одно замечательное открытие, принадлежащее П. А. Мак-Магону [*Amer. J. Math.*, **35** (1913), 281–322]. Определим *индекс* перестановки $a_1 a_2 \dots a_n$ как сумму всех j , таких, что $a_j > a_{j+1}$, $1 \leq j < n$. Например, индекс перестановки 5 9 1 8 2 6 4 7 3 равен $2 + 4 + 6 + 8 = 20$. Индекс случайно совпал с числом инверсий. Если составить список всех 24 перестановок множества $\{1, 2, 3, 4\}$, а именно

Перестановка	Индекс	Инверсии	Перестановка	Индекс	Инверсии
1 2 3 4	0	0	3 1 2 4	1	2
1 2 4 3	3	1	3 1 4 2	4	3
1 3 2 4	2	1	3 2 1 4	3	3
1 3 4 2	2	2	3 2 4 1	4	4
1 4 2 3	2	2	3 4 1 2	2	4
1 4 3 2	2	3	3 4 2 1	5	5
2 1 3 4	1	1	4 1 2 3	1	3
2 1 4 3	3	2	4 1 3 2	2	4
2 3 1 4	2	2	4 2 1 3	3	4
2 3 4 1	1	3	4 2 3 1	4	5
2 4 1 3	2	8	4 3 1 2	3	5
2 4 3 1	1	5	4 3 2 1	6	6

то видно, что число перестановок, имеющих данный индекс k , равно числу перестановок, имеющих k инверсий.

На первый взгляд этот факт может показаться почти очевидным, однако после некоторых размышлений он начинает казаться чуть ли не мистическим, и не видно никакого простого прямого его доказательства. Мак-Магон нашел следующее остроумное косвенное доказательство: пусть $J(a_1 a_2 \dots a_n)$ —индекс перестановки $a_1 a_2 \dots a_n$, и соответствующая производящая функция есть

$$H_n(z) = \sum z^{J(a_1 a_2 \dots a_n)}, \tag{14}$$

где сумма берется по всем перестановкам множества $\{1, 2, \dots, n\}$. Мы хотели бы доказать, что $H_n(z) = G_n(z)$. Для этого определим взаимно однозначное соответствие между n -ками (q_1, q_2, \dots, q_n) неотрицательных целых чисел, с одной стороны, и упорядоченными парами n -ок

$$((a_1, a_2, \dots, a_n), (p_1, p_2, \dots, p_n)),$$

с другой стороны; здесь $a_1 a_2 \dots a_n$ —перестановка множества $\{1, 2, \dots, n\}$ и $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$. Это соответствие будет удовлетворять условию

$$q_1 + q_2 + \dots + q_n = J(a_1 a_2 \dots a_n) + (p_1 + p_2 + \dots + p_n). \tag{15}$$

Производящая функция $\sum z^{q_1+q_2+\dots+q_n}$, где сумма берется по всем n -кам неотрицательных целых чисел (q_1, q_2, \dots, q_n) , равна $Q_n(z) = 1/(1-z)^z$; а производящая функция $\sum z^{p_1+p_2+\dots+p_n}$, где сумма берется по всем n -кам целых чисел (p_1, p_2, \dots, p_n) , таких, что $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$, равна, как показано в упр. 15,

$$P_n(z) = 1/(1-z)(1-z^2)\dots(1-z^n). \tag{16}$$

Существование взаимно однозначного соответствия, которое удовлетворяет условию (15) и которое мы собираемся установить, доказывает равенство $Q_n(z) = H_n(z)P_n(z)$, т.е.

$$H_n(z) = Q_n(z)/P_n(z) = G_n(z).$$

Требуемое соответствие определяется с помощью алгоритма "сортировки". Начав с пустого списка, при $k = 1, 2, \dots, n$ (в таком порядке) вставляем в этот список следующим образом: пусть после $k - 1$ шагов в списке содержатся элементы p_1, p_2, \dots, p_{k-1} , где $p_1 \geq p_2 \geq \dots \geq p_{k-1}$, и определена перестановка $a_1 a_2 \dots a_n$ множества $\{n, n - 1, \dots, n - k + 2\}$. Пусть j —единственное целое число, такое, что $p_j > q_k \geq p_{j+1}$; если $q_k \geq p_1$, то полагаем $j = 0$, а если $p_{k-1} > q_k$, то полагаем $j = k - 1$. Вставим теперь q_k в список между p_j и p_{j+1} , а целое число $(n - k + 1)$ —в перестановку между a_j и a_{j+1} . Продолав это для всех k , получим перестановку $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$ и n -ку чисел (p_1, p_2, \dots, p_n) , таких, что $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$ и

$$p_j > p_{j+1}, \quad \text{если } a_j > a_{j+1}.$$

Наконец, для $1 \leq j < n$ вычтем единицу из всех чисел p_1, \dots, p_j при всех j , таких, что $a_j > a_{j+1}$. Полученная пара $((a_1, a_2, \dots, a_n), (p_1, p_2, \dots, p_n))$ удовлетворяет условию (15).

Пусть, например, $n = 6$ и $(q_1, \dots, q_6) = (3, 1, 4, 0, 0, 1)$. Построение происходит следующим образом:

k	$p_1 \dots p_k$	$a_1 \dots a_k$
1	3	6
2	3 1	6 5
3	4 3 1	4 6 5
4	4 3 1 0	4 6 5 3
5	4 3 1 0 0	4 6 5 2 3
6	4 3 1 1 0 0	4 6 1 5 2 3

После заключительной корректировки получаем $(p_1, \dots, p_6) = (2, 1, 0, 0, 0, 0)$.

Нетрудно проверить, что этот процесс обратим; таким образом, требуемое соответствие установлено и теорема Мак-Магона доказана. Аналогичное взаимно однозначное соответствие встретится нам в п. 5.1.4.

Упражнения

- [10] Какова таблица инверсий для перестановки 2 7 1 8 4 5 9 3 6? Какой перестановке соответствует таблица инверсий 5 0 1 2 1 2 0 0?

2. [M15] Решением задачи Иосифа, сформулированной в упр. 1.3.2–22, является перестановка множества $\{1, 2, \dots, n\}$; решение для приведенного там примера ($n = 8, m = 4$)—перестановка 5 4 6 1 3 8 7 2. Соответствующая этой перестановке таблица инверсий—3 6 3 1 0 0 1 0. Найдите простое рекуррентное соотношение для элементов $b_1 b_2 \dots b_n$ таблицы инверсий в общей задаче Иосифа для n человек, если казнят каждого m -го человека.
3. [18] Пусть перестановке $a_1 a_2 \dots a_n$ соответствует таблица инверсий $b_1 b_2 \dots b_n$; какой перестановке $\bar{a}_1 \bar{a}_2 \dots \bar{a}_n$ соответствует таблица инверсий

$$(n - 1 - b_1)(n - 2 - b_2) \dots (0 - b_n)?$$

- >4. [20] Придумайте алгоритм, годный для реализации на ЭВМ, который по данной таблице инверсий $b_1 b_2 \dots b_n$, удовлетворяющей условиям (3), строил бы соответствующую перестановку $a_1 a_2 \dots a_n$. [Указание: вспомните методы работы со связанной памятью.]
5. [35] Для выполнения на типичной ЭВМ алгоритм из упр. 4 требует времени, приблизительно пропорционального n^2 ; можно ли создать алгоритм, время работы которого было бы существенно меньше n^2 ?
- >6. [26] Придумайте алгоритм вычисления таблицы инверсий $b_1 b_2 \dots b_n$, соответствующей данной перестановке $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$, время работы которого на типичной ЭВМ было бы порядка $n \log n$.
7. [20] Помимо таблицы $b_1 b_2 \dots b_n$, определенной в этом пункте, можно определить некоторые другие типы таблиц инверсий, соответствующих данной перестановке $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$. В этом упражнении мы рассмотрим три других типа таблиц инверсий, которые возникают в приложениях.

Пусть c_j —число инверсий, первая компонента которых равна j , т. е. число элементов, меньших j и расположенных правее j . [Перестановке (1) соответствует таблица 0 0 0 1 4 2 1 5 7; ясно, что $0 \leq c_j < j$.] Пусть $B_j = b_{a_j}$ и $C_j = c_{a_j}$.

Покажите, что при $1 \leq j \leq n$ справедливы неравенства $0 \leq B_j < j$ и $0 \leq C_j \leq n - j$; покажите также, что перестановку $a_1 a_2 \dots a_n$ можно однозначно определить, если задана или таблица $c_1 c_2 \dots c_n$, или $B_1 B_2 \dots B_n$, или $C_1 C_2 \dots C_n$.

8. [M24] Сохраним обозначения упр. 7; пусть $a'_1 a'_2 \dots a'_n$ —перестановка, обратная к $a_1 a_2 \dots a_n$ и пусть соответствующие ей таблицы инверсий— $b'_1 b'_2 \dots b'_n, c'_1 c'_2 \dots c'_n, B'_1 B'_2 \dots B'_n$ и $C'_1 C'_2 \dots C'_n$. Найдите как можно больше интересных соотношений между $a_j, b_j, c_j, B_j, C_j, a'_j, b'_j, c'_j, B'_j, C'_j$.
- >9. [M21] Докажите, что в обозначениях упр. 7 перестановка $a_1 a_2 \dots a_n$ обратна самой себе тогда и только тогда, когда $b_j = C_j$ при $1 \leq j \leq n$.
10. [BM20] Рассмотрите рис. 1 как многогранник в трехмерном пространстве. Чему равен диаметр усеченного октаэдра (расстояние между вершинами 1234 и 4321), если все ребра имеют единичную длину?
11. [M25] (а) Пусть $\pi = a_1 a_2 \dots a_n$ —перестановка множества $\{1, 2, \dots, n\}$, $E(x) = \{(a_i, a_j) | i < j, a_i > a_j\}$ —множество ее инверсий, а

$$\bar{E}(\pi) = \{(a_i, a_j) | i > j, a_i > a_j\}$$

—множество ее "неинверсий". Докажите, что $E(\pi)$ и $\bar{E}(\pi)$ транзитивны. [Множество S упорядоченных пар называется *транзитивным*, если для любых (a, b) и (b, c) , принадлежащих S , пара (a, c) также принадлежит S .] (б) Обратно, пусть E —любое транзитивное подмножество множества $T = \{(x, y) | 1 \leq y < x \leq n\}$, дополнение которого $T \setminus E$ транзитивно. Докажите, что существует перестановка π , такая, что $E(\pi) = E$.

12. [M28] Используя обозначения предыдущего упражнения, докажите, что если π_1 и π_2 —перестановки, а E —минимальное транзитивное множество, содержащее $E(\pi_1) \cup E(\pi_2)$, то \bar{E} —тоже транзитивное множество. [Следовательно, если мы будем говорить, что π_1 находится "над" π_2 , когда $E(\pi_1) \subseteq E(\pi_2)$, то определена решетка перестановок; существует единственная "самая низкая" перестановка, находящаяся "над" двумя данными перестановками. Диаграмма решетки при $n = 4$ представлена на рис. 1.]

Упражнения

1. [M23] Известно, что в разложении определителя половина членов выписывается со знаком $+$, а половина—со знаком $-$. Другими словами, при $n \geq 2$ перестановок с *четным* числом инверсий ровно столько же, сколько с *нечетным*. Покажите, что вообще при $n \geq t$ количество перестановок с числом инверсий, конгруэнтным $t \pmod t$, равно $n!/t$, независимо от того, каково целое число t .

2. [M24] (Ф. Франклин.) Разбиение числа n на k различных частей— это представление n в виде суммы $n = p_1 + p_2 + \dots + p_k$, где $p_1 > p_2 > \dots > p_k > 0$. Например, разбиения числа 7 на различные части таковы: 7, 6 + 1, 5 + 2, 4 + 3, 4 + 2 + 1.

Picture: Рис. 2. Соответствие Франклина между разбиениями на различные части.

Пусть $f_k(n)$ —число разбиений n на k различных частей. Докажите, что $\sum_k (-1)^k f_k(n) = 0$, если только n не представляется в виде $(3j^2 \pm j)/2$ при некотором неотрицательном целом j ; в этом случае сумма равна $(-1)^j$. Например, для $n = 7$ сумма равна $-1 + 3 - 1 = 1$, потому что $7 = (3 \cdot 2^2 + 2)/2$. [Указание. Представьте разбиения в виде массива точек, в i -й строке которого имеется p_i точек, $1 \leq i \leq k$. Найдите наименьшее j , такое, что $p_{j+1} < p_j - 1$, и обведите крайние правые точки первых j строк. Если $j < p_k$, то эти j точек можно, как правило изъять из массива, повернуть на 45° и поместить в новую, $(k + 1)$ -ю строку. С другой стороны, если $j \geq p_k$, то обычно можно изъять из массива k -ю строку точек, повернуть ее на 45° и поместить справа от обведенных точек (рис. 2). В результате этого процесса в большинстве случаев разбиения с четным числом строк и разбиения с нечетным числом строк группируются в пары, таким образом, в сумме надо учитывать только непарные разбиения.]

Замечание. В качестве следствия получаем формулу Эйлера

$$\begin{aligned} (1 - z)(1 - z^2)(1 - z^3) \dots &= 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \dots = \\ &= \sum_{-\infty < j < \infty} (-1)^j z^{(3j^2+j)/2}. \end{aligned}$$

Так как производящая функция для обычных разбиений (не обязательно на различные части) равна $\sum p(n)z^n = 1/(1 - z)(1 - z^2)(1 - z^3) \dots$, то получаем неочевидное рекуррентное соотношение для числа разбиений $p(n) = p(n - 1) + p(n - 2) - p(n - 5) - p(n - 7) + p(n + 12) + p(n + 15) - \dots$.

3. [M29] Докажите, что (16)—производящая функция для числа разбиений на не более чем n частей, т. е. докажите, что коэффициент при z^m в $1/(1 - z)(1 - z^2) \dots (1 - z^n)$ равен числу способов представить m в виде суммы $p_1 + p_2 + \dots + p_n$, где $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$. [Указание. Нарисуйте точки, как в упр. 14, и покажите, что существует взаимно однозначное соответствие между n -ками чисел (p_1, p_2, \dots, p_n) , такими, что $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$, и последовательностями (P_1, P_2, P_3, \dots) , такими, что $n \geq P_1 \geq P_2 \geq P_3 \geq \dots \geq 0$, обладающее тем свойством, что $p_1 + p_2 + \dots + p_n = P_1 + P_2 + P_3 + \dots$. Иными словами, разбиениям на не более чем n частей соответствуют разбиения на части, не превосходящие n .]
4. [M25](Л. Эйлер.) Докажите следующие тождества, интерпретируя обе части соотношений в терминах разбиений:

$$\begin{aligned} \prod_{k \geq 0} \frac{1}{(1 - q^k z)} &= \frac{1}{(1 - z)(1 - qz)(1 - q^2 z) \dots} = \\ &= 1 + \frac{z}{1 - q} + \frac{z^2}{(1 - q)(1 - q^2)} + \dots = \sum_{n \geq 0} z^n / \prod_{1 \leq k \leq n} (1 - q^k). \end{aligned}$$

$$\begin{aligned} \prod_{k \geq 0} (1 + q^k z) &= (1 + z)(1 + qz)(1 + q^2 z) \dots = \\ &= 1 + \frac{z}{1 - q} + \frac{z^2 q}{(1 - q)(1 - q^2)} + \dots = \\ &= \sum_{n \geq 0} z^n q^{n(n-1)/2} / \prod_{1 \leq k \leq n} (1 - q^k). \end{aligned}$$

5. [20] Каковы 24 четверки (q_1, q_2, q_3, q_4) , для которых в соответствии Мак-Магона, определенном в конце этого пункта, $(p_1, p_2, p_3, p_4) = (0, 0, 0, 0)$?
6. [M30] (Т. Хиббард, *САСМ*, 6 (1963), 210.) Пусть $n > 0$, и предположим, что последовательность n -битовых целых чисел X_0, \dots, X_{2^n-1} длины 2^n получена случайным образом, причем каждый бит каждого числа независимо принимает значение 1 с вероятностью p . Рассмотрим последовательность $X_0 \oplus 0, X_1 \oplus 1, \dots, X_{2^n-1} \oplus (2^n - 1)$, где \oplus —операция "исключающее или" над бинарными представлениями. Так, если $p = 0$, то последовательность будет $0, 1, \dots, 2^n - 1$, а если $p = 1$, то она будет $2^n - 1, \dots, 1, 0$; если же $p = \frac{1}{2}$, то каждый элемент последовательности—случайное число между 0 и $2^n - 1$. Вообще же при разных p это хороший способ получения последовательности случайных целых чисел со смещенным числом инверсий, в то время как распределение элементов последовательности, рассматриваемой как единое целое, равномерно.

- Определите среднее число инверсий в такой последовательности как функцию от вероятности p .
- [М36] (Д. Фоата.) Дайте прямое доказательство теоремы Мак-Магона об индексах: найдите точное взаимно однозначное соответствие, которое переводит перестановку n элементов, имеющую индекс k , в перестановку, имеющую k инверсий и тот же самый крайний правый элемент.
 - [М43] Следующее знаменитое тождество, принадлежащее Якоби [Fundamenta Nova Theoriæ Functionum Ellipticorum (1829), § 64], лежит в основе многих замечательных соотношений, содержащих эллиптические функции:

$$\begin{aligned} \prod_{k \geq 1} (1 - u^k v^{k-1})(1 - u^{k-1} v^k)(1 - u^k v^k) &= \\ &= (1 - u)(1 - v)(1 - uv)(1 - u^2 v)(1 - uv^2)(1 - u^2 v^2) \dots = \\ &= 1 - (u + v) + (u^3 v + uv^3) - (u^6 v^3 + u^3 v^6) + \dots = \\ &= 1 + \sum_{n \geq 1} (-1)^n (u^{(n+1)n/2} v^{(n-1)n/2} + u^{(n-1)n/2} v^{(n+1)n/2}). \end{aligned}$$

Если, например, положить $u = z$, $v = z^2$, то получится формула Эйлера из упр. 14. Если положить $z = \sqrt{u/v}$, $q = \sqrt{uv}$, то получим

$$\prod_{k \geq 1} (1 - q^{2k-1} z)(1 - q^{2k-1} z^{-1})(1 - q^{2k}) = \sum_{-\infty < n < \infty} (-1)^n z^n q^{n^2}.$$

Существует ли комбинаторное доказательство тождества Якоби, аналогичное доказательству Франклина для частного случая упр. 14? (Таким образом, нужно рассмотреть "комплексные разбиения")

$$m + ni = (p_1 + q_1 i) + (p_2 + q_2 i) + \dots + (p_k + q_k i),$$

где $p_j + q_j i$ — различные ненулевые комплексные числа; p_j, q_j — неотрицательные целые числа, причем $|p_j - q_j| \leq 1$. Согласно тождеству Якоби, число таких представлений с четными k равно числу представлений с нечетными k , если только m и n не являются соседними треугольными числами! Какими еще замечательными свойствами обладают комплексные разбиения?

- >9. [М25] (Г. Д. Кнотт.) Покажите, что перестановку $a_1 \dots a_n$ можно получить с помощью стека в смысле упр. 2.2.1–5 или 2.3.1–6 тогда и только тогда, когда $C_j \leq C_{j+1} + 1$ при $1 \leq j < n$ (см. обозначения в упр. 7).
- [М28] (К. Мейер.) Мы знаем, что если m и n — взаимно простые числа, то последовательность $(m \bmod n) (2m \bmod n) \dots ((n-1)m \bmod n)$ представляет собой перестановку множества $\{1, 2, \dots, n-1\}$. Покажите, что число инверсий в этой перестановке можно выразить через суммы Дедекинда (ср. с п. 3.3.3).

5.1.2. *Перестановки мультимножества

До сих пор мы рассматривали перестановки *множества* элементов; это частный случай перестановок *мультимножества*. (Мультимножество — это то же самое, что и множество, но в нем могут содержаться одинаковые элементы. Некоторые основные свойства мультимножеств обсуждались в п. 4.6.3.)

Рассмотрим, например, мультимножество

$$M = \{a, a, a, b, b, c, d, d, d, d\}, \quad (1)$$

в котором содержится 3 элемента a , 2 элемента b , 1 элемент c и 4 элемента d . Повторения элементов можно указать и другим способом:

$$M = \{3 \cdot a, 2 \cdot b, c, 4 \cdot d\}. \quad (2)$$

Перестановка мультимножества — это некоторое расположение его элементов в ряд, например

$$c a b d d a b d a d.$$

С другой стороны, такую последовательность можно назвать цепочкой букв, содержащей 3 буквы a , 2 буквы b , 1 букву c и 4 буквы d .

Сколько существует перестановок мультимножества M ? Если бы мы рассматривали все элементы M как различные, обозначив их $a_1, a_2, a_3, b_1, b_2, c_1, d_1, d_2, d_3, d_4$, то получили бы $10! = 3\,628\,800$

перестановок, но после отбрасывания индексов многие из них оказались бы одинаковыми. Фактически каждая перестановка M встретила бы ровно $3!2!1!4! = 288$ раз, поскольку в любой перестановке M индексы при буквах a можно расставить 3! способами, при b (независимо)—2! способами, при c —одним способом, а при d —соответственно 4! способами. Поэтому число перестановок M равно

$$\frac{10!}{3!2!1!4!} = 12\,600.$$

В применении к общему случаю те же рассуждения доказывают, что число перестановок любого мультимножества равно мультиномиальному коэффициенту

$$\binom{n}{n_1, n_2, \dots} = \frac{n!}{n_1!n_2!\dots},$$

где n_1 —число элементов первого типа, n_2 —число элементов второго типа и т. д., а $n = n_1 + n_2 + \dots$ —общее число элементов.

Количество перестановок множества было известно еще в древние времена. В древнееврейской Книге Творения (около 100 г. н. э.)⁵, наиболее раннем литературном произведении иудейского философского мистицизма, даны верные значения первых семи факториалов, после чего говорится: "Продолжай и получишь числа, которые уста не могут произнести, а ухо не может воспринять." [Sefer Yezirah, ed. by R. Mordecai Atia (Jerusalem: Sh. Monson, 1962), стих 52 (стр. 107–108); ср. также с Solomon Gandz, Studies in Hebrew Astronomy and Mathematics (New York: Ktav, 1970), 494–496. Книга Творения была основана на считавшихся важными отношениях между семью планетами, семью согласными звуками с двойным произношением, семью отверстиями в голове человека и семью днями сотворения мира.] Это первый известный в истории подсчет числа перестановок. Второй встречается в индийском классическом произведении Ануйогадвара-сутра (около 500 г. н. э.), привело 97, где приводится формула числа перестановок шести элементов, которые не расположены ни в возрастающем, ни в убывающем порядке:

$$6 \times 5 \times 4 \times 3 \times 2 \times 1 = 2.$$

[См. G. Chakravarti, Bull. Calcutta Math. Soc., 24 (1932), 79–88. Ануйогадвара-сутра—одна из книг канон джайнизма, религиозной секты, распространенной в Индии.]

Соответствующее правило для мультимножеств впервые, по-видимому, встречается в книге Лилавати, написанной Бхаскарой Ахарьей (ок. 1150 г.), разд. 270–271. У Бхаскары это правило сформулировано весьма сжато и проиллюстрировано лишь двумя простыми примерами $\{2, 2, 1, 1\}$ и $\{4, 8, 5, 5, 5\}$. В результате в английском переводе это правило не сформулировано корректно, впрочем, имеются некоторые сомнения относительно того, понимал ли сам Бхаскара, о чем он говорил. Вслед за этим правилом Бхаскара приводит интересную формулу

$$\frac{(4 + 8 + 5 + 5 + 5) \times 120 \times 11111}{5 \times 6}$$

для суммы 20 чисел $48\,555 + 45855 + \dots$.

Верное правило для нахождения числа перестановок в случае, когда только один элемент может повторяться, найдено независимо немецким ученым иезуитом Атанасиусом Кирхером в его многотомном труде о музыке Musurgia Universalis (Rome, 1650), том 2, стр. 5–7. Кирхера интересовал вопрос о количестве мелодий, которые можно создать из данного набора нот; для этого он придумал то, что называл "музарифметикой". На стр. 18–21 своего труда он дает верное значение числа перестановок мультимножества $\{m \cdot C, n \cdot D\}$ при нескольких значениях m и n , хотя описал он свой метод вычислений лишь для случая $n = 1$.

Общее правило (3) появилось позже в книге Жана Престэ Eléments de Mathématique (Paris, 1675), стр. 351–352, которая содержит одно из первых изложений комбинаторной математики, написанных в западной Европе. Престэ верно сформулировал правило для случая произвольного мультимножества, но проиллюстрировал его лишь простым примером $\{a, a, b, b, c, c\}$. Он особо отметил, что деление на сумму факториалов, которое он считал естественным обобщением правила Кирхера, было бы ошибкой. Несколько лет спустя Джон Валлис в своей книге Treatise of Algebra (Oxford, 1685), том 2, стр. 117–118, обсудил это правило несколько более подробно.

⁵ Книга Творения (Йоцира)—одна из основополагающих книг каббалистики.— Прим. перев.

В 1965 г. Доминик Фоата ввел одно интересное понятие, так называемое "соединительное произведение"⁶, которое позволило распространить многие известные результаты, касающиеся обычных перестановок, на общий случай перестановок мультимножества. [См. Publ. Inst. Statistique, Univ. Paris, 14 (1965), 81–241. а также Lecture Notes in Math., 85 (Springer, 1969).] Предполагая, что элементы мультимножества каким-то способом линейно упорядочены, можно рассмотреть *двустрочное обозначение*, например

$$\begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}.$$

Здесь верхняя строка содержит элементы M в неубывающем порядке, и нижняя—это сама перестановка. *Соединительное произведение* $\alpha \top \beta$ двух перестановок мультимножеств α и β —это перестановка, которая получается, если (а) взять двустрочные обозначения для α и β , (б) записать соответствующие строки в одну и (с) отсортировать столбцы так, чтобы элементы верхней строки расположились в неубывающем порядке. Сортировка должна быть устойчивой в том смысле, что взаимное расположение элементов нижней строки сохраняется, если соответствующие элементы верхней строки равны. Например, $c a d a b \top b d d a d = c a b d d a b d a d$, так как

$$\begin{pmatrix} a & a & b & c & d \\ c & a & d & a & b \end{pmatrix} \top \begin{pmatrix} a & b & d & d & d \\ b & d & d & a & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}. \quad (5)$$

Нетрудно видеть, что операция соединительного произведения ассоциативна, т. е.

$$\alpha \top \beta \top \gamma = \alpha \top (\beta \top \gamma), \quad (6)$$

и что она подчиняется законам сокращения

$$\begin{aligned} \text{если } \pi \top \alpha = \pi \top \beta, \text{ то } \alpha = \beta, \\ \text{если } \alpha \top \pi = \beta \top \pi, \text{ то } \alpha = \beta. \end{aligned}$$

Существует "единичный элемент"

$$\alpha \top \varepsilon = \varepsilon \top \alpha = \alpha, \quad (8)$$

где ε —пустая перестановка, "расположение в ряд" элементов пустого множества. Закон коммутативности, вообще говоря, не выполняется (см. упр. 2), тем не менее

$$\alpha \top \beta = \beta \top \alpha, \quad \text{если } \alpha \text{ и } \beta \text{ не содержат общих букв.} \quad (9)$$

Аналогичным способом и понятие *цикла* можно распространить на случай, когда элементы могут повторяться. Будем записывать в виде

$$(x_1 x_2 \dots x_n) \quad (10)$$

перестановку, двустрочное представление которой получается путем устойчивой сортировки столбцов

$$\begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_2 & x_3 & \dots & x_1 \end{pmatrix} \quad (11)$$

по верхним элементам. Например,

$$\begin{aligned} (d b d d a c a a b d) &= \begin{pmatrix} d & b & d & d & a & c & a & a & b & d \\ b & d & d & a & c & a & a & b & d & d \end{pmatrix} = \\ &= \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix} \end{aligned}$$

так что перестановка (4) фактически является циклом. Мы могли бы описать этот цикл словесно, сказав что-нибудь вроде "d переходит в b, переходит в d, переходит в d, переходит в . . . переходит в d и возвращается обратно". Заметим, что эти обобщенные циклы не обладают всеми свойствами обычных циклов; $(x_1 x_2 \dots x_n)$ не обязательно то же самое, что и $(x_2 \dots x_n x_1)$.

В п. 1.3.3 мы выяснили, что каждую перестановку множества можно единственным с точностью до порядка сомножителей образом представить в виде произведения непересекающихся циклов, где произведение перестановок определяется законом композиции. Легко видеть, что *произведение непересекающихся циклов—то же самое, что их соединительное произведение*; это наводит на мысль

⁶ В оригинале—"intercalation product".—Прим. перев.

о том, что можно будет обобщить полученные ранее результаты, если найти единственное (в каком-то смысле) представление и для произвольной перестановки мультимножества в виде соединительного произведения циклов. В действительности существуют по крайней мере два естественных способа сделать это, и каждый из них имеет важные приложения.

Равенство (5) дает один способ представления $c a b d d a b d a d$ в виде соединительного произведения более коротких перестановок; рассмотрим общую задачу о нахождении всех разложений $\pi = \alpha \top \beta$ данной перестановки π . Для исследования этого вопроса полезно рассмотреть конкретную перестановку, скажем

$$\pi = \begin{pmatrix} a & a & b & b & b & b & b & c & c & c & d & d & d & d & d \\ d & b & c & b & c & a & c & d & a & d & d & b & b & b & d \end{pmatrix}, \quad (12)$$

Если можно записать π в виде $\alpha \top \beta$, где α содержит по крайней мере одну букву a , то самое левое a в верхней строке двустрочного представления α должно оказаться над d , значит, перестановка α должна содержать по крайней мере одну букву d . Если взглянуть теперь на самое левое d в верхней строке α , то увидим точно так же, что оно должно оказаться над d , значит, в α должны содержаться по меньшей мере две буквы d . Посмотрев на второе d , видим, что α содержит также b . Одно-единственное предположение о том, что α есть левый сомножитель π , содержащий букву a , приводит к такому промежуточному результату:

$$\begin{pmatrix} a & & b & & d & d & & & & & & & & & & \\ & \dots & & \dots & & & & \dots & & & & & & & & \\ d & & & & & d & b & & & & & & & & & \end{pmatrix}. \quad (13)$$

Продолжая рассуждать точно так же и далее, обнаружим, что буква b в верхней строке (13) должна оказаться над c и т. д. В конце концов этот процесс вновь приведет нас к букве a , и мы сможем, если захотим, отождествить ее с первой буквой a . Только что проведенное рассуждение, по существу, доказывает, что любой левый сомножитель в разложении перестановки (12), содержащий a , имеет вид $(d d b c d b b c a) \top \alpha'$, где α' — некоторая перестановка. (Удобно записывать a не в начале, а в конце цикла; это допустимо, поскольку буква a только одна.) Аналогично, если бы мы предположили, что α содержит букву b , то вывели бы, что $\alpha = (c d d b) \top \alpha''$, где α'' — некоторая перестановка.

В общем случае эти рассуждения показывают, что *если есть какое-нибудь разложение $\alpha \top \beta = \pi$, где α содержит данную букву y , то существует единственный цикл вида*

$$(x_1 \dots x_n y), \quad n \geq 0, x_1, \dots, x_n \neq y, \quad (14)$$

который является левым сомножителем в разложении перестановки α . Такой цикл легко отыскать, зная π и y ; это самый короткий левый сомножитель в разложении перестановки π , содержащий букву y . Одно из следствий этого наблюдения дает

Теорема А. *Пусть элементы мультимножества M линейно упорядочены отношением " $<$ ". Каждая перестановка π мультимножества M имеет единственное представление в виде соединительного произведения*

$$\pi = (x_{11} \dots x_{1n_1} y_1) \top (x_{21} \dots x_{2n_2} y_2) \top \dots \top (x_{t1} \dots x_{tn_t} y_t), \quad t \geq 0, \quad (15)$$

удовлетворяющее следующим двум условиям:

$$\begin{aligned} y_1 &\leq y_2 \leq \dots \leq y_t; \\ y_i &< x_{ij} \text{ при } 1 \leq j \leq n_i, 1 \leq i \leq t. \end{aligned} \quad (16)$$

(Иными словами, в каждом цикле последний элемент меньше любого другого, и последние элементы циклов образуют неубывающую последовательность.)

Доказательство. При $\pi = \varepsilon$ получим требуемое разложение, положив $t = 0$. В противном случае пусть y_1 — минимальный элемент π ; определим $(x_{11} \dots x_{1n_1} y_1)$ — самый короткий левый сомножитель разложения π , содержащий y_1 , как в рассмотренном примере. Теперь $\pi = (x_{11} \dots x_{1n_1} y_1) \top \rho$, где ρ — некоторая перестановка; применив индукцию по длине перестановки, можем написать

$$\rho = (x_{21} \dots x_{2n_2} y_2) \top \dots \top (x_{t1} \dots x_{tn_t} y_t), \quad t \geq 1,$$

где условия (16) выполнены. Тем самым доказано существование такого разложения.

Докажем единственность разложения (15), удовлетворяющего условиям (16). Ясно, что $t = 0$ тогда и только тогда, когда π — пустая перестановка ε . При $t > 0$ из (16) следует, что y_1 — минимальный

элемент перестановки π и что $(x_{11} \dots x_{1n_1} y_1)$ — самый короткий левый сомножитель, содержащий y_1 . Поэтому $(x_{11} \dots x_{1n_1} y_1)$ определяется однозначно; доказательство единственности такого представления завершается применением индукции и законов сокращения (7). ■

Например, "каноническое" разложение перестановки (12), удовлетворяющее данным условиям, таково:

$$(d d b c d b b c a) \top (b a) \top (c d b) \top (d), \tag{17}$$

если $a < b < c < d$.

Важно отметить, что на самом деле в этом определении можно отбросить скобки и знаки операции \top , и это не приведет к неоднозначности! Каждый цикл заканчивается появлением наименьшего из оставшихся элементов. Таким образом, наше построение связывает с исходной перестановкой

$$\pi' = d d b c d b b c a b a c d b d$$

перестановку

$$\pi = d b c b c a c d a d d b b b d.$$

Если в двустрочном представлении π содержится столбец вида $\begin{smallmatrix} y \\ x \end{smallmatrix}$, где $x < y$, то в связанной с π перестановке присутствует соответствующая пара соседних элементов $\dots y x \dots$. Так, в нашем примере π содержит три столбца вида $\begin{smallmatrix} d \\ b \end{smallmatrix}$, а в π' трижды встречается пара $d b$. Вообще из этого построения вытекает замечательная

Теорема В. Пусть M — мультимножество. Существует взаимно однозначное соответствие между перестановками M , такое, что если π соответствует π' , то выполняются следующие условия:

- а) крайний левый элемент π' равен крайнему левому элементу π ;
- б) для всех пар участвующих в перестановке элементов (x, y) ,

таких, что $x < y$, число вхождений столбца $\begin{smallmatrix} y \\ x \end{smallmatrix}$ в двустрочное представление перестановки π равно числу случаев, когда в перестановке π' элемент x следует непосредственно за y .

Если M — множество, то это, по существу, "нестандартное соответствие", обсуждавшееся в конце п. 1.3.3, с незначительными изменениями. Более общий результат теоремы В полезен при подсчете числа перестановок специальных типов, поскольку часто проще решить задачу с ограничениями, наложенными на двустрочное представление, чем эквивалентную задачу с ограничениями на пары соседних элементов.

П. А. Мак-Магон рассмотрел задачи этого типа в своей выдающейся книге *Combinatory Analysis* (Cambridge Univ. Press, 1915), том 1, стр. 168–186. Он дал конструктивное доказательство теоремы В в частном случае, когда M содержит элементы лишь двух различных типов, скажем a и b , его построение для этого случая, по существу, совпадает с приведенным здесь, но представлено в совершенно ином виде. Для случая трех различных элементов a, b, c Мак-Магон дал сложное неконструктивное доказательство теоремы В; общий случай впервые доказал Фоата в 1965 г.

В качестве нетривиального примера применения теоремы В найдем число цепочек букв a, b, c , содержащих ровно

$$\begin{aligned} & A \text{ вхождений буквы } a; \\ & B \text{ вхождений буквы } b; \\ & C \text{ вхождений буквы } c; \\ & k \text{ вхождений пары стоящих рядом букв } c a; \\ & l \text{ вхождений пары стоящих рядом букв } c b; \\ & m \text{ вхождений пары стоящих рядом букв } b a; \end{aligned} \tag{18}$$

Из теоремы следует, что это то же самое, что найти число двустрочных массивов вида

$$\begin{array}{c} \begin{array}{ccc} \overbrace{a \dots a}^A & \overbrace{b \dots b}^B & \overbrace{c \dots c}^C \\ \underbrace{\square \dots \square} & \underbrace{\square \dots \square} & \underbrace{\square \dots \square} \end{array} \\ \underbrace{A-k-m \text{ букв } a \quad m \text{ букв } a}_{B-l \text{ букв } b} \quad \underbrace{k \text{ букв } a}_{l \text{ букв } b} \\ \underbrace{\hspace{10em}}_{C \text{ букв } c} \end{array} \tag{19}$$

Буквы a можно расположить во второй строке

$$\binom{A}{A-k-m} \binom{B}{m} \binom{C}{k} \quad \text{способами;}$$

после этого буквы b можно разместить в оставшихся позициях

$$\binom{B+k}{B-l} \binom{C-k}{l} \quad \text{способами.}$$

Остальные свободные места нужно заполнить буквами c ; следовательно, искомое число равно

$$\binom{A}{A-k-m} \binom{B}{m} \binom{C}{k} \binom{B+k}{B-l} \binom{C-k}{l}. \quad (20)$$

Вернемся к вопросу о нахождении всех разложений данной перестановки. Существует ли такой объект, как "простая" перестановка, которая не разлагается на множители, отличные от нее самой и ε ? Обсуждение, предшествующее теореме А, немедленно приводит к выводу о том, что *перестановка будет простой тогда и только тогда, когда она есть цикл без повторяющихся элементов*, так как, если перестановка является таким циклом, наше рассуждение доказывает, что не существует левых множителей, кроме ε и самого цикла. Если же перестановка содержит повторяющийся элемент y , то всегда можно выделить нетривиальный цикл в качестве левого сомножителя, в котором элемент y встречается всего однажды.

Если перестановка не простая, то ее можно разлагать на все меньшие и меньшие части, пока не будет получено произведение простых перестановок. Можно даже показать, что такое разложение единственно с точностью до порядка записи коммутирующих сомножителей.

Теорема С. *Каждую перестановку мультимножества можно записать в виде произведения*

$$\sigma_1 \top \sigma_2 \top \dots \top \sigma_t, \quad t \geq 0, \quad (21)$$

где σ_j — циклы, не содержащие повторяющихся элементов. Это представление единственно в том смысле, что любые два таких представления одной и той же перестановки можно преобразовать одно в другое, последовательно меняя местами соседние непересекающиеся циклы.

Термин "непересекающиеся циклы" относится к циклам, не имеющим общих элементов. В качестве примера можно проверить, что перестановка

$$\begin{pmatrix} a & a & b & b & c & c & d \\ b & a & a & c & d & b & c \end{pmatrix}$$

разлагается на множители ровно пятью способами:

$$\begin{aligned} (a b) \top (a) \top (c d) \top (b c) &= (a b) \top (c d) \top (a) \top (b c) = \\ &= (a b) \top (c d) \top (b c) \top (a) = \\ &= (c d) \top (a b) \top (a) \top (b c) = \\ &= (c d) \top (a b) \top (b c) \top (a). \end{aligned} \quad (22)$$

Доказательство. Нужно установить, что выполняется сформулированное в теореме свойство единственности. Применим индукцию по длине перестановки; тогда достаточно доказать, что если ρ и σ — два различных цикла, не содержащие повторяющихся элементов, и

$$\rho \top \alpha = \sigma \top \beta,$$

то ρ и σ — непересекающиеся циклы, и

$$\alpha = \sigma \top \theta, \quad \beta = \rho \top \theta,$$

где θ — некоторая перестановка.

Пусть y — произвольный элемент цикла ρ , тогда y любого левого сомножителя в разложении $\sigma \top \beta$, содержащего этот элемент y , будет левый сомножитель ρ . Значит, если ρ и σ имеют общий элемент, то цикл σ должен быть кратен ρ ; следовательно, $\sigma = \rho$ (так как они простые), что противоречит

нашему предположению. Следовательно, цикл, содержащий y и не имеющий общих элементов с σ , должен быть левым сомножителем в разложении β . Применив законы сокращения (7), завершим доказательство. ■

В качестве иллюстрации теоремы С рассмотрим перестановки мультимножества $M = \{A \cdot a, B \cdot b, C \cdot c\}$, состоящего из A элементов a , B элементов b и C элементов c . Пусть $N(A, B, C, m)$ — число перестановок мультимножества M , двустрочное представление которых *не содержит* столбцов вида $\begin{smallmatrix} a & b & c \\ a & b & c \end{smallmatrix}$ и содержит ровно m столбцов вида $\begin{smallmatrix} a \\ a \end{smallmatrix}$. Отсюда следует, что имеется ровно $A - m$ столбцов вида $\begin{smallmatrix} a \\ a \end{smallmatrix}$, $B - m$ столбцов вида $\begin{smallmatrix} b \\ b \end{smallmatrix}$, $C - B + m$ столбцов вида $\begin{smallmatrix} c \\ a \end{smallmatrix}$, $C - A + m$ столбцов вида $\begin{smallmatrix} b \\ c \end{smallmatrix}$ и $A + B - C - m$ столбцов вида $\begin{smallmatrix} b \\ a \end{smallmatrix}$; следовательно,

$$N(A, B, C, m) = \binom{A}{m} \binom{B}{C - A + m} \binom{C}{B - m}. \quad (23)$$

Теорема С предлагает другой способ для подсчета этих перестановок: коль скоро столбцы $\begin{smallmatrix} a \\ a \end{smallmatrix}$, $\begin{smallmatrix} b \\ b \end{smallmatrix}$, $\begin{smallmatrix} c \\ c \end{smallmatrix}$ исключены, то в разложении перестановки единственно возможны такие простые множители:

$$(a b), (a c), (b c), (a b c), (a c b). \quad (24)$$

Каждая пара этих циклов имеет хотя бы одну общую букву, значит, разложение единственно. Если цикл $(a b c)$ встречается в разложении k раз, то из нашего предыдущего предположения следует, что $(a b)$ встречается $m - k$ раз, $(b c)$ встречается $C - A + m - k$ раз, $(a c)$ встречается $C - B + m - k$ раз и $(a c b)$ встречается $A + B - C - 2m + k$ раз. Следовательно, $N(A, B, C, m)$ равно числу перестановок этих циклов (мультиномиальному коэффициенту), просуммированному по всем значениям k :

$$\begin{aligned} N(A, B, C, m) &= \sum_k \frac{(C + m - k)!}{(m - k)!(C - A + m - k)!(C - B + m - k)!k!(A + B - C - 2m + k)!} = \\ &= \sum_k \binom{m}{k} \binom{A}{m} \binom{A - m}{C - B + m - k} \binom{C + m - k}{A}. \end{aligned} \quad (25)$$

Сравнивая это с (23), обнаруживаем, что должно выполняться тождество

$$\sum_k \binom{m}{k} \binom{A - m}{C - B + m - k} \binom{C + m - k}{A} = \binom{B}{C - A + m} \binom{C}{B - m}. \quad (26)$$

Оказывается, с этим тождеством мы встречались в упр. 1.2.6-31:

$$\sum_j \binom{M - R + S}{j} \binom{N + R - S}{N - j} \binom{R + j}{M + N} = \binom{R}{M} \binom{S}{N}, \quad (27)$$

где $M = A + B - C - m$, $N = C - B + m$, $R = B$, $S = C$, а $j = C - B + m - k$.

Аналогично можно подсчитать число перестановок мультимножества $\{A \cdot a, B \cdot b, C \cdot c, D \cdot d\}$, если количество столбцов различных типов в них задано следующим образом:

Столбец:	a	a	b	b	c	c	d	d
	d	b	a	c	b	d	a	c
Количество:	r	$A - r$	q	$B - q$	$B - A + r$	$D - r$	$A - q$	$D - A + q$

(28)

(Здесь $A + C = B + D$.) Возможными циклами в разложении такой перестановки на простые множители будут

Цикл:	$(a b)$	$(b c)$	$(c d)$	$(d a)$	$(a b c d)$	$(d c b a)$
Количество:	$A - r - s$	$B - q - s$	$D - r - s$	$A - q - s$	s	$q - A + r + s$

(29)

при некотором s (см. упр. 12). В этом случае циклы $(a b)$ и $(c d)$ коммутируют, так же как и циклы $(b c)$ и $(d a)$, поэтому необходимо подсчитать число различных разложений на простые множители. Оказывается (см. упр. 10), всегда существует единственное разложение, такое, что цикл $(a b)$ никогда не следует непосредственно за $(c d)$, а $(b c)$ не встречается сразу после $(d a)$. Отсюда, пользуясь результатом упр. 13, получаем тождество

$$\begin{aligned} \sum_{s,t} \binom{B}{t} \binom{A - q - s}{A - r - s - t} \binom{B + D - r - s - t}{B - q - s} \times \frac{D!}{(D - r - s)!(A - q - s)!s!(q - A + r + s)!} = \\ = \binom{A}{r} \binom{B + D - A}{D - r} \binom{B}{q} \binom{D}{A - q}. \end{aligned}$$

Вынося из обеих частей множитель $\binom{D}{A-q}$ и слегка упрощая факториалы, приходим к сложному на вид пятипараметрическому тождеству биномиальных коэффициентов:

$$\sum_{s,t} \binom{B}{t} \binom{A-r-t}{s} \binom{B+D-r-s-t}{D+q-r-t} \times \binom{D-A+q}{D-r-s} \binom{A-q}{r+t-q} = \binom{A}{r} \binom{B+D-A}{D-r} \binom{B}{q}. \quad (30)$$

Пользуясь тождеством (27), можно выполнить суммирование по s , а получившаяся сумма по t легко вычисляется. Таким образом, после всей проделанной работы нам не повезло обнаружить какое-либо тождество, которое мы бы еще не умели выводить. Но мы по крайней мере научились подсчитывать число перестановок определенного вида двумя различными способами, а эти методы подсчета — хорошая подготовка к решению задач, которые еще впереди.

Упражнения

1. [M05] *Да или нет?* Пусть M_1 и M_2 — мультимножества. Если α — перестановка M_1 , а β — перестановка M_2 , то $\alpha \uplus \beta$ — перестановка $M_1 \cup M_2$.
2. [10] Соединительное произведение перестановок $cadab$ и $bddad$ вычислено в (5); найдите соединительное произведение $bddad \uplus cadab$, которое получается, если сомножители поменять местами.
3. [M13] Верно ли утверждение, обратное (9)? Иначе говоря, если перестановки α и β коммутативны относительно операции соединительного произведения, то следует ли из этого, что они не содержат общих букв?
4. [M11] Каноническое разложение перестановки (12) в смысле теоремы А при $a < b < c < d$ задается формулой (17). Найдите соответствующее каноническое разложение в случае, когда $d < c < b < a$.
5. [M23] В условии (b) теоремы В требуется, чтобы $x < y$; что будет, если ослабить это требование, заменив его на $x \leq y$?
6. [M15] Сколько существует цепочек, состоящих ровно из m букв a и n букв b , таких, что ровно k букв b стоят непосредственно перед буквами a и нет никаких других букв, кроме a и b ?
7. [M21] Сколько цепочек из букв a, b, c , удовлетворяющих условиям (18), начинается с буквы a ? с буквы b ? с буквы c ?
- >8. [20] Найдите все разложения перестановки (12) на два множителя $\alpha \uplus \beta$.
9. [33] Напишите программы для ЭВМ, которые бы производили разложения заданной перестановки мультимножества, описанные в теоремах А и С.
- >10. [M30] *Да или нет?* Согласно теореме С, разложение на простые множители не вполне однозначно, тем не менее можно следующим образом обеспечить единственность. "Существует линейное упорядочение \prec на множестве простых перестановок, такое, что каждая перестановка мультимножества имеет единственное разложение $\sigma_1 \uplus \sigma_2 \uplus \dots \uplus \sigma_n$ на простые множители, удовлетворяющее условию $\sigma_i \preceq \sigma_{i+1}$, если σ_i коммутирует с σ_{i+1} , при $1 \leq i < n$ ".
11. [M26] Пусть $\sigma_1, \sigma_2, \dots, \sigma_t$ — циклы без повторяющихся элементов. Определим частичное упорядочение \prec на множестве t элементов $\{x_1, \dots, x_t\}$, полагая $x_i \prec x_j$, если $i < j$ и σ_i имеет по крайней мере одну общую букву с σ_j . Докажите следующую связь между теоремой С и понятием "топологической сортировки" (п. 2.2.3): *число различных разложений перестановки $\sigma_1 \uplus \sigma_2 \uplus \dots \uplus \sigma_t$ на простые множители равно количеству способов топологически отсортировать данное частичное упорядочение.* (Например, в соответствии с (22) существует пять способов топологически отсортировать упорядочение $x_1 \prec x_3, x_2 \prec x_4, x_1 \prec x_4$.) Обратное, если на множестве из t элементов задано какое-то частичное упорядочение, то существует множество циклов

$$\{\sigma_1, \sigma_2, \dots, \sigma_t\},$$

которое определяет это частичное упорядочение указанным способом.

12. [M16] Покажите, что соотношения (29) являются следствиями предположений (28).
13. [M21] Докажите, что число перестановок мультимножества $\{A \cdot a, B \cdot b, C \cdot c, D \cdot d, E \cdot e, F \cdot f\}$, не содержащих пар стоящих рядом букв ca и db , равно

$$\sum_t \binom{D}{A-t} \binom{A+B+E+F}{t} \binom{A+B+C+E+F-t}{B} \binom{C+D+E+F}{C, D, E, F}.$$

14. [M30] Один из способов определить перестановку π^{-1} , "обратную" перестановке π , который подсказан нам другими определениями этого пункта, это поменять местами строки двустрочного

представления π и затем выполнить устойчивую сортировку столбцов, так чтобы элементы верхней строки расположились в неубывающем порядке. Например, если $a < b < c < d$, то из этого определения следует, что $cabddabdad^{-1} = acdadabdd$.

Исследуйте свойства этой операции обращения; имеется ли, например, какая-нибудь простая связь между ней и соединительным произведением? Можно ли подсчитать число перестановок, таких, что $\pi = \pi^{-1}$?

>15. [M25] Докажите, что перестановка $a_1 \dots a_n$ мультимножества

$$\{n_1 \cdot x_1, n_2 \cdot x_2, \dots, n_m \cdot x_m\},$$

где $x_1 < x_2 < \dots < x_n$ и $n_1 + n_2 + \dots + n_m = n$, является циклом тогда и только тогда, когда направленный граф с вершинами $\{x_1, x_2, \dots, x_m\}$ и дугами из x_j в $a_{n_1+\dots+n_j}$ содержит ровно один ориентированный цикл. В этом случае число способов представить перестановку в виде цикла равно длине этого ориентированного цикла. Например, направленным графом, соответствующим перестановке

$$\begin{pmatrix} a & a & a & b & b & c & c & c & d & d \\ d & c & b & a & c & a & a & b & d & c \end{pmatrix}, \text{ будет}$$

Picture: p. 48

а два способа представить перестановку в виде цикла—это $(baddcacabc)$ и $(caddcacbab)$.

16. [M35] В предыдущем пункте, формула (8), мы нашли производящую функцию для *инверсий* перестановок в частном случае, когда в перестановке участвуют элементы множества. Покажите, что в общем случае перестановок *мультимножества* производящая функция для инверсий равна " z -мультиномиальному коэффициенту"

$$\binom{n}{n_1, n_2, \dots}_z = \frac{n!_z}{n_1!_z n_2!_z \dots}, \quad \text{где } m!_z = (1-z)(1-z^2)\dots(1-z^m).$$

[Ср. с (3); z -биномиальные коэффициенты определены в формуле (1.2.6-37).]

17. [M24] Пользуясь производящей функцией, найденной в упр. 16, вычислите среднее значение и дисперсию для числа инверсий в случайной перестановке мультимножества.

18. [M30] (П. А. Мак-Магон.) *Индекс* перестановки $a_1 a_2 \dots a_n$ был определен в предыдущем пункте; мы доказали, что число перестановок данного множества, имеющих данный индекс k , равно числу перестановок, имеющих k инверсий. Верно ли это для перестановок заданного мультимножества?

19. [BM28] Определим *функцию Мёбиуса* $\mu(\pi)$ перестановки π : она равна 0, если π содержит повторяющиеся элементы, и $(-1)^k$ в противном случае, если π —произведение k простых перестановок. (Ср. с определением обычной функции Мёбиуса, упр. 4.5.2-10.) (а) Докажите, что если $\pi \neq \varepsilon$, то

$$\sum \mu(\lambda) = 0,$$

где сумма берется по всем перестановкам λ , являющимся левыми сомножителями в разложении π (т. е. $\lambda \top \rho = \pi$, где ρ —некоторая перестановка) (б) Докажите, что если $x_1 < x_2 < \dots < x_m$ и $\pi = x_{i_1} x_{i_2} \dots x_{i_n}$, где $1 \leq i_l \leq m$ при $1 \leq j \leq n$, то

$$\mu(\pi) = (-1)^m \varepsilon(i_1 i_2 \dots i_n), \quad \text{где } \varepsilon(i_1 i_2 \dots i_n) = \text{sign} \prod_{1 \leq j < k \leq n} (i_k - i_j).$$

20. [BM33] (Д. Фоата.) Пусть (a_{ij}) —произвольная матрица действительных чисел. Пользуясь обозначениями упр. 19 (б), определим $\nu(\pi) = a_{i_1 j_1} \dots a_{i_n j_n}$, если двустрочное представление перестановки π таково:

$$\begin{pmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_n} \\ x_{j_1} & x_{j_2} & \dots & x_{j_n} \end{pmatrix}.$$

Эта функция полезна при вычислении производящих функций для перестановок мультимножества, потому что $\sum \nu(\pi)$, где сумма берется по всем перестановкам π мультимножества

$$\{n_1 \cdot x_1, \dots, n_m \cdot x_m\},$$

будет производящей функцией для числа перестановок, удовлетворяющих некоторым ограничениям. Например, если положить $a_{ij} = z$ при $i = j$ и $a_{ij} = 1$ при $i \neq j$, то $\sum \nu(\pi)$ —производящая

функция для числа ”неподвижных точек” (столбцов, в которых верхний и нижний элементы равны). Чтобы можно было исследовать $\sum \nu(\pi)$ для всех мультимножеств одновременно, рассмотрим функцию

$$G = \sum \pi \nu(\pi),$$

где сумма берется по всем π из множества $\{x_1, \dots, x_m\}^*$ всех перестановок мультимножеств, содержащих элементы x_1, \dots, x_m , и посмотрим на коэффициенты при $x_1^{n_1} \dots x_m^{n_m}$ в G .

В этой формуле для G будем рассматривать π как произведение переменных x . Например, при $m = 2$ имеем

$$\begin{aligned} G &= 1 + x_1 \nu(x_1) + x_2 \nu(x_2) + x_1 x_1 \nu(x_1 x_1) + x_1 x_2 \nu(x_1 x_2) + x_2 x_1 \nu(x_2 x_1) + x_2 x_2 \nu(x_2 x_2) + \dots = \\ &= 1 + x_1 a_{11} + x_2 a_{22} + x_1^2 a_{11}^2 + x_1 x_2 a_{11} a_{22} + x_1 x_2 a_{21} a_{12} + x_2^2 a_{22}^2 + \dots \end{aligned}$$

Таким образом, коэффициент при $x_1^{n_1} \dots x_m^{n_m}$ в G равен сумме $\sum \nu(\pi)$ по всем перестановкам π мультимножества $\{n_1 \cdot x_1, \dots, n_m \cdot x_m\}$. Нетрудно видеть, что этот коэффициент равен также коэффициенту при $x_1^{n_1} \dots x_m^{n_m}$ в выражении

$$(a_{11}x_1 + \dots + a_{1m}x_m)^{n_1} (a_{21}x_1 + \dots + a_{2m}x_m)^{n_2} \dots (a_{m1}x_1 + \dots + a_{mm}x_m)^{n_m}.$$

Цель этого упражнения — доказать то, что П. А. Мак-Магон в своей книге *Combinatory Analysis* (1915), том 1, разд. 3, назвал ”основной теоремой”, а именно формулу

$$G = 1/D, \quad \text{где } D = \det \begin{pmatrix} 1 - a_{11}x_1 & -a_{12}x_2 & \dots & -a_{1m}x_m \\ -a_{21}x_1 & 1 - a_{22}x_2 & \dots & -a_{2m}x_m \\ \vdots & & & \vdots \\ -a_{m1}x_1 & -a_{m2}x_2 & \dots & 1 - a_{mm}x_m \end{pmatrix}.$$

Например, если $a_{ij} = 1$ при всех i, j , то эта формула дает

$$G = 1/(1 - (x_1 + x_2 + \dots + x_m)),$$

и коэффициент при $x_1^{n_1} \dots x_m^{n_m}$ оказывается равным $(n_1 + \dots + n_m)!/n_1! \dots n_m!$, как ему и положено.

Для доказательства основной теоремы Мак-Магона покажите, что (а) $\nu(\pi \uparrow \rho) = \nu(\pi)\nu(\rho)$; (б) в обозначениях упр. 19 $D = \sum \pi \mu(\pi) \nu(\pi)$, где сумма берется по всем перестановкам π из множества

$$\{x_1, \dots, x_m\}^*,$$

и, наконец, (с) $D \cdot G = 1$.

5.1.3. *Отрезки

В п. 3.3.2 мы рассмотрели ”неубывающие отрезки” как один из методов, позволяющих проверить случайность последовательности. Если поместить вертикальные черточки с обоих концов перестановки $a_1 a_2 \dots a_n$, а также между a_j и a_{j+1} , когда $a_j > a_{j+1}$, то *отрезками* будут называться сегменты, ограниченные парами черточек. Например, в перестановке

$$|3 \ 5 \ 7 \ | \ 1 \ 6 \ 8 \ 9 \ | \ 4 \ 2$$

—четыре отрезка. Мы нашли среднее число отрезков длины k в случайной перестановке множества $\{1, 2, \dots, n\}$, а также ковариацию числа отрезков длины j и длины k . Отрезки важны при изучении алгоритмов сортировки, так как они представляют упорядоченные сегменты данных. Поэтому-то мы теперь вновь вернемся к вопросу об отрезках. Обозначим через

$$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \tag{1}$$

число перестановок множества $\{1, 2, \dots, n\}$, имеющих ровно k возрастающих отрезков. Такие числа $\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle$ возникают и в других контекстах; их обычно называют *числами Эйлера*, потому что Эйлер обсудил их в своей знаменитой книге *Institutiones calculi differentialis* (St. Petersburg, 1755), 485–487 [Euler, *Opera Omnia*, (1) 10 (1913), 373–375]; их следует отличать от ”эйлеровых чисел”, о которых идет речь в упр. 5.1.4-22.

Из любой данной перестановки множества $\{1, 2, \dots, n - 1\}$ можно образовать n новых перестановок, вставляя элемент n во все возможные места. Если в исходной перестановке содержалось

k отрезков, то ровно k новых перестановок будут иметь k отрезков; в остальных $n - k$ перестановках будет по $k + 1$ отрезков, поскольку всякий раз, когда n вставляется не в конец уже существующего отрезка, число отрезков увеличивается на единицу. Например, среди шести перестановок, полученных из перестановки 3 1 2 4 5,

$$\begin{array}{ccc} 6 3 1 2 4 5, & 3 6 1 2 4 5, & 3 1 6 2 4 5, \\ 3 1 2 6 4 5, & 3 1 2 4 6 5, & 3 1 2 4 5 6; \end{array}$$

все, кроме второй и последней, содержат по три отрезка вместо исходных двух. Отсюда имеем рекуррентное соотношение

$$\langle n \rangle_k = k \langle n-1 \rangle_k + (n-k+1) \langle n-1 \rangle_{k-1}, \quad \text{где } n \text{ целое, } n \geq 1; k \text{ целое.} \quad (2)$$

Условимся, что

$$\langle 0 \rangle_k = \delta_{1k}, \quad (3)$$

т. е. будем считать, что в пустой перестановке содержится один отрезок. Читатель, возможно, найдет небезынтересным сравнить (2) с рекуррентным соотношением для чисел Стирлинга [формулы (1.2.6-42)]. В табл. 1 приведены числа Эйлера для малых n .

В табл. 1 можно заметить некоторые закономерности. По определению имеем

$$\langle n \rangle_0 + \langle n \rangle_1 + \dots + \langle n \rangle_n = n!; \quad (4)$$

$$\langle n \rangle_0 = 0, \quad \langle n \rangle_1 = 1; \quad (5)$$

$$\langle n \rangle_n = 1 \quad \text{при } n \geq 1. \quad (6)$$

Таблица 1

n	Числа Эйлера								
	$\langle n \rangle_0$	$\langle n \rangle_1$	$\langle n \rangle_2$	$\langle n \rangle_3$	$\langle n \rangle_4$	$\langle n \rangle_5$	$\langle n \rangle_6$	$\langle n \rangle_7$	
0	0	1	0	0	0	0	0	0	
1	0	1	0	0	0	0	0	0	
2	0	1	1	0	0	0	0	0	
3	0	1	4	1	0	0	0	0	
4	0	1	11	11	1	0	0	0	
5	0	1	26	66	26	1	0	0	
6	0	1	57	302	302	57	1	0	
7	0	1	120	1191	2416	1191	120	1	

Выполняется также свойство симметрии

$$\langle n \rangle_k = \langle n \rangle_{n+1-k}, \quad n \geq 1, \quad (7)$$

которое вытекает из того факта, что каждой перестановке $a_1 a_2 \dots a_n$, содержащей k отрезков, соответствует перестановка $a_n \dots a_2 a_1$, содержащая $n + 1 - k$ отрезков.

Другое важное свойство чисел Эйлера выражается формулой

$$\sum_k \langle n \rangle_k \binom{m+k-1}{n} = m^n, \quad n \geq 0, \quad (8)$$

которую можно доказать, используя понятие сортировки. Рассмотрим m^n последовательностей $a_1 a_2 \dots a_n$, где $1 \leq a_i \leq m$. Любую такую последовательность можно устойчиво отсортировать таким образом, чтобы элементы расположились в неубывающем порядке:

$$a_{i_1} \leq a_{i_2} \leq \dots \leq a_{i_n}, \quad (9)$$

где $i_1 i_2 \dots i_n$ — однозначно определенная перестановка множества $\{1, 2, \dots, n\}$, такая, что $i_j < i_{j+1}$, если $a_{i_j} = a_{i_{j+1}}$; другими словами, из $i_j > i_{j+1}$ следует $a_{i_j} < a_{i_{j+1}}$. Покажем, что если в перестановке $i_1 i_2 \dots i_n$ содержится k отрезков, то число соответствующих ей последовательностей $a_1 a_2 \dots a_n$ равно $\binom{m+n-k}{n}$; тем самым будет доказана формула (8), если заменить k на $n + 1 - k$.

Пусть, например, $n = 9$, $i_1 i_2 \dots i_n = 357168942$ и требуется подсчитать число последовательностей $a_1 a_2 \dots a_9$, таких, что

$$1 \leq a_3 \leq a_5 \leq a_7 < a_1 \leq a_6 \leq a_8 \leq a_9 < a_4 < a_2 \leq m; \quad (10)$$

оно равно числу последовательностей $b_1 b_2 \dots b_9$, таких, что

$$1 \leq b_1 < b_2 < b_3 < b_4 < b_5 < b_6 < b_7 < b_8 < b_9 \leq m + 5,$$

так как можно положить $b_1 = a_3$, $b_2 = a_5 + 1$, $b_3 = a_7 + 2$, $b_4 = a_1 + 2$, $b_5 = a_6 + 3$ и т. д. Число способов, которыми можно выбрать элементы b , равно просто-напросто числу способов выбрать 9 предметов из $m + 5$, т. е. $\binom{m+5}{9}$; аналогичное доказательство годится для произвольных n и k и любой перестановки $i_1 i_2 \dots i_n$ с k отрезками.

Так как в обеих частях равенства (8) стоят полиномы от m , то вместо m можно подставить любое действительное число, получив интересное выражение степеней через последовательные биномиальные коэффициенты:

$$x^n = \binom{n}{1} \binom{x}{n} + \binom{n}{2} \binom{x+1}{n} + \dots + \binom{n}{n} \binom{x+n-1}{n}, \quad n \geq 1. \quad (11)$$

Например,

$$x^3 = \binom{x}{3} + 4 \binom{x+1}{3} + \binom{x+2}{3}.$$

В основном благодаря именно этому свойству числа Эйлера весьма полезны при изучении дискретной математики. Положив в (11) $x = 1$, докажем еще раз, что

$$\binom{n}{n} = 1,$$

поскольку биномиальные коэффициенты обращаются в 0 во всех слагаемых, кроме последнего. Положив $x = 2$, получим

$$\binom{n}{n-1} = 2^n - n - 1, \quad n \geq 1. \quad (12)$$

Подставив $x = 3, 4, \dots$, убедимся, что все числа $\binom{n}{k}$ полностью определяются соотношением (11), и придем к формуле, впервые найденной Эйлером:

$$\begin{aligned} \binom{n}{k} &= k^n - (k-1)^n \binom{n+1}{1} + (k-2)^n \binom{n+1}{2} - \dots + (-1)^k 0^n \binom{n+1}{k} = \\ &= \sum_{0 \leq j \leq k} (-1)^j (k-j)^n \binom{n+1}{j}, \quad n \geq 0, k \geq 0. \end{aligned} \quad (13)$$

Изучим теперь производящую функцию для отрезков. Если положить

$$g_n(z) = \sum_k \binom{n}{k} z^k / n!, \quad (14)$$

то коэффициент при z^k равен вероятности того, что случайная перестановка множества $\{1, 2, \dots, n\}$ содержит ровно k отрезков. Поскольку k отрезков в перестановке столь же вероятны, как и $n+1-k$, то среднее число отрезков должно равняться $\frac{1}{2}(n+1)$, и, следовательно, $g'_n(1) = \frac{1}{2}(n+1)$. В упр. 2(b) показано, что имеет место простая формула для *всех* производных функции $g_n(z)$ в точке $z = 1$:

$$g_n^{(m)}(1) = \left\{ \begin{matrix} n+1 \\ n+1-m \end{matrix} \right\} / \binom{n}{m}, \quad n \geq m. \quad (15)$$

Так, в частности, дисперсия $g''_n(1) + g'_n(1) - g'_n(1)^2$ равна $(n+1)/12$ при $n \geq 2$, что указывает на довольно устойчивое распределение около среднего значения. (Мы нашли эту же величину в упр. 3.3.2-18; там она называлась $\text{covar}(R'_1, R'_1)$.) Функция $g_n(z)$ — полином, поэтому с помощью формулы (15) можно разложить ее в ряд Тейлора

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leq k \leq n} (z-1)^{n-k} k! \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} = \\ &= \frac{1}{n!} \sum_{0 \leq k \leq n} z^{k+1} (1-z)^{n-k} k! \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\}. \end{aligned} \quad (16)$$

Второе равенство следует из первого, так как по свойству симметрии (7)

$$g_n(z) = z^{n+1}g_n(1/z), \quad n \geq 1. \quad (17)$$

Из рекуррентного соотношения для чисел Стирлинга

$$\left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} = (k+1) \left\{ \begin{matrix} n \\ k+1 \end{matrix} \right\} + \left\{ \begin{matrix} n \\ k \end{matrix} \right\}$$

получаются два чуть более простых представления:

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leq k \leq n} z(z-1)^{n-k} k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \\ &= \frac{1}{n!} \sum_{0 \leq k \leq n} z^k (1-z)^{n-k} k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\}. \end{aligned} \quad (18)$$

Производящая функция от двух переменных⁷

$$g(z, x) = \sum_{n \geq 0} g_n(z) x^n = \sum_{k, n \geq 0} \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle z^k x^n / n! \quad (19)$$

равна, следовательно,

$$z \sum_{k, n \geq 0} \frac{((z-1)x)^n}{(z-1)^k} \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \frac{k!}{n!} = z \sum_{k \geq 0} \left(\frac{e^{(z-1)x} - 1}{z-1} \right)^k = \frac{z(1-z)}{e^{(z-1)x} - z}; \quad (20)$$

это еще одно соотношение, обсуждавшееся Эйлером.

Другие свойства чисел Эйлера можно найти в обзорной статье Л. Карлица [*Math. Magazine*, **33** (1959), 247–260]; см. также Дж. Риордан, Введение в комбинаторный анализ, ИЛ, М., 1963, стр. 50, 253, 254.

Рассмотрим теперь длину отрезков; какова в среднем длина отрезка? В п. 3.3.2 мы уже изучили математическое ожидание числа отрезков данной длины; средняя длина отрезка равна примерно 2 в согласии с тем фактом, что в случайной перестановке длины n содержится в среднем $\frac{1}{2}(n+1)$ отрезков. Применительно к алгоритмам сортировки полезна несколько отличная точка зрения; рассмотрим длину k -го слева отрезка перестановки при $k = 1, 2, \dots$

Какова, например, длина первого (крайнего слева) отрезка случайной перестановки $a_1 a_2 \dots a_n$? Его длина всегда ≥ 1 ; она ≥ 2 ровно в половине случаев (а именно, если $a_1 < a_2$). Его длина ≥ 3 ровно для $1/6$ случаев (если $a_1 < a_2 < a_3$). Вообще его длина $\geq m$ с вероятностью $q_m = 1/m!$ при $1 \leq m \leq n$. Следовательно, вероятность того, что длина этого отрезка равна в точности m , есть

$$\begin{aligned} p_m &= q_m - q_{m+1} = 1/m! - 1/(m+1)! \quad \text{при } 1 \leq m < n; \\ p_n &= 1/n!. \end{aligned} \quad (21)$$

Средняя длина равна, таким образом,

$$\begin{aligned} p_1 + 2p_2 + \dots + np_n &= \\ &= (q_1 - q_2) + 2(q_2 - q_3) + \dots + (n-1)(q_{n-1} - q_n) + nq_n = \\ &= q_1 + q_2 + \dots + q_n = \\ &= \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!}. \end{aligned} \quad (22)$$

Предел при $n \rightarrow \infty$ равен $e - 1 = 1.71828\dots$, а для конечных n это значение равно $e - 1 - \delta_n$, где δ_n довольно мало:

$$\delta_n = \frac{1}{(n+1)!} \left(1 + \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} + \dots \right) < \frac{e-1}{(n+1)!}.$$

⁷ В оригинале "super generating function". Соответствующего термина в отечественной литературе не существует. — Прим. ред.

Поэтому для практических целей удобно рассмотреть отрезки случайной *бесконечной* последовательности различных чисел

$$a_1, a_2, a_3, \dots;$$

под "случайностью" последовательности мы здесь подразумеваем то, что все $n!$ возможных взаимных расположении первых n элементов последовательности равновероятны. Так что средняя длина первого отрезка случайной бесконечной последовательности равна $e - 1$.

Несколько усовершенствовав этот метод, можно установить среднюю длину k -го отрезка случайной последовательности. Пусть q_{km} — вероятность того, что общая длина первых k отрезков $\geq m$; тогда q_{km} равно величине $1/m!$, умноженной на число перестановок множества $\{1, 2, \dots, m\}$, содержащих не более k отрезков:

$$q_{km} = \left(\binom{m}{1} + \dots + \binom{m}{k} \right) / m!. \quad (23)$$

Вероятность того, что общая длина первых k отрезков равна m , есть $q_{km} - q_{k(m+1)}$. Следовательно, обозначив через L_k среднюю длину k -го отрезка, находим, что

$$\begin{aligned} L_1 + \dots + L_k &= (\text{средняя общая длина первых } k \text{ отрезков}) = \\ &= (q_{k1} - q_{k2}) = 2(q_{k2} - q_{k3}) + 3(q_{k3} - q_{k4}) + \dots = \\ &= q_{k1} + q_{k2} + q_{k3} + \dots \end{aligned}$$

Вычитая $L_1 + \dots + L_{k-1}$ и используя значения q_{km} , из (23) получаем нужную нам формулу:

$$L_k = \frac{1}{1!} \binom{1}{k} + \frac{1}{2!} \binom{2}{k} + \frac{1}{3!} \binom{3}{k} + \dots = \sum_{m \geq 1} \binom{m}{k} \frac{1}{m!}. \quad (24)$$

Поскольку $\binom{0}{k} = 0$ при $k \neq 1$, значение L_k оказывается равным коэффициенту при z^k в производящей функции $g(z, 1) - z$. (см. (19)); таким образом, имеем

$$L(z) = \sum_{k \geq 0} L_k z^k = \frac{z(1-z)}{e^{z-1} - z} - z. \quad (25)$$

Из формулы Эйлера (13) получим представление L_k в виде полинома от e :

$$\begin{aligned} L_k &= \sum_{m \geq 0} \sum_{0 \leq j \leq k} (-1)^{k-j} \binom{m+1}{k-j} \frac{j^m}{m!} = \\ &= \sum_{0 \leq j \leq k} (-1)^{k-j} \sum_{m \geq 0} \binom{m}{k-j} \frac{j^m}{m!} + \sum_{0 \leq j \leq k} (-1)^{k-j} \sum_{m \geq 0} \binom{m}{k-j-1} \frac{j^m}{m!} = \\ &= \sum_{0 \leq j \leq k} \frac{(-1)^{k-j} j^{k-j}}{(k-j)!} \sum_{n \geq 0} \frac{j^n}{n!} + \sum_{0 \leq j \leq k} \frac{(-1)^{k-j} j^{(k-j-1)}}{(k-j-1)!} \sum_{n \geq 0} \frac{j^n}{n!} = \\ &= k \sum_{0 \leq j \leq k} \frac{e^j (-1)^{k-j} j^{k-j-1}}{(k-j)!}. \end{aligned} \quad (26)$$

Эту формулу для L_k впервые получила Б. Дж. Гэсснер [САСМ, 10 (1967), 89–93]. Имеем, в частности,

$$\begin{aligned} L_1 &= e - 1 && \approx 1.71828 \dots; \\ L_2 &= e^2 - 2e && \approx 1.95249 \dots; \\ L_3 &= e^3 - 3e^2 + \frac{3}{2}e && \approx 1.99579 \dots \end{aligned}$$

Итак, следует ожидать, что второй отрезок будет длиннее первого, а третий отрезок будет в среднем еще длиннее! На первый взгляд это может показаться странным, но после минутного размышления становится ясно, что, поскольку первый элемент второго отрезка будет скорее всего малым числом (именно это служит причиной окончания первого отрезка), у второго отрезка больше шансов оказаться длиннее, чем у первого. Тенденция такова, что первый элемент третьего отрезка будет даже меньше, чем первый элемент второго отрезка.

Числа L_k важны в теории сортировки посредством выбора с замещением (п. 5.4.1), поэтому интересно подробно изучить их значения. В табл. 2, вычисленной Дж. У. Ренчем (мл.), приведены

первые 18 значений L_k с точностью до 15 десятичных знаков. Рассуждения, приведенные в предыдущем абзаце, могут вызвать подозрение, что $L_{k+1} > L_k$; на самом же деле значения колеблются, то возрастают, то убывают. Заметим, что L_k быстро приближаются к предельному значению 2; весьма примечательно то, что эти нормированные полиномы от трансцендентного числа e так быстро сходятся к рациональному числу $2!$ Полиномы (26) представляют некоторый интерес и с точки зрения численного анализа, будучи прекрасным примером потери значащих цифр при вычитании почти равных чисел; используя 19-значную арифметику с плавающей точкой, Гэсснер пришла к неверному заключению о том, что $L_{12} > 2$, а Ренч отметил, что 42-значная арифметика с плавающей точкой дает L_{28} лишь с точностью до 29 значащих цифр.

Таблица 2

Средние длины отрезков			
k	L_k	k	L_k
1	1.71828 18284 59045+	10	2.00000 00012 05997+
2	1.95249 24420 12560–	11	2.00000 00001 93672+
3	1.99579 13690 84285–	12	1.99999 99999 99909+
4	2.00003 88504 76806–	13	1.99999 99999 97022–
5	2.00005 75785 89716+	14	1.99999 99999 99719+
6	2.00000 50727 55710–	15	2.00000 00000 00019+
7	1.99999 96401 44022+	16	2.00000 00000 00006+
8	1.99999 98889 04744+	17	2.00000 00000 00000+
9	1.99999 99948 43434–	18	2.00000 00000 00000 –

Асимптотическое поведение L_k можно определить, исходя из простых положений теории функций комплексного переменного.

Picture: Рис. 3 Корни уравнения $e^{z-1} = z$. Пунктирная линия соответствует уравнению $e^{x-1} \cos y = x$, сплошная линия—уравнению $e^{x-1} \sin y = y$.

Знаменатель в (25) обращается в нуль лишь при $e^{z-1} = z$, т. е. (полагая $z = x + iy$) когда

$$e^{x-1} \cos y = x \text{ и } e^{x-1} \sin y = y. \quad (27)$$

На рис. 3, где нанесены оба графика этих уравнений, видно, что они пересекаются в точках $z = z_0, z_1, \bar{z}_1, z_2, \bar{z}_2, \dots$; здесь $z_0 = 1$,

$$z_1 = (3.08884 30156 13044-) + (7.46148 92856 54255-)i \quad (28)$$

и при больших k мнимая часть $\Im(z_{k+1})$ равна приблизительно $\Im(z_k) + 2\pi$. Так как

$$\lim_{z \rightarrow z_k} \left(\frac{1-z}{e^{z-1}-z} \right) (z-z_k) = -1 \quad \text{при } k > 0$$

и этот предел равен -2 при $k = 0$, то функция

$$R_m(z) = L(z) + \frac{2}{z-z_0} + \frac{z_1}{z-z_1} + \frac{\bar{z}_1}{z-\bar{z}_1} + \frac{z_2}{z-z_2} + \frac{\bar{z}_2}{z-\bar{z}_2} + \dots + \frac{z_m}{z-z_m} + \frac{\bar{z}_m}{z-\bar{z}_m}$$

не имеет особенностей в комплексной плоскости при $|z| < |z_{m+1}|$. Значит, $R_m(z)$ можно разложить в степенной ряд $\sum_k \rho_k z^k$, который сходится абсолютно при $|z| < |z_{m+1}|$; отсюда следует, что $\rho_k M^k \rightarrow 0$ при $k \rightarrow \infty$, где $M = |z_{m+1}| - \varepsilon$. Коэффициентами для $L(z)$ служат коэффициенты разложения функции

$$\frac{2}{1-z} + \frac{1}{1-z/z_1} + \frac{1}{z/\bar{z}_1} + \dots + \frac{1}{z-z/z_m} + \frac{1}{z-z/\bar{z}_m} + R_m z,$$

а именно

$$L_n = 2 + 2r_1^{-n} \cos n\theta_1 + 2r_2^{-n} \cos n\theta_2 + \dots + 2r_m^{-n} \cos n\theta_m + O(r_{m+1}^{-n}), \quad (29)$$

если положить

$$z_k = r_k e^{i\theta_k}. \quad (30)$$

Отсюда можно проследить асимптотическое поведение L_n . Имеем

$$\begin{aligned} r_1 &= 8.07556 64528 89526-, \\ \theta_1 &= 1.17830 39784 74668+; \\ r_2 &= 14.35457-, \quad \theta_2 = 1.31269-; \\ r_3 &= 20.62073+, \quad \theta_3 = 1.37428-; \\ r_4 &= 26.88795+, \quad \theta_4 = 1.41050-; \end{aligned} \quad (31)$$

таким образом, главный вклад в $L_n - 2$ дают r_1 и θ_1 , и ряд (29) сходится очень быстро. Приведенные здесь значения r_1 и θ_1 найдены Дж. У. Ренчем (мл.) Дальнейший анализ [W. W. Hooker, *CASM*, 12 (1969), 411–413] показывает, что $R_m(z) \rightarrow -z$ при $m \rightarrow \infty$; следовательно, ряд $2 \sum_{k \geq 0} z_k^{-n} \cos n\theta_k$ действительно *сходится* к L_n при $n > 1$.

Можно провести более тщательное исследование вероятностей, чтобы полностью определить распределение вероятностей для длины k -го отрезка и для общей длины первых k отрезков (см. упр. 9, 10, 11), Оказывается сумма $L_1 + \dots + L_k$ асимптотически приближается к $2k - 1/3$.

В заключение этого пункта рассмотрим свойства отрезков в случае, когда в перестановке допускаются одинаковые элементы. Бесчисленные пасьянсы, которым посвящал свои досуги знаменитый американский астроном 19-го века Саймон Ньюкомб, имеют непосредственное отношение к интересующему нас вопросу. Он брал колоду карт и складывал их в одну стопку до тех пор, пока они шли в неубывающем порядке по старшинству; как только следующая карта оказывалась младше предыдущей, он начинал новую стопку. Он хотел найти вероятность того, что в результате вся колода окажется разложенной в заданное количество стопок.

Задача Саймона Ньюкомба состоит, следовательно, в нахождении распределения вероятностей для отрезков случайной перестановки мультимножества. В общем случае ответ довольно сложен (см. упр. 12), хотя мы уже видели, как решать задачу в частном случае, когда все карты различны по старшинству. Мы удовлетворимся здесь выводом формулы для *среднего* числа стопок в этом пасьянсе.

Пусть имеется m различных типов карт и каждая встречается ровно p раз. Например, в обычной колоде для бриджа $m = 13$, а $p = 4$, если пренебрегать различием масти. Замечательную симметрию обнаружил в этом случае П. А. Мак-Магон [Combinatory Analysis (Cambridge, 1915), том 1, стр. 212–213]: число перестановок с $k + 1$ отрезками равно числу перестановок с $mp - p - k + 1$ отрезками. Это соотношение легко проверить при $p = 1$ (формула (7)), однако при $p > 1$ оно кажется довольно неожиданным.

Можно доказать это свойство симметрии, установив взаимно однозначное соответствие между перестановками, такое, что каждой перестановке с $k + 1$ отрезками соответствует другая, с $mp - p - k + 1$ отрезками. Мы настойчиво рекомендуем читателю самому попробовать найти такое соответствие, прежде чем двигаться дальше.

Какого-нибудь очень простого соответствия на ум не приходит; доказательство Мак-Магона основано на производящих функциях, а не на комбинаторном построении. Однако установленное Фoaтой соответствие (теорема 5.1.2B) позволяет упростить задачу, так как там утверждается существование взаимно однозначного соответствия между перестановками с $k + 1$ отрезками и перестановками, в двустрочном представлении которых содержится ровно k столбцов $\begin{smallmatrix} y \\ x \end{smallmatrix}$, таких, что $x < y$.

Пусть дано мультимножество $\{p \cdot 1, p \cdot 2, \dots, p \cdot m\}$; рассмотрим перестановку с двустрочным обозначением

$$\left(\begin{array}{cccccccccc} 1 & \dots & 1 & 2 & \dots & 2 & \dots & m & \dots & m \\ x_{11} & \dots & x_{1p} & x_{21} & \dots & x_{2p} & \dots & x_{m1} & \dots & x_{mp} \end{array} \right). \quad (32)$$

Сопоставим с ней другую перестановку того же мультимножества:

$$\left(\begin{array}{cccccccccc} 1 & \dots & 1 & 2 & \dots & 2 & \dots & m & \dots & m \\ x'_{11} & \dots & x'_{1p} & x'_{m1} & \dots & x'_{mp} & \dots & x'_{21} & \dots & x'_{2p} \end{array} \right), \quad (33)$$

где $x' = m + 1 - x$. Если перестановка (32) содержит k столбцов вида $\begin{smallmatrix} y \\ x \end{smallmatrix}$, таких, что $x < y$, то (33) содержит $(m - 1)p - k$ таких столбцов, потому что необходимо рассмотреть лишь случай $y > 1$, а неравенство $x < y$ эквивалентно $x' \geq m + 2 - y$. Поскольку (32) соответствует перестановке с $k + 1$ отрезками, а (33)—перестановке с $mp - p - k + 1$ отрезками и преобразование, переводящее (32) в (33), обратимо (оно же переводит (33) в (32)), то тем самым доказано условие симметрии Мак-Магона. Пример этого построения содержится в упр. 14.

В силу свойства симметрии среднее число отрезков в случайной перестановке должно равняться $\frac{1}{2}((k + 1) + (mp - p - k + 1)) = 1 + \frac{1}{2}p(m - 1)$. Например, для стандартной колоды среднее число стопок в пасьянсе Ньюкомба равно 25 (так что раскладывание этого пасьянса вряд ли покажется столь уж увлекательным занятием).

На самом деле, используя весьма простое рассуждение, можно определить среднее число отрезков в общем случае для *любого* данного мультимножества $\{n_1 \cdot x_1, n_2 \cdot x_2, \dots, n_m \cdot x_m\}$, где все x различны. Пусть $n = n_1 + n_2 + \dots + n_m$, и представим себе, что все перестановки $a_1 a_2 \dots a_n$ этого мультимножества записаны в явном виде; посмотрим, как часто a_i оказывается больше a_{i+1} при каждом фиксированном значении i , $1 \leq i < n$. Число случаев, когда $a_i > a_{i+1}$, равно в точности половине числа случаев, когда $a_i \neq a_{i+1}$, и нетрудно видеть, что $a_i = a_{i+1} = x_j$ ровно в $Nn_j(n_j - 1)/n(n - 1)$ случаях, где N —общее число перестановок. Следовательно, $a_i = a_{i+1}$ ровно в

$$\frac{N}{n(n - 1)}(n_1(n_1 - 1) + \dots + n_m(n_m - 1)) = \frac{N}{n(n - 1)}(n_1^2 + \dots + n_m^2 - n)$$

случаях, а $a_i > a_{i+1}$ ровно в

$$\frac{N}{2n(n-1)}(n^2 - (n_1^2 + \dots + n_m^2))$$

случаях. Суммируя по i и прибавляя N , потому что в каждой перестановке один отрезок кончается элементом a_n , получим общее число отрезков во всех N перестановках:

$$N \left(\frac{n}{2} - \frac{1}{2n}(n_1^2 + \dots + n_m^2) + 1 \right). \quad (34)$$

Поделив на N , получим искомое среднее число отрезков.

Так как отрезки важны при изучении "порядковых статистик", имеется весьма обширная литература, посвященная им, в том числе и некоторым другим типам отрезков, не рассмотренным здесь. Дополнительную информацию можно найти в книге Ф. Н. Дэвид и Д. Э. Бартона *Combinatorial Chance* (London: Griffin, 1962), гл. 10, и в обзорной статье Д. Э. Бартона и К. Л. Мэллоуза [*Annals of Math. Statistics*, 36 (1965), 236–260]. Дальнейшие связи между числами Эйлера и перестановками рассматриваются в работе Д. Фоаты и М. П. Шюценберже *Théorie Géométrique des Polynômes Eulériens* (Lecture Notes in Math., 138 (Berlin: Springer, 1970), 94 стр.).

Упражнения

- [M26] Выведите формулу Эйлера (13).
- >2. [M22] (а) Попробуйте дальше развить идею, использованную в тексте при доказательстве тождества (8): рассмотрите последовательности $a_1 a_2 \dots a_n$, содержащие ровно q различных элементов, и докажите, что

$$\sum_k \langle n \rangle_k \binom{k-1}{n-q} = \left\{ \begin{matrix} n \\ q \end{matrix} \right\} q!.$$

(б) Используя это тождество, докажите, что

$$\sum_k \langle n \rangle_k \binom{k}{m} = \left\{ \begin{matrix} n+1 \\ n+1-m \end{matrix} \right\} (n-m)! \quad \text{при } n \geq m.$$

- [BM25] Вычислите сумму $\sum_k \langle n \rangle_k (-1)^k$.
- [M21] Чему равна сумма

$$\sum_k (-1)^k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} k! \binom{n-k}{m}?$$

- [M20] Найдите значение $\langle \binom{p}{k} \rangle \pmod p$, если p — простое число.
- >6. [M21] Мистер Тупица заметил, что из формул (4) и (13) можно получить

$$n! = \sum_{k \geq 0} \langle n \rangle_k = \sum_{k \geq 0} \sum_{j \geq 0} (-1)^{k-j} \binom{n+1}{k-j} j^n.$$

Произведя суммирование сначала по k , затем по j , он обнаружил, что $\sum_{k \geq 0} (-1)^{k-j} \binom{n+1}{k-j} = 0$ при всех $j \geq 0$, отсюда $n! = 0$ при любом $n \geq 0$. Не допустил ли он какой-нибудь ошибки?

- [BM40] Является ли распределение вероятностей для отрезков, задаваемое формулой (14), асимптотически нормальным? (Ср. с упр. 1.2.10-13.)
- [M24] (П. А. Мак-Магон) Покажите, что вероятность того, что длина первого отрезка достаточно длинной перестановки есть l_1 , длина второго есть l_2, \dots , а длина k -го отрезка $\geq l_k$, равна

$$\det \begin{pmatrix} 1/l_1! & 1/(l_1+l_2)! & 1/(l_1+l_2+l_3)! & \dots & 1/(l_1+l_2+l_3+\dots+l_k)! \\ 1 & 1/l_2! & 1/(l_2+l_3)! & \dots & 1/(l_2+l_3+\dots+l_k)! \\ 0 & 1 & 1/l_3! & \dots & 1/(l_3+\dots+l_k)! \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & 1/l_k! \end{pmatrix}.$$

- [M30] Пусть $h_k(z) = \sum p_{km} z^m$, где p_{km} — вероятность того, что общая длина первых k отрезков (бесконечной) случайной последовательности равна m . Найдите "простые" выражения для $h_1(z)$, $h_2(z)$ и для производящих функций $h(z, x) = \sum_k h_k(z) x^k$ от двух переменных.

10. [BM30] Определите асимптотическое поведение среднего значения и дисперсии распределения $h_k(z)$ из предыдущего упражнения при больших k .
11. [M40] Пусть $H_k(z) = \sum p_{km} z^m$, где p_{km} — вероятность того, что длина k -го отрезка в случайной (бесконечной) последовательности равна m . Выразите $H_1(z)$, $H_2(z)$ и производящую функцию $H(z, x) = \sum_k H_k(z) x^k$ от двух переменных через известные функции.
12. [M33] (П.А. Мак-Магон.) Обобщите формулу (13) на случай перестановок мультимножества, доказав, что число перестановок мультимножества $\{n_1 \cdot 1, n_2 \cdot 2, \dots, n_m \cdot m\}$, имеющих ровно k отрезков, равно

$$\sum_{0 \leq j \leq k} (-1)^j \binom{n+1}{j} \binom{n_1-1+k-j}{n_1} \binom{n_2-1+k-j}{n_2} \dots \binom{n_m-1+k-j}{n_m},$$

где $n = n_1 + n_2 + \dots + n_m$.

13. [O5] Каким будет среднее число стопок в пасьянсе Ньюкомба, если пользоваться обычной колодой для бриджа (из 52 карт), игнорируя старшинство карт, но считая, что

$$\clubsuit < \diamond < \heartsuit < \spadesuit?$$

14. [M17] Перестановка 3 1 1 1 2 3 1 4 2 3 3 4 2 2 4 4 содержит 5 отрезков; найдите с помощью приведенного в тексте построения для условия симметрии Мак-Магона соответствующую перестановку с 9-ю отрезками.
- >15. [M21] (*Перемежающиеся отрезки.*) В классической литературе 19-го века по комбинаторному анализу не изучался вопрос об отрезках в перестановках, которые рассматриваем мы, но было несколько статей, посвященных попеременно возрастающим и убывающим "отрезкам". Так, считалось, что перестановка 5 3 2 4 7 6 1 8 содержит 4 отрезка

$$532, \quad 247, \quad 761, \quad 18.$$

(Первый отрезок будет возрастающим или убывающим в зависимости от того, $a_1 < a_2$ или $a_1 > a_2$; таким образом, перестановки $a_1 a_2 \dots a_n$, $a_n \dots a_2 a_1$ и $(n+1-a_1)(n+1-a_2) \dots (n+1-a_n)$ все содержат одинаковое число перемежающихся отрезков.) Максимальное число отрезков этого типа в перестановке n элементов равно $n-1$.

Найдите среднее число перемежающихся отрезков в случайной перестановке множества $\{1, 2, \dots, n\}$.

[Указание: разберите вывод формулы (34).]

16. [M30] Продолжим предыдущее упражнение. Пусть $\langle\langle n \rangle\rangle_k$ — число перестановок множества $\{1, 2, \dots, n\}$, которые имеют ровно k перемежающихся отрезков. Найдите рекуррентное соотношение, с помощью которого можно вычислить таблицу значений $\langle\langle n \rangle\rangle_k$; найдите также соответствующее рекуррентное соотношение для производящей функции $G_n(z) = \sum_k \langle\langle n \rangle\rangle_k z^k / n!$. Используя это последнее рекуррентное соотношение, найдите простую формулу для дисперсии числа перемежающихся отрезков в случайной перестановке множества $\{1, 2, \dots, n\}$.
17. [M25] Существует всего 2^n последовательностей $a_1 a_2 \dots a_n$, где каждый элемент a_j — либо 0, либо 1. Сколько среди них последовательностей, содержащих ровно k отрезков (т. е. содержащих ровно $k-1$ элементов a_j , таких, что $a_j > a_{j+1}$)?
18. [M27] Существует всего $n!$ последовательностей $a_1 a_2 \dots a_n$, где каждый элемент a_j — целое число, лежащее в диапазоне $0 \leq a_j \leq n-j$; сколько среди них последовательностей, содержащих ровно k отрезков (т. е. содержащих ровно $k-1$ элементов a_j , таких, что $a_j > a_{j+1}$)?
- >19. [M26] (Дж. Риордан.) (а) Сколькими способами можно расположить n неатакующих ладей (т.е. никакие две не должны находиться на одной вертикали

Picture: Рис. 4. Неатакующие ладьи на шахматной доске при заданном числе ладей ниже главной диагонали.

или горизонтали) на шахматной доске размера $n \times n$ так, чтобы ровно k из них находились на заданной стороне от главной диагонали? (б) Сколькими способами можно расположить k неатакующих ладей на заданной стороне от главной диагонали шахматной доски размера $n \times n$?

Например, на рис. 4 показан один из 15619 способов расположить восемь неатакующих ладей на обычной шахматной доске с тремя ладьями на незаштрихованном участке ниже главной диагонали, а также один из 1050 способов расположить три неатакующие ладьи на треугольной доске.

- >20. [M21] Говорят, что перестановка требует k чтений, если ее нужно просмотреть k раз, слева направо, чтобы прочитать все элементы в неубывающем порядке. Например, перестановка

требует четырех чтений: при первом чтении получаем 1, 2, 3; при втором—4, 5, 6, 7; затем 8; затем 9. Найдите связь между отрезками и чтениями.

21. [M22] Если перестановка $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$ содержит k отрезков и требует j чтений в смысле упр. 20, то что можно сказать о перестановке $a_n \dots a_2 a_1$?
22. [M26] (Л. Карлиц, Д. П. Розель и Р. А. Скоувилл.) Покажите, что не существует перестановки множества $\{1, 2, \dots, n\}$ с $n + 1 - r$ отрезками, требующей s чтений, если $rs < n$, однако такая перестановка существует, если $n \geq n + 1 - r \geq s \geq 1, rs \geq n$.
23. [BM42] (Вальтер Вейсблум.) "Удлиненные отрезки" перестановки $a_1 a_2 \dots a_n$ получаются, если вставлять вертикальные черточки в тех местах, где нарушается установившаяся монотонность; удлиненные отрезки бывают как возрастающими, так и убывающими в зависимости от того, в каком порядке расположены первые два элемента, так что длина каждого удлиненного отрезка (кроме, возможно, последнего) ≥ 2 . Например, перестановка 75|62|389|14 содержит четыре удлиненных отрезка. Найдите средние длины первых двух удлиненных отрезков бесконечной перестановки и докажите, что в пределе длина удлиненного отрезка равна

$$\left(1 + \operatorname{ctg} \frac{1}{2}\right) / \left(3 - \operatorname{ctg} \frac{1}{2}\right) \approx 2.4202.$$

24. [M30] Выразите в виде функции от p среднее число отрезков в последовательностях, полученных методом, который описан в упр. 5.1.1-18.

5.1.4. *Табло и инволюции

В заключение нашего обзора комбинаторных свойств перестановок обсудим некоторые замечательные отношения, связывающие перестановки с массивами целых чисел, называемыми *табло*. *Табло Янга формы* (n_1, n_2, \dots, n_m) , где $n_1 \geq n_2 \geq \dots \geq n_m \geq 0$, это расположение $n_1 + n_2 + \dots + n_m$ различных целых чисел в массив строк, выровненных по левому краю, где в i -й строке содержится n_i элементов; при этом в каждой строке элементы возрастают слева направо, а элементы каждого столбца возрастают сверху вниз. Например,

1	2	5	9	10	15
3	6	7	13		
4	8	12	14		
11					

(1)

—табло Янга формы $(6, 4, 4, 1)$. Такие расположения ввел Альфред Янг в 1900 г. в качестве вспомогательного средства при изучении матричного представления перестановок. [См., например, D. E. Rutherford, *Substitutional Analysis* (New York: Hafner, 1968)]. Для краткости мы будем вместо "табло Янга" говорить просто "табло".

Инволюция—это перестановка, обратная самой себе. Например, существует 10 инволюций множества $\{1, 2, 3, 4\}$:

$$\begin{aligned} & \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix} \\ & \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix} \end{aligned} \quad (2)$$

Термин "инволюция" первоначально использовался в классических задачах геометрии; инволюции в общем смысле, рассматриваемые здесь, были впервые изучены Х. А. Роте, когда он ввел понятие обратной перестановки (см. п. 5.1.1).

Может показаться странным, что мы рассматриваем табло и инволюции вместе, но существует удивительная связь между этими понятиями, не имеющими, казалось бы, друг к другу никакого отношения: *число инволюций множества $\{1, 2, \dots, n\}$ равно числу табло, которые можно сформировать из элементов $\{1, 2, \dots, n\}$* . Например, из $\{1, 2, 3, 4\}$ можно сформировать ровно 10 табло

1	2	3	4	1	3	4	1	4	1	3	1	2	4
2				2			2		2		3		
				3			3		3		4		
1	2			1	2	3	1	3	1	2	1	2	3
3				4			2	4	3	4	3	4	4
4													

(3)

что соответствует 10 инволюциям (2).

Эта связь между инволюциями и табло отнюдь не очевидна, и, по-видимому, не существует простого способа ее доказать. Доказательство, которое мы обсудим, включает интересный алгоритм построения табло, обладающий и некоторыми другими неожиданными свойствами; он основан на особой процедуре вставки в табло новых элементов.

Предположим, например, что нужно вставить элемент 8 в табло

1	3	5	9	12	16
2	6	10	15		
4	13	14			
11					
17					

(4)

Метод, которым мы будем пользоваться, состоит в том, чтобы сначала поместить 8 в 1-ю строку, в клетку, ранее занимаемую 9, поскольку 9—наименьший из элементов этой строки, больших 8. Элемент 9 "вытесняется" вниз, в строку 2, где он замещает 10. Затем 10 "вытесняет" 13 из 3-й строки в 4-ю и, поскольку в 4-й строке нет элемента больше 13, процесс завершается добавлением 13 в правый конец строки 4. Наше табло преобразовалось в

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(5)

Точное описание этого процесса с доказательством того, что он всегда сохраняет свойства табло, содержится в алгоритме I.

Алгоритм I. (Вставка в табло.) Пусть $P = (P_{ij})$ —табло целых положительных чисел, а x —целое положительное число, не содержащееся в P . Этот алгоритм преобразует P в другое табло, содержащее x наряду с исходными элементами P . Новое табло имеет ту же форму, что и старое, с той лишь разницей, что на пересечении строки s и столбца t появился новый элемент, где s и t —величины, определяемые алгоритмом.

(В этом алгоритме замечания в круглых скобках предназначены для доказательства его правильности, поскольку по индукции легко проверить, что эти замечания справедливы и что массив P продолжает оставаться табло на протяжении всего процесса. Для удобства будем предполагать, что табло ограничено нулями сверху и слева и символами ∞ справа и снизу, так что элемент P_{ij} определен при всех $i, j \geq 0$. Если ввести отношение

$$a \lesssim b \quad \text{тогда и только тогда, когда } a < b \text{ или } a = b = \infty, \tag{6}$$

то неравенства для табло можно выразить в удобной форме:

$$P_{ij} = 0, \quad \text{если } i = 0 \text{ или } j = 0; \\ P_{ij} \lesssim P_{i(j+1)} \text{ и } P_{ij} \lesssim P_{(i+1)j} \quad \text{при всех } i, j \geq 0. \tag{7}$$

Утверждение " $x \notin P$ " означает, что либо $x = \infty$, либо x не равно P_{ij} ни при каких $i, j \geq 0$.)

- 11 [Ввести x .] Установить $i \leftarrow 1, x_1 \leftarrow x$, а j установить равным наименьшему значению, такому, что $P_{1j} = \infty$.
- 12 [Найти x_{i+1} .] (В этот момент $P_{(i-1)j} < x_i < P_{ij}$ и $x_i \notin P$.) Если $x_i < P_{i(j-1)}$, то уменьшить j на 1 и повторить этот шаг. В противном случае установить $x_{i+1} \leftarrow P_{ij}$ и $r_i \leftarrow j$.
- 13 [Заменить на x_i .] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Установить $P_{ij} \leftarrow x_i$.
- 14 [$x_{i+1} = \infty$?] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j-1)}$, $P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$, $r_i = j$ и $x_{i+1} \notin P$.) Если $x_{i+1} \neq \infty$, то увеличить i на 1 и вернуться к шагу 12.
- 15 [Определить s, t .] Установить $s \leftarrow i, t \leftarrow j$ и закончить работу алгоритма. (К этому моменту выполняются условия

$$P_{st} \neq \infty, P_{(s+1)t} = P_{s(t+1)} = \infty. \tag{8}$$



Заметим, что этот алгоритм определяет "последовательность вытеснений"

$$x = x_1 < x_2 < \dots < x_s < x_{s+1} = \infty, \quad (9)$$

а также вспомогательную последовательность индексов столбцов

$$r_1 \geq r_2 \geq \dots \geq r_s = t; \quad (10)$$

при этом элемент P_{ir_i} меняет свое значение с x_{i+1} на x_i , $1 \leq i \leq s$. Например, когда в табло (4) вставлялся элемент 8, последовательностью вытеснений была 8, 9, 10, 13, ∞ , а вспомогательной последовательностью—4, 3, 2, 2. Мы могли бы переформулировать алгоритм так, чтобы он расходовал меньше временной памяти (необходимо хранить только текущие значения j, x_i, x_{i+1}); последовательности (9) и (10) были введены с той лишь целью, чтобы можно было доказать некоторые интересные факты, касающиеся этого алгоритма.

Основное свойство алгоритма I, которым мы не преминем воспользоваться, состоит в том, что алгоритм можно "прокрутить" в обратном направлении: по заданным s и t , определенным в шаге I5, можно преобразовать P к исходному виду, найдя и удалив элемент x , который был вставлен. Рассмотрим, например, (5), и пусть нам известно, что элемент 13 занимает позицию, которая раньше была пуста. Тогда элемент 13, должно быть, был вытеснен вниз из 3-й строки числом 10, потому что 10—наибольший элемент в этой строке, меньший 13; аналогично, элемент 10, по-видимому, был вытеснен из 2-й строки числом 9, которое в свою очередь было вытеснено из 1-й строки числом 8. Таким образом, от (5) можно вернуться обратно к (4). В следующем алгоритме подробно описан этот процесс.

Алгоритм D. (Удаление из табло.) По заданному табло P и целым положительным числам s и t , удовлетворяющим условиям (8), этот алгоритм преобразует P в другое табло почти такой же формы, но с ∞ на пересечении столбца t и строки s . Определенный алгоритмом элемент x удаляется из табло P .

(Как и в алгоритме I, пояснения в круглых скобках включены для облегчения доказательства того, что массив P продолжает оставаться табло на протяжении всего процесса.)

D1 [Ввести s, t .] Установить $j \leftarrow t, i \leftarrow s, x_{s+1} \leftarrow \infty$.

D2 [Найти x_i .] (В этот момент $P_{ij} < x_{i+1} \lesssim P_{(i+1)j}$ и $x_{i+1} \notin P$.) Если $P_{i(j+1)} < x_{i+1}$, то увеличить j на 1 и повторить этот шаг. В противном случае установить $x_i \leftarrow P_{ij}$ и $r_i \leftarrow j$.

D3 [Заменить на x_{i+1} .] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j+1)}, P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$ и $r_i = j$.) Установить $P_{ij} \leftarrow x_{i+1}$.

D4 [$i = 1$?] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}, P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Если $i > 1$, то уменьшить t на 1 и вернуться к шагу D2.

D5 [Определить x .] Установить $x \leftarrow x_1$ и закончить работу алгоритма. (Теперь $0 < x < \infty$.) ■

Пояснения к алгоритмам I и D, заключенные в круглые скобки, не только полезны для доказательства того факта, что эти алгоритмы сохраняют структуру табло, но они позволяют убедиться, что алгоритмы I и D взаимно обратны. Если сначала выполнить алгоритм I с данным табло P и некоторым целым положительным числом $x \notin P$, то он вставит x в P и определит целые положительные числа s, t , удовлетворяющие условиям (8). Алгоритм D, примененный к полученному результату, восстановит значения x и первоначальный вид P . Обратно, если сначала выполнить алгоритм D с данным табло P и некоторыми целыми положительными числами s, t , удовлетворяющими условиям (8), то он модифицирует P , удалив некоторое целое положительное число x . Алгоритм I, примененный к полученному результату, восстановит значения s, t и первоначальный вид P . Причина заключается в том, что содержание пояснений к шагам I3 и D4 совпадает так же, как и к шагам I4 и D3; они однозначно определяют значение j . Следовательно, вспомогательные последовательности (9), (10) одинаковы, в обоих случаях.

Теперь мы подготовлены к доказательству основного свойства табло.

Теорема А. Существует взаимно однозначное соответствие между множеством всех перестановок множества $\{1, 2, \dots, n\}$ и множеством всех упорядоченных пар табло (P, Q) , где P и Q —табло одинаковой формы из элементов $\{1, 2, \dots, n\}$.

(Пример к этой теореме содержится в приведенном ниже доказательстве.)

Доказательство. Удобнее доказать несколько более общий результат. По произвольному двустрочному массиву

$$\begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}, \quad \begin{array}{l} q_1 < q_2 < \dots < q_n, \\ p_1, p_2, \dots, p_n \text{ все разные,} \end{array} \quad (11)$$

построим соответствующие два табло P и Q , где P состоит из элементов $\{p_1, p_2, \dots, p_n\}$, а Q —из элементов $\{q_1, q_2, \dots, q_n\}$, причем P и Q имеют одинаковую форму.

Пусть P и Q вначале пусты. При $i = 1, 2, \dots, n$ (именно в таком порядке) сделаем следующую операцию: вставим P_i в табло P при помощи алгоритма I; затем установим $Q_{st} \leftarrow q_i$, где s и t определяют вновь заполненную позицию в P .

Например, если задана перестановка $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$, действуем следующим образом:

	P		Q
Вставка 7:	7		1
Вставка 2:	2 7		1 3
Вставка 9:	2 9 7		1 5 3
Вставка 5:	2 5 7 9		1 5 3 6
Вставка 3:	2 3 5 9 7		1 5 3 6 8

(12)

Следовательно, пара табло (P, Q) , соответствующая перестановке $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$, такова:

$$P = \begin{pmatrix} 2 & 3 \\ 5 & 9 \\ 7 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & 5 \\ 3 & 6 \\ 8 \end{pmatrix}. \tag{13}$$

Из построения ясно, что P и Q всегда имеют одинаковую форму. Кроме того, поскольку элементы всегда добавляются на границу Q и в возрастающем порядке, то Q —табло.

Обратно, если заданы два табло одинаковой формы, то соответствующий двустрочный массив (11) можно построить так. Пусть

$$q_1 < q_2 < \dots < q_n$$

—элементы Q . Пусть при $i = n, \dots, 2, 1$ (именно в таком порядке) p_i —элемент x , который удаляется из P по алгоритму D с использованием значений s, t , таких, что $Q_{st} = q_i$.

Например, если применить это построение к табло (13) и производить вычисления, обратные (12), до тех пор, пока P не исчерпается, то получится массив $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$.

Поскольку алгоритмы I и D взаимно обратны, то взаимно обратны и описанные здесь два построения; таким образом, требуемое взаимно однозначное соответствие установлено. ■

Соответствие, определенное в доказательстве теоремы А, обладает множеством поразительных свойств, и теперь мы приступим к выводу некоторых из них. Убедительная просьба к читателю, прежде чем двигаться дальше, испытать себя на примерах из упр. 1, чтобы освоиться с этими построениями.

Как только элемент вытеснен из строки 1 в строку 2, он уже больше не влияет на строку 1; кроме того, строки 2, 3, ... строятся из последовательности "вытесненных" элементов точно так же, как строки 1, 2, ... строятся из исходной перестановки. Эти факты наводят на мысль о том, что на построение в теореме А можно взглянуть иначе, обращая внимание лишь на первые строки P и Q .

Например, перестановка $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$ вызывает следующие действия над строкой 1 [ср. с (12)]:

- 1: Вставить 7, установить $Q_{11} \leftarrow 1$.
 - 3: Вставить 2, вытеснить 7.
 - 5: Вставить 9, установить $Q_{12} \leftarrow 5$.
 - 6: Вставить 5, вытеснить 9.
 - 8: Вставить 3, вытеснить 5.
- (14)

Таким образом, первая строка P —это 23, а первая строка Q —это 15. Кроме того, остальные строки P и Q составляют табло, соответствующие "вытесненному" двустрочному массиву

$$\begin{pmatrix} 3 & 6 & 8 \\ 7 & 9 & 5 \end{pmatrix}, \tag{15}$$

Чтобы понять, как строится строка 1, можно изучить элементы, попадающие в данный столбец этой строки. Будем говорить, что пара (q_i, p_i) принадлежит классу t относительно двустрочного массива

$$\begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}, \quad \begin{array}{l} q_1 < q_2 < \dots < q_n, \\ p_1, p_2, \dots, p_n \text{ все разные,} \end{array} \quad (16)$$

если $p_i = P_{1t}$ после применения алгоритма I последовательно к p_1, p_2, \dots, p_i , начиная с пустого табло P . (Напомним, что алгоритм I всегда вставляет данный элемент в 1-ю строку.)

Легко видеть, что (q_i, p_i) принадлежит классу 1 тогда и только тогда, когда p_i имеет $(i-1)$ инверсий, т. е. $p_i = \min\{p_1, p_2, \dots, p_i\}$ — "левосторонний минимум". Если в массиве (16) вычеркнуть столбцы класса 1, то получится другой двустрочный массив:

$$\begin{pmatrix} q'_1 & q'_2 & \dots & q'_m \\ p'_1 & p'_2 & \dots & p'_m \end{pmatrix} \quad (17)$$

такой, что пара (q, p) принадлежит классу t относительно (17) тогда и только тогда, когда она принадлежит классу $t+1$ относительно массива (16). Операция перехода от (16) к (17) соответствует удалению крайней левой позиции строки 1. Это дает систематический способ определения классов.

Например, в перестановке $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$ левосторонними минимумами являются элементы 7 и 2,

так что класс 1 — это $\{(1, 7), (3, 2)\}$; в оставшемся массиве $\begin{pmatrix} 5 & 6 & 8 \\ 9 & 5 & 3 \end{pmatrix}$ все элементы минимальны, так что класс 2 — это $\{(5, 9), (6, 5), (8, 3)\}$. В "вытесненном" массиве (15) класс 1 — это $\{(3, 7), (8, 5)\}$, а класс 2 — $\{(6, 9)\}$.

Для любого фиксированного t элементы класса t можно пометить

$$(q_{i_1}, p_{i_1}), \dots, (q_{i_k}, p_{i_k}),$$

так чтобы выполнялись неравенства

$$\begin{array}{l} q_{i_1} < q_{i_2} < \dots < q_{i_k}, \\ p_{i_1} > p_{i_2} > \dots > p_{i_k}, \end{array} \quad (18)$$

поскольку при работе алгоритма вставки позиция табло P_{1t} принимает убывающую последовательность значений p_{i_1}, \dots, p_{i_k} . В конце построения

$$p_{1t} = p_{i_k}, \quad Q_{1t} = q_{i_1}, \quad (19)$$

а вытесненный двустрочный массив, которым определяются строки 2, 3, ... табло P и Q , содержит столбцы

$$\begin{pmatrix} q_{i_2} & q_{i_3} & \dots & q_{i_k} \\ p_{i_1} & p_{i_2} & \dots & p_{i_{k-1}} \end{pmatrix}, \quad (20)$$

а также другие столбцы, аналогичным образом полученные из других классов.

Эти наблюдения приводят к простому методу вычисления P и Q вручную (см. упр. 3), а также предоставляют средства для доказательства одного весьма неожиданного результата.

Теорема В. Если в построении из теоремы А перестановка

$$\begin{pmatrix} 1 & 2 & \dots & n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$$

соответствует табло (P, Q) , то обратная ей перестановка соответствует табло (Q, P) .

Это довольно удивительный факт, потому что в теореме А табло P и Q формируются совершенно разными способами, и обратная перестановка получается в результате весьма причудливой перестановки столбцов двустрочного массива.

Доказательство. Предположим, у нас есть двустрочный массив (16); поменяв местами его строки и отсортировав столбцы так, чтобы элементы новой верхней строки расположились в неубывающем порядке, получим "обратный" массив

$$\begin{aligned} \begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}^{-1} &= \begin{pmatrix} p_1 & p_2 & \dots & p_n \\ q_1 & q_2 & \dots & q_n \end{pmatrix} = \\ &= \begin{pmatrix} p'_1 & p'_2 & \dots & p'_n \\ q'_1 & q'_2 & \dots & q'_n \end{pmatrix}, \quad \begin{array}{l} p'_1 < p'_2 < \dots < p'_n; \\ q'_1, q'_2, \dots, q'_n \text{ все разные.} \end{array} \end{aligned} \quad (21)$$

Покажем, что эта операция соответствует замене (P, Q) на (Q, P) в построении из теоремы А.

В упр. 2 наши замечания об определении классов переформулированы таким образом, что класс, к которому относится пара (q_i, p_i) , не зависит от того факта, что элементы q_1, q_2, \dots, q_n расположены в возрастающем порядке. Поскольку результирующие условия симметричны относительно p и q , то операция (21) не нарушает структуру классов; если (q, p) принадлежит классу t относительно (16), то (p, q) принадлежит классу t относительно (21). Поэтому, если разместить элементы этого последнего класса t так, чтобы

$$\begin{aligned} p_{i_k} &< \dots < p_{i_2} < p_{i_1}, \\ q_{i_k} &> \dots > q_{i_2} > q_{i_1}, \end{aligned} \tag{22}$$

[ср. с (18)], то получим

$$P_{1t} = q_{i_1}, Q_{1t} = p_{i_k}, \tag{23}$$

как в (19), а столбцы

$$\begin{pmatrix} p_{i_{k-1}} & \dots & p_{i_2} & p_{i_1} \\ q_{i_k} & \dots & q_{i_3} & q_{i_2} \end{pmatrix} \tag{24}$$

войдут в вытесненный массив, как в (20). Следовательно, первые строки P и Q меняются местами. Кроме того, вытесненный двустрочный массив для (21) является обратным по отношению к вытесненному двустрочному массиву для (16), так что доказательство завершается применением индукции по числу строк в табло. ■

Следствие. *Количество табло, которые можно сформировать из элементов $\{1, 2, \dots, n\}$, равно количеству инволюций множества $\{1, 2, \dots, n\}$.*

Доказательство. Если π —инволюция, соответствующая паре табло (P, Q) , то $\pi = \pi^{-1}$ соответствует паре (Q, P) . Следовательно, $P = Q$. Обратно, если π —какая-либо перестановка, соответствующая паре (P, P) , то π^{-1} тоже соответствует паре (P, P) ; отсюда $\pi = \pi^{-1}$. Таким образом, существует взаимно однозначное соответствие между инволюциями π и табло P . ■

Ясно, что элемент в левом верхнем углу табло всегда наименьший. Это наводит на мысль о возможном способе сортировки множества чисел. Сначала можно составить из них табло, многократно применяя алгоритм I, в результате наименьший элемент окажется в углу. Затем этот наименьший элемент удаляется, а остальные элементы переразмещаются так, чтобы образовалось другое табло; потом удаляется новый минимальный элемент и т.д.

Поэтому давайте посмотрим, что происходит, когда мы удаляем угловой элемент из табло

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(25)

После удаления 1 на освободившееся место необходимо поставить 2. Затем можно поднять 4 на место двойки, однако 11 нельзя поднять на место 4, но на это место можно подвинуть 10, а потом 13 на место 10. В общем случае приходим к следующей процедуре.

Алгоритм S. (Удаление углового элемента.) Этот алгоритм удаляет элемент из левого верхнего угла табло P и перемещает остальные элементы так, чтобы сохранились свойства табло. Используются те же обозначения, что и в алгоритмах I и D.

S1 [Начальная установка.] Установить $r \leftarrow 1, s \leftarrow 1$.

S2 [Конец?] Если $P_{rs} = \infty$, то процесс завершен.

S3 [Сравнить.] Если $P_{(r+1)s} \lesssim P_{r(s+1)}$, то перейти к шагу S5. (Сравниваем элементы справа и снизу от свободного места и передвигаем меньший из них.)

S4 [Подвинуть влево.] Установить $P_{rs} \leftarrow P_{r(s+1)}, s \leftarrow s + 1$ и вернуться к S3.

S5 [Подвинуть вверх.] Установить $P_{rs} \leftarrow P_{(r+1)s}, r \leftarrow r + 1$ и вернуться к S2. ■

Легко доказать, что после удаления углового элемента с помощью алгоритма S, P —по-прежнему табло (см. упр. 10). Таким образом, применяя алгоритм S до тех пор, пока P не исчерпается, можно прочесть его элементы в возрастающем порядке. К сожалению, этот алгоритм сортировки оказывается не столь эффективным, как другие алгоритмы, которые нам еще предстоит рассмотреть. Минимальное время его работы пропорционально $n^{1.5}$; аналогичные алгоритмы, использующие вместо табло деревья, затрачивают время порядка $n \log n$.

Алгоритм S, хотя и не приводит к особенно эффективному алгоритму сортировки, обладает некоторыми очень интересными свойствами.

Теорема С. (М. П. Шюценберже.) Если P —табло, построенное, как в теореме А, из перестановки $a_1 a_2, \dots, a_n$, и $a_i = \min\{a_1, a_2, \dots, a_n\}$, то алгоритм S преобразует P в табло, соответствующее перестановке $a_1 \dots a_{i-1} a_{i+1} \dots a_n$.

Доказательство. См. упр. 13. ■

Давайте после применения алгоритма S поместим на вновь освободившееся место удаленный элемент в скобках, указав таким образом, что на самом деле он не является частью табло. Применив, например, эту процедуру к табло (25), мы получили бы

2	3	5	8	12	16
4	6	9	15		
10	13	14			
11	(1)				
17					

а еще два применения приводят к

4	5	8	12	16	(2)
6	9	14	15		
10	13	(3)			
11	(1)				
17					

Продолжая до тех пор, пока все элементы не окажутся в скобках, и убрав скобки, получим конфигурацию

17	15	14	13	11	2
16	10	6	4		
12	5	3			
9	1				
8					

 (26)

имеющую ту же форму, что и исходное табло (25). Эту конфигурацию можно назвать *двойственным табло*, потому что она похожа на табло с той лишь разницей, что применяется "двойственное отношение порядка" ($<$ и $>$ поменялись ролями). Обозначим двойственное табло, полученное из P таким способом, через P^S .

Табло P определяется из P^S единственным образом. В самом деле, исходное табло можно получить из P^S при помощи того же самого алгоритма (с обратным отношением порядка, поскольку P^S —двойственное табло). Например, применение к (26) двух шагов этого алгоритма дает

15	14	13	11	2	(16)
12	10	6	4		
9	5	3			
8	1				
(17)					

и в конце концов опять получается табло (25). Этот замечательный результат—одно из следствий нашей следующей теоремы.

Теорема D. (К. Шенстед, М. П. Шюценберже.) Пусть

$$\begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix} \quad (27)$$

—двустрочный массив, соответствующий паре табло (P, Q) .

а) Если пользоваться двойственным (обратным) отношением порядка для q , но не для p , то двустрочный массив

$$\begin{pmatrix} q_n & \dots & q_2 & q_1 \\ p_n & \dots & p_2 & p_1 \end{pmatrix} \quad (28)$$

соответствует паре $(P^T, (Q^S)^T)$.

(Как обычно, через T обозначена операция транспонирования; P^T —табло, а $(Q^S)^T$ — двойственное табло, поскольку элементы q расположены в обратном порядке.)

b) Если пользоваться двойственным отношением порядка для p , но не для q , то двустрочный массив (37) соответствует паре $((P^S)^T, Q^T)$.

c) Если пользоваться двойственным отношением порядка как для p , так и для q , то двустрочный массив (28) соответствует паре (P^S, Q^S) .

Доказательство. Не известно простого доказательства этой теоремы. То, что случай (а) соответствует паре (P^T, X) , где X —некоторое двойственное табло, доказано в упр. 6; следовательно, по теореме В, случай (b) соответствует паре (Y, Q^T) , где Y —некоторое двойственное табло той же формы, что и P^T .

Пусть $p_i = \min\{p_1, \dots, p_n\}$; так как p_i —"наибольший" элемент при двойственном отношении порядка, то он окажется на границе Y и не вытесняет никаких элементов при построении из теоремы А. Таким образом, если последовательно вставлять элементы $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$, применяя двойственное отношение порядка, то получится $Y - \{p_i\}$, т.е. Y , из которого удален элемент p_i . По теореме С, если последовательно вставлять элементы $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$, применяя обычное отношение порядка, построим табло $d(P)$, которое получается путем применения к P алгоритма S. Индукция по n дает $Y - \{p_i\} = (d(P)^S)^T$. Но поскольку

$$(P^S)^T - \{p_i\} = (d(P)^S)^T \quad (29)$$

по определению операции S , а Y имеет ту же форму, что и $(P^S)^T$, то должно иметь место равенство $Y = (P^S)^T$.

Тем самым доказано утверждение (b); (а) получается применением теоремы В. Последовательное применение (а) и (b) показывает, что случай (c) соответствует паре $((P^T)^S)^T, ((Q^S)^T)^T$, а это равно (P^S, Q^S) , так как $(P^S)^T = (P^T)^S$ вследствие симметрии операции S по отношению к строкам и столбцам. ■

Эта теорема, в частности, устанавливает два удивительных факта, касающихся алгоритма вставки в табло. Если в результате последовательной вставки различных элементов p_1, \dots, p_n в пустое табло получается табло P , то в результате вставки этих элементов в обратном порядке — p_n, \dots, p_1 , получится *транспонированное* табло P^T . Если же мы не только станем вставлять элементы в порядке p_n, \dots, p_1 , но и поменяем ролями $<$ и $>$, а также 0 и ∞ , то получим двойственное табло P^S . Настоятельно рекомендуем читателю испытать эти процессы на нескольких простых примерах. Необычная природа этих совпадений может вызвать подозрение о вмешательстве каких-то колдовских сил. До сих пор не известно какого-либо простого объяснения подобных явлений; кажется, не существует простого способа доказать даже то, что случай (c) соответствует табло той же формы, что P и Q .

Соответствие, устанавливаемое теоремой А, найдено Ж. Робинсоном [*American J. Math.*, **60** (1938), 745–760, Sec. 5] в несколько иной и довольно туманной форме как часть решения весьма сложной задачи из теории групп. Нетрудно проверить, что его построение в сущности идентично приведенному здесь. Он сформулировал теорему В без доказательства. Много лет спустя К. Шенстед независимо заново открыл это соответствие, которое он сформулировал по существу в той же форме, какую использовали мы [*Canadian J. Math.*, **13** (1961), 179–191]. Он также доказал "P"-часть теоремы D (а). М. П. Шюценберже [*Math. Scand.*, **12** (1963), 117–128] доказал теорему В и "Q"-часть теоремы D (а), из которой следуют (b) и (c). Это соответствие можно распространить и на перестановки мультимножеств; случай, когда p_1, \dots, p_n не обязательно различны, рассмотрел Шенстед, а "максимальное" обобщение на случай, когда p , и q могут содержать повторяющиеся элементы, исследовано Кнуттом [*Pacific J. Math.*, **34** (1970), 709–727].

Обратимся теперь к родственному вопросу: сколько табло, составленных из элементов $\{1, 2, \dots, n\}$, имеют данную форму (n_1, n_2, \dots, n_m) , где $n_1 + n_2 + \dots + n_m = n$? Обозначим это число через $f(n_1, n_2, \dots, n_m)$; оно должно удовлетворять соотношениям

$$f(n_1, n_2, \dots, n_m) = 0, \quad \text{если не выполнено условие } n_1 \geq n_2 \geq \dots \geq n_m \geq 0; \quad (30)$$

$$f(n_1, n_2, \dots, n_m, 0) = f(n_1, n_2, \dots, n_m); \quad (31)$$

$$f(n_1, n_2, \dots, n_m) = f(n_1 - 1, n_2, \dots, n_m) + f(n_1, n_2 - 1, \dots, n_m) + \dots + f(n_1, n_2, \dots, n_m - 1), \\ \text{если } n_1 \geq n_2 \geq \dots \geq n_m \geq 1. \quad (32)$$

Последнее рекуррентное соотношение вытекает из того факта, что при удалении наибольшего элемента табло всегда снова получается табло; например, количество табло формы $(6, 4, 4, 1)$ равно $f(5, 4, 4, 1) +$

$f(6, 3, 4, 1) + f(6, 4, 3, 1) + f(6, 4, 4, 0) = f(5, 4, 4, 1) + f(6, 4, 3, 1) + f(6, 4, 4)$, потому что всякое табло формы $(6, 4, 4, 1)$ из элементов $\{1, 2, \dots, 15\}$ получается в результате вставки элемента 15 в подходящее место в табло формы $(5, 4, 4, 1)$, $(6, 4, 3, 1)$ или $(6, 4, 4)$. Изобразим это на схеме

Picture: р. 80, (33)

Функция $f(n_1, n_2, \dots, n_m)$, удовлетворяющая таким соотношениям, имеет довольно простой вид:

$$f(n_1, n_2, \dots, n_m) = \frac{\Delta(n_1 + m - 1, n_2 + m - 2, \dots, n_m)n!}{(n_1 + m - 1)!(n_2 + m - 2)! \dots n_m!} \quad \text{при } n_1 + m - 1 \geq n_2 + m - 2 \geq \dots \geq n_m, \quad (34)$$

где через Δ обозначен детерминант

$$\Delta(x_1, x_2, \dots, x_m) = \det \begin{pmatrix} x_1^{m-1} & x_2^{m-1} & \dots & x_m^{m-1} \\ \vdots & \vdots & & \vdots \\ x_1^2 & x_2^2 & & x_m^2 \\ x_1 & x_2 & & x_m \\ 1 & 1 & \dots & 1 \end{pmatrix} = \prod_{1 \leq i < j \leq m} (x_i - x_j) \quad (35)$$

Формулу (34) вывел Ф. Фробениус [Sitzungsberichte Preuss. Akad. der Wissenschaften (Berlin, 1900), 516–534, Sec. 3], изучая эквивалентную задачу теории групп; он использовал довольно глубокую аргументацию, опирающуюся на теорию групп. Комбинаторное доказательство независимо нашел Мак-Магон [*Philosophical Trans.*, A-209 (London, 1909), 153–175]. Эту формулу можно доказать по индукции, так как (30) и (31) доказываются без труда, а формула (32) получается, если положить $y = -1$ в тождестве упр. 17.

Из теоремы А в связи с этой формулой для числа табло вытекает замечательное тождество. Взяв сумму по всевозможным формам табло, получим

$$\begin{aligned} n! &= \sum_{\substack{k_1 \geq \dots \geq k_n \geq 0 \\ k_1 + \dots + k_n = n}} f(k_1, \dots, k_n)^2 = \\ &= n!^2 \sum_{\substack{k_1 \geq \dots \geq k_n \geq 0 \\ k_1 + \dots + k_n = n}} \frac{\Delta(k_1 + n - 1, \dots, k_n)^2}{(k_1 + n - 1)!^2 \dots k_n!^2} = \\ &= n!^2 \sum_{\substack{q_1 > q_2 > \dots > q_n \geq 0 \\ q_1 + \dots + q_n = (n+1)n/2}} \frac{\Delta(q_1, \dots, q_n)^2}{q_1!^2 \dots q_n!^2}; \end{aligned}$$

отсюда

$$\sum_{\substack{q_1 + \dots + q_n = (n+1)n/2 \\ q_1 \dots q_n \geq 0}} \frac{\Delta(q_1, \dots, q_n)^2}{q_1!^2 \dots q_n!^2} = 1. \quad (36)$$

В последней сумме отсутствуют неравенства $q_1 > q_2 > \dots > q_n$, потому что слагаемые — симметричные относительно q функции, обращающиеся в 0 при $q_i = q_j$. Аналогичное тождество появляется в упр. 24.

Формулу числа табло можно выразить несколько более интересным способом, если ввести понятие "уголков". В *уголок*, соответствующий

Picture: Рис 5. Уголки и длины уголков.

клетке табло, входит сама эта клетка плюс все клетки, лежащие снизу и справа от нее. Например, заштрихованный участок на рис. 5 — уголок, соответствующий клетке $(2, 3)$ строки 2 и столбца 3; он состоит из шести клеток. В каждой клетке на рис. 5 записана длина соответствующего ей уголка.

Если табло имеет форму (n_1, n_2, \dots, n_m) , где $n_m \geq 1$, то длина самого длинного уголка равна $n_1 + m - 1$. Дальнейшее исследование длин уголков показывает, что строка 1 содержит все длины $n_1 + m - 1, n_1 + m - 2, \dots, 1$, кроме $(n_1 + m - 1) - (n_m), (n_1 + m - 1) - (n_{m-1} + 1), \dots, (n_1 + m - 1) - (n_2 + m - 2)$. Например, на рис. 5 длины уголков в 1-й строке суть 12, 11, 10, ..., 1, за исключением 10, 9, 6, 3, 2; эти исключения соответствуют пяти несуществующим уголкам, начинающимся в несуществующих клетках $(6, 3), (5, 3), (4, 5), (3, 7), (2, 7)$ и заканчивающимся в клетке $(1, 7)$. Аналогично j -я строка содержит все длины уголков $n_j + m - j, \dots, 1$, кроме $(n_j + m - j) - (n_m), \dots, (n_j + m - j) - (n_{j+1} + m - j - 1)$. Отсюда следует, что произведение длин всех уголков равно

$$(n_1 + m - 1)! \dots (n_m)! / \Delta(n_1 + m - 1, \dots, n_m).$$

Это выражение входит в формулу (34); отсюда следует

Теорема Н. (Дж. С. Фрэйм, Ж. Робинсон, Р. М. Тролл.) *Количество табло данной формы, составленных из элементов $\{1, 2, \dots, n\}$, равно $n!$, деленному на произведение длин уголков.* ■

Такой простой результат заслуживает и простого доказательства; однако (как и для большинства других фактов, касающихся табло) более прямого доказательства не известно. Каждый элемент табло—минимальный в своем уголке; если заполнить клетки табло случайным образом, то вероятность того, что в клетке (i, j) окажется минимальный элемент соответствующего уголка, есть величина, обратная длине уголка. Перемножение этих вероятностей по всем i, j дает теорему Н. Однако такое рассуждение ошибочно, поскольку эти вероятности отнюдь не являются независимыми. Все известные доказательства теоремы Н основаны на одном неубедительном индуктивном рассуждении, которое на самом деле не объясняет, почему же теорема верна (так как в нем совершенно не используются свойства уголков).

Существует интересная связь между теоремой Н и перечислением деревьев, которое рассматривалось в гл. 2. Мы видели, что бинарным деревьям с n узлами соответствуют перестановки, которые можно получить с помощью стека, и что таким перестановкам соответствуют последовательности $a_1 a_2 \dots a_{2n}$ литер S и X , такие, что количество литер S никогда не бывает меньше количества литер X , если читать последовательность слева направо. (См. упр. 2.3.1-6 и 2.2.1-3.) Таким последовательностям естественным образом сопоставляются табло формы (n, n) ; в 1-ю строку помещаются индексы i , такие, что $a_i = S$, а во 2-ю строку—индексы, при которых $a_i = X$. Например, последовательности

$$S S S X X S S X X S X X$$

соответствует табло

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 6 & 7 & 10 \\ \hline 4 & 5 & 8 & 9 & 11 & 12 \\ \hline \end{array} \tag{37}$$

Условие, налагаемое на столбцы, удовлетворяется в таком табло в том и только том случае, когда при чтении слева направо число литер X никогда не превышает числа литер S . По теореме Н количество всевозможных табло формы (n, n) равно

$$(2n)!/(n+1)!n!;$$

следовательно, таково же и число бинарных деревьев с n узлами (что согласуется с формулой (2.3.4.4-13)). Более того, если воспользоваться табло формы (n, m) при $n \geq m$, то при помощи этого рассуждения можно решить и более общую "задачу о баллотировке", рассмотренную в ответе к упр. 2.2.1-4. Таким образом, теорема Н в качестве простых частных случаев включает в себя некоторые весьма сложные задачи о перечислении.

Всякому табло A формы (n, n) из элементов $\{1, 2, \dots, 2n\}$ соответствуют два табло (P, Q) одинаковой формы. Следующий способ построения такого соответствия предложен Мак-Магоном [Combinatory Analysis, 1 (1915), 130–131]. Пусть P состоит из элементов $\{1, \dots, n\}$, расположенных, как в A , а Q получается, если взять остальные элементы A , повернуть всю конфигурацию на 180° и заменить $n+1, n+2, \dots, 2n$ на соответственно $n, n-1, \dots, 1$. Например, табло (37) распадается на

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 6 \\ \hline 4 & 5 & & \\ \hline \end{array} \text{ и } \begin{array}{|c|c|c|c|} \hline & & 7 & 10 \\ \hline 8 & 9 & 11 & 12 \\ \hline \end{array};$$

после поворота и перенумерации имеем

$$P = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 6 \\ \hline 4 & 5 & & \\ \hline \end{array}, \quad Q = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 4 & 5 \\ \hline 3 & 6 & & \\ \hline \end{array}. \tag{38}$$

Наоборот, каждой паре табло одинаковой формы, состоящих из n элементов и из не более двух строк, соответствует табло формы (n, n) . Следовательно (из упр. 7), *число перестановок $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$, не содержащих убывающих подпоследовательностей $a_i > a_j > a_k$ при $i < j < k$, равно числу бинарных деревьев с n узлами.* Интересное взаимно однозначное соответствие между такими перестановками и бинарными деревьями, установленное более прямым способом, чем тот окольный путь посредством алгоритма I, которым воспользовались мы, нашел Д. Ротом [Inf. Proc. Letters, 4 (1975), 58–61]; аналогично, существует довольно прямое соответствие между бинарными деревьями и перестановками, не содержащими подпоследовательностей $a_j > a_k > a_i$ и $i < j < k$ (см. упр. 2.2.1-5).

Число способов заполнить табло формы $(6, 4, 4, 1)$, очевидно, равно числу способов пометить вершины направленного графа

числами $\{1, 2, \dots, 15\}$ так, чтобы метка вершины u была меньше метки вершины v , если $u \rightarrow v$. Иначе говоря, оно равно числу способов топологически отсортировать частичное упорядочение (39) в смысле п. 2.2.3.

В общем случае можно задать тот же вопрос для произвольного направленного графа, не содержащего ориентированных циклов. Было бы хорошо, если бы существовала какая-нибудь простая формула, обобщающая теорему Н на случай произвольного графа; однако не все графы обладают такими приятными свойствами, как графы, соответствующие табло. Другие классы направленных графов, для которых задача о разметке вершин имеет простое решение, частично обсуждаются в упражнениях в конце этого пункта. Там также имеются упражнения, в которых показано, что в ряде случаев направленных графов *не* существует простой формулы, соответствующей теореме Н. Например, число способов пометить вершины не всегда является делителем $n!$.

Завершим наши исследования подсчетом общего числа табло, которые можно сформировать из n различных элементов. Обозначим это число через t_n . По следствию из теоремы В t_n равно числу инволюций множества $\{1, 2, \dots, n\}$. Перестановка обратна самой себе тогда и только тогда, когда ее представление с помощью циклов состоит только из единичных циклов (неподвижных точек) и циклов из двух элементов (транспозиций). Поскольку в t_{n-1} из t_n инволюций (n) —неподвижная точка, а в t_{n-2} из них (j, n) —цикл из двух элементов при некотором фиксированном $j < n$, то получаем формулу

$$t_n = t_{n-1} + (n-1)t_{n-2}, \quad (40)$$

которую вывел Роте в 1800 г. для получения таблицы значений t_n при малых n .

Вычислим t_n другим способом. Пусть имеется k циклов из двух элементов и $n - 2k$ единичных циклов. Существует $\binom{n}{2k}$ способов выбрать неподвижные точки, и мультиномиальный коэффициент $(2k)!/(2!)^k$ равен числу способов организовать остальные элементы в k различных транспозиций. Поделив на $k!$, чтобы сделать эти транспозиции неразличимыми, получим

$$t_n = \sum_{k \geq 0} t_n(k), \quad t_n(k) = \frac{n!}{(n-2k)!2^k k!}. \quad (41)$$

К сожалению, насколько это известно, такую сумму уже нельзя упростить, поэтому, для того чтобы лучше понять поведение величины t_n , мы применим два косвенных подхода:

а) Можно найти производящую функцию

$$\sum_n t_n z^n / n! = e^{z+z^2/2}; \quad (42)$$

см. упр. 25.

б) Можно определить асимптотическое поведение величины t_n . Это поучительная задача, ее решение содержит несколько общих методов, которые будут полезны и в связи с другими вопросами; поэтому мы заключим этот пункт анализом асимптотического поведения t_n .

Первый шаг в анализе асимптотического поведения (41) состоит в выделении слагаемых, дающих основной вклад в сумму. Поскольку

$$\frac{t_n(k+1)}{t_n(k)} = \frac{(n-2k)(n-2k-1)}{2(k+1)}, \quad (43)$$

можно видеть, что слагаемые постепенно возрастают, когда k меняется от $k=0$ до k , равного приблизительно $\frac{1}{2}(n - \sqrt{n})$, а затем убывают до нуля, когда k достигает значения, равного примерно $\frac{1}{2}n$. Ясно, что основной вклад дают члены в окрестности $k = \frac{1}{2}(n - \sqrt{n})$. Для анализа, однако, удобнее, когда основной вклад достигается при значении 0, поэтому представим k в виде

$$k = \frac{1}{2}(n - \sqrt{n}) + x \quad (44)$$

и исследуем величину $t_n(k)$ как функцию от x .

Чтобы избавиться от факториалов в $t_n(k)$, полезно воспользоваться формулой Стирлинга (1.2.11.2-18). Для этой цели удобно (как мы вскоре увидим) ограничить область изменения x промежутком

$$-(n^{\varepsilon+1/4}) \leq x \leq n^{\varepsilon+1/4}, \quad (45)$$

где ε равно, скажем, 0.001, так что можно включить слагаемое погрешности. После довольно трудоемких вычислений, которые на самом деле должны были быть проделаны вычислительной машиной, получается формула

$$t_n(k) = \exp \left(\frac{1}{2}n \ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4} \ln n - \frac{2x^2}{\sqrt{n}} - \frac{1}{4} - \frac{1}{2} \ln \pi - \frac{4}{3}x^3/n + 2x/\sqrt{n} + \frac{1}{3}/\sqrt{n} - \frac{4}{3}x^4/n\sqrt{n} + O(n^{5\varepsilon-3/4}) \right). \quad (46)$$

Ограничение (45) на x оправдывается тем, что можно положить $x = \pm n^{\varepsilon+1/4}$ для получения верхней оценки всех отброшенных слагаемых, именно

$$e^{-2n^{2\varepsilon}} \cdot \exp \left(\frac{1}{2}n \ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4} \ln n - \frac{1}{4} - \frac{1}{2} \ln \pi + O(n^{3\varepsilon-1/4}) \right), \quad (47)$$

а если умножить это на n , то получится верхняя оценка суммы исключенных слагаемых. Верхняя оценка имеет меньший порядок, чем те слагаемые, которые мы вычислим для x в ограниченном промежутке (45), благодаря множителю $\exp(-2n^{2\varepsilon})$, который много меньше любого полинома от n . Очевидно, можно отбросить в сумме множитель

$$\exp \left(\frac{1}{2}n \ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4} \ln n - \frac{1}{4} - \frac{1}{2} \ln \pi + \frac{1}{3}/\sqrt{n} \right), \quad (48)$$

после чего нам останется только просуммировать

$$\begin{aligned} & \exp \left(-2x^2/\sqrt{n} - \frac{4}{3}x^3/n + 2x/\sqrt{n} - \frac{4}{3}x^4/n\sqrt{n} + O(n^{5\varepsilon-3/4}) \right) = \\ & = \exp \left(\frac{-2x^2}{\sqrt{n}} \right) \left(1 - \frac{4}{3} \frac{x^3}{n} + \frac{8}{9} \frac{x^6}{n^2} \right) \times \left(1 + 2 \frac{x}{\sqrt{n}} + 2 \frac{x^2}{n} \right) \left(1 - \frac{4}{3} \frac{x^4}{n\sqrt{n}} \right) (1 + O(n^{9\varepsilon-3/4})) \end{aligned} \quad (49)$$

по всем $x = \alpha, \alpha+1, \dots, \beta-2, \beta-1$, где $-\alpha$ и β (не обязательно целые) равны приблизительно $n^{\varepsilon+1/4}$. Если сместить интервал суммирования, то формулу суммирования Эйлера (1.2.11.2-10) можно записать в виде

$$\sum_{\alpha \leq x < \beta} f(x) = \int_{\alpha}^{\beta} f(x) dx - \frac{1}{2}f(x) \Big|_{\alpha}^{\beta} + \frac{1}{2}B_2 \cdot \frac{f'(x)}{1!} \Big|_{\alpha}^{\beta} + \dots + \frac{1}{m+1}B_{m+1} \cdot \frac{f^{(m)}(x)}{m!} \Big|_{\alpha}^{\beta} + R_{m+1}. \quad (50)$$

Здесь $|R_m| \leq (4/(2\pi)^m) \int_{\alpha}^{\beta} |f^{(m)}(x)| dx$. Полагая $f(x) = x^t \times \exp(-2^2/\sqrt{n}x)$, где t —фиксированное неотрицательное целое число, найдем, что $\int_{\alpha}^{\beta} f(x) dx$ отличается от $\int_{-\infty}^{\infty} f(x) dx$ на $O(\exp(-2\pi^{\varepsilon}))$, так что можно взять $\alpha = -\infty, \beta = \infty$. В этом случае формула суммирования Эйлера даст асимптотический ряд для $\sum f(x)$ при $n \rightarrow \infty$, поскольку

$$f^{(m)}(x) = n^{(t-m)/4} g^{(m)}(n^{-1/4}x), \quad g(y) = y^t e^{-2y^2}, \quad (51)$$

и $g(y)$ —хорошая функция, не зависящая от n ; отсюда видно, что $R_m = O(n^{(t+1-m)/4})$. Так как $f^{(m)}(-\infty) = f^{(m)}(\infty) = 0$, мы доказали, что

$$\sum_{\alpha \leq x < \beta} f(x) = \int_{-\infty}^{\infty} f(x) dx + O(n^{-m}) \quad \text{для любого } m \geq 0; \quad (52)$$

на самом деле при этом конкретном выборе $f(x)$ существен только интеграл! Интеграл нетрудно вычислить (см. упр. 26); таким образом, мы можем выполнить умножение и сложение в формуле (49) и получить

$$\begin{aligned} & \sqrt{\frac{\pi}{2}} n^{1/4} \left(1 - \frac{1}{24} n^{-1/4} + O(n^{-1/2}) \right); \\ t_n = & \frac{1}{\sqrt{2}} n^{n/2} e^{-n/2 + \sqrt{n} - 1/4} \left(1 + \frac{7}{24} n^{-1/2} + O(n^{-3/4}) \right). \end{aligned} \quad (53)$$

В действительности слагаемые в $O(n^{-3/4})$ содержат еще экспоненту от 9ε , но из наших выкладок ясно, что величина 9ε должна исчезнуть, если произвести вычисления с большей точностью. В принципе этот метод можно усилить и вместо $O(n^{-3/4})$ получить $O(n^{-k})$ при любом k . Этот асимптотический ряд для t_n впервые нашли (другим способом) Мозер и Уаймэн [*Canadian. J. Math.*, 7 (1955), 159–168]. Другие примеры и усиления метода, примененного для вывода формулы (53), см. в конце п. 5.2.2.

Упражнения

1. [16] Какие табло (P, Q) соответствуют двустрочному массиву

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 4 & 9 & 5 & 7 & 1 & 2 & 8 & 3 \end{pmatrix}$$

в построении из теоремы А? Какой двустрочный массив соответствует паре табло

$$B = \begin{array}{|c|c|c|} \hline 1 & 4 & 7 \\ \hline 2 & 8 & \\ \hline 5 & 9 & \\ \hline \end{array}, \quad C = \begin{array}{|c|c|c|} \hline 1 & 3 & 7 \\ \hline 4 & 5 & \\ \hline 8 & 9 & \\ \hline \end{array} ?$$

2. [M21] Докажите, что (q, p) принадлежит классу t относительно массива (16) тогда и только тогда, когда t равно максимальному числу индексов i_1, \dots, i_t , таких, что

$$p_{i_1} < p_{i_2} < \dots < p_{i_t} = p, \quad q_{i_1} < q_{i_2} < \dots < q_{i_t} = q.$$

- >3. [M24] Покажите, что соответствие в теореме А можно также установить путем построения такой таблицы:

Строка 0	1	3	5	6	8
Строка 1	7	2	9	5	3
Строка 2	∞	7	∞	9	5
Строка 3	∞	∞	∞	7	∞
Строка 4	∞	∞	∞	∞	∞

Здесь в строках 0 и 1 записан данный двустрочный массив. При $k \geq 1$ строка $k + 1$ образуется из строки k следующим образом:

- а) Установить $p \leftarrow \infty$.
- б) Пусть столбец j —самый левый столбец, в котором строка k содержит целое число $< p$, а соответствующее место в строке $k + 1$ не заполнено. Если такого столбца нет и $p = \infty$, то строка $k + 1$ заполнена; если такого столбца нет и $p < \infty$, то возвратиться к (а).
- в) Вставить p в столбец j в строке $k + 1$, потом установить p равным элементу столбца j и строки k и вернуться к (б).

После того как таблица таким образом построена, строка k табло P составляется из тех целых чисел строки k этой таблицы, которые отсутствуют в строке $(k + 1)$; строка k табло Q строится из тех целых чисел строки 0, которые находятся в столбцах, содержащих ∞ в строке $(k + 1)$.

4. [M26] (М. П. Шюценберже). Пусть π —инволюция с k неподвижными точками. Покажите, что табло, соответствующее π в доказательстве следствия из теоремы В, содержит ровно k столбцов нечетной длины.
- >5. [M30] Пусть $a_1 \dots a_{j-1} a_j \dots a_n$ —перестановка различных элементов и $1 < j \leq n$. Перестановка $a_1 \dots a_{j-2} a_j a_{j-1} a_{j+1} \dots a_n$, получающаяся из исходной, если поменять местами a_{j-1} и a_j , называется "допустимой", если либо

- i) $j \geq 3$ и a_{j-2} лежит между a_{j-1} и a_j , либо
- ii) $j < n$ и a_{j+1} лежит между a_{j-1} и a_j .

Например, в перестановке 1546837 можно произвести ровно три допустимые замены: можно поменять местами 1 и 5, поскольку $1 < 4 < 5$; можно поменять местами 8 и 3, поскольку $3 < 6 < 8$ (или $3 < 7 < 8$); однако нельзя менять местами 5 и 4 или 3 и 7.

- а) Докажите, что при допустимой замене не меняется табло P , которое получается из перестановки путем последовательной вставки элементов a_1, a_2, \dots, a_n в первоначально пустое табло.

- b) Обратно, покажите, что любые две перестановки, соответствующие одному и тому же табло P , можно преобразовать одну в другую последовательностью одной или более допустимых замен. [Указание: покажите, что если табло P имеет форму (n_1, n_2, \dots, n_m) , то любую соответствующую ему перестановку при помощи последовательности допустимых замен можно преобразовать в "каноническую перестановку" $P_{m1} \dots P_{mn_m} \dots P_{21} \dots P_{2n_2} P_{11} \dots P_{1n_1}$.]
- >6. [M22] Пусть P —табло, соответствующее перестановке $a_1 a_2 \dots a_n$; с помощью упр. 5 докажите, что табло P^T соответствует перестановке $a_n \dots a_2 a_1$.
7. [M 20] (К. Шенстед). Пусть P —табло, соответствующее перестановке $a_1 a_2 \dots a_n$. Докажите, что число столбцов в P равно длине s максимальной возрастающей подпоследовательности $a_{i_1} < a_{i_2} < \dots < a_{i_c}$, где $i_1 < i_2 < \dots < i_c$; число строк в P равно длине r максимальной убывающей подпоследовательности $a_{j_1} > a_{j_2} > \dots > a_{j_r}$, где $j_1 < j_2 < \dots < j_r$.
8. [M18] (П. Эрдеш, Г. Секереш). Докажите, что в любой перестановке, состоящей из более чем n^2 элементов, имеется монотонная подпоследовательность длины более n ; однако существуют перестановки n^2 элементов, не содержащие монотонных подпоследовательностей длины более n . [Указание: см. предыдущее упражнение.]
9. [M24] Продолжая упр. 8, найдите "простую" формулу точного числа перестановок множества $\{1, 2, \dots, n^2\}$, не содержащих монотонных подпоследовательностей длины более чем n .
10. [M20] Докажите, что массив P является табло по окончании работы алгоритма S, если он был табло до этого.
11. [20] Можно ли восстановить исходный вид табло P по окончании работы алгоритма S, если известны только значения r и s ?
12. [M24] Сколько раз выполняется шаг S3, если алгоритм S многократно применяется для исключения всех элементов из табло P формы (n_1, n_2, \dots, n_m) ? Каково минимальное значение этой величины по всем формам, таким, что $n_1 + n_2 + \dots + n_m = n$?
13. [M28] Докажите теорему С.
14. [M43] Найдите более прямое доказательство части (с) теоремы D.
15. [M20] Сколько перестановок мультимножества $\{l \cdot a, m \cdot b, n \cdot c\}$ обладают тем свойством, что если читать перестановку слева направо, то число прочитанных букв c никогда не превышает числа букв b , которое в свою очередь не превышает числа букв a ? (Например, перестановка $a a b c a b b c a c a$ обладает этим свойством.)
16. [M08] Сколькими способами можно топологически отсортировать частичное упорядочение, представляемое графом (39)?
17. [BM25] Пусть

$$g(x_1, x_2, \dots, x_n; y) = x_1 \Delta(x_1 + y, x_2, \dots, x_n) + x_2 \Delta(x_1, x_2 + y, \dots, x_n) + \dots + x_n \Delta(x_1, x_2, \dots, x_n + y).$$

Докажите, что

$$g(x_1, x_2, \dots, x_n; y) = \left(x_1 + x_2 + \dots + x_n + \binom{n}{2} y \right) \Delta(x_1, x_2, \dots, x_n).$$

[Указание: полином g однородный (все слагаемые имеют одинаковую суммарную степень) и антисимметричный по x (знак g изменится, если поменять местами x_i и x_j).]

18. [BM30] Обобщая упр. 17, вычислите при $m \geq 0$ сумму

$$x_1^m \Delta(x_1 + y, x_2, \dots, x_n) + \Delta x_2^m(x_1, x_2 + y, \dots, x_n) + \dots + x_n^m \Delta(x_1, x_2, \dots, x_n + y).$$

19. [M40] Найдите формулу для определения числа способов, которыми можно заполнить массив, подобный табло, но без первых двух клеток в первой строке; например, массив такой формы:

Picture: 1. p.90

(Элементы в строках и столбцах должны располагаться в возрастающем порядке, как в обычных табло.)

Иначе говоря, сколько из $f(n_1, n_2, \dots, n_m)$ табло формы (n_1, n_2, \dots, n_m) , составленных из элементов $\{1, 2, \dots, n_1 + \dots + n_m\}$, содержат элементы 1 и 2 в первой строке?

- >20. [M24] Докажите, что число способов пометить узлы данного бинарного дерева элементами $\{1, 2, \dots, n\}$ так, чтобы метка каждого узла была меньше метки любого из его потомков, равно частному от деления $n!$ на произведение "длин поддеревьев", т. е. количеств узлов в каждом поддереве. (Ср. с теоремой Н.). Например, число способов пометить узлы дерева

Picture: 1. p.90

равно $10!/10 \cdot 4 \cdot 5 \cdot 1 \cdot 2 \cdot 3 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 9 \cdot 8 \cdot 7 \cdot 6$.

21. [BM31] (Р. М. Тролл). Пусть числа $n_1 > n_2 > \dots > n_m$ описывают форму "сдвинутого табло", в котором строка $i + 1$ начинается на одну позицию, правее, чем строка i ; например, сдвинутое табло формы $(7, 5, 4, 1)$ изображено на диаграмме

Picture: 3. p.90

Докажите, что число способов заполнить сдвинутое табло формы (n_1, n_2, \dots, n_m) числами $1, 2, \dots, n = n_1 + n_2 + \dots + n_m$ так, чтобы числа во всех строках и столбцах располагались в возрастающем порядке, равно частному от деления $n!$ на произведение "длин обобщенных уголков"; на рисунке заштрихован обобщенный уголок длины 11, соответствующий клетке строки 1 и столбца 2. (Уголки в левой части массива, имеющей вид "перевернутой лестницы", имеют форму буквы U, повернутой на 90° , а не буквы L.) Итак, существует

$$17!/12 \cdot 11 \cdot 8 \cdot 7 \cdot 5 \cdot 4 \cdot 1 \cdot 9 \cdot 6 \cdot 5 \cdot 3 \cdot 2 \cdot 5 \cdot 4 \cdot 2 \cdot 1 \cdot 1$$

способов заполнить изображенную выше форму так, чтобы элементы во всех строках и столбцах располагались в возрастающем порядке.

- >22. [BM30] (Д. Андрэ). Чему равно число A_n способов заполнить числами $\{1, 2, \dots, n\}$ массив из n ячеек

Picture: p.91

так, чтобы во всех строках и столбцах они располагались в возрастающем порядке? Найдите производящую функцию $g(z) = \sum A_n z^n / n!$

23. [M39] Сколькими способами можно заполнить массив формы (n_1, n_2, \dots, n_m) элементами множества $\{1, 2, \dots, N\}$, если допускаются одинаковые элементы, причем в строках элементы должны располагаться в неубывающем порядке, а в столбцах — в строго возрастающем? Например, простую форму из m строк $(1, 1, \dots, 1)$ можно заполнить $\binom{N}{m}$ способами; форму из одной строки m можно заполнить $\binom{m+N-1}{m}$ способами; форму $(2, 2)$ можно заполнить $\frac{1}{3} \binom{N+1}{2} \binom{N}{2}$ способами.

24. [M28] Докажите, что

$$\sum_{\substack{q_1 + \dots + q_n = t \\ 0 \leq q_1, \dots, q_n \leq m}} \binom{m}{q_1} \dots \binom{m}{q_n} \Delta(q_1, \dots, q_n)^2 = n! \binom{nm - (n^2 - n)}{t - \frac{1}{2}(n^2 - n)} \binom{m}{n-1} \binom{m}{n-2} \dots \binom{m}{0} \Delta(n-1, \dots, 0)^2.$$

[Указания: докажите, что $\Delta(k_1 + n - 1, \dots, k_n) = \Delta(m - k_n + n - 1, \dots, m - k_1)$; разложите табло формы $n \times (m - n + 1)$ способом, аналогичным (38), и преобразуйте сумму, как при выводе тождества (36).]

25. [M20] Почему (42) является производящей функцией для инволюций?
 26. [BM21] Вычислите $\int_{-\infty}^{\infty} x^t \exp(-2x^2/\sqrt{n}) dx$ при неотрицательном целом t .
 27. [M24] Пусть Q — табло Янга из элементов $\{1, 2, \dots, n\}$, и пусть элемент t находится в строке r_i и столбце c_i . Мы говорим, что i "выше" j , если $r_i < r_j$.

- a) Докажите, что при $1 \leq i < n$ элемент i выше $i + 1$ тогда и только тогда, когда $c_i \geq c_{i+1}$.
 b) Пусть Q такое, что (P, Q) соответствуют перестановке

$$\begin{pmatrix} 1 & 2 & \dots & n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$$

Докажите, что i выше $i + 1$ тогда и только тогда, когда $a_i > a_{i+1}$. (Следовательно, можно найти число отрезков перестановки, зная только Q . Этот результат получен Шюценберже.)

- c) Докажите, что при $1 \leq i < n$ элемент i выше $i + 1$ в Q тогда и только тогда, когда $i + 1$ выше i в Q^S .

28. [M47] Каково асимптотическое поведение средней длины максимальной возрастающей последовательности в случайной перестановке множества $\{1, 2, \dots, n\}$? (Это средняя длина первой строки в соответствии из теоремы А. Обширные таблицы, вычисленные Р. М. Баером и П. Брокком

[*Math. Comp.*, 22 (1968), 385–410], в связи с тем, что они назвали ”естественной сортировкой”, позволяют предположить, что среднее l_n равно примерно $2\sqrt{n}$; Л. Шепп и Б. Логан доказали, что $\liminf_{n \rightarrow \infty} l_n/\sqrt{n} \geq 2$ (в печати).

29. [M50] Исследуйте трехмерные массивы, с тем чтобы понять, какие свойства двумерных таблиц можно обобщить.
30. [M42] (М. П. Шюценберже). Покажите, что операция перехода от P к P^S —частный случай операции, которую можно связать с любым конечным частично упорядоченным множеством, а не только с таблицей. Пометьте элементы частично упорядоченного множества целыми числами $\{1, 2, \dots, n\}$ так, чтобы эта система меток была согласована с частичным упорядочением. Найдите двойственную систему меток, аналогичную (26), путем последовательного удаления меток $1, 2, \dots$, передвигая при этом другие метки способом, подобным алгоритму S, и помещая метки (1), (2), ... на освободившиеся места. Покажите, что эта операция, если ее многократно применять к двойственной системе меток с обратным отношением порядка для чисел, дает исходную систему меток; исследуйте другие свойства этой операции
31. [BM30] Пусть x_n —число способов разместить n взаимно неатакующих ладей на шахматной доске размера $n \times n$ таким образом, что расположение не меняется при отражении доски относительно одной из главных диагоналей и при повороте на 180° . Найдите асимптотическое поведение x_n .

5.2. Внутренняя сортировка

Начнем обсуждение хорошего ”сортирования” с маленького эксперимента: как бы вы решили следующую задачу программирования?

”Ячейки памяти $R + 1, R + 2, R + 3, R + 4$ и $R + 5$ содержат пять чисел. Напишите программу, которая перерасмещает, если нужно, эти числа так, чтобы они расположились в возрастающем порядке.”

(Если вы уже знакомы с какими-либо методами сортировки, постарайтесь, пожалуйста, на минуту забыть о них; вообразите, что вы решаете такую задачу впервые, не имея никаких предварительных знаний о том, как к ней подступиться.)

Прежде чем читать дальше, настоятельно просим вас найти хоть какое-нибудь решение этой задачи.

.....

Время, затраченное на решение приведенной выше задачи, окупится с лихвой, когда вы продолжите чтение этой главы. Возможно, ваше решение окажется одним из следующих:

A. Сортировка вставками. Элементы просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов. (Именно таким способом игроки в бридж упорядочивают свои карты, беря по одной.)

B. Обменная сортировка. Если два элемента расположены не по порядку, то они меняются местами. Этот процесс повторяется до тех пор, пока элементы не будут упорядочены.

C. Сортировка посредством выбора. Сначала выделяется наименьший (или, быть может, наибольший) элемент и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся и т. д.

D. Сортировка подсчетом. Каждый элемент сравнивается со всеми остальными; окончательное положение элемента определяется подсчетом числа меньших ключей.

E. Специальная сортировка, которая хороша для пяти элементов, указанных в задаче, но не поддается простому обобщению на случай большего числа элементов.

F. Ленивое решение. Вы не откликнулись на наше предложение и отказались решать задачу. Жаль, но теперь, когда вы прочли так много, ваш шанс упущен.

G. Новая суперсортировка, которая представляет собой определенное усовершенствование известных методов. (Пожалуйста, немедленно сообщите об этом автору.)

Если бы эта задача была сформулирована, скажем, для 1000 элементов, а не для 5, то вы могли бы открыть и некоторые более тонкие методы, о которых будет упомянуто ниже. В любом случае, приступая к новой задаче, разумно найти какую-нибудь очевидную процедуру, а затем попытаться улучшить ее. Случаи A, B и C приводят к важным классам методов сортировки, которые представляют собой усовершенствования сформулированных выше простых идей.

Было изобретено множество различных алгоритмов сортировки, и в этой книге рассматривается около 25 из них. Это пугающее количество методов на самом деле лишь малая толика всех алгоритмов, придуманных до сих пор; многие, уже устаревшие методы мы вовсе не будем рассматривать или упомянем лишь вскользь. Почему же так много методов сортировки? Для программирования это частный случай вопроса: "Почему существует так много методов x ?", где x пробегает множество всех задач. Ответ состоит в том, что каждый метод имеет свои преимущества и недостатки, поэтому он оказывается эффективнее других при некоторых конфигурациях данных и аппаратуры. К сожалению, неизвестен "наилучший" способ сортировки; существует *много* наилучших методов в зависимости от того, что сортируется, на какой машине, для какой цели. Говоря словами Редьярда Киплинга, "существует 9 и еще 60 способов сложить песню племени, и каждый из них в отдельности хорош".

Полезно изучить характеристики каждого метода сортировки, чтобы можно было производить разумный выбор для конкретных приложений. К счастью, задача изучения этих алгоритмов не столь уж громоздка, поскольку между ними существуют интересные взаимосвязи.

В начале этой главы мы ввели основную терминологию и обозначения, которые и будем использовать при изучении сортировки. Записи

$$R_1, R_2, \dots, R_N \quad (1)$$

должны быть отсортированы в неубывающем порядке по своим ключам K_1, K_2, \dots, K_N , по существу, путем нахождения перестановки $p(1) p(2) \dots p(N)$, такой, что

$$K_{p(1)} \leq K_{p(2)} \leq \dots \leq K_{p(N)}. \quad (2)$$

В этом параграфе рассматривается *внутренняя сортировка*, когда число записей, подлежащих сортировке, достаточно мало, так что весь процесс можно провести в оперативной памяти ЭВМ.

В одних случаях может понадобиться физически переразместить записи в памяти так, чтобы их ключи были упорядочены; в других можно обойтись вспомогательной таблицей некоторого

Picture: Рис. 6. Сортировка таблицы адресов.

вида, которая определяет перестановку. Если записи и/или ключи занимают несколько слов памяти, то часто лучше составить новую таблицу адресов (ссылок), которые указывают на записи, и работать с этими адресами, не перемещая громоздкие записи. Такой метод называется *сортировкой таблицы адресов*

Picture: Рис. 7. Сортировка списка.

(рис. 6). Если ключи короткие, а сопутствующая информация в записях велика, то для повышения скорости ключи можно вынести в таблицу адресов; это называется *сортировкой ключей*. Другие схемы сортировки используют вспомогательное поле связи, которое включается в каждую запись. Связи обрабатываются таким образом, что в результате все записи оказываются связанными в линейный список, в котором каждая связь указывает на следующую по порядку запись. Это называется *сортировкой списка* (рис. 7).

После сортировки таблицы адресов или сортировки списка можно по желанию расположить записи в неубывающем порядке. Для этого имеется несколько способов, требующих дополнительной памяти для хранения всего одной записи (см. упр. с 10 по 12); или же можно просто переместить записи в новую область памяти, если она может вместить все эти записи. Последний способ обычно вдвое быстрее первого, но требует почти в два раза больше памяти. Во многих приложениях вовсе не обязательно перемещать записи, так как поля связи, как правило, вполне приемлемы для операций с последовательной адресацией.

Все методы сортировки, которые мы исследуем "досконально", будут проиллюстрированы четырьмя способами: посредством

- a) словесного описания алгоритма,
- b) блок-схемы,
- c) программы для машины MIX,
- d) примера применения этого метода сортировки к заданному множеству чисел.

[В тех примерах, где это уместно, будет обрабатываться множество из 16 чисел, которые автор, пользуясь набором десятичных игральных костей, выбрал случайным образом 19 марта 1963 г.; ср. с упр. 3.1–1 (с).]

Из соображений удобства программы для машины MIX будут, как правило, написаны в предположении, что ключ числовой и что он помещается в одном слове; иногда мы даже будем ограничивать значения ключей так, чтобы они занимали лишь часть слова. Отношением порядка < будет обычное арифметическое отношение порядка, а записи будут состоять из одного ключа, без сопутствующей информации. В этих предположениях программы получаются короче, проще для понимания, и не представляет труда распространить их на общий случай (например, применяя сортировку таблиц адресов). Вместе с MIX-программами приводится анализ времени выполнения соответствующего алгоритма сортировки.

Сортировка подсчетом. Чтобы проиллюстрировать способ, которым мы будем изучать методы внутренней сортировки, рассмотрим идею "подсчета", упомянутую в начале этого параграфа. Этот простой метод основан на том, что j -й ключ в окончательно упорядоченной последовательности превышает ровно $(j - 1)$ из остальных ключей. Иначе говоря, если известно, что некоторый ключ превышает ровно 27 других, то после сортировки соответствующая запись должна занять 28-е место. Таким образом, идея состоит в том, чтобы сравнить попарно все ключи и подсчитать, сколько из них меньше каждого отдельного ключа.

Очевидный способ выполнить сравнения —

$$((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \leq j \leq N) \text{ при } 1 \leq i \leq N,$$

но легко видеть, что более половины этих действий излишни, поскольку не нужно сравнивать ключ сам с собой и после сравнения K_a с K_b уже не надо сравнивать K_b с K_a . Поэтому достаточно

$$((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \leq j \leq i) \text{ при } 1 < i \leq N.$$

Итак, приходим к следующему алгоритму.

Алгоритм С. (Сравнение и подсчет.) Этот алгоритм сортирует записи R_1, \dots, R_N по ключам K_1, \dots, K_N , используя для подсчета числа ключей, меньших данного, вспомогательную таблицу COUNT[1], ..., COUNT[N]. После завершения алгоритма величина COUNT[j] + 1 определяет окончательное положение записи R_j .

- C1 [Сбросить счетчики.] Установить COUNT[1], ..., COUNT[N] равными нулю.
- C2 [Цикл по i .] Выполнить шаг C3 при $i = N, N - 1, \dots, 2$; затем завершить работу алгоритма.
- C3 [Цикл по j .] Выполнить шаг C4 при $j = i - 1, i - 2, \dots, 1$.
- C4 [Сравнить K_i, K_j .] Если $K_i < K_j$, то увеличить COUNT[j] на 1; в противном случае увеличить COUNT[i] на 1. ■

Заметим, что в этом алгоритме записи не перемещаются. Он аналогичен сортировке таблицы адресов, поскольку таблица COUNT определяет конечное расположение записей; но эти методы несколько различаются, потому что COUNT[j] указывает то место, куда нужно переслать запись R_j , а не ту запись, которую надо переслать на место R_j . (Таким образом, таблица COUNT определяет перестановку, обратную $p(1) \dots p(n)$; см. п. 5.1.1.)

В рассуждении, предшествующем этому алгоритму, мы не учитывали, что ключи могут быть равными. Это, вообще говоря, серьезное упущение, потому что если бы равным ключам соответствовали равные счетчики, то заключительное перемещение записей было бы довольно сложным. К счастью, как показано в упр. 2, алгоритм С дает верный результат независимо от числа равных ключей.

Picture: Рис. 8. Алгоритм С: сравнение и подсчет.

Таблица 1

Сортировка подсчетом (алгоритм С)																
ключи:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
COUNT(нач.):	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT($i = N$):	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	12
COUNT($i = N - 1$):	0	0	0	0	2	0	2	0	0	0	0	0	0	0	13	12
COUNT($i = N - 2$):	0	0	0	0	3	0	3	0	0	0	0	0	0	11	13	12
COUNT($i = N - 3$):	0	0	0	0	4	0	4	0	1	0	0	0	9	11	13	12
COUNT($i = N - 4$):	0	0	1	0	5	0	5	0	2	0	0	7	9	11	13	12
COUNT($i = N - 5$):	1	0	2	0	6	1	6	1	3	1	2	7	9	11	13	12
...																
COUNT($i = 2$):	6	1	8	0	15	3	14	4	10	5	2	7	9	11	13	12

Программа С. (Сравнение и подсчет.) В следующей реализации алгоритма С для машины MIX предполагается, что запись R_j находится в ячейке INPUT + j , а COUNT[j]—в ячейке COUNT + j , где $1 \leq j \leq N$; rI1 $\equiv i$; rI2 $\equiv j$; rA $\equiv K_i \equiv R_i$; rX \equiv COUNT[i].

START	ENT1	N	1	C1 . Сбросить счетчики.
	STZ	COUNT, 1	N	COUNT[i] \leftarrow 0.
	DEC1	1	N	
	J1P	*-2	N	$N \geq i > 0$.
	ENT1	N	1	C2. Цикл по i .
	JMP	1F	1	
2H	LDA	INPUT, 1	$N - 1$	
	LDX	COUNT, 1	$N - 1$	
3H	CMPA	INPUT, 2	A	C4. Сравнить K_i, K_j .
	JGE	4F	A	Переход, если $K_i \geq K_j$.
	LD3	COUNT, 2	B	COUNT[j]
	INC3	1	B	+1
	ST3	COUNT, 2	B	\rightarrow COUNT[j]
	JMP	5F	B	
4H	INCX	1	$A - B$	COUNT[i] + 1 \rightarrow COUNT[i].
5H	DEC2	1	A	C3. Цикл по j .
	J2P	3B	A	
	STX	COUNT, 1	$N - 1$	
	DEC1	1	$N - 1$	
1H	ENT2	-1, 1	N	$N \geq i > j > 0$.
	J2P	2B	N	

■

Время работы этой программы равно $13N + 6A + 5B - 4$ единиц, где N —число записей, A —число способов выбрать 2 предмета из N , т. е. $\binom{N}{2} = (N^2 - N)/2$, а B —число пар индексов, таких, что $j < i$, и $K_j > K_i$. Таким образом, B —число инверсий перестановки K_1, \dots, K_N ; эта величина подробно анализировалась в п. 5.1.1, и было найдено [формулы (5.1.1–12,13)], что для неравных ключей, расположенных в случайном порядке,

$$B = \left(\min 0, \text{ave} \frac{(N^2 - N)}{4}, \max \frac{(N^2 - N)}{2}, \text{dev} \frac{\sqrt{N(N-1)(N+2.5)}}{6} \right).$$

Следовательно, выполнение программы С занимает от $3N^2 + 10N - 4$ до $5.5N^2 + 7.5N - 4$ единиц времени, а среднее время работы находится посередине между этими двумя границами. Например, для данных табл. 1 имеем $N = 16$, $A = 120$, $B = 41$, значит, программа С отсортирует их за время 1129*u*. Модификацию программы С, обладающую несколько иными временными характеристиками, см. в упр. 5.

Множитель N^2 , которым определяется время работы, свидетельствует о том, что алгоритм С не дает эффективного способа сортировки, когда N велико; при удвоении числа записей время увеличивается в четыре раза. Поскольку этот метод требует сравнения всех пар ключей (K_i, K_j), то нет очевидного способа исключить зависимость от N^2 , тем не менее мы увидим дальше в этой главе, что, пользуясь "разделением и обменом", можно снизить порядок среднего времени работы до $N \log N$. Алгоритм С интересен для нас не эффективностью, а главным образом своей простотой; его описание служит примером того стиля, в котором будут описаны более сложные (и более эффективные) методы.

Существует другая разновидность сортировки посредством подсчета, которая действительно весьма важна с точки зрения эффективности; она применима в основном в том случае, когда имеется много равных ключей и все они попадают в диапазон $u \leq K_j \leq v$, где значение $(v - u)$ невелико. Эти предположения представляются весьма ограничительными, но на самом деле мы увидим немало приложений этой идеи; например, если применить этот метод лишь к старшим цифрам ключей, а не ко всем ключам целиком, то файл окажется частично отсортированным, и будет уже сравнительно просто довести дело до конца.

Чтобы понять принцип, предположим, что все ключи лежат между 1 и 100. При первом просмотре файла можно подсчитать, сколько имеется ключей, равных 1, 2, ..., 100, а при втором просмотре можно уже располагать записи в соответствующих местах области вывода. В следующем алгоритме

все это описано более подробно.

Picture: Рис. 9. Алгоритм D: распределяющий подсчет.

Алгоритм D. (Распределяющий подсчет.) Этот алгоритм в предположении, что все ключи—целые числа в диапазоне $u \leq K_j \leq v$ при $1 \leq j \leq N$, сортирует записи R_1, \dots, R_N , используя вспомогательную таблицу $\text{COUNT}[u], \dots, \text{COUNT}[v]$. В конце работы алгоритма все записи в требуемом порядке переносятся в область вывода S_1, \dots, S_N .

D1 [Сбросить счетчики.] Установить $\text{COUNT}[u], \dots, \text{COUNT}[v]$ равными нулю.

D2 [Цикл по j .] Выполнить шаг **D3** при $1 \leq j \leq N$, затем перейти к шагу **D4**.

D3 [Увеличить $\text{COUNT}[K_j]$.] Увеличить значение $\text{COUNT}[K_j]$ на 1.

D4 [Суммирование.] (К этому моменту значение $\text{COUNT}[i]$ есть число ключей, равных i .) Установить $\text{COUNT}[i] \leftarrow \text{COUNT}[i] + \text{COUNT}[i - 1]$ при $i = u + 1, u + 2, \dots, v$.

D5 [Цикл по j .] (К этому моменту значение $\text{COUNT}[i]$ есть число ключей, меньших или равных i , в частности $\text{COUNT}[v] = N$.) Выполнить шаг **D6** при $j = N, N - 1, \dots, 1$ и завершить работу алгоритма.

D6 [Вывод R_j .] Установить $i \leftarrow \text{COUNT}[K_j]$, $S_i \leftarrow R_j$, и $\text{COUNT}[K_j] \leftarrow i - 1$. ■

Пример применения этого алгоритма приведен в упр. 6; программу для машины MIX можно найти в упр. 9. При сформулированных выше условиях это очень быстрая процедура сортировки.

Сортировка посредством сравнения и подсчета, как в алгоритме C, впервые упоминается в работе Э. Х. Фрэнда [JACM, 3 (1965), 152], хотя он и не заявил о ней как о своем собственном изобретении. Распределяющий подсчет, как в алгоритме D, впервые разработан Х. Сьювордом в 1954 г. и использовался при поразрядной сортировке, которую мы обсудим позже (см. п. 5.2.5); этот метод также был опубликован под названием "Mathsort" в работе W. Feurzig, CACM, 3 (1960), 601.

Упражнения

- [15] Будет ли работать алгоритм C, если в шаге C2 значение C будет изменяться от 2 до N, а не от N до 2? Что произойдет, если в шаге C3 значение j будет изменяться от 1 до i - 1?
- [21] Покажите, что алгоритм C работает правильно и при наличии одинаковых ключей. Если $K_j = K_i$ и $j < i$, то где окажется в конце концов R_j —до или после R_i ?
- >3. [21] Будет ли алгоритм C работать правильно, если в шаге C4 заменить проверку " $K_i < K_j$ " на " $K_i \leq K_j$ "?
- [16] Напишите MIX-программу, которая завершит сортировку, начатую программой C; ваша программа должна поместить ключи в ячейки $\text{OUTPUT} + 1, \dots, \text{OUTPUT} + N$ в возрастающем порядке. Сколько времени затратит на это ваша программа?
- [22] Улучшится ли программа C в результате следующих изменений?

Ввести новую строку 8a: INCX 0, 2
Изменить строку 10: JGE 5F
Изменить строку 14: DECX 1
Исключить строку 15.

- [18] Промоделируйте вручную работу алгоритма D, показывая промежуточные результаты, получающиеся при сортировке 16 записей

5T, 0C, 5U, 00, 9, 1N, 8S, 2R, 6A, 4A, 1G, 5L, 6T, 61, 70, 7N.

Здесь цифры—это ключи, а буквы—сопутствующая информация в записях.

- [13] Является ли алгоритм D алгоритмом "устойчивой" сортировки?
- [15] Будет ли алгоритм D работать правильно, если в шаге D5 значение j будет изменяться от 1 до N, а не от N до 1?
- [23] Напишите MIX-программу для алгоритма D, аналогичную программе C и упр. 4. Выразите время работы вашей программы в виде функции от N и $(v - u)$.
- [25] Предложите эффективный алгоритм, который бы заменял N величин (R_1, \dots, R_N) соответственно на $(R_{p(1)}, \dots, R_{p(N)})$, если даны значения R_1, \dots, R_N и перестановка $p(1) \dots p(N)$ множества $\{1, \dots, N\}$. Постарайтесь не использовать излишнего пространства памяти. (Эта задача возникает, когда требуется переразместить в памяти записи после сортировки таблицы адресов, не расходуя память на $2N$ записей.)
- [M27] Напишите MIX-программу для алгоритма из упр. 10 и проанализируйте ее эффективность.

- >12. [25] Предложите эффективный алгоритм перераспределения в памяти записей R_1, \dots, R_N в отсортированном порядке после завершения сортировки списка (рис. 7). Постарайтесь не использовать излишнего пространства памяти.
- >13. [27] Алгоритму D требуется пространство для $2N$ записей R_1, \dots, R_N и S_1, \dots, S_N . Покажите, что можно обойтись пространством для N записей R_1, \dots, R_N , если вместо шагов D5 и D6 подставить новую упорядочивающую процедуру. (Таким образом, задача состоит в том, чтобы разработать алгоритм, который бы перераспределял записи R_1, \dots, R_N , основываясь на значениях

$$\text{COUNT}[u], \dots, \text{COUNT}[v]$$

после выполнения шага D4, не используя дополнительной памяти; это, по существу, обобщение задачи, рассмотренной в упр. 10.)

5.2.1. Сортировка вставками

Одно из важных семейств-методов сортировки основано на упомянутом в начале § 5.2 способе, которым пользуются игроки в бридж; предполагается, что перед рассмотрением записи R_j предыдущие записи R_1, \dots, R_{j-1} уже упорядочены, и R_j вставляется в соответствующее место. На основе этой схемы возможны несколько любопытных вариаций.

Простые вставки. Простейшая сортировка вставками относится к наиболее очевидным. Пусть $1 < j \leq N$ и записи R_1, \dots, R_{j-1} уже размещены так, что

$$K_1 \leq K_2 \leq \dots \leq K_{j-1}.$$

(Напомним, что в этой главе через K_j обозначается ключ записи R_j .) Будем сравнивать по очереди K_j с K_{j-1}, K_{j-2}, \dots до тех пор, пока не обнаружим, что запись R_j следует вставить между R_i и R_{i+1} ; тогда подвинем записи R_{i+1}, \dots, R_{j-1} на одно место вверх и поместим новую запись в позицию $i + 1$. Удобно совмещать операции сравнения и перемещения, перемежая их друг с другом, как показано в следующем алгоритме; поскольку запись R_j как бы "проникает на положенный ей уровень", этот способ часто называют *просеиванием*, или *погружением*.

Алгоритм S. (Сортировка простыми вставками.) Записи R_1, \dots, R_N перераспределяются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$.

S1 [Цикл. по j .] Выполнить шаги от S2 до S5 при $j = 2, 3, \dots, N$ и после этого завершить алгоритм.

Picture: Рис. 10. Алгоритм S: простые вставки.

- S2 [Установить i, K, R .] Установить $i \leftarrow j - 1, K \leftarrow K_j, R \leftarrow R_j$. (В последующих шагах мы попытаемся вставить запись R в нужное место, сравнивая K с K_i при убывающих значениях i .)
- S3 [Сравнить K с K_i .] Если $K \geq K_i$, то перейти к шагу S5. (Мы нашли искомое место для записи R .)
- S4 [Переместить R_i , уменьшить i .] Установить $R_{i+1} \leftarrow R_i, i \leftarrow i - 1$. Если $i > 0$, то вернуться к шагу S3. (Если $i = 0$, то K — наименьший из рассмотренных до сих пор ключей, а, значит, запись R должна занять первую позицию.)
- S5 [R на место R_{i+1} .] Установить $R_{i+1} \leftarrow R$. ■

В табл. 1 показано применение алгоритма S к шестнадцати числам, взятым нами для примеров. Этот метод чрезвычайно просто реализуется на вычислительной машине; фактически следующая MIX-программа — самая короткая из всех приемлемых программ сортировки в этой книге.

Программа S. (Сортировка простыми вставками). Записи, которые надо отсортировать, находятся в ячейках $\text{INPUT} + 1, \dots, \text{INPUT} + N$; они сортируются в той же области по ключу, занимающему одно слово целиком. Здесь $rI1 \equiv j - N; rI2 \equiv i, rA \equiv R \equiv K$; предполагается, что $N \geq 2$.

START	ENT1	2-N	1	S1. Цикл по j . $j \leftarrow 2$.
2H	LDA	INPUT+N, 1	$N - 1$	S2. Установить i, K, R
	ENT2	N-1, 1	$N - 1$	$i \leftarrow j - 1$.
3H	CMPA	INPUT, 2	$B + N - 1 - A$	S3. Сравнить K, K_i
	JGE	5F	$B + N - 1 - A$	K S5, если $K \geq K_i$.
4H	LDX	INPUT, 2	B	S4. Переместить R_i , уменьшить i
	STX	INPUT+1, 2	B	$R_{i+1} \leftarrow R_i$.
	DEC2	1	B	$i \leftarrow i - 1$.
	J2P	3B	B	K S3, если $i > 0$.
5H	STA	INPUT+1, 2	$N - 1$	S5. R на место $R_i + 1$.
	INC1	1	$N - 1$	
	J1NP	2B	$N - 1$	$2 \leq j \leq N$.



Время работы этой программы равно $9B + 10N - 3A - 9$ единиц, где N —число сортируемых записей, A —число случаев, когда в шаге S4 значение i убывает до 0, а B —число перемещений. Ясно, что A равно числу случаев, когда $K_j < (K_1, \dots, K_{j-1})$ при $1 < j \leq N$, т. е. это число левосторонних минимумов—величина, которая была тщательно исследована в п. 1.2.10. Немного подумав, нетрудно понять, что B —тоже известная величина: число перемещений при фиксированном j равно числу инверсий ключа K_j , так что B равно числу инверсий перестановки K_1, K_2, \dots, K_N . Следовательно, согласно формулам (1.2.10–16) и (5.1.1–12, 13),

$$A = (\min 0, \text{ave } H_N - 1, \max N - 1, \text{dev } \sqrt{H_n - H_n^{(2)}});$$

$$B = (\min 0, \text{ave}(N^2 - N)/4, \max(N^2 - N)/2, \text{dev } \sqrt{N(N - 1)(N + 2.5)/6}),$$

а среднее время работы программы S в предположении, что ключи различны и расположены в случайном порядке, равно $(2.25N^2 + 7.75N - 3H_N - 6)u$. В упр. 33 показано, как можно чуть-чуть уменьшить это время.

Приведенные в качестве примера в табл. 1 данные содержат 16 элементов; имеются два левосторонних минимума, 087 и 061, и, как мы видели в предыдущем пункте, 41 инверсия. Следовательно, $N = 16$, $A = 2$, $B = 41$, а общее время сортировки равно $514u$.

Таблица 1

Пример применения простых вставок

^503	:087														
087	503^	:512													
^087	503	512	:061												
061	087	503	512^	:908											
061	087^	503	512	908	:170										
061	087	170	503	512^	908	:897									
.															
061	087	154	170	275	426	503	509	512	612	653	677^	765	897	908	:703
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Бинарные вставки и двухпутевые вставки. Когда при сортировке простыми вставками обрабатывается j -я запись, ее ключ сравнивается в среднем примерно с $j/2$ ранее отсортированными ключами; поэтому общее число сравнений равно приблизительно $(1 + 2 + \dots + N)/2 \approx N^2/4$, а это очень много, даже если N умеренно велико. В п. 6.2.1 мы изучим методы ”бинарного поиска”, которые указывают, куда вставлять j -й элемент после приблизительно $\log_2 j$ соответствующим образом выбранных сравнений. Например, если вставляется 64-я запись, можно сначала сравнить ключ K_{64} с K_{32} ; затем, если он меньше, сравниваем его с K_{16} , если больше—с K_{48} и т. д., так что место для R_{64} будет найдено после всего лишь шести сравнений. Общее число сравнений для N вставляемых элементов равно приблизительно $N \log_2 N$, что существенно лучше, чем $N^2/4$; в п. 6.2.1 показано, что соответствующая программа не обязательно намного сложнее, чем программа для простых вставок. Этот метод называется *бинарными вставками*. Он упоминался Джоном Мочли еще в 1946 г., в первой публикации по машинной сортировке.

Неприятность состоит в том, что бинарные вставки решают задачу только наполовину: после того, как мы нашли, куда вставлять запись R_j , все равно нужно подвинуть примерно $j/2$ ранее отсортированных записей, чтобы освободить место для R_j так что общее время работы остается, по существу, пропорциональным N^2 . Некоторые вычислительные машины (например, IBM 705) имеют встроенные инструкции ”переброски”, выполняющие операции перемещения с большой скоростью, но с ростом N зависимость от N^2 в конце концов начинает преобладать. Например, анализ, проведенный Х. Нэгдером [CASM, 3 (1960), 618–620], показывает, что не следует рекомендовать бинарные вставки при сортировке более $N = 128$ записей на машине IBM 705, если каждая запись состоит из 80 литер; аналогичный анализ применим и к другим машинам.

Разумеется, изобретательный программист может придумать какие-нибудь способы, позволяющие сократить число необходимых перемещений; первый такой прием, предложенный в начале 50-х годов, проиллюстрирован в табл. 2. Здесь первый элемент помещается в середину области вывода, и место для последующих элементов освобождается при помощи сдвигов влево или вправо, туда, куда удобнее. Таким образом удастся сэкономить примерно половину времени работы по сравнению

с простыми вставками за счет некоторого усложнения программы. Можно применять этот метод, используя не больше памяти, чем требуется для N записей (см. упр. 6), но мы не станем дольше задерживаться на таких "двухпутевых" вставках, так как были разработаны гораздо более интересные методы.

Таблица 2

Двухпутевые вставки

			\wedge 503				
		087	503 \wedge				
	\wedge 087	503	512				
	061	087	503	512 \wedge			
	061	087 \wedge	503	512	908		
	061	087	170	503	512	908	
	061	087	170 \wedge	503	512	897	908
061	087	170	276	503	512	897	908

Метод Шелла. Для алгоритма сортировки, который каждый раз перемещает запись только на одну позицию, среднее время работы будет в лучшем случае пропорционально N^2 , потому что в процессе сортировки каждая запись должна пройти в среднем через $N/3$ позиций (см. упр. 7). Поэтому, если мы хотим получить метод, существенно превосходящий по скорости простые вставки, то необходим некоторый механизм, с помощью которого записи могли бы перемещаться большими скачками, а не короткими шажками.

Такой метод предложен в 1959 г. Дональдом Л. Шеллом [CASM, 2 (July, 1959), 30–32]; мы будем называть его *сортировкой с убывающим шагом*. В табл. 3 проиллюстрирована общая идея, лежащая в основе этого метода. Сначала делим 16 записей на 8 групп по две записи в каждой группе: (R_1, R_9) , (R_2, R_{10}) , ..., (R_8, R_{16}) . В результате сортировки каждой группы записей по отдельности приходим ко второй строке табл. 3,

Picture: Таблица 3

это называется "первым просмотром". Заметим, что элементы 154 и 512 поменялись местами, а 908 и 897 переместились вправо. Разделим теперь записи на четыре группы по четыре в каждой: (R_1, R_5, R_9, R_{13}) , ..., $(R_4, R_8, R_{12}, R_{16})$ —и опять отсортируем каждую группу в отдельности; этот второй просмотр приводит к третьей строке таблицы. При третьем просмотре сортируются две группы по восемь записей; процесс завершается четвертым просмотром, во время которого сортируются все 16 записей. В каждом из этих промежуточных процессов сортировки участвуют либо сравнительно короткие файлы, либо уже сравнительно хорошо упорядоченные файлы, поэтому на каждом этапе можно пользоваться простыми вставками; записи довольно быстро достигают своего конечного положения.

Последовательность шагов 8, 4, 2, 1 не следует считать неизбежной, можно пользоваться *любой* последовательностью h_t, h_{t-1}, \dots, h_1 , в которой последний шаг h_1 равен 1. Например, в табл. 4 будет показана сортировка тех же данных с шагами 7, 5, 3, 1. Одни последовательности оказываются гораздо лучше других; мы обсудим выбор последовательностей шагов позже.

Алгоритм D. (Сортировка с убывающим шагом.) Записи R_1, \dots, R_N переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Для управления процессом сортировки используется вспомогательная последовательность шагов h_t, h_{t-1}, \dots, h_1 , где $h_1 = l$; правильно выбрав эти приращения, можно значительно сократить время сортировки. При $t = 1$ этот алгоритм сводится к алгоритму S.

- D1 [Цикл по s .] Выполнить шаг D2 при $s = t, t - 1, \dots, 1$, после чего завершить работу алгоритма.
- D2 [Цикл по j .] Установить $h \leftarrow h_s$ и выполнить шаги D3, ..., D6 при $h < j \leq N$. (Для сортировки элементов, отстоящих друг от друга на h позиций, воспользуемся простыми вставками и в результате получим $K_i \leq K_{i+h}$ при $1 \leq i \leq N - h$. Шаги D3, ..., D6, по существу, такие же, как соответственно S2, ..., S5 в алгоритме S.)
- D3 [Установить i, K, R .] Установить $i \leftarrow j - h, K \leftarrow K_j, R \leftarrow R_j$.
- D4 [Сравнить K, K_i .] Если $K \geq K_i$, то перейти к шагу D6.
- D5 [Переместить R_i , уменьшить i .] Установить $R_{i+h} \leftarrow R_i$, затем $i \leftarrow i - h$. Если $i > 0$, то возвратиться к шагу D4.
- D6 [R на место R_{i+h} .] Установить $R_{i+h} \leftarrow R$. ■

Соответствующая MIX-программа не намного длиннее, чем наша программа для простых вставок. Строки 08–19 этой программы перенесены из программы S в более общий контекст алгоритма D.

Программа D. (Сортировка с убывающим шагом.) Предполагается, что шаги сортировки хранятся во вспомогательной таблице и h_s находится в ячейке $H + s$; все шаги сортировки меньше N . Содержимое регистров: $rI1 \equiv j - N$; $rI2 \equiv i$; $rA \equiv R \equiv K$; $rI3 \equiv s$; $rI4 \equiv h$. Заметим, что эта программа сама себя изменяет. Это сделано для того, чтобы добиться более эффективного выполнения внутреннего цикла.

START	ENT3	T	1	D1. Цикл по s . $s \leftarrow t$.
1H	LD4	H, 3	T	D2. Цикл по j . $h \leftarrow h_s$.
	ENT1	INPUT, 4	T	Изменить адреса в трех
	ST1	6F(0:2)	T	инструкциях в
	ST1	7F(0:2)	T	основном цикле.
	ENN1	-N, 4	T	$rI1 \leftarrow N - h$.
	ST1	4F(0:2)	T	
	ENT1	1-N, 4	T	$j \leftarrow h + 1$.
2H	LDA	INPUT+N, 1	$NT - S$	D3. Установить i, K, R .
4H	ENT2	N-H, 1	$NT - S$	$i \leftarrow j - h$. (Изменяемая инструкция)
5H	CMPA	INPUT, 2	$B + NT - S - A$	D4. Сравнить K, K_i .
	JOE	7F	$B + NT - S - A$	К шагу D6, если $K \geq K_i$.
	LDX	INPUT, 2	B	D5. Переместить R_i , уменьшить i .
6H	STX	INPUT+H, 2	B	$R_{i+h} \leftarrow R_i$. (Изменяемая инструкция)
	DEC2	0, 4	B	$i \leftarrow i - h$.
	J2P	5B	B	К шагу D4, если $i > 0$.
7H	STA	INPUT+H, 2	$NT - S$	D6. R на место R_{i+h} . (Изменяемая инструкция)
8H	INC1	1	$NT - S$	$j \leftarrow j + 1$.
	J1NP	2B	$NT - S$	К D3, если $j \leq N$.
	DEC3	1	T	
	J3P	1B	T	$t \geq s \geq 1$.

***Анализ метода Шелла.** Чтобы выбрать хорошую последовательность шагов сортировки для алгоритма D, нужно проанализировать время работы как функцию от этих шагов. Это приводит к очень красивым, но еще не до конца решенным математическим задачам; никому до сих пор не удалось найти наилучшую возможную последовательность шагов для больших N . Тем не менее известно довольно много интересных фактов о поведении сортировки методом Шелла с убывающим шагом, и мы здесь их кратко изложим; подробности можно найти в приведенных ниже упражнениях. [Читателям, не имеющим склонности к математике, лучше пропустить следующие несколько страниц, до формул (8).]

Счетчики частот выполнения в программе D показывают, что на время выполнения влияют пять факторов: размер файла N , число просмотров (т.е. число шагов) $T = t$, сумма шагов

$$S = h_1 + \dots + h_t,$$

число сравнений $B + NT - S - A$ и число перемещений B . Как и при анализе программы S, здесь A равно числу левосторонних минимумов, встречающихся при промежуточных операциях сортировки, а B равно числу инверсий в подфайлах. Основным фактором, от которого зависит время работы, является величина B , поэтому на нее мы и обратим главным образом свое внимание. При анализе будет предполагаться, что ключи различны и первоначально расположены в случайном порядке.

Назовем операцию шага D2 " h -сортировкой". Тогда сортировка методом Шелла состоит из h_t -сортировки, за которой следует h_{t-1} -сортировка, ..., за которой следует h_1 -сортировка. Файл, в котором $K_i \leq K_{i+h}$ при $1 \leq i \leq N - h$, будем называть h -упорядоченным.

Рассмотрим сначала простейшее обобщение простых вставок, когда имеются всего два шага $h_2 = 2$ и $h_1 = 1$. Во время второго просмотра имеем 2-упорядоченную последовательность ключей $K_1 K_2 \dots K_N$. Легко видеть, что число перестановок $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$, таких, что $a_i \leq a_{i+2}$ при $l \leq i \leq n - 2$, равно

$$\binom{n}{\lfloor n/2 \rfloor},$$

так как существует всего одна 2-упорядоченная перестановка для каждого выбора $\lfloor n/2 \rfloor$ элементов, расположенных в четных позициях $a_2 a_4, \dots$, тогда остальные $\lceil n/2 \rceil$ элементов попадают в позиции с нечетными номерами. После 2-сортировки случайного файла с одинаковой вероятностью может получиться любая 2-упорядоченная перестановка. Каково среднее число инверсий во всех таких перестановках?

Picture: Рис. 11. Соответствие между 2-упорядочением и путями на решетке. Курсивом набраны веса, соответствующие числу инверсий в 2-упорядоченной перестановке.

Пусть A_n — суммарное число инверсий во всех 2-упорядоченных перестановках множества $\{1, 2, \dots, n\}$. Ясно, что $A_1 = 0, A_2 = 1, A_3 = 2$, а из рассмотрения шести случаев

$$1\ 3\ 2\ 4 \quad 1\ 2\ 3\ 4 \quad 1\ 2\ 4\ 3 \quad 2\ 1\ 3\ 4 \quad 2\ 1\ 4\ 3 \quad 3\ 1\ 4\ 2$$

находим, что $A_4 = 1 + 0 + 1 + 1 + 2 + 3 = 8$. Чтобы исследовать A_n в общем случае, рассмотрим решетчатую диаграмму на рис. 11 для $n = 15$. В такой диаграмме 2-упорядоченную перестановку можно представить в виде пути из верхней левой угловой точки $(0, 0)$ в нижнюю правую угловую точку $(\lfloor n/2 \rfloor, \lceil n/2 \rceil)$, если делать k -й шаг пути вправо или вниз в соответствии с тем, находится ли k в четной или нечетной позиции перестановки. Этим правилом определяется взаимно однозначное соответствие между 2-упорядоченными перестановками и n -шаговыми путями из одного угла решетчатой диаграммы в другой. Например, изображенный на рисунке путь соответствует перестановке

$$2\ 1\ 3\ 4\ 6\ 5\ 7\ 10\ 8\ 11\ 9\ 12\ 14\ 13\ 15. \tag{1}$$

Далее, вертикальным отрезкам пути можно приписать "веса", как показано на диаграмме; отрезку, ведущему из точки (i, j) в точку $(i + 1, j)$ приписывается вес $|i - j|$. Немного подумав, читатель убедится в том, что сумма этих весов вдоль каждого пути равна числу инверсий в соответствующей перестановке (см. упр. 12). Так, например, перестановка (1) содержит $1 + 0 + 1 + 0 + 1 + 2 + 1 = 6$ инверсий.

Если $a \leq a'$ и $b \leq b'$, то число допустимых путей из (a, b) в (a', b') равно числу способов перемешать $a' - a$ вертикальных отрезков с $b' - b$ горизонтальными, а именно

$$\binom{a' - a + b' - b}{a' - a}.$$

Следовательно, число перестановок, для которых соответствующие пути проходят через вертикальный отрезок из (i, j) в $(i + 1, j)$, равно

$$\binom{i + j}{i} \binom{n - i - j - 1}{\lfloor n/2 \rfloor - j}.$$

Умножая это значение на вес данного отрезка и суммируя по всем отрезкам, получаем

$$A_{2n} = \sum_{\substack{0 \leq i \leq n \\ 0 \leq j \leq n}} |i - j| \binom{i + j}{i} \binom{2n - i - j - 1}{n - j};$$

$$A_{2n+1} = \sum_{\substack{0 \leq i \leq n \\ 0 \leq j \leq n}} |i - j| \binom{i + j}{i} \binom{2n - i - j}{n - j}; \tag{2}$$

Знаки абсолютной величины в этих суммах несколько усложняют вычисления, но в упр. 14 показано, что величина A_n имеет удивительно простой вид: $\lfloor n/2 \rfloor 2^{n-2}$. Следовательно, среднее число инверсий в случайной 2-упорядоченной перестановке равно

$$\lfloor n/2 \rfloor 2^{n-2} / \binom{n}{\lfloor n/2 \rfloor}.$$

По формуле Стирлинга эта величина асимптотически приближается к $\sqrt{\pi/128} n^{3/2} \approx 0.15 n^{3/2}$. Как легко видеть, максимальное число инверсий равно

$$\binom{\lfloor n/2 \rfloor + 1}{2} \approx \frac{1}{8} n^2.$$

Полезно исследовать распределение числа инверсий более тщательно, рассмотрев производящие функции

$$h_1(z) = 1, \quad h_2(z) = 1 + z,$$

$$h_3(z) = 1 + 2z, \quad h_4(z) = 1 + 3z + z^2 + z^3, \dots, \tag{3}$$

как в упр. 15. Таким образом, найдем, что стандартное отклонение тоже пропорционально $n^{3/2}$, так что число инверсий не слишком устойчиво распределено около своего среднего значения. Рассмотрим теперь общий двухпроходный случай алгоритма D, когда шаги сортировки равны h и 1.

Теорема Н. Среднее число инверсий в h -упорядоченной перестановке множества $\{1, 2, \dots, n\}$ равно

$$f(n, h) = \frac{2^{2q-1}q!q!}{(2q+1)!} \left(\binom{h}{2} q(q+1) + \binom{r}{2} (q+1) - \frac{1}{2} \binom{h-r}{2} q \right), \quad (4)$$

где $q = \lfloor n/h \rfloor$, $r = n \bmod h$. Эта теорема принадлежит Дугласу Ханту [Bachelor's thesis, Princeton University (April, 1967)]. Заметим, что формула справедлива и при $h \geq n$: $f(n, h) = \frac{1}{2} \binom{n}{2}$.

Доказательство. В h -упорядоченной перестановке содержится r упорядоченных подпоследовательностей длины $q+1$ и $h-r$ подпоследовательностей длины q . Каждая инверсия образуется из элементов двух различных подпоследовательностей, а каждая пара различных упорядоченных подпоследовательностей в случайной h -упорядоченной перестановке определяет случайную 2-упорядоченную перестановку. Поэтому среднее число инверсий равно сумме средних значений числа инверсий во всех парах различных подпоследовательностей, а именно

$$\binom{r}{2} \frac{A_{2q+2}}{\binom{2q+2}{q+1}} + r(h-r) \frac{A_{2q+1}}{\binom{2q+1}{q}} + \binom{h-r}{2} \frac{A_{2q}}{\binom{2q}{q}} = f(n, h). \blacksquare$$

Следствие. Если последовательность приращений удовлетворяет условию

$$h_{s+1} \bmod h_s = 0 \quad \text{при } t > s \geq 1, \quad (5)$$

то среднее число операций перемещения в алгоритме D равно

$$\sum_{t \geq s \geq 1} (r_s f(q_s + 1, h_{s+1}/h_s) + (h_s - r_s) f(q_s, h_{s+1}/h_s)), \quad (6)$$

где $r_s = N \bmod h_s$, $q_s = \lfloor N/h_s \rfloor$, $h_{t+1} = Nh_t$, а функция f определяется формулой (4).

Доказательство. Процесс h_s -сортировки состоит из сортировки простыми вставками $r_s(h_{s+1}/h_s)$ -упорядоченных подфайлов длины $q_s + 1$ и $(h_s - r_s)$ таких подфайлов длины q_s . Поскольку мы предполагаем, что исходная перестановка случайна и все ее элементы различны, то из условий делимости следует, что каждый из подфайлов — "случайная" (h_{s+1}/h_s) -упорядоченная перестановка в том смысле, что все (h_{s+1}/h_s) -упорядоченные перестановки равновероятны. \blacksquare

Условие (5) этого следствия всегда выполняется для *двухпроходной* сортировки методом Шелла, когда шаги равны соответственно h и 1. Пусть $q = \lfloor N/h \rfloor$, а $r = N \bmod h$, тогда среднее значение величины B в программе D равно

$$rf(q+1, N) + (h-r)f(q, N) + f(N, h) = \frac{r}{2} \binom{q+1}{2} + \frac{h-r}{2} \binom{q}{2} + f(N, h).$$

В первом приближении функция $f(n, h)$ равна $(\sqrt{\pi}/8)n^{3/2}h^{1/2}$; ср. с гладкой кривой на рис. 12. Следовательно, время работы

Picture: Рис. 12. Среднее число инверсий $f(n, h)$ в h -упорядоченном файле из n элементов для случая $n = 64$.

двухпроходной программы D примерно пропорционально $2N^2/h + \sqrt{\pi N^3 h}$. Поэтому наилучшее значение h равно приблизительно $\sqrt[3]{16N/\pi} \approx 1.72\sqrt[3]{N}$; при таком выборе h среднее время работы пропорционально $N^{5/3}$.

Таким образом, даже применяя метод Шелла с всего двумя шагами, можно существенно сократить время по сравнению с простыми вставками, с $O(N^2)$ до $O(N^{1.667})$. Ясно, что можно добиться лучших результатов, если использовать больше шагов. В упр. 18 обсуждается оптимальный выбор h_t, \dots, h_1 при фиксированном t в случае, когда значения h ограничены условием делимости; время работы при больших N сокращается до $O(N^{1.5+\varepsilon/2})$ где $\varepsilon = 1/(2^t - 1)$. Если мы пользуемся приведенными выше формулами, барьер $N^{1.5}$ преодолеть невозможно, потому что при последнем просмотре в сумму инверсий всегда вносится вклад

$$f(N, h_2) \approx (\sqrt{\pi}/8)N^{3/2}h_2^{1/2}.$$

Но интуиция подсказывает, что, если шаги h_t, \dots, h_1 не будут удовлетворять условию делимости (5), можно достичь большего. Например, при последовательном выполнении 8-, 4- и 2-сортировок невозможно взаимодействие между ключами в четных и нечетных позициях; поэтому на долю заключительной 1-сортировки останется $O(N^{3/2})$ инверсий. В то же время при последовательном выполнении 7-, 5- и 3-сортировок файл перетряхивается так, что при заключительной 1-сортировке не может встретиться более $2N$ инверсий! (См. упр. 26.) На самом деле наблюдается удивительное явление.

Теорема К. После h -сортировки k -упорядоченный файл остается k -упорядоченным.

Таким образом, файл, который был сначала 7-отсортирован, а потом 5-отсортирован, становится одновременно 7- и 5-упорядоченным. А если мы подвергнем его 3-сортировке, то полученный файл будет 7-, 5- и 3-упорядочен. Примеры проявления этого замечательного свойства можно найти в табл. 4.

Picture: Таблица 4. Сортировка с убывающим шагом (7, 5, 3, 1)

Доказательство. В упр. 20 показано, что эта теорема вытекает из следующей леммы:

Лемма Л. Пусть m, n, r — неотрицательные целые числа, а x_1, \dots, x_{m+r} и y_1, \dots, y_{n+r} — произвольные последовательности чисел, такие, что

$$y_1 \leq x_{m+1}, y_2 \leq x_{m+2}, \dots, y_r \leq x_{m+r}. \quad (7)$$

Если последовательности x и y отсортировать независимо, так что $x_1 \leq \dots \leq x_{m+r}$ и $y_1 \leq \dots \leq y_{n+r}$, то соотношения (7) останутся в силе.

Доказательство. Известно, что все, кроме, быть может, m элементов последовательности x , превосходят (т. е. больше или равны) некоторые элементы последовательности y , причем различные элементы x превосходят различные элементы y . Пусть $1 \leq j \leq r$. Так как после сортировки элемент x_{m+j} превосходит $m+j$ элементов x , то он превосходит по крайней мере j элементов y , а значит, он превосходит j наименьших элементов y . Следовательно, после сортировки имеем $x_{m+j} \geq y_j$. ■ ■

Из теоремы К видно, что при сортировке желательно пользоваться взаимно простыми значениями шагов, однако непосредственно из нее не следуют точные оценки числа перемещений, выполняемых алгоритмом D. Так как число перестановок множества $\{1, 2, \dots, n\}$, одновременно h - и k -упорядоченных, не всегда является делителем $n!$, то понятно, что теорема К объясняет далеко не все; в результате k - и h -сортировок некоторые k - и h -упорядоченные файлы получаются чаще других. Более того, не существует очевидного способа отыскать "наихудший случай" для алгоритма D при произвольной последовательности шагов сортировки h_t, \dots, h_1 . Поэтому до сих пор все попытки анализа этого алгоритма в общем случае были тщетны; по существу, все, что нам известно, — это приближенное асимптотическое поведение максимального времени работы в некоторых случаях.

Теорема Р. Если $h_s = 2^s - 1$ при $1 \leq s \leq t = \lfloor \log_2 N \rfloor$, то время работы алгоритма D есть $O(N^{3/2})$

Доказательство. Достаточно найти оценку B_s числа перемещений при s -м просмотре, такую, чтобы $B_t + \dots + B_1 = O(N^{3/2})$. Для первых $t/2$ просмотров при $t \leq s \leq t/2$ можно воспользоваться очевидной оценкой $B_s = O(h_s(N/h_s)^2)$, а для последующих просмотров можно применить результат упр. 23: $B_s = O(Nh_{s+2}h_{s+1}/h_s)$. Следовательно, $B_t + \dots + B_1 = O(N(2 + 2^2 + \dots + 2^{t/2} + 2^{t/2} + \dots + 2)) = O(N^{3/2})$. ■

Эта теорема принадлежит А. А. Папернову и Г. В. Стасевичу [Проблемы передачи информации, 1,3 (1965), 81–98]. Она дает верхнюю оценку максимального времени работы алгоритма, а не

Таблица 5

Анализ алгоритма D при $N = 8$							
Шаги	A_{ave}	B_{ave}	S	T	время машины MIX		
1	1.718	14.000	1	1	204.85u		
2	1	2.667	9.657	3	2	235.91u	
3	1	2.917	9.100	4	2	220.16u	
4	1	3.083	10.000	5	2	217.75u	
5	1	2.601	10.000	6	2	210.00u	
6	1	2.135	10.667	7	3	206.60u	
7	1	1.718	12.000	8	2	208.85u	
4	2	1	3.500	8.324	7	3	272.32u
5	3	1	3.301	8.167	9	3	251.60u
3	2	1	3.320	7.829	6	3	278.50u

просто оценку среднего времени работы. Этот результат нетривиален, поскольку максимальное время работы в случае, когда приращения h удовлетворяют условию делимости (5), имеет порядок N^2 , а в упр. 24 доказано, что показатель $3/2$ уменьшить нельзя.

Интересное улучшение по сравнению с теоремой Р обнаружил в 1969 г. Воган Пратт. Если все шаги сортировки выбираются из множества чисел вида $2^p 3^q$, меньших N , то время работы алгоритма D будет порядка $N(\log N)^2$. В этом случае также можно внести в алгоритм несколько существенных упрощений. К сожалению, метод Пратта требует сравнительно большого числа просмотров, так что это не лучший способ выбора шагов, если только N не очень велико; см. упр. 30 и 31.

Рассмотрим *общее* время работы программы D , именно $(9B + 10NT + 13T - 10S - 3A + 1)u$. В табл. 5 показано среднее время работы для различных последовательностей шагов при $N = 8$. Каждый элемент таблицы можно вычислить с помощью формул, приведенных выше или в упр. 19, за исключением случаев, когда шаги равны 5 3 1 и 3 2 1; для этих двух случаев было проведено тщательное исследование всех $8!$ перестановок. Заметим, что при таком малом значении N в общем времени работы преобладают вспомогательные операции, поэтому наилучшие результаты получаются при $t = 1$; следовательно, при $N = 8$ лучше всего пользоваться простыми вставками. (Среднее время работы программы S при $N = 8$ равно всего $191.85u$.) Любопытно, что наилучший результат в двухпроходном алгоритме достигается при $h_2 = 6$, поскольку большая величина 5 оказывается важнее малой величины B . Аналогично три шага 3 2 1 минимизируют среднее число перемещений, но это не самая лучшая последовательность для трех просмотров. Быть может, интересно привести некоторые "наихудшие" перестановки, максимизирующие число перемещений, так как общий способ построения таких перестановок до сих пор не известен:

$$\begin{aligned} h_3 = 5, \quad h_2 = 3, \quad h_1 = 1: & \quad 8 \ 5 \ 2 \ 6 \ 3 \ 7 \ 4 \ 1 \quad (19 \text{ перемещений}) \\ h_3 = 3, \quad h_2 = 2, \quad h_1 = 1: & \quad 8 \ 3 \ 5 \ 7 \ 2 \ 4 \ 6 \ 1 \quad (17 \text{ перемещений}) \end{aligned}$$

С ростом N наблюдается несколько иная картина. В табл. 6 показаны приближенные значения числа перемещений для различных последовательностей шагов при $N = 1000$. Первые несколько последовательностей удовлетворяют условию делимости (5), так что можно воспользоваться формулой (6); для получения средних значений при других последовательностях шагов применялись эмпирические тесты. Были сгенерированы пять файлов по 1000 случайных элементов, и каждый файл сортировался с каждой последовательностью шагов.

Эти данные позволяют выявить некоторые характеристики, но поведение алгоритма D все еще остается неясным. Шелл первоначально предполагал использовать шаги $\lfloor N/2 \rfloor$, $\lfloor N/4 \rfloor$, $\lfloor N/8 \rfloor$, ... но это нежелательно, если двоичное представление числа N содержат длинные цепочки нулей. Лазарус и Фрэнк [САСМ, 3 (1960), 20–22] предложили использовать, по существу, ту же последовательность, но добавляя 1 там, где это необходимо, чтобы сделать все шаги нечетными. Хиббард [САСМ, 6 (1963), 206–213] предложил шаги вида $2^k - 1$; Папернов и Стасевич предложили последовательность $2^k + 1$. Среди других естественных последовательностей, использованных для получения табл. 6, — последовательности $(2^k - (-1)^k)/3$, $(3^k - 1)/2$ и числа Фибоначчи.

Минимальное число перемещений 6600 наблюдается для шагов вида $2^k + 1$, но важно понимать, что надо учитывать не только число перемещений, хотя именно оно асимптотически доминирует в общем времени работы. Так как время работы программы D равно $9B + 10NT + \dots$ единиц, ясно, что экономия одного просмотра примерно эквивалентна сокращению числа перемещений на $\frac{10}{9}N$; мы готовы добавить 1100 перемещений, если за счет этого удастся сэкономить один просмотр. Поэтому представляется неразумным начинать с h_t , большего, чем, скажем, $N/3$, поскольку большой шаг не убавит числа последующих перемещений настолько, чтобы оправдать первый просмотр.

Picture: Таблица 6.

Большое число экспериментов с алгоритмом D провели Джеймс Петерсон и Дэвид Л. Рассел в Стэнфордском университете в 1971 г. Они обнаружили, что для среднего числа перемещений B хорошим приближением при $100 \leq N \leq 60\,000$ служат следующие формулы:

$$\begin{aligned} 1.09N^{1.27} \text{ или } .30N(\ln N)^2 - 1.35N \ln N & \text{ для последовательности шагов } 2^k + 1, \dots, 9, 5, 3, 1; \\ 1.22N^{1.26} \text{ или } .29N(\ln N)^2 - 1.26N \ln N & \text{ для последовательности шагов } 2^k - 1, \dots, 15, 7, 3, 1; \\ 1.12N^{1.28} \text{ или } .36N(\ln N)^2 - 1.73N \ln N & \text{ для последовательности шагов } (2^k \pm 1)/3, \dots, 11, 5, 3, 1; \\ 1.66N^{1.25} \text{ или } .33N(\ln N)^2 - 1.26N \ln N & \text{ для последовательности шагов } (3^k - 1)/2, \dots, 40, 13, 4, 1. \end{aligned}$$

Например, при $N = 20\,000$ для всех этих типов шагов получаем соответственно $B \approx 31\,000$, $33\,000$, $35\,000$, $39\,000$. Табл. 7 дает типичную картину того, как распределяются перемещения по просмотрам в трех из этих экспериментов. Любопытно, что *оба* вида функций $\alpha N(\ln N)^2 + \beta N \ln N$ и βN^α , кажется, довольно хорошо согласуются с наблюдаемыми данными, хотя степенная функция была существенно лучше при меньших значениях N . Дальнейшие эксперименты были выполнены для

последовательности шагов $2^k - 1$ со значением N , достигающим 250 000; сорок пять испытаний с $N = 250\,000$ дали значение $B \approx 7\,900\,000$ при наблюдаемом стандартном отклонении 50 000. "Наиболее подходящими" формулами для диапазона $100 \leq N \leq 250\,000$ оказались соответственно $1.21N^{1.26}$ и $.39N(\ln N)^2 - 2.33N \ln N$. Так как коэффициенты в представлении степенной функцией остались почти такими же, в то время как коэффициенты в логарифмическом представлении довольно резко изменились, то разумно предположить, что именно степенная функция описывает истинное асимптотическое поведение метода Шелла.

Таблица 7
Количества перемещений по просмотрам (примеры для $N = 20000$)

h_s	B_s	h_s	B_s	h_s	B_s
4095	19460	4097	19550	3280	25210
2047	15115	2049	14944	1093	28324
1023	15869	1025	15731	364	35477
511	18891	513	18548	121	47158
255	22306	257	21827	40	62110
127	27400	129	27814	13	88524
63	35053	65	33751	4	74599
31	34677	33	34303	1	34666
15	51054	17	46044		
7	40382	9	35817		
3	24044	5	19961		
1	16789	3	9628		
		1	13277		

Эти эмпирические данные ни коим образом не исчерпывают всех возможностей, и мы не получили оснований для решительных заключений о том, какие же последовательности шагов являются наилучшими для алгоритма D. Поскольку приращения вида $(3^k - 1)/2$ не увеличивают существенно числа перемещений и поскольку для них требуется лишь примерно $5/8$ числа просмотров, необходимых для шагов других типов, то, очевидно, *разумно выбирать последовательность шагов следующим образом:*

$$\text{Принять } h_t = 1, h_{s+1} = 3h_s + 1 \text{ и остановиться на } h_t, \text{ когда } h_{t+2} \geq N. \quad (8)$$

Вставки в список. Оставим теперь метод Шелла и рассмотрим другие пути усовершенствования простых вставок. Среди общих способов улучшения алгоритма один из самых важных основывается на тщательном анализе структур данных, поскольку реорганизация структур данных, позволяющая избежать ненужных операций, часто дает существенный эффект. Дальнейшее обсуждение этой общей идеи можно найти в § 2.4, в котором изучается довольно сложный алгоритм. Посмотрим, как она применяется к такому нехитрому алгоритму, как простые вставки. Какова наиболее подходящая структура данных для алгоритма S?

Сортировка простыми вставками состоит из двух основных операций:

- i) просмотра упорядоченного файла для нахождения наибольшего ключа, меньшего или равного данному ключу;
- ii) вставки новой записи в определенное место упорядоченного файла.

Файл—это, очевидно, линейный список, и алгоритм S обрабатывает его, используя последовательное распределение (п. 2.2.2); поэтому для выполнения каждой операции вставки необходимо переместить примерно половину записей. С другой стороны, нам известно, что для вставок идеально подходит связанное распределение (п. 2.2.3), так как при этом требуется изменить лишь несколько связей; другая операция—последовательный просмотр—при связанном распределении почти так же проста, как и при последовательном. Поскольку списки всегда просматриваются в одном и том же направлении, то достаточно иметь списки с одной связью. Таким образом, приходим к выводу, что "правильная" структура данных для метода простых вставок—линейные списки с одной связью. Удобно также изменить алгоритм S, чтобы список просматривался в возрастающем порядке.

Алгоритм L. (Вставки в список.) Предполагается, что записи R_1, \dots, R_N содержат ключи K_1, \dots, K_N и "поля связи" L_1, \dots, L_N , в которых могут храниться числа от 0 до N ; имеется также еще одно поле связи L_0 в некоторой искусственной записи R_0 в начале файла. Алгоритм устанавливает поля связи так, что записи оказываются связанными в возрастающем порядке. Так, если $p(1) \dots p(N)$ —"устойчивая" перестановка, такая, что $K_{p(1)} \leq \dots \leq K_{p(N)}$, то в результате применения алгоритма

получим

$$L_0 = p(1); L_{p(i)} = p(i+1) \text{ при } 1 \leq i < N; L_{p(N)} = 0. \quad (9)$$

- L1** [Цикл по j .] Установить $L_0 \leftarrow N, L_N \leftarrow 0$. (L_0 служит "головой" списка, а 0—пустой связью; следовательно, список, по существу, циклический.) Выполнить шаги от **L2** до **L5** при $j = N - 1, N - 2, \dots, 1$ и завершить работу алгоритма.
- L2** [Установить p, q, K .] Установить $p \leftarrow L_0, q \leftarrow 0, K \leftarrow K_j$. (В последующих шагах мы вставим запись R_j в нужное место в связанном списке путем сравнения ключа K с предыдущими ключами в возрастающем порядке. Переменные p и q служат указателями на текущее место в списке, причем $p = L_q$, так что q всегда на один шаг отстает от p .)
- L3** [Сравнить K, K_p .] Если $K \leq K_p$, то перейти к шагу **L5**. (Мы нашли искомое положение записи R в списке между R_q и R_p .)
- L4** [Продвинуть p, q .] Установить $q \leftarrow p, p \leftarrow L_q$. Если $p > 0$, то возвратиться к шагу **L3**. (Если $p = 0$, то K —наибольший ключ, обнаруженный до сих пор; следовательно, запись R должна попасть в конец списка, между R_q и R_0 .)
- L5** [Вставить в список.] Установить $L_q \leftarrow j, L_j \leftarrow p$. ■

Таблица 8

Пример применения алгоритма вставок в список																	
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K_i	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
L_j	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0
L_j	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	15
L_j	14	—	—	—	—	—	—	—	—	—	—	—	—	—	16	0	15

Этот алгоритм важен не только как простой метод сортировки, но и потому, что он часто встречается как часть других алгоритмов обработки списков. В табл. 8 показаны первые несколько шагов сортировки шестнадцати чисел, выбранных нами для примеров.

Программа L. (Вставки в список.) Предполагается, что ключ K_j хранится в поле $\text{INPUT} + j(0:3)$, а L_j —в поле $\text{INPUT} + j(4:5)$. Значения регистров: $\text{rI1} \equiv j; \text{rI2} \equiv p; \text{rI3} \equiv q; \text{rA}(0:3) \equiv K$.

KEY	EQU	0:3		
LINK	EQU	4:5		
START	ENT1	N	1	L1. Цикл по $j, j \leftarrow N$.
	ST1	INPUT(LINK)	1	$L_0 \leftarrow N$.
	STZ	INPUT+N(LINK)	1	$L_N \leftarrow 0$.
	JMP	6F	1	Переход к уменьшению j .
2H	LD2	INPUT(LINK)	$N - 1$	L2. Установить $p, q, K. p \leftarrow L_0$.
	ENT3	0	$N - 1$	$q \leftarrow 0$.
	LDA	INPUT, 1	$N - 1$	$K \leftarrow K_j$.
3H	CMPL	INPUT, 2 (KEY)	$B + N - 1 - A$	L3. Сравнить K, K_p .
	JLE	5F	$B + N - 1 - A$	K L5, если $K \leq K_p$.
4H	ENT3	0, 2	B	L4. Продвинуть $p, q. q \leftarrow p$.
	LD2	INPUT, 3 (LINK)	B	$p \leftarrow L_q$.
	J2P	3B	B	K L3, если $p > 0$.
5H	ST1	INPUT, 3 (LINK)	$N - 1$	L5. Вставить в список. $L_q \leftarrow j$.
	ST2	INPUT, 1 (LINK)	$N - 1$	$L_j \leftarrow p$.
6H	DEC1	1	N	
	J1P	2B	N	$N > j \geq 1$.

■

Время работы этой программы равно $7B + 14N - 3A - 6$ единиц, где N —длина файла, A —число правосторонних максимумов, а B —число инверсий в исходной перестановке. (Ср. с анализом программы S. Заметьте, что программа L не перемещает записи в памяти; это можно сделать, как в упр. 5.2-12, затратив дополнительно $20N$ единиц времени.) Программа S требует $(9B + 10N - 3A - 9)u$, а так как B равно примерно $\frac{1}{4}N^2$, то нетрудно видеть, что за счет дополнительного пространства памяти, выделенного под поля связи, удалось сэкономить примерно 22% времени работы. Аккуратно

программируя, можно сберечь еще 22% (см. упр. 33), но время работы все же останется пропорциональным N^2 .

Подведем итог сделанному до сих пор. Мы начали с алгоритма S—простого и естественного алгоритма сортировки, который выполняет около $\frac{1}{4}N^2$ сравнений и около $\frac{1}{4}N^2$ перемещений. Мы усовершенствовали его, рассмотрев бинарные вставки, при которых выполняется около $N \log_2 N$ сравнений и $\frac{1}{4}N^2$ перемещений. Несколько изменив структуру данных, применив ”двухпутевые вставки”, сумели сократить число перемещений до $\frac{1}{8}N^2$. При сортировке методом Шелла ”с убывающим шагом” число сравнений и перемещений снижается примерно до $N^{1.3}$ для тех значений N , которые встречаются на практике; при $N \rightarrow \infty$ это число можно сократить до порядка $N(\log_2 N)^2$. Дальнейшее стремление улучшить алгоритм—путем применения связанной структуры данных—привело нас к вставкам в список, при которых выполняется около $\frac{1}{4}N^2$ сравнений, 0 перемещений и $2N$ изменений связей.

Можно ли соединить лучшие свойства этих методов, сократив число сравнений до порядка $N \log N$, как при бинарных вставках, и исключив при этом перемещения данных, как при вставках

Picture: Рис. 13. Пример схемы Уилера для вставок в дерево.

в список? Ответ утвердительный: это достигается переходом к древовидной структуре. Такая возможность была впервые исследована около 1957 г. Д. Дж. Уилером, который предложил использовать двухпутевые вставки до тех пор, пока не появится необходимость перемещать данные. Тогда вместо того, чтобы их перемещать, вставляется указатель на новую область памяти, и тот же самый метод рекуррентно применяется ко всем элементам, которые нужно вставить в эту новую область памяти. Оригинальный метод Уилера [см. A. S. Douglas, *Comp. J.*, 2 (1959), 5] представляет собой сложную комбинацию последовательной и связанной памяти с узлами переменного размера; для наших шестнадцати чисел было бы сформировано дерево, показанное на рис. 13. Аналогичную, но более простую схему вставки в дерево с использованием бинарных деревьев независимо разработал около 1958 г. К.М. Бернес-Ли [см. *Comp. J.*, 3 (1960), 174, 184]. Сам этот метод и его модернизации весьма важны как для сортировки, так и для поиска, поэтому подробно они обсуждаются в гл. 6.

Еще один путь улучшить простые вставки—попытаться вставлять несколько элементов одновременно. Если, например, имеется файл из 1000 элементов и 998 из них уже отсортированы, то алгоритм S выполнит еще два просмотра файла (вставив сначала R_{999} , а потом R_{1000}). Очевидно, можно сэкономить время, если сначала сравнить ключи K_{999} с K_{1000} чтобы выяснить, который из них больше, а потом вставить их *оба* за один просмотр файла. Комбинированная операция такого рода требует около $(2/3)N$ сравнений и перемещений (ср. с упр. 3.4.2–5) вместо двух просмотров, примерно по $N/2$ сравнений и перемещений каждый.

Иначе говоря, обычно бывает полезно ”группировать” операции, которые требуют длительного поиска, чтобы можно было выполнить несколько операций вместе. Если довести эту идею до ее естественного завершения, то мы заново откроем для себя сортировку посредством слияния, настолько важную, что ей посвящен отдельный пункт.

Сортировка с вычислением адреса. Теперь уж мы, несомненно, исчерпали все возможные способы усовершенствовать метод простых вставок, но давайте подумаем еще! Представьте себе, что вам дали чистый лист бумаги и собираются диктовать какие-то слова. Ваша задача—записать их в алфавитном порядке и вернуть листок с отсортированным списком слов. Услышав слово на букву А, вы будете стремиться записать его ближе к верхнему краю страницы, тогда как слово на букву Я будет, по-видимому, помещено ближе к нижнему краю страницы и т. д. Аналогичный способ применяется при расстановке книг на полке по фамилиям авторов, если книги берутся с пола в случайном порядке: ставя книгу на полку, вы оцениваете ее конечное положение, сокращая таким образом число необходимых сравнений и перемещений. (Эффективность процесса повышается, если на полке имеется немного больше места, чем это абсолютно необходимо.) Такой метод машинной сортировки впервые предложили Исаак и Синглтон, [*JACM*, 3 (1956), 169–174]; он получил дальнейшее развитие в работе Кронмэла и Тартара [*Proc. ACM Nat'l Conf.*, 21 (1966), 331–337].

Сортировка с вычислением адреса обычно требует дополнительного пространства памяти либо для того, чтобы оставить достаточно свободного места и не делать много лишних перемещений, либо для хранения вспомогательных таблиц, которые бы позволяли учитывать неравномерность распределения ключей. (См. сортировку распределяющим подсчетом (алгоритм 5.2D), которая является разновидностью сортировки с вычислением адреса.) Дополнительное пространство будет, по-видимому, использоваться наилучшим образом, если отвести его под поля связи, как в методе вставок в список. К тому же отпадает необходимость выделять отдельные области для ввода и вывода; все операции можно выполнить в одной и той же области памяти.

Основная идея—так обобщить метод вставок в список, чтобы располагать не одним списком, а *несколькими*. Каждый список содержит ключи из определенного диапазона. Мы делаем важное предположение о том, что ключи распределены довольно равномерно и не ”скапливаются” хаотически в каких-то диапазонах. Множество всех возможных значений ключей разбивается на M отрезков и предполагается, что данный ключ попадает в данный отрезок с вероятностью $1/M$. Отводим дополнительную память под головы M списков, а каждый список строим, как при простых вставках в список.

Нет необходимости приводить здесь этот алгоритм со всеми подробностями. Достаточно вначале установить все головы списков равными Λ , а для каждого вновь поступившего элемента предварительно решить, в какой из M отрезков он попадает, после чего вставить его в соответствующий список, как в алгоритме L.

Чтобы проиллюстрировать этот метод в действии, предположим, что наши 16 ключей разделены на $M = 4$ диапазона 0–249, 250–499, 500–749, 750–999. В процессе сортировки получаются следующие конфигурации:

Списки	Пришло 4 элемента	Пришло 8 элементов	Пришло 12 элементов	Конечное состояние
1:	061, 087	061, 087, 170	061, 087, 154, 170	061, 087, 154, 170
2:		275	275, 426	275, 426
3:	503, 512	503, 512	503, 509, 512, 653	503, 509, 512, 612 653, 677, 703
4:		897, 908	897, 908	765, 897, 908

Благодаря применению связанной памяти не возникает проблемы распределения памяти при использовании списков переменной длины.

Программа M. (Вставки в несколько списков.) В этой программе делаются те же предположения, что и в программе L, но ключи должны быть *неотрицательными*; так что

$$0 \leq \text{KEY} < (\text{BYTESIZE})^3.$$

Этот диапазон делится в программе на M равных частей путем умножения каждого ключа на подходящую константу. Головы списков хранятся в ячейках $\text{HEAD} + 1, \dots, \text{HEAD} + M$.

KEY	EQU	1:3		
LINK	EQU	4:5		
START	ENT2	M	1	
	STZ	HEAD, 2	M	HEAD[p] ← Λ .
	DEC2	1	M	
	J2P	*-2	M	$M \geq p \geq 1$.
	ENT1	N	1	$j \leftarrow N$.
2H	LDA	INPUT, 1 (KEY)	N	
	MUL	=M(1:3)=	N	$rA \leftarrow \lfloor M \times K_j / \text{BYTESIZE}^3 \rfloor$.
	STA	*+1(1:2)	N	
	ENT3	0	N	$q \leftarrow rA$.
	INC3	HEAD+1-INPUT	N	$q \leftarrow \text{LOC}(\text{HEAD}[q])$.
	LDA	INPUT, 1	N	$K \leftarrow K_j$.
	JMP	4F	N	Установить p.
3H	CMPA	INPUT, 2 (KEY)	$B + N - A$	
	JLE	5F	$B + N - A$	Вставить, если $K \leq K_p$.
	ENT3	0, 2	B	$q \leftarrow p$.
4H	LD2	INPUT, 3 (LINK)	$B + N$	$p \leftarrow \text{LINK}(q)$.
	J2P	3B	$B + N$	Переход, если не конец списка.
5H	ST1	INPUT, 3 (LINK)	N	$\text{LINK}(q) \leftarrow \text{LOC}(R_j)$.
	ST2	INPUT, 1 (LINK)	N	$\text{LINK}(\text{LOC}(R_j)) \leftarrow p$.
6H	DEC1	1	N	
	J1P	2B	N	$N \geq j \geq 1$.

Эта программа написана для произвольного значения M , но лучше зафиксировать M , положив его равным некоторому удобному значению; можно, например, положить $M = \text{BYTESIZE}$, тогда головы

списков можно опустошить одной-единственной командой MOVE, а последовательность команд 08–11, реализующих умножение, заменить командой LD3 INPUT, 1(1:1). Наиболее заметное отличие программы M от программы L состоит в том, что в программе M нужно рассматривать случай пустого списка, когда не надо делать сравнений.

Сколько же времени мы экономим, имея M списков вместо одного? Общее время работы программы M равно $7B + 31N - 3A + 4M + 2$ единиц, где M —число списков, N —число сортируемых записей, A и B равны соответственно числу правосторонних максимумов и числу инверсий среди ключей, принадлежащих каждому списку. (При анализе этого алгоритма в отличие от всех других анализов в данном пункте мы считаем крайний правый элемент непустой перестановки правосторонним максимумом, а не игнорируем его.) Мы уже изучили величины A и B при $M = 1$, когда их средние значения равны соответственно H_N и $\frac{1}{2} \binom{N}{2}$. Согласно предположению о распределении ключей, вероятность того, что данный список в конце сортировки будет содержать ровно n элементов, есть "биномиальная" вероятность

$$\binom{N}{n} \left(\frac{1}{M}\right)^n \left(1 - \frac{1}{M}\right)^{N-n}. \quad (10)$$

Поэтому средние значения величин A и B в общем случае равны

$$A_{ave} = M \sum_n \binom{N}{n} \left(\frac{1}{M}\right)^n \left(1 - \frac{1}{M}\right)^{N-n} H_n; \quad (11)$$

$$B_{ave} = M \sum_n \binom{N}{n} \left(\frac{1}{M}\right)^n \left(1 - \frac{1}{M}\right)^{N-n} \binom{n}{2}. \quad (12)$$

По теореме 1.2.7A

$$\sum_n \binom{N}{n} (M-1)^{-n} H_n = \left(1 - \frac{1}{M}\right)^{-N} (H_N - \ln M) + \varepsilon, \quad 0 < \varepsilon = \sum_{n>N} \frac{1}{n} \left(1 - \frac{1}{M}\right)^{n-N} < \frac{M-1}{N+1};$$

следовательно,

$$A_{ave} = M(H_N - \ln M) + \delta, \quad 0 < \delta < \frac{M^2}{N+1} \left(1 - \frac{1}{M}\right)^{N+1}. \quad (13)$$

(Эта формула практически бесполезна, если $M \approx N$. Более подробно асимптотическое поведение величины A_{ave} при $M = N/\alpha$ обсуждается в упр. 5.2.2-57.) Сумму (12) легко вычислить с помощью тождества

$$\binom{N}{n} \binom{n}{2} = \binom{N}{2} \binom{N-2}{n-2},$$

которое представляет собой частный случай тождества (1.2.6-20); получаем

$$B_{ave} = \frac{1}{2M} \binom{N}{2}. \quad (14)$$

(Стандартное отклонение для величины B см. в упр. 37.) Следовательно, общее время работы программы M при фиксированном M и $N \rightarrow \infty$ равно

$$\begin{array}{ll} \min & 31N + M + 2, \\ \text{ave} & 1.75N^2/M + 31N - 3MH_N + 3M \ln M + 4M - 3 - 1.75N/M + 2, \\ \max & 3.50N^2 + 24.5N + 4M + 2. \end{array} \quad (15)$$

Заметим, что если M не слишком велико, то среднее время работы сокращается в M раз. При $M = 10$ сортировка будет примерно в 10 раз быстрее, чем при $M = 1$! Однако максимальное время работы гораздо больше среднего. Таким образом, подтверждается необходимость выполнения предположения о равномерности распределения ключей, так как наихудший случай имеет место, когда все ключи попадают в один список.

Если положить $M = N$, то среднее время работы будет примерно $34.36N$, при $M = \frac{1}{2}N$ оно несколько больше, равно приблизительно $34.52N$, а при $M = N/10$ оно равно приблизительно $48.04N$. (Заметим, что $10N$ из этих единиц времени машины MIX тратятся на одну лишь команду умножения!) Мы получили метод сортировки с временем работы порядка N при условии, что ключи довольно равномерно распределены в области изменения.

Упражнения

1. [10] Является ли алгоритм S алгоритмом "устойчивой" сортировки?
2. [11] Будет ли алгоритм S правильно сортировать числа, если в шаге S3 отношение " $K \geq K_i$ " заменить на " $K > K_i$ "?
- >3. [30] Является ли программа S самой короткой программой сортировки для машины MIX, или можно написать более короткую программу, которая бы давала тот же результат?
- >4. [M20] Найдите минимальное и максимальное время работы программы S как функцию от N .
- >5. [M27] Найдите производящую функцию $g_N(z) = \sum_{k \geq 0} p_{Nk} z^k$ для общего времени работы программы S, где p_{Nk} — вероятность того, что на выполнение программы S уйдет ровно k единиц времени при заданной исходной случайной перестановке множества $\{1, 2, \dots, N\}$. Вычислите также стандартное отклонение времени работы для данного значения N .
6. [33] Для двухпутевых вставок, проиллюстрированных в табл. 2, по-видимому, необходимо, помимо области ввода, содержащей N записей, иметь область вывода, в которой может храниться $2N + 1$ записей. Покажите, что можно выполнять двухпутевые вставки, имея как для ввода, так и для вывода пространство, достаточное для хранения всего $N + 1$ записей.
7. [M20] Пусть $a_1 a_2 \dots a_n$ — случайная перестановка множества $\{1, 2, \dots, n\}$; каково среднее значение величины $|a_1 - 1| + |a_2 - 2| + \dots + |a_n - n|$? (Оно равно произведению n на среднее чистое расстояние, на которое перемещается запись в процессе сортировки.)
8. [10] Является ли алгоритм D алгоритмом "устойчивой" сортировки?
9. [20] Какие значения A и B и какое общее время работы программы D соответствуют табл. 3 и 4? Проанализируйте достоинства метода Шелла по сравнению с простыми вставками в этом случае.
- >10. [22] В случае, когда $K_j \geq K_{j-h}$, в шаге D3 алгоритм D предписывает выполнение множества ненужных действий. Покажите, как можно изменить программу D, чтобы избежать этих избыточных вычислений, и обсудите преимущества такого изменения.
11. [M10] Какой путь на решетке (аналогичной представленной на рис. 11) соответствует перестановке 1 2 5 3 7 4 8 6 9 11 10 12?
12. [M20] Докажите, что сумма вертикальных весов пути на решетке равна числу инверсий соответствующей 2-упорядоченной перестановки.
- >13. [M22] Поясните, как нужно приписать веса горизонтальным отрезкам вместо вертикальных, чтобы сумма горизонтальных весов пути на решетке равнялась числу инверсий в соответствующей 2-упорядоченной перестановке.
14. [M24] (а) Покажите, что для сумм, определяемых равенством (2), $A_{2n+1} = 2A_{2n}$. (б) Если положить $r = -s - 1$, $t = 1$, то общее тождество из упр. 1.2.6-26 упрощается до

$$\sum_k \binom{2k+s}{k} z^k = \frac{1}{\sqrt{1-4z}} \left(\frac{1 - \sqrt{1-4z}}{2z} \right)^s.$$

Рассмотрев сумму $\sum_n A_{2n} z^n$ покажите, что

$$A_{2n} = n \cdot 4^{n-1}.$$

- >15. [BM33] Пусть $g_n(z)$, $\bar{g}_n(z)$, $h_n(z)$, $\bar{h}_n(z)$ равны $\sum z^{\text{общий вес пути}}$, где сумма берется по всем путям длины $2n$ на решетке из $(0, 0)$ в (n, n) , удовлетворяющим некоторым ограничениям на вершины, через которые эти пути проходят: для $h_n(z)$ нет ограничений, для $g_n(z)$ пути не должны проходить через вершины (i, j) , такие, что $i > j$; $\bar{h}_n(z)$ и $\bar{g}_n(z)$ определяются аналогично, но не допускаются также и вершины (i, i) при $0 < i < n$. Таким образом,

$$\begin{aligned} g_0(z) &= 1, & g_1(z) &= z, & g_2(z) &= z^3 + z^2; & \bar{g}_1(z) &= z, & \bar{g}_2(z) &= z^3; \\ h_0(z) &= 1, & h_1(z) &= z + 1, & h_2(z) &= z^3 + z^2 + 3z + 1; \\ \bar{h}_1(z) &= z + 1, & \bar{h}_2(z) &= z^3 + z. \end{aligned}$$

Найдите рекуррентные соотношения, определяющие эти функции, и воспользуйтесь этими рекуррентными соотношениями для доказательства равенства

$$h_n''(1) + h_n'(1) = \frac{7n^3 + 4n^2 + 4n}{30} \binom{2n}{n}.$$

(Отсюда легко находится точная формула дисперсии числа инверсий в случайной 2-упорядоченной перестановке множества $\{1, 2, \dots, 2n\}$; она асимптотически приближается к $(\frac{7}{30} - \frac{\pi}{16}) n^3$.)

16. [M24] Найдите формулу максимального числа инверсий в h -упорядоченной перестановке множества $\{1, 2, \dots, n\}$. Каково максимально возможное число перемещений, выполняемых алгоритмом D, если шаги сортировки удовлетворяют условию делимости (5)?
17. [M21] Покажите, что если $N = 2^t$ и $h_s = 2^{s-1}$ при $t \geq s \geq 1$, то существует единственная перестановка множества $\{1, 2, \dots, N\}$, максимизирующая число перемещений, выполняемых алгоритмом D. Найдите простой способ описания этой перестановки.
18. [BM24] При больших значениях N сумму (6) можно оценить величиной

$$\frac{1}{4} \frac{N^2}{h_t} + \frac{\sqrt{\pi}}{8} \left(\frac{N^{3/2} h_t^{1/2}}{h_{t-1}} + \dots + \frac{N^{3/2} h_2^{1/2}}{h_1} \right).$$

Каковы действительные числа h_t, \dots, h_1 , минимизирующие это выражение, если N и t фиксированы, а $h_1 = 1$?

- >19. [M25] Каково среднее значение величины A в анализе времени работы программы D, если шаги сортировки удовлетворяют условию делимости (5)?
20. [M20] Покажите, что теорема K следует из леммы L.
21. [M25] Пусть h и k — взаимно простые целые положительные числа. Покажите, что наибольшее целое число, которое нельзя представить в виде $ah + bk$, где a и b — неотрицательные целые числа, равно $hk - h - k$. Следовательно, если файл является одновременно h -упорядоченным и k -упорядоченным, то $K_i \leq K_j$ при $j - i \geq (h - 1)(k - 1)$.
22. [M30] Докажите, что любое целое число $\geq 2^h(2^h - 1)$ можно представить в виде $a_0(2^h - 1) + a_1(2^{h+1} - 1) + a_2(2^{h+2} - 1) + \dots$, где a_j — неотрицательные целые числа; но число $2^h(2^h - 1) - 1$ нельзя представить в таком виде. Более того, существует ровно $2^{h-1}(2^h + h - 3)$ целых положительных чисел, которые нельзя представить в таком виде.
Найдите аналогичные формулы для случая, когда в этом представлении выражение $2^k - 1$ заменено на $2^k + 1$.
- >23. [M22] Докажите, что если h_{s+2} и h_{s+1} взаимно просты, то число перемещений, выполняемых алгоритмом D при просмотре с шагом h_s , есть $O(Nh_{s+2}h_{s+1}/h_s)$. (Указание: см. упр. 21.)
24. [M42] Докажите, что теорема P — наилучшая из возможных теорем в том смысле, что показатель $3/2$ нельзя уменьшить.
- >25. [M22] Сколько существует перестановок множества $\{1, 2, \dots, n\}$, являющихся одновременно 3-упорядоченными и 2-упорядоченными? Каково максимальное число инверсий в такой перестановке? Чему равно суммарное число инверсий во всех таких перестановках?
26. [M35] Может ли 3-, 5- и 7-упорядоченный файл из N элементов содержать более N инверсий?
27. [M50] Предложите эффективный алгоритм построения перестановок, которые требуют максимального числа операций перемещения в алгоритме D при заданных $N, h_t, h_{t-1}, \dots, h_1$.
28. [15] Какая из последовательностей шагов, указанных в табл. 6, наилучшая для программы D с точки зрения суммарного времени работы?
29. [M42] Найдите для $N = 1000$ и различных значений t эмпирические значения h_t, \dots, h_1 , которые, быть может, минимизируют среднее число операций перемещения в алгоритме D в том смысле, что если изменить одну из величин h , не меняя остальных, то среднее число перемещений возрастет.
30. [M23] (В. Пратт.) Покажите, что если множество шагов есть $\{2^p 3^q \mid 2^p 3^q < N\}$, то число просмотров равно примерно $\frac{1}{2}(\log_2 N)(\log_3 N)$ и на каждый просмотр приходится не более $N/2$ операций перемещения. В действительности при любом просмотре, если $K_{j-h} > K_j$, то всегда $K_{j-3h}, K_{j-2h} \geq K_j < K_{j-h} \leq K_{j+h}, K_{j+2h}$, так что можно просто поменять местами K_{j-h} и K_j и увеличить j на $2h$, сэкономив в алгоритме D 2 сравнения. (Указание: см. упр. 25.)
- >31. [25] Напишите MIX-программу для алгоритма сортировки, предложенного В. Праттом (упр. 30). Выразите ее время работы через величины A, B, S, T, N , аналогичные соответствующим величинам в программе D.
32. [10] Каково будет окончательное содержимое связей $L_0 L_1 \dots L_{16}$, если провести до конца сортировку списка вставками в табл. 8?
- >33. [25] Найдите способ усовершенствовать программу L, чтобы ее время работы определялось величиной $5B$, а не $7B$, где B — число инверсий.
34. [M10] Проверьте формулу (10).
35. [18] Предположим, что размер байта машины MIX равен 100 и шестнадцать ключей в табл. 8 равны на самом деле 503 000, 087 000, 512 000, ..., 703 000. Определите время работы программ M и L с этими данными при $M = 4$.

36. [BM16] Если программа выполняется приблизительно за $A/M + B$ единиц времени и использует $C + M$ ячеек памяти, то при каком выборе M достигается оптимальное значение произведения пространства на время?
37. [M25] Пусть $g_n(z)$ — производящая функция для числа инверсий в случайной перестановке n элементов (ср. с формулой (5.1.1-11)). Пусть $g_{NM}(z)$ — соответствующая производящая функция для величины B в программе M . Покажите, что

$$\sum_{N \geq 0} g_{NM}(z) M^N w^N / N! = \left(\sum_{n \geq 0} g_n(z) w^n / n! \right)^M,$$

и воспользуйтесь этой формулой для вывода дисперсии величины B .

38. [BM23] (Р. М. Карп.) Пусть $F(x)$ — некоторая функция распределения вероятностей, причем $F(0) = 0$ и $F(1) = 1$. Докажите, что если ключи K_1, K_2, \dots, K_N независимо выбираются случайным образом из этого распределения и $M = cN$, где c — константа, а $N \rightarrow \infty$, то время работы программы M есть $O(N)$, если только F — достаточно гладкая функция. (Ключ K вставляется в список j , если $\lfloor MK \rfloor = j - 1$; это случается с вероятностью $F(j/M) - F((j-1)/M)$. В тексте рассматривался лишь случай $F(x) = x$, $0 \leq x \leq 1$.)

Упражнения

1. [M25] (Беспорядок в библиотеке.) Небрежные читатели часто ставят книги на полки в библиотеке не на свое место. Один из способов измерить степень беспорядка в библиотеке — посмотреть, какое минимальное количество раз пришлось бы брать книгу с одного места и вставлять ее в другое место до тех пор, пока книги не окажутся расположенными в правильном порядке.

Пусть $\pi = a_1 a_2 \dots a_n$ — перестановка множества $\{1, 2, \dots, n\}$. "Операция удаления-вставки" заменяет π на

$$a_1 a_2 \dots a_{i-1} \dots a_j a_i a_{j+1} \dots a_n$$

или на

$$a_1 \dots a_j a_i a_{j+1} \dots a_{i-1} a_{i+1} \dots a_n$$

при некоторых i и j . Пусть $\text{dis}(\pi)$ — минимальное число операций удаления-вставки, необходимое для упорядочения перестановки π . Можно ли выразить $\text{dis}(\pi)$ через более простые характеристики π ?

2. [40] Проведите эксперименты со следующей модификацией сортировки с убывающим шагом, имеющей целью ускорение "внутреннего цикла": для $s = t, t - 1, \dots, 2$ и для $j = h_s + 1, h_s + 2, \dots, N$ поменять местами $R_{j-h_s} \leftrightarrow R_j$, если $K_{j-h_s} > K_j$. (Таким образом, результат обменов не распространяется; не производится сравнений $K_{j-h_s} : K_{j-2h_s}$. Записи не обязательно будут h_s -отсортированы, но эти предварительные просмотры способствуют сокращению числа инверсий.) Сортировка завершается применением простых вставок.

5.2.2. Обменная сортировка

Мы подошли теперь ко второму из семейств алгоритмов сортировки, упомянутых в самом начале § 5.2, — к методам "обменов" или "транспозиций", предусматривающих систематический обмен местами между элементами пар, в которых нарушается упорядоченность, до тех пор, пока таких пар не останется.

Процесс простых вставок (алгоритм 5.2.1S) можно рассматривать как обменную сортировку: мы берем новую запись R_j и, по существу, меняем местами с соседями слева до тех пор, пока она не займет нужного места. Таким образом, классификация методов сортировки по таким семействам, как "вставки", "обмены", "выбор" и т. д., не всегда четко определена. В этом пункте мы рассмотрим четыре типа методов сортировки, для которых обмены являются основной характеристикой: *обменную сортировку с выбором* ("метод пузырька"), *обменную сортировку со слиянием* (параллельную сортировку Бэтчера), *обменную сортировку с разделением* ("быструю сортировку" Хоара), *поразрядную обменную сортировку*.

Метод пузырька. Пожалуй, наиболее очевидный способ обменной сортировки — это сравнивать K_1 с K_2 , меняя местами R_1 и R_2 , если их ключи не упорядочены, затем проделать то же самое с R_2 и R_3 , R_3 и R_4 и т. д. При выполнении этой последовательности операций записи с большими ключами будут продвигаться вправо, и на самом деле запись с наибольшим ключом займет положение R_N . При многократном выполнении этого процесса соответствующие записи попадут в позиции R_{N-1} , R_{N-2} и т. д., так что в конце концов все записи будут упорядочены.

На рис. 14 показано действие этого метода сортировки на шестнадцати ключах 503 087 512... 703. Файл чисел удобно

Picture: Рис. 14. Сортировка методом пузырька в действии.

представлять не горизонтально, а вертикально, чтобы запись R_N была сверху, а R_1 —снизу. Метод назван "методом пузырька", потому что большие элементы, подобно пузырькам, "всплывают" на соответствующую позицию в противоположность "методу погружения" (т. е. методу простых вставок), в котором элементы погружаются на соответствующий уровень. Метод пузырька известен также под более прозаическими именами, такими, как "обменная сортировка с выбором" или метод "распространения". Нетрудно видеть, что после каждого просмотра файла все записи, начиная с самой последней, которая участвовала в обмене, и выше, должны занять свои окончательные позиции, так что их не нужно проверять при последующих просмотрах. На рис. 14 такого рода продвижения алгоритма отмечены черточками. Заметим, например, что после четвертого просмотра стало известно, что еще пять записей заняли свои окончательные позиции. При последнем, просмотре вообще не было произведено обменов. Теперь, сделав эти наблюдения, мы готовы сформулировать алгоритм.

Алгоритм В. (Метод пузырька.) Записи R_1, \dots, R_N переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$.

- В1** [Начальная установка BOUND.] Установить $\text{BOUND} \leftarrow N$. (BOUND—индекс самого верхнего элемента, о котором еще не известно, занял ли он уже свою окончательную позицию; таким образом, мы отмечаем, что к этому моменту еще ничего не известно.)
- В2** [Цикл по j .] Установить $t \leftarrow 0$. Выполнить шаг **В3** при $j = 1, 2, \dots, \text{BOUND} - 1$. Затем перейти к шагу **В4**. (Если $\text{BOUND} = 1$, то сразу перейти к **В4**.)
- В3** [Сравнение/обмен $R_j : R_{j+1}$]⁸. Если $K_j > K_{j+1}$, то поменять местами $R_j \leftrightarrow R_{j+1}$ и установить $t \leftarrow j$.
- В4** [Были ли обмены?] Если $t = 0$, то завершить работу алгоритма. В противном случае установить $\text{BOUND} \leftarrow t$ и возвратиться к шагу **В2**. ■

Picture: Рис. 15. Блок-схема сортировки методом пузырька.

Программа В. (Метод пузырька.) Как и в предыдущих MIX-программах этой главы, мы предполагаем, что сортируемые элементы находятся в ячейках $\text{INPUT} + 1, \dots, \text{INPUT} + N$; регистры: $\text{rI1} \equiv t$; $\text{rI2} \equiv j$.

START	ENT1	N	1	В1. Начальная установка BOUND. $t \leftarrow N$.
1H	ST1	BOUND (1:2)	A	$\text{BOUND} \leftarrow t$.
	ENT2	1	A	В2. Цикл. по j .
	ENT1	0	A	$t \leftarrow 0$.
	JMP	BOUND	A	Выход, если $j \geq \text{BOUND}$.
3H	LDA	INPUT, 2	C	В3. Сравнение/обмен $R_j : R_{j+1}$.
	CMPA	INPUT+1, 2	C	
	JLE	2F	C	Если $K_j \leq K_{j+1}$, то без обмена.
	LDX	INPUT+1, 2	B	R_{j+1}
	STX	INPUT, 2	B	$\rightarrow R_j$.
	STA	INPUT+1, 2	B	(прежнее R_j) $\rightarrow R_{j+1}$
	ENT1	0, 2	B	$t \leftarrow j$.
9H	INC2	1	C	$j \leftarrow j + 1$.
BOUND	ENTX	-*, 2	A + C	$\text{rX} \leftarrow j - \text{BOUND}$. (Изменяемая инструкция)
	JXN	3B	A + C	$1 \leq j < \text{BOUND}$.
4H	J1P	1B	A	В4. Были ли обмены? К В2 , если $t > 0$.

Анализ метода пузырька. Очень полезно исследовать время работы алгоритма В. Оно определяется тремя величинами: числом просмотров A , числом обменов B и числом сравнений C . Если исходные ключи различны и расположены в случайном порядке, то можно предположить, что они образуют случайную перестановку множества $\{1, 2, \dots, n\}$. Понятие таблицы инверсий (п. 5.1.1) приводит к простому способу описания действия каждого просмотра при сортировке методом пузырька.

⁸ Здесь, как и ранее, двоеточие используется для обозначения оператора сравнения.—Прим. ред.

Теорема I. Пусть $a_1 a_2 \dots a_n$ — перестановка множества $\{1, 2, \dots, n\}$, а $b_1 b_2 \dots b_n$ — соответствующая таблица инверсий. Если в результате очередного просмотра при сортировке методом пузырька (алгоритм В) перестановка $a_1 a_2 \dots a_n$ преобразуется в $a'_1 a'_2 \dots a'_n$, то соответствующая таблица инверсий $b'_1 b'_2 \dots b'_n$ получается из $b_1 b_2 \dots b_n$ уменьшением на единицу каждого нулевого элемента.

Доказательство. Если перед a_i имеется больший элемент, то a_i поменяется местами с наибольшим из предшествующих элементов, так что b_i уменьшится на единицу. С другой стороны, если перед a_i нет элемента, большего a_i , то a_i никогда не поменяется местами с большим элементом, так что b_{a_i} останется 0. ■

Итак, можно разобраться в том, что происходит в процессе сортировки методом пузырька, изучая последовательность таблиц инверсий между просмотрами. Вот как выглядят, например, таблицы инверсий для рис. 14:

Просмотр 1	3	1	8	3	4	5	0	4	0	3	2	2	3	2	1	0
Просмотр 2	2	0	7	2	3	4	0	3	0	2	1	1	2	1	0	0
Просмотр 3	1	0	6	1	2	3	0	2	0	1	0	0	1	0	0	0
	0	0	5	0	1	2	0	1	0	0	0	0	0	0	0	0

(1)

и т. д. Поэтому, если $b_1 b_2 \dots b_n$ — таблица инверсий исходной перестановки, то должны выполняться равенства

$$A = 1 + \max(b_1, b_2, \dots, b_n); \tag{2}$$

$$B = b_1 + b_2 + \dots + b_n; \tag{3}$$

$$C = c_1 + c_2 + \dots + c_n, \tag{4}$$

где c_j — значение BOUND – 1 перед началом j -го просмотра. Используя таблицы инверсий, запишем

$$c_j = \max\{b_i + i \mid b_i \geq j - 1\} - j \tag{5}$$

(см. упр. 5). Следовательно, в примере (1) $A = 9, B = 41, C = 15 + 14 + 13 + 12 + 7 + 5 + 4 + 3 + 2 = 75$. Общее время сортировки на машине MIX для рис. 14 равно $1030u$.

Распределение величины B (суммарное число инверсий в случайной перестановке) нам уже хорошо известно; таким образом, остается проанализировать величины A и C .

Вероятность того, что $A \leq k$, равна произведению $1/n!$ на число таблиц инверсий, не содержащих компонент $\geq k$, именно $k^{n-k}k!$ при $1 \leq k \leq n$. Следовательно, вероятность того, что потребуется ровно k просмотров, равна

$$A_k = \frac{1}{n!} (k^{n-k}k! - (k-1)^{n-k+1}(k-1)!). \tag{6}$$

Теперь можно вычислить среднее значение $\sum k A_k$, производя суммирование по частям, получаем

$$A_{ave} = n + 1 - \sum_{0 \leq k \leq n} \frac{k^{n-k}k!}{n!} = n + 1 + P(n), \tag{7}$$

где $P(n)$ — функция, асимптотическое поведение которой, как было показано в п. 1.2.11, описывается формулой $\sqrt{\pi n/2} - \frac{2}{3} + O(1/\sqrt{n})$. Формула (7) была найдена без доказательства Э. Х. Фрэндом [JACM, 3 (1956), 150]; доказательство в своей докторской диссертации привел Говард Б. Демут [(Stanford University: October, 1956), 64–68]. Стандартное отклонение величины A см. в упр. 7.

Суммарное число сравнений C исследовать несколько сложнее, и мы рассмотрим только C_{ave} . Пусть $f_j(k)$ — число таких таблиц инверсий $b_1 \dots b_n$ (n фиксировано), что при $1 \leq i \leq n$ либо $b_i < j - 1$, либо $b_i + i - j \leq k$; тогда

$$f_j(k) = (j+k)!(j-1)^{n-j-k} \quad \text{при } 0 \leq k \leq n-j. \tag{8}$$

(См. упр. 8.) Среднее значение c_j в (5) равно $\sum k(f_j(k) - f_j(k-1))/n!$; суммируя по частям, а затем по j , получаем формулу

$$C_{ave} = \binom{n+1}{2} - \frac{1}{n!} \sum_{\substack{1 \leq j \leq n \\ 0 \leq k \leq n-j}} f_j(k) = \binom{n+1}{2} - \frac{1}{n!} \sum_{0 \leq r < s \leq n} s!r^{n-s}. \tag{9}$$

Определить асимптотическое значение здесь не так просто, и мы вернемся к нему в конце этого пункта.

Чтобы подвести итог нашего анализа метода пузырька, запишем формулы, выведенные выше (а также ниже), следующим образом:

$$A = (\min 1, \text{ave } N - \sqrt{\pi N/2} + O(1), \max N); \quad (10)$$

$$B = (\min 0, \text{ave } \frac{1}{4}(N^2 - N), \max \frac{1}{2}(N^2 - N)); \quad (11)$$

$$C = \left(\min N - 1, \text{ave } \frac{1}{2}(N^2 - N \ln N - (\gamma + \ln 2 - 1)N) + O(\sqrt{N}), \max \frac{1}{2}(N^2 - N) \right). \quad (12)$$

Во всех случаях минимум достигается, когда исходный файл уже упорядочен, а максимум—когда записи расположены в обратном порядке; таким образом, время работы для машины MIX равно $8A + 7B + 8C + 1 = (\min 8N + 1, \text{ave } 5.75N^2 + O(N \ln N), \max 7.5N^2 + 0.5N + 1)$.

Усовершенствования метода пузырька. Мы потратили много усилий на анализ метода пузырька, и, хотя способы, применявшиеся при вычислениях, поучительны, результаты разочаровывают, поскольку они говорят о том, что метод пузырька вовсе не так уж хорош. По сравнению с простыми вставками (алгоритм 5.2.1S) метод пузырька описывается более сложной программой и требует примерно в 2 раза больше времени!

Можно предложить несколько путей улучшения метода пузырька. Например, на рис. 14 первое сравнение в 4-м просмотре излишне, так же как и первые два сравнения в 5-м просмотре и первые три в 6-м и 7-м просмотрах. Заметим также, что за один просмотр элемент не может переместиться более чем на одну позицию влево; так что если наименьший элемент вначале находился в правом конце, то мы вынуждены выполнить максимальное число сравнений. Это наводит на мысль о "шейкер-сортировке", когда файл просматривается попеременно в обоих направлениях (рис. 16). При таком подходе среднее число сравнений несколько сокращается. К. Э. Айверсон [A Programming Language (Wiley, 1962), 218–219] сделал интересное в этом отношении наблюдение: если j —такой индекс, что R_j и R_{j+1} не меняются местами при двух последовательных просмотрах

Picture: Рис. 16. "Шейкер-сортировка".

в противоположных направлениях, то записи R_j и R_{j+1} должны занимать свои окончательные позиции, и они могут не участвовать в последующих сравнениях. Например, просматривая перестановку 4 3 2 1 8 6 9 7 5 слева направо, получаем 3 2 1 4 6 8 7 5 9: записи R_4 и R_5 не поменялись местами. При просмотре последней перестановки справа налево R_4 все еще меньше (новой) записи R_5 ; следовательно, можно сразу же сделать вывод о том, что записи R_4 и R_5 могут и не участвовать ни в одном из последующих, сравнений.

Однако ни одно из этих усовершенствований не приводит к лучшему алгоритму, чем алгоритм сортировки простыми вставками, а мы уже знаем, что даже он не годится при больших N . Другая идея состоит в том, чтобы избежать большинства обменов. Так как большая часть элементов во время обменов просто сдвигается на один шаг влево, то можно было бы достичь того же эффекта, иначе рассматривая массив—сместив начало отсчета! Но полученный алгоритм не превосходит метода простого выбора (алгоритм 5.2.3S), который мы изучим позже.

Короче говоря, метод пузырька, кажется, не обладает никакими достоинствами, за которые его можно было бы порекомендовать, если не считать легко запоминающегося названия и интересных теоретических задач, к которым он приводит.

Параллельная сортировка Бэтчера. Если мы хотим получить алгоритм обменной сортировки, время работы которого имеет порядок, меньший N^2 , то необходимо подбирать для сравнений некоторые пары несоседних ключей (K_i, K_j); иначе придется

Picture: Рис. 17. Алгоритм М.

выполнить столько обменов, сколько инверсий имеется в исходной перестановке, а среднее число инверсий равно $(N^2 - N)/4$. В 1964 г. К. Э. Бэтчер [Proc. AFIPS Spring Joint Computer Conference, 32 (1968), 307–314] открыл один искусный способ запрограммировать последовательность сравнений, предназначенную для отыскания возможных обменов. Его метод далеко не очевиден. В самом деле, только для того, чтобы показать его справедливость, требуется весьма сложное доказательство, так как выполняется относительно мало сравнений. Мы обсудим два доказательства, одно в этом пункте и одно в п. 5.3.4.

Схема сортировки Бэтчера несколько напоминает сортировку Шелла, но сравнения выполняются по-новому, так что распространение обменов не обязательно. Так, можно сравнить табл. 1 с табл. 5.2.1-3. Сортировка Бэтчера действует как 8-сортировка, 4-сортировка, 2-сортировка и 1-сортировка, но сравнения не перекрываются. Так как в алгоритме Бэтчера, по существу, происходит слияние пар отсортированных подпоследовательностей, то его можно назвать "обменной сортировкой со слиянием".

Алгоритм М. (Обменная сортировка со слиянием.) Записи R_1, \dots, R_N переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Предполагается, что $N \geq 2$.

M1 [Начальная установка p .] Установить $p \leftarrow 2^{t-1}$, где $t = \lceil \log_2 N \rceil$ — наименьшее целое число, такое, что $2^t \geq N$. (Шаги **M2**, ..., **M5** будут выполняться с $p = 2^{t-1}, 2^{t-2}, \dots, 1$.)

M2 [Начальная установка q, r, d .] Установить $q \leftarrow 2^{t-1}, r \leftarrow 0, d \leftarrow p$.

M3 [Цикл по t .] Для всех t , таких, что $0 \leq i < N - d$ и $i \wedge p = r$, выполнить шаг **M4**. Затем перейти к шагу **M5**. (Здесь через $i \wedge p$ обозначена операция "логическое и" над двоичными представлениями чисел i и p ; все биты результата равны 0, кроме тех, для которых в соответствующих позициях i и p находятся единичные биты. Так, $13 \wedge 21 = (1101)_2 \wedge (10101)_2 = (00101)_2 = 5$. К этому моменту d — нечетное кратное p (т. е. частное от деления d на p нечетно), а p — степень двойки, так что $i \wedge p \neq (i + d) \wedge p$; отсюда следует, что действия шага **M4** можно выполнять при всех нужных значениях i в любом порядке или даже одновременно.)

M4 [Сравнение/обмен $R_{i+1} : R_{i+d+1}$.] Если $K_{i+1} > K_{i+d+1}$, то поменять местами $R_{i+1} \leftrightarrow R_{i+d+1}$.

M5 [Цикл по q .] Если $q \neq p$, то установить $d \leftarrow q - p, q \leftarrow q/2, r \leftarrow p$ и возвратиться к шагу **M3**.

M6 [Цикл по p .] (К этому моменту перестановка $K_1 K_2 \dots K_N$ будет p -упорядочена.) Установить $p \leftarrow \lfloor p/2 \rfloor$. Если $p > 0$, то возвратиться к шагу **M2**. ■

В табл. 1 этот метод проиллюстрирован при $N = 16$. Заметим, что алгоритм сортирует N элементов, по существу, путем независимой сортировки подфайлов R_1, R_3, R_5, \dots и R_2, R_4, R_6, \dots , после чего выполняются шаги **M2**, ..., **M5** с $p = 1$, чтобы слить две отсортированные последовательности.

Чтобы доказать, что магическая последовательность сравнений/обменов, описанная в алгоритме М, действительно сортирует любой файл $R_1 R_2 \dots R_N$, мы должны показать только, что в результате выполнения шагов от **M2** до **M5** с $p = 1$ будет слит любой 2-упорядоченный файл $R_1 R_2 \dots R_N$. Для этой цели можно воспользоваться методом решеточных диаграмм из п. 5.2.1 (см. рис. 11); каждая 2-упорядоченная перестановка множества $\{1, 2, \dots, N\}$ соответствует на решетке единственному пути из вершины $(0, 0)$ в $(\lceil N/2 \rceil, \lfloor N/2 \rfloor)$. На рис. 18(a) показан

Picture: Таблица 1. p.139

пример при $N = 16$, соответствующий перестановке 1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12. Действие шага **M3** при $p = 1, q = 2^{t-1}, r = 0, d = 1$ состоит в сравнении (и, возможно, обмене) записей $R_1 : R_2, R_3 : R_4$ и т. д. Этой операции соответствует простое преобразование решеточного пути: "перегиб" относительно диагонали, если необходимо, так, чтобы путь нигде не проходил выше диагонали. (См. рис. 18(b) и доказательство в упр. 10.) Действие последующих повторений шага **M3** с $p = r = 1$ и $d = 2^{t-1} - 1, 2^{t-2} - 1, \dots, 1$ состоит в сравнении/обмене записей $R_2 : R_{2+d}, R_4 : R_{4+d}$ и т. д., и опять имеется простая геометрическая интерпретация: путь "перегибается" относительно прямой, расположенной на $(d+l)/2$ единиц ниже диагонали (см. рис. 18(c) и (d)). В конце концов приходим к пути, изображенному на рис. 18(e), который соответствует полностью отсортированной перестановке. На этом "геометрическое доказательство" справедливости алгоритма Бэтчера завершается; этот алгоритм можно было бы назвать сортировкой посредством перегибов!

МIX-программа для алгоритма М приведена в упр. 12. К сожалению, количество вспомогательных операций, необходимых для управления последовательностью сравнений, весьма велико, так что программа менее эффективна, чем другие методы, которые мы разбирали. Однако алгоритм обладает одним важным компенсирующим качеством: все сравнения/обмены, определяемые данной итерацией шага **M3**, можно выполнять *одновременно* на ЭВМ или логических схемах, которые допускают параллельные

Picture: Рис. 18. Геометрическая интерпретация метода Бэтчера, $N = 16$.

вычисления. С такими параллельными операциями сортировка выполняется за $\frac{1}{2} \lceil \log_2 N \rceil (\lceil \log_2 N \rceil + 1)$ шагов, и это один из самых быстрых известных общих методов. Например, *1024 элемента можно отсортировать методом Бэтчера всего за 55 параллельных шагов*. Его ближайшим соперником является метод Пратта (см. упр. 5.2.1–30), который затрачивает 40 или 73 шага в зависимости от

того, как считать: если мы готовы допустить перекрытие сравнений до тех пор, пока не потребуется выполнять перекрывающиеся обмены, то для сортировки 1024 элементов методом Пратта требуется всего 40 циклов сравнения/обмена. Дальнейшие пояснения см. в п. 5.3.4.

”Быстрая сортировка”. В методе Бэтчера последовательность сравнений предопределена: каждый раз сравниваются одни и те же пары ключей независимо от того, что мы могли узнать о файле из предыдущих сравнений. Это утверждение в большой мере справедливо и применительно к методу пузырька, хотя алгоритм В и использует в ограниченной степени полученные сведения, с тем чтобы сократить количество работы в правом конце файла. Обратимся теперь к совсем иной стратегии, при которой используется результат каждого сравнения, чтобы определить, какие ключи сравнивать следующими. Такая стратегия не годится для параллельных вычислений, но она может оказаться плодотворной для вычислительных машин, работающих последовательно.

Итак, рассмотрим следующую схему сравнений/обменов. Имеются два указателя i и j , причем вначале $i = l$, а $j = N$. Сравним $K_i : K_j$, и если обмен не требуется, то уменьшим j на единицу и повторим этот процесс. После первого обмена увеличим i на единицу и будем продолжать сравнения, увеличивая i , пока не произойдет еще один обмен. Тогда опять уменьшим j и т. д.; будем ”сжигать свечку с обоих концов”, пока не станет $i = j$. Посмотрим, например, что произойдет с нашим файлом из шестнадцати чисел:

Дано:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	
																	уменьшить j
1-й обмен	154	087	512	061	908	170	897	275	653	426	503	509	612	677	765	703	
																	увеличить i
2-й обмен	154	087	503	061	908	170	897	275	653	426	512	509	612	677	765	703	
																	уменьшить j
3-й обмен	154	087	426	061	908	170	897	275	653	503	512	509	612	677	765	703	
																	увеличить i
4-й обмен	154	087	426	061	503	170	897	275	653	908	512	509	612	677	765	703	
																	уменьшить j
5-й обмен	154	087	426	061	275	170	897	503	653	908	512	509	612	677	765	703	
																	увеличить i
6-й обмен	154	087	426	061	275	170	503	897	653	808	512	509	612	677	765	703	
																	уменьшить j

(Чтобы выявить состояние i и j , ключи K_i и K_j напечатаны жирным шрифтом.) Заметим, что в каждом сравнении этого примера участвует ключ **503**; вообще в каждом сравнении будет участвовать исходный ключ K_1 , потому что он будет продолжать обмениваться местами с другими ключами каждый раз, когда мы переключаем направление. К моменту, когда $i = j$, исходная запись R_1 займет свою окончательную позицию, потому что, как нетрудно видеть, слева от нее не будет больших ключей, а справа — меньших. Исходный файл окажется разделен таким образом, что первоначальная задача сведется к двум более простым: сортировке файла $R_1 \dots R_{i-1}$ и (независимо) сортировке файла $R_{i+1} \dots R_N$. К каждому из этих подфайлов можно применить тот же самый метод.

Таблица 2

”Быстрая сортировка”

	(l, r)	Стек
[503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703]	(1, 16)	—
[154 087 426 061 275 170] 503 [897 653 908 512 509 612 677 765 703]	(1, 6)	(8, 16)
[061 087] 154 [426 275 170] 503 [897 653 908 512 509 612 677 765 703]	(1, 2)	(4, 6)(8, 16)
061 087 154 [426 275 170] 503 [897 653 908 512 509 612 677 765 703]	(4, 6)	(8, 16)
061 087 154 [170 275] 426 503 [897 653 908 512 509 612 677 765 703]	(4, 5)	(8, 16)
061 087 154 170 275 426 503 [897 653 908 512 509 612 677 765 703]	(8, 16)	—
061 087 154 170 275 426 503 [703 653 765 512 509 612 677] 897 908	(8, 14)	—
061 087 154 170 275 426 503 [677 653 612 512 509] 703 765 897 908	(8, 12)	—
061 087 154 170 275 426 503 [509 653 612 512] 677 703 765 897 908	(8, 11)	—
061 087 154 170 275 426 503 509 [653 612 512] 677 703 765 897 908	(9, 11)	—
061 087 154 170 275 426 503 509 [512 612] 653 677 703 765 897 908	(9, 10)	—
061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908	—	—

В табл. 2 показано, как выбранный нами для примеров файл полностью сортируется при помощи этого метода за 11 стадий. В скобки заключены подфайлы, которые еще предстоит отсортировать; в машине эти подфайлы можно представлять двумя переменными l и r (границы рассматриваемого в данный момент файла) и стеком дополнительных пар (l_k, r_k) . Каждый раз, когда файл подразделяется,

мы помещаем в стек *большой* из подфайлов и начинаем обрабатывать оставшийся, и так до тех пор, пока не придем к тривиально коротким файлам; как показано в упр. 20, такая процедура гарантирует, что в стеке никогда не будет находиться более, чем примерно $\log_2 N$ элементов.

Только что описанную процедуру сортировки можно назвать обменной сортировкой *с разделением*; она принадлежит Ч. Э. Р. Хоару, интереснейшая статья которого [Comp. J., 5 (1962), 10–15]—одно из наиболее исчерпывающих из когда-либо опубликованных сообщений об этом методе. Хоар окрестил свой метод "quicksort" ("быстрая сортировка"), и это название вполне соответствует действительности, так как, например, весь процесс сортировки, показанный в табл. 2, требует всего 48 сравнений (меньше любого другого встречавшегося уже метода, за исключением бинарных вставок, требующих 47 сравнений).

Picture: Рис 19. Обменная сортировка с разделением ("быстрая сортировка").

Во всех сравнениях на данной стадии участвует один и тот же ключ, так что его можно хранить в регистре. Кроме того, количество перемещений данных весьма умеренно: при вычислениях табл. 2 произведено всего 17 обменов, причем большинство из них—просто "полуобмены" (простые пересылки), так как один из ключей все время остается в регистре, и его не нужно записывать до самого конца стадии.

Вспомогательные операции (требуемые для управления стеком и переменными i, j) не сложны, но из-за них процедура быстрой сортировки посредством разделений пригодна в основном для больших значений N ; поэтому короткие подфайлы желательно сортировать особым образом, как это делается в следующем алгоритме.

Алгоритм Q. (Обменная сортировка с разделением.) Записи R_1, \dots, R_N переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Нужен вспомогательный стек для хранения не более чем $\log_2 N$ элементов. Этот алгоритм соответствует процедуре "быстрой сортировки" посредством разделений, приведенной выше, с небольшими изменениями в целях повышения эффективности:

- a) Предполагается наличие искусственных ключей $K_0 = -\infty$ и $K_{N+1} = +\infty$, таких, что

$$K_0 \leq K_i \leq K_{N+1} \quad \text{при } 1 \leq i \leq N. \quad (13)$$

(Равенство допускается.)

- b) Подфайлы, состоящие из M и менее элементов, сортируются простыми вставками, где $M \geq 1$ —параметр, который выбирается, как описано ниже.
- c) На некоторых стадиях делается одно или два дополнительных сравнения (допускается перекрытие указателей i, j), чтобы основные циклы сравнения выполнялись настолько быстро, насколько это возможно.
- d) Записи с одинаковыми ключами меняются местами, хотя это не является строго необходимым. (Эта идея, принадлежащая Р. К. Синглтону, способствует разделению подфайлов почти пополам, если имеются равные ключи; см. упр. 18.)

Q1 [Начальная установка.] Опустошить стек и установить $l \leftarrow 1; r \leftarrow N$.

Q2 [Начать новую стадию.] (Нам хотелось бы отсортировать файл R_l, \dots, R_r ; из самого существа алгоритма вытекает, что $r \geq l - 1, K_{l-1} \leq K_i \leq K_{r+1}$ при $l \leq i \leq r$.) Если $r - l < M$, то перейти к шагу Q8. В противном случае установить $i \leftarrow l, j \leftarrow r, K \leftarrow K_l, R \leftarrow R_l$. (Обсуждение наилучшего выбора R и K приведено ниже.)

Q3 [Сравнить $K : K_j$.] Если $K < K_j$, то уменьшить j на 1 и повторить этот шаг.

Q4 [Переслать R на место R_i .] (К этому моменту K_i —ключ $\geq K$, находящийся не на своем месте, и $K \geq K_j$.) Если $j \leq i$, то установить $R_i \leftarrow R$ и перейти к шагу Q7. В противном случае установить $R_i \leftarrow R_j$ и увеличить i на 1.

Q5 [Сравнить $K_i : K$.] Если $K_i < K$, то увеличить i на 1 и повторить этот шаг.

Q6 [Переслать R на место R_j .] (К этому моменту K_j —ключ $\leq K$, находящийся не на своем месте, и $K \leq K_i$.) Если $j \leq i$, то установить $R_j \leftarrow R$ и $i \leftarrow j$. В противном случае установить $R_j \leftarrow R_i$, уменьшить j на 1 и перейти к шагу Q3.

Q7 [Поместить в стек.] (Теперь подфайл $R_l \dots R_i \dots R_r$ разделен так, что $K_k \leq K_i$ при $l \leq k \leq i$ и $K_i \leq K_k$ при $i \leq k \leq r$.) Если $r - i \geq i - l$, то поместить в стек $(i + 1, r)$ и установить $r \leftarrow i - 1$. В противном случае поместить в стек $(l, i - 1)$ и установить $l \leftarrow i + 1$. (Каждый элемент стека (a, b) —это заявка на сортировку подфайла $R_a \dots R_b$ в одной из последующих стадий.) Вернуться к шагу Q2.

Q8 [Сортировка простыми вставками.] Для $j = l + 1, l + 2, \dots$ до $j > r$ выполнять следующие операции: установить $K \leftarrow K_j, R \leftarrow R_j, i \leftarrow j - 1$; затем установить $R_{i+1} \leftarrow R_i, i \leftarrow i - 1$ нуль или более раз

до тех пор, пока не выполнится условие $K_i \leq K$; затем установить $R_{i+1} \leftarrow R$. (Это, по существу, алгоритм 5.2.1S, примененный к подфайлу из M или менее элементов.)

Q9 [Взять из стека.] Если стек пуст, то сортировка завершена; в противном случае взять верхний элемент стека (l', r') , установить $l \leftarrow l', r \leftarrow r'$ и возвратиться к шагу **Q2**. ■

Соответствующая MIX-программа довольно велика, но не сложна; на самом деле большая часть команд относится к шагу **Q7**, в котором проводятся весьма простые манипуляции с переменными.

Программа Q. (Обменная сортировка с разделением.) Записи, которые предстоит отсортировать, находятся в ячейках $INPUT + 1, \dots, INPUT + N$; предполагается, что в ячейках $INPUT$ и $INPUT + N + 1$ содержатся значения, соответственно минимально, и максимально допустимые в машине MIX. Стек располагается в ячейках $STACK + 1, STACK + 2, \dots$; точное число ячеек, которое необходимо отвести под стек, обсуждается в упр. 20. Значения регистров: $rI1 \equiv l, rI2 \equiv r, rI3 \equiv i, rI4 \equiv j, rI6 \equiv$ размер стека, $rA \equiv K \equiv R$.

A	EQU	2:3		Первая компонента элемента стека.
B	EQU	4:5		Вторая компонента элемента стека.
START	ENT1	1	1	Q1. Начальная установка. $l \leftarrow 1$.
	ENT2	N	1	$r \leftarrow N$.
	ENT6	0	1	Опустошить стек.
2H	ENTX	0,2	$2A + 1$	Q2. Начать новую стадию.
	DECX	M,1	$2A + 1$	$rX \leftarrow r - l - M$.
	JXN	8F	$2A + 1$	К шагу Q8 , если размер подфайла $\leq M$.
	ENT3	0,1	A	$i \leftarrow l$.
	ENT4	0,2	A	$j \leftarrow r$.
	LDA	INPUT,3	A	$K \leftarrow K_i$.
	JMP	3F	A	К шагу Q3 .
0H	LDX	INPUT,3	B	
	STX	INPUT,4	B	$R_j \leftarrow R_i$.
	DEC4	1	$C' - A$	$j \leftarrow j - 1$.
3H	CMPA	INPUT,4	C'	Q3. Сравнить $K : K_j$.
	JL	*-2	C'	Если $<$, то уменьшить j и повторить.
4H	ENTX	0,3	$B + A$	Q4. Переслать R на место R_i .
	DECX	0,4	$B + A$	
	JXNN	7F	$B + A$	К шагу Q7 , если $i \geq j$.
	LDX	INPUT,4	$B + X$	
	STX	INPUT,3	$B + X$	$R_i \leftarrow R_j$.
	INC3	1	C''	$i \leftarrow i + 1$.
5H	CMPA	INPUT,3	C''	Q5. Сравнить $K_i : K$.
	JG	*-2	C''	Если $<$, то увеличить i и повторить.
6H	ENTX	0,3	$B + X$	Q6. Переслать R на место R_i .
	DECX	0,4	$B + X$	
	JXN	0B	$B + X$	К шагу Q3 , если $i < j$.
	ENT3	0,4	X	$i \leftarrow j$.
7H	STA	INPUT,3	A	Q7. Поместить в стек. $R_i \leftarrow R$.
	ENTA	0,2	A	
	DECA	0,3	A	
	DECA	0,3	A	
	INCA	0,1	A	
	INC6	1	A	$rI6 \leftarrow rI6 + 1$.
	JANN	1F	A	Переход, если $r - i \geq i - l$.
	ST1	STACK,6(A)	A'	
	DEC3	1	A'	
	ST3	STACK,6(B)	A'	$(l, i - 1) \Rightarrow$ стек.
	ENT1	2,3	A'	$l \leftarrow i + 1$.
	JMP	2B	A'	К шагу Q2 .
1H	INC3	1	$A - A'$	
	ST3	STACK,6(A)	$A - A'$	
	ST2	STACK,6(B)	$A - A'$	$(i + 1, r) \Rightarrow$ стек.
	ENT2	-2,3	$A - A'$	$r \leftarrow i - 1$.

	JMP	2B	$A - A'$	К шагу Q2.
8H	DEC2	0, 1	$A + 1$	Q8. Сортировка простыми вставками.
	J2NP	9F	$A + 1$	К шагу Q9, если $r \leq l$.
	ENT4	1, 1	$A + 1 - L$	$j \leftarrow l + 1$.
1H	LDA	INPUT, 4	D	$R \leftarrow R_j$.
	ENT3	-1, 4	D	$i \leftarrow j - 1$.
	JMP	*+4	D	Переход и цикл.
	LDX	INPUT, 3	E	
	STX	INPUT+1, 3	E	$R_{i+1} \leftarrow R_i$.
	DEC3	1	E	$i \leftarrow i - 1$.
	CMPA	INPUT, 3	$D + E$	
	JL	*-4	$D + E$	Переход, если $K < K_i$.
	STA	INPUT+1, 3	D	$R_{i+1} \leftarrow R$.
	INC4	1	D	$j \leftarrow j + 1$.
	DEC2	1	D	
	J2P	1B	D	Повторить $r - l$ раз.
9H	LD1	STACK, 6 (A)	$A + 1$	Q9. Взять из стека.
	LD2	STACK, 6 (B)	$A + 1$	$(l, r) \leftarrow$ стек.
	DEC6	1	$A + 1$	$rI6 \leftarrow rI6 - 1$.
	J6NN	2B	$A + 1$	К шагу Q2, если стек не был пуст.

Анализ "быстрой сортировки". Информацию о времени выполнения, приведенную вместе с программой Q, нетрудно вывести из закона Кирхгофа, или закона сохранения (см. п. 1.3.3), и из того факта, что все помещенное в стек в конце концов оттуда извлекается. Общее время работы зависит от следующих величин:

$$\begin{aligned}
 A &= \text{число стадий, на которых данный подфайл содержит более } M \text{ элементов;} \\
 B &= \text{число присваиваний } R_j \leftarrow R_i \text{ в шаге Q6;} \\
 C' &= \text{число выполнений шага Q3;} \\
 C'' &= \text{число выполнений шага Q5;} \\
 D &= \text{число присваиваний } R \leftarrow R_j \text{ в шаге Q8;} \\
 E &= \text{число присваиваний } R_{i+1} \leftarrow R_i \text{ в шаге Q8;} \\
 L &= \text{число случаев, когда в шаге Q8 } r \leq l; \\
 X &= \text{число операций } i \leftarrow j \text{ в шаге Q6.}
 \end{aligned} \tag{14}$$

Мы не будем анализировать величины C' и C'' независимо, а рассмотрим лишь

$$C = C' + C'' = \text{общее число сравнений,} \tag{15}$$

так как время работы для C' и C'' одинаково. Изучив семь величин: A, B, C, D, E, L и X , мы сможем разумно выбрать параметр M , которым определяется "порог" между сортировкой простыми вставками и разделением. (Если читатель не математик, он может пропустить все вплоть до формул (25).)

Как и при анализе большинства других алгоритмов в этой главе, мы будем предполагать, что сортируемые ключи различны; в упр. 18 показано, что наличие равных ключей не снижает существенно эффективности алгоритма Q; в действительности присутствие равных ключей, по-видимому, даже способствует ее увеличению. Так как метод зависит только от относительного расположения ключей, то можно предполагать также, что это просто числа $\{1, 2, \dots, N\}$, расположенные в некотором порядке.

Когда мы впервые достигаем шага Q2, сортировка сводится к простым вставкам, если $N \leq M$, а это мы уже изучили; поэтому предположим, что $N > M$. Необходимо лишь разобраться в вычислениях, которые в первый раз приводят к шагу Q7; нетрудно видеть, что при разделении записи в обоих подфайлах $R_1 \dots R_{i-1}$ и $R_{i+1} \dots R_N$ будут расположены в случайном порядке, если только записи исходного файла были расположены в случайном порядке. Значит, вклад последующих вычислений можно определить, применив индукцию по N .

Пусть s — значение первого ключа K_1 , и предположим, что ровно t из ключей K_1, \dots, K_s превосходят s . Пусть

$$h = \begin{cases} 1, & \text{если } K_s < s; \\ 0, & \text{если } K_s \geq s. \end{cases} \tag{16}$$

Иначе говоря, $h = 1$, если ключ, который вначале занимал позицию s (куда попадет ключ K_1), будет перемещен влево. Напомним, что сортируемые ключи являются целыми числами из множества $\{1, 2, \dots, N\}$.

Если $s = 1$, то нетрудно видеть, что произойдет во время первой стадии разделения. Шаг Q3 выполнится N раз, после чего шаг Q4 приведет нас к Q7. Таким образом, первая стадия в этом случае даст такие вклады: $A = 1, B = 0, C = N, X = 0$. Аналогичные, но несколько более сложные рассуждения для случая $s > 1$ (см. упр. 21) показывают, что вкладами первой стадии в суммарное время выполнения в общем случае будут

$$A = 1, \quad B = t, \quad C = N + 1 - \delta_{s1}, \quad X = h \quad \text{при } 1 < s \leq N. \quad (17)$$

К этому необходимо добавить вклады последующих стадий, во время которых упорядочиваются подфайлы соответственно из $s - 1$ и $N - S$ элементов.

Если предположить, что в исходном файле записи располагались в случайном порядке, то можно выписать формулы, которыми определяются производящие функции для распределений вероятностей величин A, B, \dots, X (см. упр. 22). Но для простоты мы изучим здесь только *средние значения* этих величин A_N, B_N, \dots, X_N как функции от N . Рассмотрим, например, среднее число сравнений C_N , выполненных в процессе разделения. Если $N \leq M$, то $C_N = 0$. При $N > M$, так как любое данное значение s встречается с вероятностью $1/N$,

$$\begin{aligned} C_N &= \frac{1}{N} \sum_{1 \leq s \leq N} (N + 1 - \delta_{s1} + C_{s-1} + C_{N-s}) = \\ &= N + 1 - \frac{1}{N} + \frac{2}{N} \sum_{0 \leq k < N} C_k. \end{aligned} \quad (18)$$

Аналогичные формулы имеют место и для остальных величин A_N, B_N, \dots, X_N (см. упр. 23).

Существует простой способ решения рекуррентных соотношений вида

$$x_n = f_n + \frac{2}{n} \sum_{0 \leq k < n} x_k \quad \text{при } n \geq m. \quad (19)$$

На первом шаге освобождаются от знака суммирования: поскольку

$$\begin{aligned} (n+1)x_{n+1} &= (n+1)f_{n+1} + 2 \sum_{0 \leq k \leq n} x_k, \\ nx_n &= nf_n + 2 \sum_{0 \leq k < n} x_k, \end{aligned}$$

то можно вычесть второе равенство из первого, получив

$$(n+1)x_{n+1} - nx_n = g_n + 2x_n, \quad \text{где } g_n = (n+1)f_{n+1} - nf_n.$$

Теперь рекуррентная зависимость принимает гораздо более простой вид:

$$(n+1)x_{n+1} = (n+2)x_n + g_n \quad \text{при } n \geq m. \quad (20)$$

Любое рекуррентное соотношение общего вида

$$a_n x_{n+1} = b_n x_n + g_n \quad (21)$$

можно свести к простой сумме, если умножить обе части на суммирующий множитель $a_0 a_1 \dots a_{n-1} / b_0 b_1 \dots b_n$. Получаем

$$y_{n+1} = y_n + c_n, \quad \text{где } y_n = \frac{a_0 \dots a_{n-1}}{b_0 \dots b_{n-1}} x_n, \quad c_n = \frac{a_0 \dots a_{n-1}}{b_0 \dots b_n} g_n. \quad (22)$$

В случае (20) суммирующий множитель равен просто $n! / (n+2)! = 1 / (n+1)(n+2)$. Таким образом, находим, что простое соотношение

$$\frac{x_{n+1}}{n+2} = \frac{x_n}{n+1} + \frac{(n+1)f_{n+1} - nf_n}{(n+1)(n+2)}, \quad n \geq m, \quad (23)$$

является следствием соотношения (19).

Если, например, положить $f_n = 1/n$, то получится неожиданный результат $x_n/(n+1) = x_m/(m+1)$ при любом $n \geq m$. Если положить $f_n = n+1$, то получится

$$\begin{aligned} x_n/(n+1) &= 2/(n+1) + 2/n + \dots + 2/(m+2) + x_m/(m+1) = \\ &= 2(H_{n+1} - H_{m+1}) + x_m/(m+1) \end{aligned}$$

при любом $n \geq m$. Комбинация этих двух решений и подстановка $m = M+1$ и $x_n = 0$ при $n \leq M$ дают формулу

$$\begin{aligned} C_N &= (N+1) \left(2H_{N+1} - 2H_{M+2} + 1 - \frac{1}{(M+1)(M+2)} \right) \approx \\ &\approx 2(N+1) \ln \left(\frac{N+1}{M+2} \right) \quad \text{при } N > M. \end{aligned} \quad (24)$$

В п. 6.2.2 мы докажем, что стандартное отклонение величины C_N асимптотически равно $\sqrt{(21-2\pi^2)}/3N$; это довольно мало по сравнению с (24).

Остальные величины можно найти аналогичным способом (см. упр. 23); имеем

$$\begin{aligned} A_N &= 2(N+1)/(M+2) - 1, \\ B_N &= \frac{1}{6}(N+1) \left(2H_{N+1} - 2H_{M+2} + 1 - \frac{6}{M+2} \right) + \frac{1}{2}, \\ D_N &= (N+1)M(M-1)/(M+2)(M+1), \\ E_N &= \frac{1}{6}(N+1)M(M-1)/(M+2), \\ L_N &= 4(N+1)/(M+2)(M+1), \\ X_N &= (N+1)/(M+2) - \frac{1}{2} \quad \text{при } N > M. \end{aligned} \quad (25)$$

Приведенное выше обсуждение показывает, что можно произвести точный анализ среднего времени выполнения весьма сложной программы, используя методы, которые мы ранее применяли лишь к более простым случаям.

Чтобы определить "наилучшее" значение M для конкретной машины, можно воспользоваться формулами (24) и (25). Программа Q для машины MIX требует $37A + 14B + 4C + 12D + 8E - L + 8X + 15$ единиц времени; это равно в среднем $\frac{1}{3}(38(N+1)H_N + (N+1)f(M)) - 19$ единиц при $N > M$, где

$$f(M) = 4M + 38H_{M+2} + 43 + \frac{84}{M+2} + \frac{48}{(M+2)(M+1)}. \quad (26)$$

Мы хотим выбрать такое значение M , при котором функция $f(M)$ достигает минимума. В данном случае

$$f(M) - f(M-1) = 4 - \frac{38}{M+2} - \frac{84}{(M+2)(M+1)} - \frac{96}{(M+2)(M+1)M},$$

и требуется найти такое значение M , чтобы $f(M) - f(M-1) \leq 0$, $f(M+1) - f(M) \geq 0$; решение $M = 9$ найти нетрудно. Если $M = 9$, то при больших N среднее время выполнения программы Q равно приблизительно $12.67(N+1) \ln N - 1.92N - 14.59$.

Таким образом, программа Q работает в среднем довольно быстро; следует, кроме того, учесть, что она требует очень мало памяти. Но каков *наихудший* случай для алгоритма Q? Существуют ли какие-нибудь исходные файлы, обрабатывать которые этим алгоритмом не эффективно? Ответ несколько обескураживает: если исходный файл уже упорядочен, а именно $K_1 < K_2 < \dots < K_N$, то каждая операция "разделения" становится почти бесполезной, так как подфайл сводится к одному элементу! Значит, в этой ситуации (которая должна быть самой простой для сортировки) быстрая сортировка превращается в отнюдь не быструю. Время ее работы становится пропорциональным N^2 , а не $N \log N$ (см. упр. 25). В отличие от других алгоритмов сортировки, которые нам встречались, алгоритм Q предпочитает неупорядоченные файлы!

В упомянутой статье Хоара предложены два способа поправить ситуацию, основывающихся на выборе лучшего значения проверяемого ключа K , который управляет разделением. Одна из его рекомендаций состоит в том, чтобы в последней части шага Q2 выбирать *случайное* целое число q между l и r ; в этом шаге можно заменить инструкции " $K \leftarrow K_l, R \leftarrow R_l$ " на

$$K \leftarrow K_q, \quad R \leftarrow R_q, \quad R_q \leftarrow R_l. \quad (27)$$

Согласно формулам (25), такие случайные целые числа придется вычислять в среднем лишь $2(N+1)/(M+2)-1$ раз, так что дополнительное время работы несущественно, а случайный выбор—хорошая защита от опасности оказаться в наихудшей ситуации.

Второе предложение Хоара состоит в том, чтобы просмотреть небольшой участок файла и найти медиану для этой совокупности данных. Такому подходу последовал Р. К. Синглтон [АСМ, 12(1969), 185–187], который предложил в качестве K_q брать медиану трех значений

$$K_l, \quad K_{\lfloor (l+r)/2 \rfloor}, \quad K_r. \quad (28)$$

Процедура Синглтона сокращает число сравнений с $2N \ln N$ примерно до $\frac{12}{7}N \ln N$ (см. упр. 29). Можно показать, что в этом случае B_N асимптотически приближается к $C_N/5$, а не к $C_N/6$, так что метод медианы несколько увеличивает время, затрачиваемое на пересылку данных, поэтому общее время работы сокращается примерно на 8%. (Подробный анализ см. в упр. 56.) Время работы в наихудшем случае все еще порядка N^2 , однако с таким медленным поведением алгоритма вряд ли когда-либо придется встретиться на практике.

У. Д. Фрэйзер и А. Ч. Мак-Келлар [АСМ, 17(1970), 496–507] предложили рассматривать совокупность гораздо большего объема из $2^k - 1$ записей, где k выбирается так, чтобы $2^k \approx N/\ln N$. Эту совокупность можно отсортировать обычным методом быстрой сортировки, после чего элементы вставляются среди остальных записей за k просмотров всего файла (в результате файл будет разделен на 2^k подфайлов, ограниченных элементами первоначальной совокупности). На заключительном этапе сортируются полученные подфайлы. Среднее число сравнений, выполняемых такой процедурой "сортировки совокупности", примерно такое же, как и для метода медианы Синглтона, когда N находится в практическом диапазоне значений, но при $N \rightarrow \infty$ оно асимптотически приближается к $N \log_2 N$.

Обменная поразрядная сортировка. Мы подходим теперь к методу, совершенно отличному от всех схем сортировки, которые рассматривались прежде; в нем используется *двоичное представление* ключей, и потому он предназначен исключительно для двоичных машин. Вместо того чтобы сравнивать между собой два ключа, в этом методе проверяется, равны ли 0 или 1 отдельные биты ключа. В других отношениях он обладает характеристиками обменной сортировки и на самом деле очень напоминает быструю сортировку. Так как он зависит от разрядов ключа, представленного в двоичной системе счисления, мы называем его "обменной поразрядной сортировкой". В общих чертах этот алгоритм можно описать следующим образом:

- 1) Последовательность сортируется *по старшему значащему двоичному биту* так, чтобы все ключи, начинающиеся с 0, оказались перед всеми ключами, начинающимися с 1. Для этого надо найти самый левый ключ K_i , начинающийся с 1, и самый правый ключ K_j , начинающийся с 0, после чего R_i и R_j меняются местами, и процесс повторяется, пока не станет $i > j$.
- 2) Пусть F_0 —множество элементов, начинающихся с 0, а F_1 —все остальные. Применим к F_0 поразрядную сортировку (начав теперь со *второго* бита слева, а не со старшего) до тех пор, пока множество F_0 не будет полностью отсортировано; затем сделаем то же с F_1 .

Например, в табл. 3 показано, как действует обменная поразрядная сортировка на наши 16 случайных чисел, записанных теперь в восьмеричной системе счисления. На стадии 1 показан исходный файл; после обменов по первому биту приходим ко второй стадии. На второй стадии сортируется первая группа по второму биту, на третьей—по третьему биту. (Читатель должен мысленно преобразовать восьмеричные числа в 10-разрядные двоичные.) Когда мы после сортировки по четвертому биту достигаем пятой стадии, то обнаруживаем, что каждая из оставшихся групп содержит всего по одному элементу,— так что эту часть файла можно больше не рассматривать. Запись "4[0232 0252]" означает, что подфайл 0232 0252 еще предстоит сортировать по четвертому биту слева. В этом конкретном случае сортировка по четвертому биту не дает ничего нового; чтобы разделить элементы, необходимо добраться до пятого бита.

Весь процесс сортировки, показанный в табл. 3, выполняется за 22 стадии; это несколько больше соответствующего числа в быстрой сортировке (табл. 2). Число проверок битов 82 также велико; но мы увидим, что число проверок битов при больших

Picture: Таблица 3

N в действительности меньше, чем число сравнений в быстрой сортировке, в предположении о равномерном распределении ключей. Общее число обменов в табл. 3 равно 17, т. е. весьма умеренно.

Заметим, что, хотя сортируются 10-битовые числа, в данном примере при проверке битов никогда не приходится идти дальше седьмого бита.

Как и при быстрой сортировке, для хранения "информации о границах" подфайлов, ожидающих сортировки, можно воспользоваться стеком. Вместо того чтобы сортировать в первую очередь наименьший из подфайлов, удобно просто продвигаться слева направо, так как размер стека в этом случае никогда не превзойдет числа битов в сортируемых ключах. В алгоритме, приведенном ниже, элемент стека (r, b) указывает на то, что подфайл с правой границей r ожидает сортировки по биту b ; левую границу можно не запоминать в стеке: она всегда задана неявно, поскольку в этой процедуре файл всегда обрабатывается слева направо.

Алгоритм R. (Обменная поразрядная сортировка.) Записи R_1, \dots, R_N переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Предполагается, что все ключи — m -разрядные двоичные числа $(a_1 a_2 \dots a_m)_2$; i -й по старшинству бит a_i называется "бит i " ключа. Требуется вспомогательный стек, вмещающий до $m - 1$ элементов. Этот алгоритм, по существу, следует процедуре обменной поразрядной сортировки с разделениями, описанной выше; возможны некоторые усовершенствования с целью повышения эффективности (они описаны далее в тексте и в упражнениях).

R1 [Начальная установка.] Опустошить стек и установить $l \leftarrow 1, r \leftarrow N, b \leftarrow 1$.

R2 [Начать новую стадию.] (Мы хотели бы теперь отсортировать подфайл $R_l \leq \dots \leq R_r$ по биту b ; по смыслу алгоритма $l \leq r$.) Если $l = r$, то перейти к шагу **R10** (так как файл, состоящий из одного слова, уже отсортирован). В противном случае установить $i \leftarrow l, j \leftarrow r$.

R3 [Проверить K_i на 1.] Проверить бит b ключа K_i . Если он равен 1, то перейти к шагу **R6**.

R4 [Увеличить i .] Увеличить i на 1. Если $i \leq j$, то возвратиться к шагу **R3**; в противном случае перейти к шагу **R8**.

R5 [Проверить K_{j+1} на 0.] Проверить бит b ключа K_{j+1} . Если он равен 0, то перейти к шагу **R7**.

R6 [Уменьшить j .] Уменьшить j на 1. Если $i \leq j$, то перейти к шагу **R5**; в противном случае перейти к шагу **R8**.

R7 [Поменять местами R_i, R_{j+1} .] Поменять местами $R_i \leftrightarrow R_{j+1}$, затем перейти к шагу **R4**.

R8 [Проверить особые случаи.] (К этому моменту стадия разделения завершена, $i = j + 1$, бит b ключей K_l, \dots, K_j равен 0, а бит b ключей K_i, \dots, K_r равен 1.) Увеличить b на 1. Если $b > m$, где m — общее число битов в ключах, то перейти к шагу **R10**. (Это означает, что подфайл $R_l \dots R_r$ отсортирован. Если в файле не может быть равных ключей, то такую проверку можно не делать.) В противном случае, если $j < l$ или $j = r$, возвратиться к шагу **R2** (все просмотренные биты оказались равными соответственно 1 или 0). Если же $j = l$, то увеличить l на 1 и перейти к шагу **R2** (встретился только один бит, равный 0).

R9 [Поместить в стек.] Поместить в стек элемент (r, b) , затем установить $r \leftarrow j$ и перейти к шагу **R2**.

R10 [Взять из стека.] Если стек пуст, то сортировка закончена. В противном случае установить $l \leftarrow r + 1$, взять из стека элемент (r', b') , установить $r \leftarrow r', b \leftarrow b'$ и возвратиться к шагу **R2**. ■

Программа R. (Обменная поразрядная сортировка.) В следующей программе для машины MIX используются, по существу, те же соглашения, что и в программе Q. Значения регистров: $rI1 \equiv l - r$, $rI2 \equiv r$, $rI3 \equiv i$, $rI4 \equiv j$, $rI5 \equiv m - b$, $rI6$ = размер стека, если не считать того, что в некоторых командах (отмеченных ниже) удобно оставить $rI3 = i - j$ или $rI4 = j - i$. Двоичной природой поразрядной сортировки объясняется использование в этой программе команд SRB (двоичный сдвиг содержимого AX вправо), JAE (переход, если содержимое A четно) и JAO (переход, если содержимое A нечетно), определенных в п. 4.5.2. Предполагается, что $N \geq 2$.

START	ENT6	0	1	R1. Начальная установка. Опустошить стек.
	ENT1	1-N	1	$l \leftarrow 1$.
	ENT2	N	1	$r \leftarrow N$.
	ENT5	M-1	1	$b \leftarrow 1$.
	JMP	1F	1	К R2 (опустить проверку $l = r$).
9H	INC6	1	S	R9. Поместить в стек. [$rI4 = l - j$]
	ST2	STACK, 6 (A)	S	
	ST5	STACK, 6 (B)	S	$(r, b) \Rightarrow$ стек.
	ENN1	0, 4	S	$rI1 \leftarrow l - j$.
	ENT2	-1, 3	S	$r \leftarrow j$.
1H	ENT3	0, 1	A	R2. Начать новую стадию. [$rI3 = i - j$]
	ENT4	0, 2	A	$i \leftarrow l, j \leftarrow r$. [$rI3 = i - j$]
3H	INC3	0, 4	C'	R3. Проверить K_i на 1.

	LDA	INPUT, 3	C'	
	SRB	0, 5	C'	младший бит $rA \leftarrow$ бит b ключа K_i .
	JAE	4F	C'	К R4, если он равен 0.
6H	DEC4	1, 3	$C'' + X$	R6. Уменьшить $j \leftarrow j - l$. [$rI4 = j - i$].
	J4N	8F	$C'' + X$	К R8, если $j < i$. [$rI4 = j - i$]
5H	INC4	0, 3	C''	R5. Проверить K_{j+1} на 0.
	LDA	INPUT+1, 4	C''	
	SRB	0, 5	C''	младший бит $rA \leftarrow$ бит b ключа K_{j+1} .
	JAO	6B	C''	К R6, если он равен 1.
7H	LDA	INPUT+1, 4	B	R7. Поменять местами R_i, R_{j+1} .
	LDX	INPUT, 3	B	
	STX	INPUT+1, 4	B	
	STA	INPUT, 3	B	
4H	DEC3	-1, 4	$C' - X$	R4. Увеличить i . $i \leftarrow i + 1$. [$rI3 = i - j$]
	J3NP	3B	$C' - X$	К R3, если $i \leq j$. [$rI3 = i - j$]
	INC3	0, 4	$A - X$	$rI3 \leftarrow i$.
8H	J5Z	0F	A	R8. Проверить особые случаи. [$rI4 =$ неизвестен]
	DEC5	1	$A - G$	К R10, если $b = m$, иначе $b \leftarrow b - 1$.
	ENT4	-1, 3	$A - G$	$rI4 \leftarrow j$.
	DEC4	0, 2	$A - G$	$rI4 \leftarrow j - r$.
	J4Z	1B	$A - G$	К R2, если $j = r$.
	DEC4	0, 1	$A - G - R$	$rI4 \leftarrow j - l$.
	J4N	1B	$A - G - R$	К R2, если $j < l$.
	J4NZ	9B	$A - G - L - R$	К R9, если $j \neq l$.
	INC1	1	K	$l \leftarrow l + 1$.
2H	J1NZ	1B	$K + S$	Переход, если $l \neq r$.
0H	ENT1	1, 2	$S + 1$	R10. Взять из стека.
	LD2	STACK, 6 (A)	$S + 1$	
	DEC1	0, 2	$S + 1$	
	LD5	STACK, 6 (B)	$S + 1$	Стек $\Rightarrow (r, b)$
	DEC6	1	$S + 1$	
	J6NN	2B	$S + 1$	К R2, если стек не был пуст.

Время работы программы обменной поразрядной сортировки зависит от

$$\begin{aligned}
 A &= \text{число стадий, в которых } l < r; \\
 B &= \text{число замещений}; \\
 C &= C' + C'' = \text{число проверок битов}; \\
 G &= \text{число случаев, когда в шаге R8 } b > m; \\
 K &= \text{число случаев, когда в шаге R8 } b \leq m, j = l; \\
 L &= \text{число случаев, когда в шаге R8 } b \leq m, j < l; \\
 R &= \text{число случаев, когда в шаге R8 } b \leq m, j = r; \\
 S &= \text{число случаев, когда что-либо помещается в стек}; \\
 X &= \text{число случаев, когда в шаге R6 } j < i.
 \end{aligned} \tag{29}$$

По закону Кирхгофа $S = A - G - K - L - R$, так что общее время выполнения равно $27A + 8B + 8C - 23G - 14K - 17L - 19R - X + 13$ единиц. За счет усложнения программы можно несколько ускорить циклы проверки битов (см. упр. 34). Обменную поразрядную сортировку можно также ускорить, если при достаточно малых значениях разности $r - l$ применять простые вставки, как это сделано в алгоритме Q, но мы не будем задерживаться на таких усовершенствованиях.

Для анализа времени выполнения обменной поразрядной сортировки запрашиваются два типа исходных данных:

- i) Пусть $N = 2^m$ и ключи, которые надо отсортировать, — просто числа $0, 1, 2, \dots, 2^m - 1$, расположенные в случайном порядке.
- ii) Пусть $m = \infty$ (неограниченная точность) и ключи, которые надо отсортировать, — независимые, равномерно распределенные в промежутке $[0, 1)$ действительные числа.

Анализ случая (i) относительно прост, поэтому он оставлен читателю в качестве упражнения (см. упр. 35). Случай (ii) сравнительно сложен, поэтому он *также* оставлен для упражнений. В следующей таблице приведены грубые оценки результатов того и другого анализов:

Величина	Случай (i)	Случай (ii)	
A	N	αN	
B	$\frac{1}{4}N \log_2 N$	$\frac{1}{4}N \log_2 N$	
C	$N \log_2 N$	$N \log_2 N$	
G	$\frac{1}{2}N$	0	
K	0	$\frac{1}{2}N$	(30)
L	0	$\frac{1}{2}(\alpha - 1)N$	
R	0	$\frac{1}{2}(\alpha - 1)N$	
S	$\frac{1}{2}N$	$\frac{1}{2}N$	
X	$\frac{1}{2}N$	$\frac{1}{4}(\alpha - 1)N$	

Здесь $\alpha = 1/(\ln 2) = 1.4427$. Заметим, что среднее число обменов, проверок битов и обращений к стеку, по существу, одинаково в обоих случаях, несмотря даже на то, что в случае (ii) число стадий на 44% больше. На сортировку N элементов в случае (ii) наша MIX-программа затратит в среднем приблизительно $14.4N \ln N$ единиц времени. Это число можно было бы сократить примерно до $11.5N \ln N$, если воспользоваться предложением из упр. 34. Соответствующая величина для программы Q равна $12.7N \ln N$, но и ее можно уменьшить до $11.7N \ln N$, если воспользоваться предложением Синглтона и выбирать медиану из трех ключей.

В случае равномерного распределения данных обменная поразрядная сортировка занимает в среднем примерно столько же времени, сколько и быстрая сортировка; фактически для машины MIX она немного быстрее, чем быстрая сортировка. В упр. 53 показано, в какой мере замедляется этот процесс при неравномерном распределении. Важно отметить, что весь наш анализ основан на предположении о том, что все ключи различны. *Обменная поразрядная сортировка в том виде, как она описана выше, не очень эффективна, если имеются одинаковые ключи*, так как она проходит через несколько стадий, требующих времени, пытаясь разделить множества одинаковых ключей, пока b не станет $> m$. Один приемлемый способ исправить этот недостаток предложен в ответе к упр. 40.

Как обменная поразрядная сортировка, так и быстрая сортировка основаны на идее разделения. Записи меняются местами до тех пор, пока файл не будет разбит на две части: левый подфайл, в котором все ключи $\leq K$ при некотором K , и правый подфайл, в котором все ключи $\geq K$. В быстрой сортировке в качестве K выбирается реальный ключ из файла, в то время как в обменной поразрядной сортировке, по существу, выбирается некоторый искусственный ключ на основе двоичных представлений. Что касается исторической стороны дела, то обменную поразрядную сортировку открыли П. Хильдебрандт, Г. Исбитц, Х. Райзинг и Ж. Шварц [JACM, 6 (1959), 156–163] примерно за год до изобретения быстрой сортировки. Возможны также и другие схемы разделения; например, Джон Маккарти предложил выбирать $K \approx \frac{1}{2}(u + v)$, если известно, что все ключи лежат в диапазоне между u и v .

Еще одну стратегию разделения предложил М. Х. ван Эмден [CACM, 13 (1970), 563–567]: вместо того чтобы выбирать K заранее, мы "узнаем", каково может быть хорошее значение K , следя в процессе разделения за изменением величин $K' = \max(K_1, \dots, K_i)$ и $K'' = \min(K_j, \dots, K_r)$. Можно увеличивать i до тех пор, пока не встретится ключ, больший K' ; затем начать уменьшать j , пока не встретится ключ, меньший K'' , после чего поменять их местами и/или уточнить значения K' и K'' . Эмпирические тесты для этой интервальной сортировки показывают, что она требует около $1.64N \ln N = 1.14N \log_2 N$ сравнений. Это единственный метод, обсуждаемый в этой книге, для поведения которого еще не найдено адекватного теоретического объяснения.

Обобщение обменной поразрядной сортировки на случай системы счисления с основанием, большим 2, обсуждается в п. 5.2.5.

* **Асимптотические методы.** Анализ алгоритмов обменной сортировки приводит к некоторым особенно поучительным математическим задачам, которые позволяют больше узнать о способах определения асимптотического поведения функции. Например, при анализе метода пузырька [формула (9)] мы столкнулись с функцией

$$W_n = \frac{1}{n!} \sum_{0 \leq r < s \leq n} s! r^{n-s}. \quad (31)$$

Каково ее асимптотическое значение?

Можно действовать так же, как при исследовании числа инволюций [формула (5.1.4–41)]. Поэтому, прежде чем двигаться дальше, полезно еще раз просмотреть обсуждение в конце п. 5.1.4.

Исследование формулы (31) показывает, что вклад при $s = n$ больше, чем вклад при $s = n - 1$, и т. д.; это наводит на мысль о замене s на $n - s$. Мы скоро увидим, что на самом деле удобнее всего применить подстановки $t = n - s + 1$, $m = n + 1$, в результате которых формула (31) примет вид

$$\frac{1}{m} W_{m-1} = \frac{1}{m!} \sum_{1 \leq t < m} (m-t)! \sum_{0 \leq r < m-t} r^{t-1}. \quad (32)$$

Для внутренней суммы существует хорошо известный асимптотический ряд, полученный из формулы суммирования Эйлера:

$$\begin{aligned} \sum_{0 \leq r < N} r^{t-1} &= \frac{N^t}{t} - \frac{1}{2}(N^{t-1} - \delta_{t1}) + \frac{B_2}{2!}(t-1)(N^{t-2} - \delta_{t2}) + \dots = \\ &= \frac{1}{t} \sum_{0 \leq j \leq k} \binom{t}{j} B_j (N^{t-j} - \delta_{tj}) + O(N^{t-k}) \end{aligned} \quad (33)$$

(см. упр. 1.2.11.2–4). Следовательно, наша задача сводится к изучению сумм вида

$$\frac{1}{m!} \sum_{1 \leq t < m} (m-t)!(m-t)^t t^k, \quad k \geq -1. \quad (34)$$

Как и в п. 5.1.4, можно показать, что при значениях t , больших $m^{1/2+\varepsilon}$, члены суммы пренебрежимо малы: $O(\exp(-n^\delta))$. Значит, можно положить $t = O(m^{1/2+\varepsilon})$ и заменить факториалы по формуле Стирлинга

$$\frac{(m-t)!(m-t)^t}{m!} = \sqrt{1 - \frac{t}{m}} \exp\left(\frac{t}{12m^2} - \left(\frac{t^2}{2m} + \frac{t^3}{3m^2} + \frac{t^4}{4m^3} + \frac{t^5}{5m^4}\right) + O(m^{-2+6\varepsilon})\right).$$

Таким образом, нас интересует асимптотическое значение суммы

$$r_k(m) = \sum_{1 \leq t < m} e^{-t^2/2m} t^k, \quad k \geq -1. \quad (35)$$

(Суммирование здесь также можно было бы распространить на весь диапазон $1 \leq t < \infty$, не изменив при этом асимптотического значения суммы, поскольку, как указано выше, входящие в сумму значения при $t > m^{1/2+\varepsilon}$ пренебрежимо малы.)

Пусть $g_k(x) = x^k e^{-x^2}$ и $f_k(x) = g_k(x/\sqrt{2m})$. По формуле суммирования Эйлера при $k \geq 0$

$$\begin{aligned} \sum_{0 \leq t < m} f_k(t) &= \int_0^m f_k(x) dx + \sum_{1 \leq j \leq p} \frac{B_j}{j!} (f_k^{(j-1)}(m) - f_k^{(j-1)}(0)) + R_p, \\ R_p &= \frac{(-1)^{p+1}}{p!} \int_0^m B_p(\{x\}) f_k^{(p)}(x) dx = \\ &= \left(\frac{1}{\sqrt{2m}}\right)^p O\left(\int_0^\infty |g_k^{(p)}(y)| dy\right) = O(m^{-p/2}); \end{aligned} \quad (36)$$

следовательно, при помощи, по существу, тех же приемов, что и раньше, можно получить асимптотический ряд для $r_k(m)$, если только $k \geq 0$. Но при $k = -1$ этот метод не годится, так как значение $f_{-1}(0)$ не определено; нельзя также просто просуммировать от 1 до m , так как остаточные члены не дают убывающих степеней m , если нижний предел равен 1. (Именно в этом суть дела, и, прежде чем двигаться дальше, читатель должен убедиться в том, что он хорошо понял задачу.)

Чтобы разрешить эту дилемму, можно положить по определению $g_{-1}(x) = (e^{-x^2} - 1)/x$, $f_{-1}(x) = g_{-1}(x/\sqrt{2m})$; тогда $f_{-1}(0) = 0$ и $r_{-1}(m)$ нетрудно получить из $\sum_{0 \leq t < m} f_{-1}(t)$. Уравнение (36) справедливо теперь и при $k = -1$, а оставшийся интеграл нам "хорошо знаком" (см. упр. 43):

$$\begin{aligned} \frac{2}{\sqrt{2m}} \int_0^m f_{-1}(x) dx &= 2 \int_0^m \frac{e^{-x^2/2m} - 1}{x} dx = \int_0^{m/2} \frac{e^{-y} - 1}{y} dy = \\ &= \int_0^1 \frac{e^{-y} - 1}{y} dy + \int_1^{m/2} \frac{e^{-y}}{y} dy - \ln(m/2) = \\ &= -\gamma - \ln(m/2) + O(e^{-m/2}). \end{aligned}$$

Теперь у нас достаточно формул и фактов для того, чтобы вывести наконец ответ, который равен, как показано в ответе к упр 44,

$$W_n = \frac{1}{2}m \ln m + \frac{1}{2}(\gamma + \ln 2)m - \frac{2}{3}\sqrt{2\pi m} + \frac{31}{36} + O(n^{-1/2}), \quad m = n + 1. \quad (37)$$

Этим завершается анализ метода пузыря.

Чтобы проанализировать обменную поразрядную сортировку, необходимо знать асимптотическое поведение при $n \rightarrow \infty$ конечной суммы

$$U_n = \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{1}{2^{k-1} - 1}. \quad (38)$$

Этот вопрос оказывается гораздо сложнее других задач об асимптотическом поведении, с которыми мы сталкивались до сих пор; элементарные методы разложения в степенные ряды, формула суммирования Эйлера и т. д. здесь бессильны. Следующий вывод был предложен Н. Г. де Брейном.

Чтобы избавиться в формуле (38) от подавляющего влияния больших множителей $\binom{n}{k}(-1)^k$, мы начнем с того, что перепишем сумму в виде бесконечного ряда:

$$U_n = \sum_{k \geq 2} \binom{n}{k} (-1)^k \sum_{j \geq 1} \left(\frac{1}{2^k - 1} \right)^j = \sum_{j \geq 1} (2^j (1 - 2^{-j})^n - 2^j + n). \quad (39)$$

Если положить $x = n/2^j$, то член ряда запишется в виде

$$2^j (1 - 2^{-j})^n - 2^j + n = \frac{n}{x} \left(\left(1 - \frac{x}{n}\right)^n - 1 + x \right).$$

При $x \leq n^\varepsilon$ имеем

$$\left(1 - \frac{x}{n}\right)^n = \exp\left(n \ln\left(1 - \frac{x}{n}\right)\right) = \exp(-x + x^2 O(n^{-1})), \quad (40)$$

а это наводит на мысль о том, чтобы приблизить (39) рядом

$$T_n = \sum_{j \geq 1} (2^j e^{-n/2^j} - 2^j + n). \quad (41)$$

Чтобы оправдать это приближение, рассмотрим разность $U_n - T_n = X_n + Y_n$, где

$$\begin{aligned} X_n &= \sum_{\substack{j \geq 1 \\ 2^j < n^{1-\varepsilon}}} (2^j (1 - 2^{-j})^n - 2^j e^{-n/2^j}) = \quad \text{[т.е. слагаемые при } x > n^\varepsilon\text{]} \\ &= \sum_{\substack{j \geq 1 \\ 2^j < n^{1-\varepsilon}}} O(ne^{-n/2^j}) = \quad \text{[так как } 0 < 1 - 2^{-j} < e^{-2^{-j}}\text{]} \\ &= O(n \log ne^{-n^\varepsilon}) \quad \text{[так как имеется } O(\log n)\text{ слагаемых];} \\ Y_n &= \sum_{\substack{j \geq 1 \\ 2^j \geq n^{1-\varepsilon}}} (2^j (1 - 2^{-j})^n - 2^j e^{-n/2^j}) = \quad \text{[слагаемые при } x \leq n^\varepsilon\text{]} \\ &= \sum_{\substack{j \geq 1 \\ 2^j \geq n^{1-\varepsilon}}} \left(e^{-n/2^j} \left(\frac{n}{2^j} \right) O(1) \right) \quad \text{[по формуле (40)]} \end{aligned}$$

Приведенное ниже рассуждение покажет, что эта последняя сумма есть $O(1)$; следовательно, $U_n - T_n = O(1)$. (См. упр. 47.)

До сих пор мы еще не использовали никаких методов, которые бы действительно отличались от применявшихся ранее, но

Picture: Рис. 20. Контур интегрирования для тождеств с гамма-функциями.

для изучения ряда T_n потребуется новая идея, основанная на простых принципах теории функций комплексного переменного. Если x — произвольное положительное число, то

$$e^{-x} = \frac{1}{2\pi i} \int_{1/2-i\infty}^{1/2+i\infty} \Gamma(z) x^{-z} dz = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Gamma\left(\frac{1}{2} + it\right) x^{-(1/2+it)} dt. \quad (42)$$

Для доказательства этого тождества рассмотрим путь интегрирования, показанный на рис. 20(a), где N, N' и M велики. Значение интеграла вдоль этого контура равно сумме вычетов внутри контура, а именно

$$\sum_{0 \leq k < M} x^{-k} \lim_{z \rightarrow -k} (z+k)\Gamma(z) = \sum_{0 \leq k < M} x^{-k} \frac{(-1)^k}{k!}.$$

Интеграл по верхней части контура есть $O\left(\int_{-\infty}^{1/2} |\Gamma(t+iN)|x^{-t} dt\right)$, и у нас имеется известная оценка

$$\Gamma(t+iN) = O(N^{t-1/2}e^{-\pi N/2}) \quad \text{при } N \rightarrow \infty.$$

[Свойства гамма-функций см., например, в книге А. Erdélyi et al., Higher Transcendental Functions, 1 (New York: McGraw-Hill, 1953), гл. 1.] Поэтому интеграл по верхнему отрезку бесконечно мал: $O\left(e^{-\pi N/2} \int_{-\infty}^{1/2} (N/x)^t dt\right)$. Интеграл по нижнему отрезку ведет себя столь же безобидно. Для вычисления интеграла по левому отрезку контура воспользуемся тем фактом, что

$$\begin{aligned} \Gamma\left(\frac{1}{2} + it - M\right) &= \Gamma\left(\frac{1}{2} + it\right) / \left(-M + \frac{1}{2} + it\right) \dots \left(-1 + \frac{1}{2} + it\right) = \\ &= \Gamma\left(\frac{1}{2} + it\right) O(1/(M-1)!). \end{aligned}$$

Следовательно, интеграл по левой стороне есть $O(x^{M-1/2}/(M-1)! \times \int_{-\infty}^{\infty} |\Gamma(\frac{1}{2} + it)| dt)$. Поэтому при $M, N, N' \rightarrow \infty$ уцелеет лишь интеграл по правой стороне; тем самым доказано тождество (42). В действительности тождество (42) остается в силе и в том случае, если заменить $\frac{1}{2}$ любым положительным числом.

Тем же самым рассуждением можно воспользоваться для вывода других полезных соотношений, содержащих гамма-функции. Величину x^{-z} можно заменить другими функциями от z ; можно также заменить другой величиной константу $\frac{1}{2}$. Например,

$$\frac{1}{2\pi i} \int_{-3/2-i\infty}^{-3/2+i\infty} \Gamma(z)x^{-z} dz = e^{-x} - 1 + x, \tag{43}$$

а это — критическая величина в формуле (41) для T_n :

$$T_n = n \sum_{j \geq 1} \frac{1}{2\pi i} \int_{-3/2-i\infty}^{-3/2+i\infty} \Gamma(z)(n/2^j)^{-1-z} dz. \tag{44}$$

Суммирование можно внести под знак интеграла, так как сходимость здесь достаточно хорошая; имеем

$$\sum_{j \leq 1} (n/2^j)^w = n^w \sum_{j \leq 1} (1/2^w)^j = n^w / (2^w - 1), \quad \text{если } \Re(w) > 0,$$

так как $|2^w| = 2^{\Re(w)} > 1$. Поэтому

$$T_n = \frac{n}{2\pi i} \int_{-3/2-i\infty}^{-3/2+i\infty} \frac{\Gamma(z)n^{-1-z}}{2^{-1-z} - 1} dz, \tag{45}$$

и остается оценить последний интеграл.

На этот раз интегрирование производится по контуру, который больше вытянут вправо, как изображено на рис. 20(b). Интеграл по верхнему отрезку есть $O\left(n^{1/2}e^{-\pi M/2} \int_{-3/2}^M N^t dt\right)$, если $2^{iN} \neq 1$, а интеграл по нижнему отрезку также пренебрежимо мал. Интеграл по правому отрезку равен $O\left(n^{-1-M} \int_{-\infty}^{\infty} |\Gamma(M+it)| dt\right)$. Фиксируя M и устремляя N, N' к ∞ , можно показать, что $-T_n/n$ есть $O(n^{-1-M})$ плюс сумма вычетов в области $-3/2 < \Re(z) < M$. Пусть $M \rightarrow \infty$, $\Gamma(z)$ имеет простые полюсы при $z = -1$ и $z = 0$, n^{-1-z} не имеет полюсов, а $1/(2^{-1-z} - 1)$ имеет простые полюсы при $z = -1 + 2\pi ik / \ln 2$.

Наибольшую трудность представляет двойной полюс в точке $z = -1$. Если $w = z + 1$ мало, то можно воспользоваться известным соотношением

$$\Gamma(z+1) = \exp(-\gamma z + \zeta(2)z^2/2 - \zeta(3)z^3/3 + \zeta(4)z^4/4 - \dots),$$

где $\zeta(s) = 1^{-s} + 2^{-s} + 3^{-s} + \dots = H_{\infty}^{(s)}$, для вывода следующих разложений:

$$\begin{aligned}\Gamma(z) &= \frac{\Gamma(w+1)}{w(w-1)} = -w^{-1} + (\gamma-1) + O(w); \\ n^{-1-z} &= 1 - (\ln n)w + O(w^2); \\ 1/(2^{-1-z} - 1) &= -w^{-1}/\ln 2 - \frac{1}{2} + O(w).\end{aligned}$$

Вычет в точке $z = -1$ равен коэффициенту при w^{-1} в произведении этих трех формул, а именно $\frac{1}{2} - (\ln n + \gamma - 1)/\ln 2$. Прибавляя остальные вычеты, получаем формулу

$$T_n/n = \frac{\ln n + \gamma - 1}{\ln 2} - \frac{1}{2} + f(n) + \frac{2}{n}, \quad (46)$$

где $f(n)$ —функция довольно необычного вида:

$$f(n) = \frac{2}{\ln 2} \sum_{k \geq 1} \Re(\Gamma(-1 - 2\pi ik/\ln 2) \exp(2\pi ik \log_2 n)). \quad (47)$$

Заметим, что $f(n) = f(2n)$. Среднее значение $f(n)$ равно 0, так как среднее значение каждого слагаемого равно 0. (Можно считать, что величина $(\log_2 n) \bmod 1$ распределена равномерно, принимая во внимание результаты о числах с плавающей точкой, полученные в п. 4.2.4.) Кроме того, поскольку $|\Gamma(-1 + it)| = |\pi/(t(1+t^2) \sinh \pi t)|^{1/2}$, нетрудно показать, что

$$f(n) < 0.0000001725;$$

таким образом, в практических приложениях $f(n)$ можно спокойно отбросить. Что касается теории, то без $f(n)$ мы не можем получить асимптотического ряда для U_n ; именно поэтому величина U_n сравнительно трудна для анализа. Итак, мы доказали, что

$$U_n = n \log_2 n + n \left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} + f(n) \right) + O(1). \quad (48)$$

Другие примеры применения этого метода гамма-функций можно найти в упр. 51–53 и в § 6.3.

Упражнения

- [M20] Пусть $a_1 \dots a_n$ —перестановка множества $\{1, \dots, n\}$, и пусть i и j таковы, что $i < j$ и $a_i > a_j$. Пусть $a'_1 \dots a'_n$ —перестановка, которая получается из $a_1 \dots a_n$, если поменять местами a_i и a_j . Может ли в $a'_1 \dots a'_n$ быть больше инверсий, чем в $a_1 \dots a_n$?
- [M25] (а) Каково минимальное число обменов, необходимое для того, чтобы отсортировать перестановку 3 7 6 9 8 1 4 5? (б) Вообще пусть дана перестановка $\pi = a_1 \dots a_n$ множества $\{1, \dots, n\}$, и пусть $\text{sch}(\pi)$ —минимальное число обменов, в результате которых перестановка π будет отсортирована в возрастающем порядке. Выразите $\text{sch}(\pi)$, через "более простые" характеристики перестановки π . (Ср. с упр. 5 2.1–39.)
- [10] Является ли устойчивой сортировка методом пузырька (алгоритм В)?
- [M23] Если в шаге В4 получится $t = 1$, то на самом деле работу алгоритма В можно сразу же заканчивать, потому что в следующем шаге В2 не выполнится никаких полезных действий. Какова вероятность того, что при сортировке случайной перестановки в шаге В4 окажется $t = 1$?
- [M25] Пусть $b_1 b_2 \dots b_n$ —таблица инверсий перестановки $a_1 a_2 \dots a_n$. Покажите, что после r просмотров сортировки методом пузырька значение переменной BOUND равно $\max\{b_i + i \mid b_i \geq r\} - r$, где $0 \leq r \leq \max(b_1, \dots, b_n)$.
- [M22] Пусть $a_1 \dots a_n$ —перестановка множества $\{1, \dots, n\}$, и пусть $a'_1 \dots a'_n$ —обратная к ней перестановка. Покажите, что число просмотров, необходимых для того, чтобы отсортировать $a_1 \dots a_n$ методом пузырька, равно $1 + \max(a'_1 - 1, a'_2 - 2, \dots, a'_n - n)$.
- [M28] Вычислите стандартное отклонение числа просмотров сортировки методом пузырька и выразите его через n и функцию $P(n)$. [Ср. с формулами (6) и (7).]
- [M24] Выведите формулу (8).
- [M48] Проанализируйте число просмотров и число сравнений в алгоритме шейкер-сортировки. (Замечание: частичная информация содержится в упр. 5.4.8–9.)

10. [M26] Пусть $a_1 a_2 \dots a_n$ —2-упорядоченная перестановка множества $\{1, 2, \dots, n\}$. (a) Каковы координаты конечных точек a_i -го шага соответствующего решеточного пути (ср. с рис. 11)? (b) Докажите, что сравнение/обмен элементов $a_1: a_2, a_3, a_4, \dots$ соответствует перегибанию пути относительно диагонали, как на рис. 18(b). (c) Докажите, что сравнение/обмен элементов $a_2: a_{2+d}, a_4: a_{4+d}, \dots$ соответствует перегибанию пути относительно линии, расположенной на m единиц ниже диагонали, как на рис. 18(c), (d) и (e), если $d = 2m - 1$.
- >11. [M25] На какой перестановке множества $\{1, 2, \dots, 16\}$ достигается максимум числа обменов в алгоритме Бэтчера?
12. [24] Напишите MIX-программу для алгоритма M, предполагая, что MIX—двоичная вычислительная машина с операциями AND и SRB. Сколько времени потребуется вашей программе, чтобы отсортировать шестнадцать записей из табл. 1?
13. [10] Устойчива ли сортировка Бэтчера?
14. [M21] Пусть $c(N)$ —число сравнений ключей, производимых при сортировке N элементов методом Бэтчера; это равно числу выполнению шага M4. (a) Покажите, что при $t \geq 1$ $c(2^t) = 2c(2^{t-1}) + (t - 1)2^{t-1} + 1$. (b) Найдите простое выражение для $c(2^t)$ как функцию от t . (Указание: рассмотрите последовательность $x_t = c(2^t)/2^t$).
15. [M38] Содержание этого упражнения—анализ функции $c(N)$ из упр. 14 и нахождение формулы для $c(N)$, если $N = 2^{e_1} + 2^{e_2} + \dots + 2^{e_r}$, $e_1 > e_2 > \dots > e_r \geq 0$. (a) Пусть $a(N) = c(N + 1) - c(N)$. Докажите, что $a(2n) = a(n) + \lfloor \log_2(2n) \rfloor$, $a(2n + 1) = a(n) + 1$; отсюда

$$a(N) = \binom{e_1 + 1}{2} - r(e_1 - 1) + (e_1 + e_2 + \dots + e_r).$$

- (b) Пусть $x(n) = a(n) - a(\lfloor n/2 \rfloor)$, и тогда $a(n) = x(n) + x(\lfloor n/2 \rfloor) + x(\lfloor n/4 \rfloor) + \dots$. Пусть $y(n) = x(1) + x(2) + \dots + x(n)$, и пусть $z(2n) = y(2n) - a(n)$, $z(2n + 1) = y(2n + 1)$. Докажите, что $c(N + 1) = z(N) + 2z(\lfloor N/2 \rfloor) + 4z(\lfloor N/4 \rfloor) + \dots$. (c) Докажите, что $y(N) = N + (\lfloor N/2 \rfloor + 1) \times (e_1 - 1) - 2^{e_1} + 2$. (d) Теперь соберите все вместе и найдите выражение $c(N)$ через показатели e_j при фиксированном значении r .
16. [BM46] Найдите асимптотическое значение среднего числа обменов в случае, когда алгоритм Бэтчера применяется к $N = 2^t$ различным элементам, расположенным в случайном порядке.
- >17. [20] Где в алгоритме Q используется то, что K_0 и K_{N+1} обладают значениями, постулированными неравенствами (13)?
- >18. [20] Объясните, как работает алгоритм Q в случае, когда все ключи в исходном файле равны. Что произойдет, если в шагах Q3 и Q5 заменить знаки " $<$ " на " \leq "?
19. [15] Будет ли алгоритм Q по-прежнему работать правильно, если вместо стека (последним включается—первым исключается) воспользоваться очередью (первым включается—первым исключается)?
20. [M20] Выразите наибольшее число элементов, которые могут одновременно оказаться в стеке во время работы алгоритма Q, в виде функции от M и N.
21. [20] Объясните, почему фаза разделения в алгоритме Q требует как раз такого числа сравнений, пересылок и т. д., как это описано в (17).
22. [M25] Пусть $p_k N$ —вероятность того, что величина A в (14) будет равна k , если алгоритм Q применяется к случайной перестановке множества $\{1, 2, \dots, N\}$, и пусть $A_N(z) = \sum_k p_k N z^k$ —соответствующая производящая функция. Докажите, что $A_N(z) = 1$ при $N \leq M$, и $A_N(z) = z(\sum_{1 \leq s \leq N} A_{s-1}(z) A_{N-s}(z))/N$ при $N > M$. Найдите аналогичные рекуррентные соотношения, определяющие другие распределения вероятностей $B_N(z), C_N(z), D_N(z), E_N(z), L_N(z), X_N(z)$.
23. [M24] Пусть $A_N, B_N, D_N, E_N, L_N, X_N$ —средние значения соответствующих величин в (14) при сортировке случайной перестановки-множества $\{1, 2, \dots, N\}$. Найдите для этих величин рекуррентные соотношения, аналогичные (18), затем разрешите эти соотношения и получите формулы (25).
24. [M21] Очевидно, в алгоритме Q производится несколько больше сравнений, чем это необходимо, потому что в шагах Q3 и Q5 может оказаться, что $i = j$ или даже $i > j$. Сколько сравнений G_N производилось бы в среднем, если бы исключались все сравнения при $i \geq j$?
25. [M20] Чему равны точные значения величин A, B, C, D, E, L, X для программы Q в случае, когда исходные ключи представляют собой упорядоченный набор чисел $1, 2, \dots, N$ в предположении, что $N > M$?
- >26. [M21] Постройте исходный файл, при котором программа Q работала бы еще медленнее, чем в упр. 25. (Попытайтесь найти действительно плохой случай.)
27. [M23] Какой исходный файл будет наилучшим для алгоритма Q? Найдите приблизительные значения величин A, B, C, D, E, X для этого случая.

28. [M26] Найдите рекуррентное соотношение, аналогичное (20), которому удовлетворяет среднее число сравнений в алгоритме Q , модифицированном Синглтоном (т. е. когда в качестве s выбирается не $s = K_1$, а медиана из $\{K_1, K_{\lfloor (N+1)/2 \rfloor}, K_N\}$).
29. [BM40] Найдите асимптотическое значение числа сравнений в алгоритме Синглтона "медиана из трех" (см. упр. 28).
30. [25] (П. Шаклетон.) При сортировке *ключей длиной в несколько слов* многие алгоритмы все более замедляются по мере того, как файл приближается к упорядоченному состоянию, поскольку для определения правильного лексикографического порядка равных или почти равных ключей требуется сравнить несколько пар слов (см. упр. 5-5). Файлы, которые встречаются на практике, часто содержат почти равные ключи, и это явление может заметно отразиться на времени сортировки.
- Такое затруднение существенно для алгоритма Q , но не для алгоритма R , поскольку там каждый раз проверяется только один бит (хотя присутствие равных и почти равных ключей может сильно увеличить время работы алгоритма R по другим причинам).
- Объясните, как можно усовершенствовать алгоритм Q , чтобы избежать этого затруднения; в подфайле, о котором известно, что старшие k слов во всех ключах содержат постоянные значения, следует проверять лишь $(k+1)$ -е слова ключей.
- >31. [20] (Ч. Э. Р. Хоар) Предположим, что нам нужно не отсортировать файл, а лишь определить m -й наименьший по величине элемент из заданного множества n элементов. Покажите, что "быструю сортировку" можно приспособить для этой цели, избежав значительной части вычислений, необходимых для полной сортировки.
32. [M40] Найдите простое выражение "в замкнутом виде" для C_{nm} — среднего числа сравнений ключей, необходимых для отыскания m -го наименьшего из n элементов по методу упр. 31. (Для простоты можно пропускать все сравнения с $i \geq j$, как в упр. 24.) Каково асимптотическое поведение величины $C_{(2m-1)m}$ — среднего числа сравнений, необходимых для нахождения медианы из $2m-1$ элементов методом Хоара?
- >33. [20] Разработайте алгоритм переразмещения чисел в некоторой заданной таблице таким образом, чтобы все отрицательные значения предшествовали положительным. Элементы не нужно полностью сортировать: достаточно просто отделить отрицательные числа от положительных. Ваш алгоритм должен производить минимально возможное число обменов.
34. [20] Как можно ускорить циклы проверки битов в обменной поразрядной сортировке (шаги от R3 до R6)?
35. [M23] Проанализируйте статистические характеристики A, B, C, G, K, L, R, S и X , которые получаются при обменной поразрядной сортировке для "исходных данных типа (i)".
36. [M27] Для любой данной последовательности чисел $\langle a_n \rangle = a_0, a_1, a_2, \dots$ определим ее *биномиальное преобразование* $\langle \hat{a}_n \rangle = \hat{a}_0, \hat{a}_1, \hat{a}_2, \dots$ правилом

$$\hat{a}_n = \sum_k \binom{n}{k} (-1)^k a_k.$$

- (a) Докажите, что $\langle \hat{\hat{a}}_n \rangle = \langle a_n \rangle$. (b) Найдите биномиальные преобразования последовательностей $\langle 1 \rangle$; $\langle n \rangle$; $\langle \binom{n}{m} \rangle$ при фиксированном m ; $\langle a^n \rangle$ при фиксированном a ; $\langle \binom{n}{m} a^n \rangle$ при фиксированных a и m . (c) При условии, что последовательность $\langle x_n \rangle$ удовлетворяет соотношению

$$x_n = a_n + 2^{1-n} \sum_{k \geq 2} \binom{n}{k} x_k \quad \text{при } n \geq 2, x_0 = x_1 = a_0 = a_1 = 0,$$

докажите, что

$$x_n = \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{2^{k-1} \hat{a}_k}{2^{k-1} - 1} = a_n + \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{\hat{a}_k}{2^{k-1} - 1}.$$

37. [M28] Определите все такие последовательности $\langle a_n \rangle$, что $\langle \hat{a}_n \rangle = \langle a_n \rangle$ в смысле упр. 36.
- >38. [M30] Найдите $A_N, B_N, C_N, G_N, K_N, L_N, R_N$, и X_N — средние значения величин (29) — в случае, когда поразрядной сортировке подвергаются "исходные данные типа (ii)". Выразите ваши ответы через N и функции

$$U_n = \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1}, \quad V_n = \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k k}{2^{k-1} - 1} = n(U_n - U_{n-1}).$$

[Указание: см. упр. 36.]

39. [20] Результаты (30) показывают, что поразрядная обменная сортировка, примененная к случайным входным данным, требует около 1.44 стадий. Докажите, что быстрая сортировка никогда не требует более N стадий, и объясните, почему при обменной поразрядной сортировке часто бывает необходимо более N стадий.
40. [21] Объясните, как нужно изменить алгоритм R, чтобы он работал достаточно эффективно и тогда, когда сортируемые файлы содержат много равных ключей.
41. [23] Дайте точное описание алгоритма "интервальной обменной сортировки" Ван Эмдена.
42. [M43] Проанализируйте алгоритм интервальной обменной сортировки из упр. 41.
43. [BM21] Докажите, что $\int_0^1 y^{-1}(e^{-y}-1) dy + \int_1^\infty y^{-1}e^{-y} dy = -\gamma$. [Указание: рассмотрите $\lim_{a \rightarrow 0^+} y^{a-1}$.]
44. [BM24] Выведите формулу (37), как это предложено в тексте настоящего пункта.
45. [BM20] Объясните, почему при $x > 0$ справедливо равенство (43).
46. [BM20] Каково значение выражения $(1/2\pi i) \int_{a-i\infty}^{a+i\infty} \Gamma(z)n^{s-z} dz / (2^{s-z} - 1)$ при условии, что s — целое положительное число и $0 < a < s$?
47. [BM21] Докажите, что $\sum_{j \geq 1} (n/2^j)e^{-n/2^j}$ — ограниченная функция от n .
48. [BM24] Найдите асимптотическое значение величины V_n , определенной в упр. 38, при помощи методов, аналогичных тем, которыми в тексте настоящего пункта исследовалась величина U_n , и получите члены до $O(1)$.
49. [BM24] Продолжите асимптотическую формулу (47) для U_n до членов порядка $O(n^{-1})$.
50. [BM24] Найдите асимптотическое значение функции

$$U_{mn} = \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{1}{m^{k-1} - 1},$$

где m — произвольное фиксированное число, большее 1. (Эта величина при m целом большем 2 потребуется при исследовании обобщений обменной поразрядной сортировки, а также при изучении алгоритмов поиска в "боровой памяти" в § 6.3.)

- >51. [BM28] Покажите, что для вывода асимптотического разложения $r_k(m)$ можно воспользоваться методом исследования асимптотических задач при помощи гамма-функций вместо формулы суммирования Эйлера (ср. с (35)). Это дает нам единообразный способ изучения $r_k(m)$ при всех k , не полагаясь на такие "трюки", как введение функции $g_{-1}(x) = (e^{-x^2} - 1)/x$.
52. [BM35] (Н. Г. де Брейн.) Каково асимптотическое поведение суммы

$$S_n = \sum_{t \geq 1} \left(\frac{2n}{n+t} \right) d(t),$$

где $d(t)$ — количество делителей числа t ? (Так, $d(1) = 1$, $d(2) = d(3) = 2$, $d(4) = 3$, $d(5) = 2$ и т. д. Этот вопрос возникает в связи с анализом алгоритма прохождения дерева; упр. 2.3.1-11.) Найдите значение величины $S_n / \binom{2n}{n}$ до членов порядка $O(n^{-1})$.

53. [BM42] Проанализируйте число проверок битов и число обменов, выполняемых при обменной поразрядной сортировке, если исходными данными служат двоичные числа с бесконечной точностью из промежутка $[0, 1)$, каждый бит которых независимо принимает значение 1 с вероятностью p . (В основном тексте пункта обсуждался лишь случай $p = \frac{1}{2}$; применявшиеся методы можно обобщить на случай произвольного p .) Рассмотрите особо случай $p = 1/\phi = .61803\dots$
54. [BM24] (С. О. Райе.) Покажите, что U_n можно записать в виде

$$U_n = (-1)^n \frac{n!}{2\pi i} \oint_C \frac{dz}{z(z-1)\dots(z-n)} \frac{1}{2^{z-1} - 1},$$

где C — замкнутая кривая, охватывающая область около точек $2, 3, \dots, n$. В результате замены C на произвольно большую окружность с центром в начале координат получаем сходящийся ряд

$$U_n = n(H_{n-1} - 1)/(\ln 2) - \frac{1}{2}n + 2 + \frac{2}{\ln 2} \sum_{m \geq 1} \Re(B(n+1, -1 + ibm)),$$

где $b = 2\pi/(\ln 2)$ и $B(n+1, -1 + ibm) = \Gamma(n+1)\Gamma(-1 + ibm)/\Gamma(n + ibm) = n! / \prod_{0 \leq k \leq n} (k - 1 + ibm)$.

- >55. [22] Покажите, как нужно изменить программу Q, чтобы в качестве разделяющего элемента выбиралась медиана из трех ключей (28).
56. [M44] Проанализируйте в среднем поведение величин, от которых зависит время работы программы Q, если программа изменена так, что она выбирает медиану из трех элементов, как в упр. 55. [(Ср. упр. 29.)]

>57. [BM24] Найдите асимптотическое значение числа правосторонних максимумов, встречающихся при сортировке вставками в несколько списков (формула (5.2.1-11)), если $M = N/\alpha$ при фиксированном α и $N \rightarrow \infty$. Выполните разложение до членов порядка $O(N^{-1})$ и выразите ваш ответ через интегральную показательную функцию $E_1(z) = \int_z^\infty e^{-t} dt/t$.

5.2.3. Сортировка посредством выбора

Еще одно важное семейство методов сортировки основано на идее многократного выбора. Вероятно, простейшая сортировка посредством выбора сводится к следующему:

- i) Найти наименьший ключ; переслать соответствующую запись в область вывода и заменить ключ значением "∞" (которое по предположению больше любого реального ключа).
- ii) Повторить шаг (i). На этот раз будет выбран ключ, наименьший из оставшихся, так как ранее наименьший ключ был заменен на ∞.
- iii) Повторять шаг (i) до тех пор, пока не будут выбраны N записей.

Заметим, что этот метод требует наличия всех исходных элементов до начала сортировки, а элементы вывода он порождает последовательно, один за другим. Картина, по существу, противоположна методу вставок, в котором исходные элементы должны поступать последовательно, но вплоть до завершения сортировки ничего не известно об окончательном выводе.

Ряд вычислительных машин (например, машины с циклической барабанной памятью) имеет встроенную команду "найти наименьший элемент", которая выполняется с большой скоростью. На таких машинах сортировка указанным методом особенно привлекательна, если только N не слишком велико.

Описанный метод требует $N - 1$ сравнений каждый раз, когда выбирается очередная запись; он также требует отдельной области вывода в памяти. Имеется очевидный способ несколько поправить ситуацию, избежав при этом использования ∞: выбранное значение можно записывать в соответствующую позицию, а запись, которая ее занимала, переносить на место выбранной. Тогда эту позицию не нужно рассматривать вновь при последующих выборах. На этой идее основан наш первый алгоритм сортировки посредством выбора.

Алгоритм S. (Сортировка посредством простого выбора.) Записи R_1, \dots, R_N переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Сортировка основана на описанном выше методе, если не считать того, что более, удобно, оказывается, выбирать сначала *наибольший* элемент, затем второй по величине и т. д.

S1 [Цикл по j .] Выполнить шаги S2 и S3 при $j = N, N - 1, \dots, 2$.

S2 [Найти $\max(K_1, \dots, K_j)$.] Найти наибольший из ключей K_j, K_{j-1}, \dots, K_1 ; пусть это будет K_i .

S3 [Поменять местами с R_j .] Взаимозаменить записи $R_i \leftrightarrow R_j$. (Теперь записи R_j, \dots, R_N занимают свои окончательные позиции.) ■

Picture: Рис. 21. Сортировка простым выбором.

В табл. 1 показано действие этого алгоритма на шестнадцать ключей, выбранных нами для примеров; элементы, претендующие на то, чтобы оказаться максимальными во время поиска в шаге S2, выделены жирным шрифтом.

Таблица 1

Сортировка посредством простого выбора															
503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	087	512	061	703	170	897	275	653	426	154	509	612	677	765	908
503	087	512	061	703	170	765	275	653	426	154	509	612	677	897	908
503	087	512	061	703	170	677	275	653	426	154	509	612	765	897	908
503	087	512	061	612	170	677	275	653	426	154	509	703	765	897	908
503	087	512	061	612	170	509	275	653	426	154	677	703	765	897	908
...															
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Соответствующая MIX-программа довольно проста.

Программа S. (Сортировка посредством простого выбора.) Как и в предыдущих программах этой главы, записи, находящиеся в ячейках от INPUT + 1 до INPUT + N, сортируются на том же месте

по ключу, занимающему полное слово. Значения регистров: $rA \equiv$ текущий максимум, $rI1 \equiv j - 1$, $rI2 \equiv k$ (индекс при поиске), $rI3 \equiv i$. Предполагается, что $N \geq 2$.

START	ENT1	N-1	1	S1. Цикл по j . $j \leftarrow N$.
2H	ENT2	0, 1	$N - 1$	S2. Найти $\max(K_1, \dots, K_j)$. $k \leftarrow j - 1$.
	ENT3	1, 1	$N - 1$	$i \leftarrow j$.
	LDA	INPUT, 3	$N - 1$	$rA \leftarrow K_i$.
8H	CMPA	INPUT, 2	A	
	JGE	*+3	A	Переход, если $K_i \geq K_k$.
	ENT3	0, 2	B	В противном случае установить $i \leftarrow k$,
	LDA	INPUT, 3	B	$rA \leftarrow K_i$.
	DEC2	1	A	$k \leftarrow k - 1$.
	J2P	8B	A	Повторить, если $k > 0$.
	LDX	INPUT+1, 1	$N - 1$	S3. Поменять местами с R_j .
	STX	INPUT, 3	$N - 1$	$R_i \leftarrow R_j$.
	STA	INPUT+1, 1	$N - 1$	$R_j \leftarrow rA$.
	DEC1	1	$N - 1$	
	J1P	2B	$N - 1$	$N \geq j \geq 2$.

■

Время работы этой программы зависит от числа элементов N , числа сравнений A и числа "правосторонних максимумов" B . Нетрудно видеть, что независимо от значений исходных ключей

$$A = \binom{N}{2} = \frac{1}{2}N(N-1). \quad (1)$$

Следовательно, переменной является только величина B . Несмотря на всю безыскусность простого выбора, не так легко произвести точный анализ величины B . В упр. с 3 по 6 показано, что

$$B = (\min 0, \text{ave}(N+1)H_n - 2N, \max \lfloor N^2/4 \rfloor); \quad (2)$$

в этом случае особенно интересным оказывается максимальное значение (стандартное отклонение величины B до сих пор не определено).

Таким образом, среднее время работы программы S равно $2.5N^2 + 3(N+1)H_N + 3.5N - 11$ единиц, т. е. она лишь немногим медленнее простых вставок (программа 5.2.1S). Интересно сравнить алгоритм S с сортировкой методом пузырька (алгоритм 5.2.2B), так как метод пузырька можно рассматривать как алгоритм выбора, в котором иногда выбирается более одного элемента за раз. По этой причине при сортировке методом пузырька производится меньше сравнений, чем при простом выборе, и она, как может показаться, предпочтительнее. Но в действительности программа 5.2.2B более чем вдвое медленнее программы S! Сортировка методом пузырька проигрывает из-за того, что в ней выполняется слишком много обменов, в то время как при сортировке простым выбором производится очень мало пересылок данных.

Усовершенствования простого выбора. Существует ли какой-нибудь способ улучшить метод выбора, используемый в алгоритме S? Возьмем к примеру поиск максимума в шаге S2: возможен ли существенно более быстрый способ нахождения максимума? Ответ на этот вопрос — *нет!*

Лемма М. В любом алгоритме нахождения максимума из n элементов, основанном на сравнении пар элементов, необходимо выполнить по крайней мере $n - 1$ сравнений.

Доказательство. Если произведено менее $n - 1$ сравнений, то найдутся по крайней мере два элемента, для которых не было обнаружено ни одного элемента, превосходящего их по величине. Следовательно, мы так и не узнаем, который из этих двух элементов больше, и, значит, не сможем определить максимум. ■

Таким образом, процесс выбора, в котором отыскивается наибольший элемент, должен состоять не менее чем из $n - 1$ шагов. Означает ли это, что для всех методов сортировки, основанных на n повторных выборах, число шагов неизбежно будет порядка n^2 ? К счастью, лемма М применима только к *первому* шагу выбора; при последующих выборах можно использовать извлеченную ранее информацию. Например, в упр. 8 показано, что сравнительно простое изменение алгоритма S наполовину сокращает число сравнений.

Рассмотрим 16 чисел, представленных в 1-й строке в табл. 1. Один из способов сэкономить время при последующих выборах—разбить все числа на четыре группы по четыре числа. Начать можно с определения наибольшего, элемента каждой группы, а именно соответственно с ключей

512, 908, 653, 765;

тогда наибольший из этих четырех элементов 908 и будет наибольшим во всем файле. Чтобы получить второй по величине элемент, достаточно просмотреть сначала остальные три элемента группы, содержащей 908; наибольший из {170, 897, 275} равен 897, и тогда наибольший среди

512, 897, 653, 765

это 897. Аналогично, для того чтобы получить третий по величине элемент, определяем наибольший из {170, 275}, а затем наибольший из

512, 275, 653, 765

и т. д. Каждый выбор, кроме первого, требует не более 6 дополнительных сравнений. Вообще, если N —точный квадрат, то можно разделить файл на \sqrt{N} групп по \sqrt{N} элементов в каждой; любой выбор, кроме первого, требует не более чем $\sqrt{N} - 1$ сравнений внутри группы ранее выбранного элемента плюс $\sqrt{N} - 1$ сравнений среди "лидеров групп". Этот метод получил название "квадратичный выбор"; общее время работы для него есть $O(N\sqrt{N})$, что существенно лучше, чем $O(N^2)$.

Метод квадратичного выбора впервые опубликован Э. Х. Фрэндом [JACM, 3 (1956), 152–154]; он указал, что эту идею можно обобщить, получив в результате метод кубического выбора, выбора четвертой степени и т. д. Например, метод кубического выбора состоит в том, чтобы разделить файл на $\sqrt[3]{N}$ больших групп, каждая из которых содержит по $\sqrt[3]{N}$ малых групп по $\sqrt[3]{N}$ записей; время работы пропорционально $N\sqrt[3]{N}$. Если развить эту идею до ее полного завершения, то мы придем к тому, что Фрэнд назвал "выбор n -й степени", основанный на структуре бинарного дерева. Время работы этого метода пропорционально $N \log N$; мы будем называть его *выбором из дерева*.

Выбор из дерева. Принципы сортировки посредством выбора из дерева будет легче воспринять, если воспользоваться аналогией с типичным "турниром с выбыванием". Рассмотрим, например, результаты соревнования по настольному теннису, показанные на рис. 22. Джим побеждает Дона, а Джо побеждает Джека; затем в следующем туре Джо выигрывает у Джима и т. д.

На рис. 22 показано, что Джо—чемпион среди восьми спортсменов, а для того, чтобы определить это, потребовалось $8 - 1 = 7$ матчей (т. е. сравнений). Дик вовсе не обязательно будет вторым по силе игроком: любой из спортсменов, у которых выиграл Джо, включая даже проигравшего в первом туре Джека, мог бы оказаться вторым по силе игроком. Второго игрока можно определить, заставив Джека сыграть с Джимом, а победителя этого матча—с Диком; всего два дополнительных матча требуется для определения второго по силе игрока, исходя из того соотношения сил, которое мы запомнили из предыдущих игр.

Вообще можно "вывести" игрока, находящегося в корне дерева, и заменить его чрезвычайно слабым игроком. Подстановка этого слабого игрока означает, что первоначально второй по силе спортсмен станет теперь наилучшим, и именно он окажется в корне, если вновь вычислить победителей в верхних уровнях дерева. Для этого нужно изменить лишь один путь, в дереве, так что для выбора следующего по силе игрока необходимо менее $\lceil \log_2 N \rceil$ дополнительных сравнений. Суммарное

Picture: Рис. 23. Пример сортировки посредством выбора из дерева...

время выполнения такой сортировки посредством выбора примерно пропорционально $N \log N$.

На рис. 23 сортировке посредством выбора из дерева подвергаются наши 16 чисел. Заметим, что для того, чтобы знать, куда вставлять следующий элемент " $-\infty$ ", необходимо помнить, откуда пришел ключ, находящийся в корне. Поэтому узлы разветвления в действительности содержат указатели или индексы, описывающие позицию ключа, а не сам ключ. Отсюда следует, что необходима память для N исходных записей, $N - 1$ слов-указателей и N выводимых записей. (Разумеется, если вывод

Picture: Рис. 24. Применение корпоративной системы выдвижений к сортировке. Каждый поднимается на свой уровень некомпетентности в иерархии.

идет на ленту или на диск, то не нужно сохранять выводимые записи в оперативной памяти.)

Чтобы оценить те замечательные усовершенствования, которые мы собираемся обсудить, в этом месте следует прервать чтение до тех пор, пока вы не освоитесь с выбором из дерева хотя бы настолько, что решение упр. 10 не составит для вас никакого труда.

Одна из модификаций выбора из дерева, введенная, по существу, К. Э. Айверсоном [A Programming Language (Wiley, 1962), 223–227], устраняет необходимость указателей, следующим образом осуществляя ”заглядывание вперед”: когда победитель матча на нижнем уровне поднимается вверх, то на нижнем уровне его сразу же можно заменить на $-\infty$; когда же победитель перемещается вверх с одного разветвления на другое, то его можно заменить игроком, который в конце концов все равно должен подняться, на его прежнее место (а именно наибольшим из двух ключей, расположенных под ним). Выполнив эту операцию столько раз, сколько возможно, приходим от рис. 23(а) к рис. 24.

Коль скоро дерево построено таким образом, можно продолжать сортировку не ”восходящим” методом, показанным на рис. 23, а ”нисходящим”: выводится элемент, находящийся в корне, перемещается вверх наибольший из его потомков, перемещается вверх наибольший из потомков последнего и т. д. Процесс начинает походить не столько на турнир по настольному теннису, сколько на корпоративную систему выдвижений.

Читатель должен уяснить, что у нисходящего метода есть важное достоинство—он позволяет избежать лишних сравнений $-\infty$ с $-\infty$. (Пользуясь восходящим методом, мы на более поздних стадиях сортировки всюду натываемся на $-\infty$, а при нисходящем методе можно на каждой стадии заканчивать преобразование дерева сразу же после занесения $-\infty$.)

Picture: Рис. 25. Последовательное распределение памяти для полного бинарного дерева.

На рис. 23 и 24 изображены *полные бинарные деревья* с 16 концевыми узлами (ср. с п. 2.3.4.5); такие деревья удобно хранить в последовательных ячейках памяти, как показано на рис. 25. Заметим, что отцом узла номер k является узел $\lfloor k/2 \rfloor$, а его потомками являются узлы $2k$ и $2k + 1$. Отсюда вытекает еще одно преимущество нисходящего метода, поскольку зачастую значительно проще продвигаться вниз от узла k к узлам $2k$ и $2k + 1$, чем вверх от узла k к узлам $k \oplus 1$ и $\lfloor k/2 \rfloor$. (Здесь через $k \oplus 1$ обозначено число $k + 1$ или $k - 1$ в зависимости от того, является ли k четным или нечетным.)

До сих пор в наших примерах выбора из дерева в той или иной мере предполагалось, что N есть степень 2; в действительности можно работать с произвольным значением N , так как полное бинарное дерево с N концевыми узлами нетрудно построить для любого N .

Мы подошли теперь к основному вопросу: нельзя ли в нисходящем методе обойтись совсем без $-\infty$? Не правда ли, было бы чудесно, если бы всю существенную информацию, имеющуюся на рис. 24, удалось расположить в ячейках 1–16 полного бинарного дерева без всяких бесполезных ”дыр”, содержащих $-\infty$? Поразмыслив, можно прийти к выводу, что эта цель в действительности достижима, причем не только исключается $-\infty$, но и появляется возможность сортировать N записей на том же месте без вспомогательной области вывода. Это приводит к еще одному важному алгоритму сортировки. Его автор Дж. У. Дж. Уильямс [САСМ, 7 (1964), 347–348] окрестил свой алгоритм ”пирамидальной сортировкой” (”heapsort”).

Пирамидальная сортировка. Будем называть файл ключей K_1, K_2, \dots, K_N ”пирамидой”, если

$$K_{\lfloor j/2 \rfloor} \geq K_j \quad \text{при } 1 \leq \lfloor j/2 \rfloor < j \leq N. \quad (3)$$

В этом случае $K_1 \geq K_2, K_1 \geq K_3, K_2 \geq K_4$ и т.д.; именно это условие выполняется на рис. 24. Из него следует, в частности, что наибольший ключ оказывается ”на вершине пирамиды”:

$$K_1 = \max(K_1, K_2, \dots, K_N). \quad (4)$$

Если бы мы сумели как-нибудь преобразовать произвольный исходный файл в пирамиду, то для получения эффективного алгоритма сортировки можно было бы воспользоваться ”нисходящей” процедурой выбора, подобной той, которая описана выше.

Эффективный подход к задаче построения пирамиды предложил Р. У. Флойд [САСМ, 7 (1964), 701]. Пусть нам удалось расположить файл таким образом, что

$$K_{\lfloor j/2 \rfloor} \geq K_j \quad \text{при } l < \lfloor j/2 \rfloor < j \leq N, \quad (5)$$

где l —некоторое число ≥ 1 . (В исходном файле это условие выполняется ”автоматически” для $l = \lfloor N/2 \rfloor$, поскольку ни один индекс j не удовлетворяет условию $\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leq N$.) Нетрудно понять, как, изменяя лишь поддерево с корнем в узле l , преобразовать файл, чтобы распространить неравенства (5) и на случай, когда $\lfloor j/2 \rfloor = l$. Следовательно, можно уменьшать l на единицу, пока в конце концов не будет достигнуто условие (3). Эти идеи Уильямса и Флойда приводят к изящному алгоритму, который заслуживает пристального изучения.

Алгоритм Н. (Пирамидальная сортировка.) Записи R_1, \dots, R_N переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \dots \leq K_N$. Сначала файл перестраивается в пирамиду, после чего вершина пирамиды многократно исключается и записывается на свое окончательное место. Предполагается, что $N \geq 2$.

Н1 [Начальная установка.] Установить $l \leftarrow \lfloor N/2 \rfloor + 1, r \leftarrow N$.

Н2 [Уменьшить l или r .] Если $l > 1$, то установить $l \leftarrow l - 1, R \leftarrow R_l, K \leftarrow K_l$. (Если $l > 1$, это означает, что происходит процесс преобразования исходного файла в пирамиду; если же $l = 1$, то это значит, что ключи K_1, K_2, \dots, K_r уже образуют пирамиду.) В противном случае установить $R \leftarrow R_r, K \leftarrow K_r, R_r \leftarrow R_1$ и $r \leftarrow r - 1$; если в результате оказалось, что $r = 1$, то установить $R_1 \leftarrow R$ и завершить работу алгоритма.

Н3 [Приготовиться к "протаскиванию".] Установить $j \leftarrow l$. (К этому моменту

$$K_{\lfloor j/2 \rfloor} \geq K_j \quad \text{при } l < \lfloor j/2 \rfloor < j \leq r, \tag{6}$$

а записи $R_k, r < k \leq N$, занимают свои окончательные места. Шаги **Н3–Н8** называются алгоритмом "протаскивания"; их

Picture: Рис. 26. Пирамидальная сортировка; пунктиром обведен алгоритм "протаскивания".

действие эквивалентно установке $R_l \leftarrow R$ с последующим перемещением записей R_l, \dots, R_r таким образом, чтобы условие (6) выполнялось и при $\lfloor j/2 \rfloor = l$.)

Н4 [Продвинуться вниз.] Установить $i \leftarrow j$ и $j \leftarrow 2j$. (В последующих шагах $i = \lfloor j/2 \rfloor$.) Если $j < r$, то перейти к шагу **Н5**; если $j = r$, то перейти к шагу **Н6**; если же $j > r$, то перейти к шагу **Н8**.

Н5 [Найти "большого" сына.] Если $K_j < K_{j+1}$, то установить $j \leftarrow j + 1$.

Н6 [Больше K ?] Если $K \geq K_j$, то перейти к шагу **Н8**.

Н7 [Поднять его вверх.] Установить $R_i \leftarrow R_j$ и возвратиться к шагу **Н4**.

Н8 [Занести R .] Установить $R_i \leftarrow R$. (На этом алгоритм "протаскивания", начатый в шаге **Н3**, заканчивается.) Возвратиться к шагу **Н2**. ■

Пирамидальную сортировку иногда описывают как \triangleleft -алгоритм; это обозначение указывает на характер изменения переменных l и r . Верхний треугольник соответствует фазе построения пирамиды, когда $r = N$, а l убывает до 1; нижний треугольник представляет фазу выбора, когда $l = 1$, а r убывает до 1. В табл. 2 показан процесс пирамидальной сортировки все тех же шестнадцати чисел. (В каждой строке изображено состояние после шага **Н2**, скобки указывают на значения переменных l и r .)

Таблица 2

Пример пирамидальной сортировки																		
K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r	K
503	087	512	061	908	170	897	[275	653	426	154	509	612	677	765	703]	8	16	275
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16	897
503	087	512	061	908	[170	897	703	653	426	154	509	612	677	765	275]	6	16	170
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16	908
503	087	512	[061	908	612	897	703	653	426	154	509	170	677	765	275]	4	16	061
503	087	[512	703	908	612	897	275	653	426	154	509	170	677	765	061]	3	16	512
503	[087	897	703	908	612	765	275	653	426	154	509	170	677	512	061]	2	16	087
[503	908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	1	16	503
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512]	908	1	15	061
[897	703	765	653	426	612	677	275	503	087	154	509	170	061]	897	908	1	14	512
[765	703	677	653	426	612	512	275	503	087	154	509	170]	765	897	908	1	13	061
[703	653	677	503	426	612	512	275	061	087	154	509]	703	765	897	908	1	12	170
[677	653	612	503	426	509	512	275	061	087	154]	677	703	765	897	908	1	11	170
[653	503	612	275	426	509	512	170	061	087]	653	677	703	765	897	908	1	10	154
[612	503	512	275	426	509	154	170	061]	612	653	677	703	765	897	908	1	9	087
[512	503	509	275	426	087	154	170]	512	612	653	677	703	765	897	908	1	8	061
[509	503	154	275	426	087	061]	509	512	612	653	677	703	765	897	908	1	7	170
[503	426	154	276	170	087]	503	509	512	612	653	677	703	765	897	908	1	6	061
[426	275	154	061	176]	426	503	509	512	612	653	677	703	765	897	908	1	5	087
[275	170	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4	087
[170	087	154]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3	061
[154	087]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2	061
[061]	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	1	061

Программа Н. (Пирамидальная сортировка.) Записи, находящиеся в ячейках с INPUT+1 по INPUT+N, сортируются при помощи алгоритма Н; регистры принимают следующие значения: rI1 \equiv l - 1, rI2 \equiv r - 1, rI3 \equiv i, rI4 \equiv j, rI5 \equiv r - l, rA \equiv K \equiv R, rX \equiv R_j.

START	ENT1	N/2	1	Н1. Начальная установка $l \leftarrow \lfloor N/2 \rfloor + 1$.
	ENT2	N-1	1	$r \leftarrow N$.
1H	DEC1	1	$\lfloor N/2 \rfloor$	$l \leftarrow l - 1$.
	LDA	INPUT+1, 1	$\lfloor N/2 \rfloor$	$R \leftarrow R_l, K \leftarrow K_l$.
3H	ENT4	1, 1	P	Н3. Приготовиться к "протаскиванию". $j \leftarrow l$.
	ENT5	0, 2	P	
	DEC5	0, 1	P	rI5 $\leftarrow r - j$.
	JMP	4P	P	К шагу Н4.
5H	LDX	INPUT, 4	B + A - D	Н5. Найти "большого" сына.
	CMPX	INPUT+1, 4	B + A - D	
	JOE	6F	B + A - D	Переход, если $K_j \geq K_{j+1}$.
	INC4	1	C	В противном случае установить $j \leftarrow j + 1$.
	DECS	1	C	
9H	LDX	INPUT, 4	C + D	rX $\leftarrow R_j$.
6H	CMPA	INPUT, 4	B + A	Н6. Больше K?
	JGE	8F	B + A	К Н8, если $K \geq K_j$.
7H	STX	INPUT, 3	B	Н7. Поднять его вверх. $R_i \leftarrow R_j$.
4H	ENT3	0, 4	B + P	Н4. Продвинуться вниз.
	DEC5	0, 4	B + P	rI5 $\leftarrow rI5 - j$.
	INC4	0, 4	B + P	$j \leftarrow j + j$.
	J5P	5B	B + P	К Н5, если $j < r$.
	J5Z	9B	P - A + D	К Н6, если $j = r$.
8H	STA	INPUT, 3	P	Н8. Занести R. $R_i \leftarrow R$.
2H	J1P	1B	P	Н2. Уменьшить l или r.
	LDA	INPUT+1, 2	N - 1	Если $l = 1$, то установить $R \leftarrow R_r, K \leftarrow K_r$.
	LDX	INPUT+1	N - 1	
	STX	INPUT+1, 2	N - 1	$R_r \leftarrow R_1$.
	DEC2	1	N - 1	$r \leftarrow r - 1$.
	J2P	3B	N - 1	К Н3, если $r > 1$.
	STA	INPUT+1	1	$R_1 \leftarrow R$.

Эта программа приблизительно лишь вдвое длиннее программы S, но при больших N она гораздо более эффективна. Ее время работы зависит от

$P = N + \lfloor N/2 \rfloor - 2 =$ число "протаскиваний";

$A =$ число протаскиваний, при которых ключ K в конце попадает во внутренний узел пирамиды;

$B =$ суммарное число ключей, просмотренных во время протаскиваний;

$C =$ число присваиваний $j \leftarrow j + 1$ в шаге Н5;

$D =$ число случаев, когда в шаге Н4 $j = r$.

Эти величины проанализированы ниже. Как показывает практика, они сравнительно мало отклоняются от своих средних значений:

$$\begin{aligned} A &\approx 0.349N; & B &\approx N \log_2 N - 1.87N; \\ C &\approx \frac{1}{2}N \log_2 N - 0.9N; & D &\approx \ln N. \end{aligned} \quad (7)$$

При $N = 1000$, например, четыре эксперимента со случайными исходными данными показали соответственно результаты $(A, B, C, D) = (371, 8055, 4056, 12), (351, 8072, 4087, 14), (341, 8094, 4017, 8), (340, 8108, 4083, 13)$. Общее время работы $7A + 14B + 4C + 20N - 2D + 15\lfloor N/2 \rfloor - 28$ равно, таким образом, в среднем примерно $16N \log_2 N + 0.2N$ единиц.

Глядя на табл. 2, трудно поверить в то, что пирамидальная сортировка так уж эффективна: большие ключи перемещаются влево прежде, чем мы успеваем отложить их вправо! Это и в самом

деле странный способ сортировки при малых N . Время сортировки 16 ключей из табл. 2 равно $1068u$, тогда как обычный метод простых вставок (программа 5.2.1S) требует всего $514u$. При сортировке простым выбором (программа S) требуется $853u$.

При больших N программа H более эффективна. Напрашивается сравнение с сортировкой методом Шелла с убывающим шагом (программа 5.2.1D) и быстрой сортировкой Хоара (программа 5.2.2Q), так как во всех трех программах сортировка производится путем сравнения ключей, причем вспомогательной памяти используется мало или она не используется вовсе. При $N = 1000$ средние времена работы равны приблизительно

160000 u для пирамидальной сортировки;
130000 u для сортировки методом Шелла;
80000 u для быстрой сортировки.

(MIX—типичный представитель большинства современных вычислительных машин, но, разумеется, на конкретных машинах получатся несколько иные относительные величины.) С ростом N пирамидальная сортировка превзойдет по скорости метод Шелла, но асимптотическая формула $16N \log_2 N \approx 23.08N \ln N$ никогда не станет лучше выражения для быстрой сортировки $12.67N \ln N$. Модификация пирамидальной сортировки, обсуждаемая в упр. 18, ускорит процесс на многих вычислительных машинах, но даже с этим усовершенствованием пирамидальная сортировка не достигнет скорости быстрой сортировки.

С другой стороны, быстрая сортировка эффективна лишь в среднем; в наихудшем случае ее время работы пропорционально N^2 . Пирамидальная же сортировка обладает тем интересным свойством, что для нее наихудший случай не намного хуже среднего. Всегда выполняются неравенства

$$A \leq 1.5N, \quad B \leq N \lfloor \log_2 N \rfloor, \quad C \leq N \lfloor \log_2 N \rfloor; \quad (8)$$

таким образом, независимо от распределения исходных данных выполнение программы H не займет более $18N \lfloor \log_2 N \rfloor + 38N$ единиц времени. Пирамидальная сортировка—первый из рассмотренных нами до сих пор методов сортировки, время работы которого *заведомо* имеет порядок $N \log N$. Сортировка посредством слияний, которая будет обсуждаться ниже, в п. 5.2.4, тоже обладает этим свойством, но она требует больше памяти.

Наибольший из включенных—первым исключается. В гл. 2 мы видели, что линейные списки часто можно осмысленно расклассифицировать по характеру производимых над ними операций включения и исключения. *Стек* ведет себя по принципу ”последним включается—первым исключается” в том смысле, что при каждом исключении удаляется самый молодой элемент списка (элемент, который был вставлен позже всех других элементов, присутствующих в данный момент в списке). Простая *очередь* ведет себя по принципу ”первым включается—первым исключается” в том смысле, что при каждом исключении удаляется самый старший из имеющихся элементов. В более сложных ситуациях, таких, как моделирование лифта в п. 2.2.5, требуется список типа ”наименьший из включенных—первым исключается”, где при каждом исключении удаляется элемент, имеющий наименьший ключ. Такой список можно назвать *приоритетной очередью*, так как ключ каждого элемента отражает его относительную способность быстро покинуть список. Сортировка посредством выбора—частный случай приоритетной очереди, над которой производится сначала N операций вставки, а затем N операций удаления.

Приоритетные очереди возникают в самых разнообразных приложениях. Например, в некоторых численных итеративных схемах повторяется выбор элемента, имеющего наибольшее (или наименьшее) значение некоторого проверочного критерия; параметры выбранного элемента изменяются, и он снова вставляется в список с новым проверочным значением, — соответствующим новым значениям параметров. Приоритетные очереди часто используются в операционных системах при планировании заданий. Другие типичные применения приоритетных очередей упоминаются в упр. 15, 29 и 36; кроме того, много примеров встретится в последующих главах.

Как же реализовать приоритетную очередь? Один из очевидных способов—поддерживать отсортированный список элементов, упорядоченных по ключам. Тогда включение нового элемента, по существу, сводится к задаче, рассмотренной нами при изучении сортировки вставками в п. 5.2.1. При другом, еще более очевидном способе работы с приоритетной очередью элементы в списке хранятся в произвольном порядке, и тогда для выбора нужного элемента, приходится осуществлять поиск наибольшего (или наименьшего) ключа каждый раз, когда необходимо произвести исключение. В обоих этих очевидных подходах неприятность состоит в том, что требуется порядка N шагов для выполнения либо операции вставки, либо операции удаления, если в списке содержится N элементов, т. е. при больших N эти операции занимают слишком много времени.

В своей статье о пирамидальной сортировке Уильяме указал на то, что пирамиды идеально подходят для приложений с большими приоритетными очередями, так как элемент можно вставить в пирамиду или удалить из нее за $O(\log N)$ шагов; к тому же все элементы пирамиды компактно располагаются в последовательных ячейках памяти. Фаза выбора в алгоритме Н—это последовательность шагов удаления в процессе типа *наибольший из включенных—первым исключается*: чтобы исключить наибольший элемент K_1 мы удаляем его и ”протаскиваем” элемент K_N в новой пирамиде из $N - 1$ элементов. (Если нужен алгоритм типа *наименьший из включенных—первым исключается*, как при моделировании лифта, то, очевидно, можно изменить определение пирамиды, заменив в (3) знак ” \geq ” на ” \leq ”; для удобства мы будем рассматривать здесь лишь случай ”наибольший из включенных—первым исключается”.) Вообще, если требуется исключить наибольший элемент, а затем вставить новый элемент x , то можно выполнить процедуру протаскивания с $l = 1$, $r = N$ и $K = x$. Если же необходимо вставить элемент без предварительного исключения, то можно воспользоваться ”восходящей” процедурой из упр. 16.

Связанное представление приоритетных очередей. Эффективный способ представления приоритетных очередей в виде связанных бинарных деревьев предложил в 1971 г. Кларк Э. Крэйн. Его метод требует наличия в каждой записи двух полей связи и короткого поля счетчика, но по сравнению с пирамидами он обладает следующими преимуществами:

- 1) Если с приоритетной очередью работают как со стекком, то операции включения и исключения более эффективны (они занимают фиксированное время, не зависящее от длины очереди).
- 2) Записи никогда не перемещаются, изменяются только указатели.
- 3) Можно слить две непересекающиеся приоритетные очереди, содержащие в общей сложности N элементов, в одну всего за $O(\log N)$ шагов.

Слегка видоизмененный метод Крэйна проиллюстрирован на рис. 27, на котором показан особый тип структуры бинарного дерева. Каждый узел содержит поле KEY, поле DIST и два поля связи—LEFT и RIGHT. Поле DIST всегда устанавливается равным длине кратчайшего пути от этого узла до концевой узла (т. е. до пустого узла Λ) дерева. Если считать, что $\text{DIST}(\Lambda) = 0$ и $\text{KEY}(\Lambda) = -\infty$, то поля KEY и DIST в этом дереве удовлетворяют следующим соотношениям:

$$\text{KEY}(P) \geq \text{KEY}(\text{LEFT}(P)), \text{KEY}(P) \geq \text{KEY}(\text{RIGHT}(P)); \quad (9)$$

$$\text{DIST}(P) = 1 + \min(\text{DIST}(\text{LEFT}(P)), \text{DIST}(\text{RIGHT}(P))); \quad (10)$$

$$\text{DIST}(\text{LEFT}(P)) \geq \text{DIST}(\text{RIGHT}(P)). \quad (11)$$

Соотношение (9) аналогично условию пирамиды (3) и служит гарантией того, что в корне дерева находится наибольший ключ, а соотношение (10)—это просто определение поля DIST, сформулированное выше. Соотношение (11) представляет собой интересное новшество: из него следует, что кратчайший путь к концевому узлу всегда можно получить, двигаясь вправо. Мы

Picture: Рис. 27. Приоритетная очередь, представленная в виде левостороннего дерева.

будем называть бинарное дерево с этим свойством левосторонним деревом, поскольку оно, как правило, сильно ”тянется” влево.

Из этих определений ясно, что равенство $\text{DIST}(P) = n$ подразумевает существование по крайней мере 2^n концевых узлов ниже P ; в противном случае нашелся бы более короткий путь от P до концевой узла. Таким образом, если в левостороннем дереве имеется N узлов, то путь, ведущий из корня вниз по направлению вправо, содержит не более чем $\lfloor \log_2(N + 1) \rfloor$ узлов. Новый узел можно вставить в приоритетную очередь, пройдя по этому пути (см. упр. 32); следовательно, в худшем случае необходимо всего $O(\log N)$ шагов. Наилучший случай достигается, когда дерево линейно (все связи RIGHT равны Λ), а наихудший случай достигается, когда дерево абсолютно сбалансировано.

Чтобы удалить узел из корня, нужно просто слить два его поддерева. Операция слияния двух непересекающихся левосторонних деревьев, на которые ссылаются указатели P и Q , по своей идее проста: если $\text{KEY}(P) \geq \text{KEY}(Q)$, то берем в качестве корня P и сливаем Q с правым поддеревом P ; при этом изменится $\text{DIST}(P)$, а $\text{LEFT}(P)$ меняется местами с $\text{RIGHT}(P)$, если это необходимо. Нетрудно составить подробное описание этого процесса (см. упр. 32).

Сравнение методов работы с приоритетными очередями. Если число узлов N мало, то для поддержания приоритетной очереди лучше всего применять один из простых методов с использованием линейных списков. Если же N велико, то, очевидно, гораздо более быстрым будет метод, время работы

которого порядка $\log N$. Поэтому большие приоритетные очереди обычно представляют в виде пирамид или левосторонних деревьев. В п. 6.2.3 мы обсудим еще один способ представления линейных списков в виде *сбалансированных деревьев*, который приводит к третьему методу, пригодному для представления приоритетных очередей, с временем работы порядка $\log N$. Поэтому уместно сравнить эти три метода.

Мы видели, что операции над левосторонними деревьями в целом несколько быстрее, чем операции над пирамидами, но пирамиды занимают меньше памяти. Сбалансированные деревья занимают примерно столько же памяти, сколько левосторонние деревья (быть может, чуть меньше); операции над ними медленнее, чем над пирамидами, а программирование сложнее, но структура сбалансированных деревьев в некоторых отношениях существенно более гибкая. Работая с пирамидами, не так просто предсказать, что произойдет с элементами, если у них равные ключи; нельзя гарантировать, что элементы с равными ключами будут обрабатываться по принципу ”последним включается—первым исключается” или ”первым включается—первым исключается”, если только ключ не расширен и не содержит дополнительного поля ”порядковый номер вставки”, и тогда равных ключей просто нет. С другой стороны, если применять сбалансированные деревья, можно легко оговорить твердые условия относительно равных ключей. Можно также выполнять такие действия, как ”вставить x непосредственно перед (или после) y ”. Сбалансированные деревья симметричны, так что в любой момент можно исключить либо наибольший, либо наименьший элемент, в то время как левосторонние деревья и пирамиды должны быть так или иначе ориентированы. (См. тем не менее упр. 31, в котором

Picture: Рис. 28. Так выглядит пирамида...

показано, как строить *симметричные* пирамиды.) Сбалансированные деревья можно использовать как для поиска, так и для сортировки; и из сбалансированного дерева можно довольно быстро удалять последовательные блоки элементов. Но два сбалансированных дерева нельзя слить менее чем за $O(N)$ шагов, в то время как два левосторонних дерева можно слить всего за $O(\log N)$ шагов.

Итак, пирамиды наиболее экономны с точки зрения памяти; левосторонние деревья хороши тем, что можно быстро слить две непересекающиеся приоритетные очереди; и, если нужно, за умеренное вознаграждение можно получить ту гибкость, какую предоставляют сбалансированные деревья.

* **Анализ пирамидальной сортировки.** Алгоритм Н до сих пор не был полностью проанализирован, но некоторые его свойства можно вывести без особого труда. Поэтому мы завершим этот пункт довольно подробным исследованием, касающимся пирамид.

На рис. 28 показана форма пирамиды из 26 элементов; каждый узел помечен двоичным числом, соответствующим его индексу в пирамиде. Звездочками в этой диаграмме помечены так называемые *особые узлы*, которые лежат на пути от 1 к N .

Одна из наиболее важных характеристик пирамиды—набор размеров ее поддеревьев. Например, на рис. 28 размеры поддеревьев с корнями в узлах 1, 2, ..., 26 равны соответственно

$$26^*, 15, 10^*, 7, 7, 6^*, 3, 3, 3, 3, 3, 3, 2^*, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1^*. \quad (12)$$

Звездочками отмечены *особые поддеревья* с корнями в особых узлах; в упр. 20 показано, что если N имеет двоичное представление

$$N = (b_n b_{n-1} \dots b_1 b_0)_2, \quad n = \lfloor \log_2 N \rfloor, \quad (13)$$

то размеры особых поддеревьев равны

$$(1b_{n-1} \dots b_1 b_0)_2, (1b_{n-2} \dots b_1 b_0)_2, \dots, (1b_1 b_0)_2, (1b_0)_2, (1)_2. \quad (14)$$

(Неособые поддеревья всегда абсолютно сбалансированы, так что их размеры всегда имеют вид $2^k - 1$. В упр. 21 показано, что среди неособых поддеревьев имеется ровно

$$\begin{aligned} \lfloor (N-1)/2 \rfloor \text{ размера } 1, & \quad \lfloor (N-2)/4 \rfloor \text{ размера } 3, \\ \lfloor (N-4)/8 \rfloor \text{ размера } 7, & \quad \dots \quad \lfloor (N-2^{n-1})/2^n \rfloor \text{ размера } (2^n - 1), \end{aligned} \quad (15)$$

Например, на рис. 28 изображено двенадцать неособых поддеревьев размера 1, шесть поддеревьев размера 3, два—размера 7 и одно—размера 15.

Пусть s_l —размер поддерева с корнем l , а M_N —мультимножество $\{s_1, s_2, \dots, s_N\}$ всех этих размеров. Используя (14) и (15), легко вычислить M_N при любом заданном N . В упр. 5.1.4–20 показано, что общее число способов построить пирамиду из целых чисел $\{1, 2, \dots, N\}$ равно

$$N!/s_1 s_2 \dots s_N = N! / \prod_{s \in M_N} s. \quad (16)$$

Например, число способов расположить 26 букв $\{A, B, C, \dots, Z\}$ на рис. 28 так, чтобы по вертикали сохранялся алфавитный порядок, равно

$$26! / (26 \cdot 10 \cdot 6 \cdot 2 \cdot 1 \cdot 1^{12} \cdot 3^6 \cdot 7^2 \cdot 15^1).$$

Теперь мы в состоянии проанализировать фазу построения пирамиды в алгоритме Н, т. е. вычисления, которые завершаются до того, как в шаге Н2 впервые выполнится условие $l = 1$. К счастью, благодаря следующей ниже теореме анализ построения пирамиды можно свести к изучению независимых операций протаскивания.

Теорема Н. Если исходными данными для алгоритма Н служит случайная перестановка множества $\{1, 2, \dots, N\}$, то в фазе построения пирамиды с одинаковой вероятностью может получиться любая из $N! / (\prod_{s \in M_N} s)$ возможных пирамид. Более того, все $\lfloor N/2 \rfloor$ операций протаскивания, выполненные за время этой фазы, "равномерны" в том смысле, что по достижении шага Н8 все s_i возможных значений переменной i равновероятны.

Доказательство. Применим метод, который в численном анализе называется методом "обратной задачи". Пусть в качестве одного из возможных результатов операции протаскивания задана пирамида $K_1 \dots K_N$ с корнем в узле l ; тогда ясно, что имеется всего s_l исходных конфигураций $K'_1 \dots K'_N$ файла, которые после протаскивания дают такой результат. Все эти исходные конфигурации имеют различные значения K'_l , следовательно, рассуждая в обратном направлении, существует ровно $s_l s_{l+1} \dots s_N$ исходных перестановок множества $\{1, 2, \dots, N\}$, которые после завершения операции протаскивания в позицию l дают конфигурацию $K_1 \dots K_N$.

Случай $l = 1$ типичен: пусть $K_1 \dots K_N$ — пирамида, и пусть $K'_1 \dots K'_N$ — файл, который преобразуется в $K_1 \dots K_N$ в результате протаскивания при $l = 1$, $K = K'_1$. Если $K = K_i$, то должны иметь место равенства $K'_i = K_{\lfloor i/2 \rfloor}$, $K'_{\lfloor i/2 \rfloor} = K_{\lfloor i/4 \rfloor}$ и т. д., при этом $K'_j = K_j$ для всех j , не лежащих на пути от 1 к i . Обратно, при любом i в результате такого построения получается файл $K'_1 \dots K'_N$, такой, что (а) операция протаскивания преобразует файл $K'_1 \dots K'_N$ в $K_1 \dots K_N$ и (б) $K_{\lfloor j/2 \rfloor} \geq K_j$ при $2 \leq \lfloor j/2 \rfloor < j \leq N$. Следовательно, возможны ровно N таких файлов $K'_1 \dots K'_N$, и операция протаскивания равномерна. (Пример доказательства этой теоремы см. в упр. 22.) ■

Обращаясь к величинам A, B, C, D в анализе программы Н, можно видеть, что равномерная операция протаскивания, производимая над поддеревом размера s , дает вклад $\lfloor s/2 \rfloor / s$ в среднее значение величины A ; ее вклад в среднее значение величины B равен

$$\frac{1}{s}(0 + 1 + 1 + 2 + \dots + \lfloor \log_2 s \rfloor) = \frac{1}{s} \left(\sum_{1 \leq k \leq s} \lfloor \log_2 k \rfloor \right) = \frac{1}{s} ((s+1) \lfloor \log_2 s \rfloor - 2^{\lfloor \log_2 s \rfloor + 1} + 2)$$

(см. упр. 1.2.4–42); и она дает вклад $2/s$ или 0 в среднее значение величины D в зависимости от того, является ли s четным или нечетным. Несколько сложнее определить соответствующий вклад в среднее значение величины C , так что эта задача предоставляется читателю (см. упр. 26). Производя суммирование по всем операциям протаскивания, находим, что среднее значение величины A за время построения пирамиды равно

$$A'_N = \sum_{s \in M_N} \lfloor s/2 \rfloor / s; \quad (17)$$

аналогичные формулы имеют место и для B, C и D , так что можно без особого труда точно вычислить эти средние значения. В следующей таблице приведены типичные результаты:

N	A'_N	B'_N	C'_N	D'_N
99	19.18	68.35	42.95	0.00
100	19.93	69.39	42.71	1.84
999	196.16	734.66	464.53	0.00
1000	196.94	735.80	464.16	1.92
9999	1966.02	7428.18	4695.54	0.00
10000	1966.82	7429.39	4695.06	1.97
10001	1966.45	7430.07	4695.84	0.00
10002	1967.15	7430.97	4695.95	1.73

Что касается асимптотики, то в M_N можно не обращать внимания на размеры особых поддеревьев, и тогда мы найдем, например, что

$$A'_N = \frac{N}{2} \cdot \frac{0}{1} + \frac{N}{4} \cdot \frac{1}{3} + \frac{N}{8} \cdot \frac{3}{7} + \dots + O(\log N) = \left(1 - \frac{1}{2}\alpha\right) N + O(\log N), \quad (18)$$

где

$$\alpha = \sum_{k \geq 1} \frac{1}{2^k - 1} = 1.60669\ 51524\ 15291\ 76378\ 33015\ 23190\ 92458\ 04805\text{—}. \quad (19)$$

(Это значение получил Дж. У. Ренч младший, пользуясь преобразованием ряда из упр. 27.) При больших N можно использовать приближенные формулы

$$\begin{aligned} A'_N &\approx 0.1967N + 0.3 \quad (N \text{ чет.}), \quad 0.1967N - 0.3 \quad (N \text{ нечет.}); \\ B'_N &\approx 0.74403N - 1.3 \ln N; \\ C'_N &\approx 0.47034N - 0.8 \ln N; \\ D'_N &\approx 1.8 \pm 0.2 \quad (N \text{ чет.}), \quad 0 \quad (N \text{ нечет.}). \end{aligned} \quad (20)$$

Нетрудно определить также максимальные и минимальные значения. Для построения пирамиды требуется всего $O(N)$ шагов (ср. с упр. 23).

Этим, по существу, завершается анализ фазы построения пирамиды в алгоритме Н. Анализ фазы выбора—совсем другая задача, которая еще ожидает своего решения! Пусть пирамидальная сортировка применяется к N элементам; обозначим через A''_N , B''_N , C''_N и D''_N средние значения величин A , B , C и D во время фазы выбора. Поведение алгоритма Н подвержено сравнительно малым колебаниям около эмпирически установленных средних значений

$$\begin{aligned} A''_N &\approx 0.152N; \\ B''_N &\approx N \log_2 N - 2.61N; \\ C''_N &\approx \frac{1}{2}N \log_2 N - 1.4N; \\ D''_N &\approx \ln N \pm 2; \end{aligned} \quad (21)$$

тем не менее, до сих пор не найдено правильного теоретического объяснения этим константам. Рассмотрев отдельные операции протаскивания, нетрудно вывести верхние оценки, указанные в неравенствах (8), хотя, если рассматривать алгоритм как целое, верхнюю оценку для C , по-видимому, следует уменьшить приблизительно до $\frac{1}{2}N \log_2 N$.

Упражнения

1. [10] Является ли сортировка посредством простого выбора (алгоритм S) "устойчивой"?
2. [15] Почему в алгоритме S оказывается более удобным находить сначала наибольший элемент, затем наибольший из оставшихся и т. д., вместо того чтобы находить сначала наименьший элемент, затем наименьший из оставшихся и т. д.?
3. [M21] (а) Докажите, что если алгоритм S применяется к случайной перестановке множества $\{1, 2, \dots, N\}$, то в результате первого выполнения шагов S2 и S3 получается случайная перестановка множества $\{1, 2, \dots, N-1\}$, за которой следует N . (Иначе говоря, файл $K_1 \dots K_{N-1}$ с одинаковой вероятностью может быть любой перестановкой множества $\{1, 2, \dots, N-1\}$.) (б) Следовательно, если через B_N обозначить среднее значение величины B в программе S, то при условии, что исходный файл упорядочен случайным образом, имеем $B_N = H_N - 1 + B_{N-1}$. [Указание: ср. со статистическими характеристиками 1.2.10–16.]
- >4. [M35] В шаге S3 алгоритма S ничего не делается, если $i = j$. Не лучше ли перед выполнением шага S3 проверить условие $i = j$? Чему равно среднее число случаев выполнения условия $i = j$ в шаге S3, если исходный файл случаен?
5. [20] Чему равно значение величины B в анализе программы S для исходного файла $N \dots 3\ 2\ 1$?
6. [M29] (а) Пусть $a_1 a_2 \dots a_N$ —перестановка множества $\{1, 2, \dots, N\}$ с C циклами, I инверсиями и такая, что при ее сортировке с помощью программы S производится B обменов на правосторонний максимум. Докажите, что $2B \leq I + N - C$. [Указание: см. упр. 5.2.2–1.] (б) Покажите, что $I + N - C \leq \lfloor n^2/2 \rfloor$; следовательно, B не превышает $\lfloor n^2/4 \rfloor$.
7. [M46] Найдите дисперсию величины B в программе S как функцию от N , считая, что исходный файл случаен.
8. [24] Покажите, что если при поиске $\max(K_1, \dots, K_j)$ в шаге S2 просматривать ключи слева направо: K_1, K_2, \dots, K_j , а не наоборот: K_j, \dots, K_2, K_1 , как в программе S, то за счет этого можно было бы сократить число сравнений при следующих повторениях шага S2. Напишите MIX-программу, основанную на этом наблюдении.
9. [M25] Чему равно среднее число сравнений, выполняемых алгоритмом из упр. 8 для случайного исходного файла?

10. [12] Как будет выглядеть дерево, изображенное на рис. 23, после того как будут выведены 14 из 16 первоначальных элементов?
11. [10] Как будет выглядеть дерево, изображенное на рис. 24, после вывода элемента 908?
12. [M20] Сколько раз будет выполнено сравнение $-\infty$ с ∞ , если применить "восходящий" метод, представленный на рис. 23, для упорядочения файла из 2^n элементов?
13. [20] (Дж. У. Дж. Уильяме.) В шаге H4 алгоритма H различаются три случая: $j < r$, $j = r$ и $j > r$. Покажите, что если $K \geq K_{r+1}$, то можно было бы так упростить шаг H4, чтобы разветвление происходило лишь по двум путям. Как надо изменить шаг H2, чтобы обеспечить в процессе пирамидальной сортировки выполнение условия $K \geq K_{r+1}$?
14. [10] Покажите, что простая очередь—частный случай приоритетной. (Объясните, какие ключи нужно присваивать элементам, чтобы процедура "наибольший из включенных—первым исключается" была эквивалентна процедуре "первым включается—первым исключается".) Является ли стек также частным случаем приоритетной очереди?
- >15. [M22] (В. Э. Чартрс.) Придумайте быстрый алгоритм построения таблицы простых чисел $\leq N$, в котором используется *приоритетная очередь* с целью избежать операций деления. [Указание. Пусть наименьший ключ в приоритетной очереди будет наименьшим нечетным непростым числом, большим, чем самое последнее нечетное число, воспринятое как кандидат в простые числа. Попытайтесь свести к минимуму число элементов в этой очереди.]
16. [20] Постройте эффективный алгоритм, который вставляет новый ключ в данную пирамиду из n элементов, порождая пирамиду из $n + 1$ элементов.
17. [20] Алгоритм из упр. 16 можно использовать для построения пирамиды взамен метода "уменьшения l до 1", применяемого в алгоритме H.
Порождают ли оба метода из одного и того же исходного файла одну и ту же пирамиду?
- >18. [21] (Р. У. Флойд) Во время фазы выбора в алгоритме пирамидальной сортировки ключ K , как правило, принимает довольно малые значения, и поэтому почти при всех сравнениях в шаге H6 обнаруживается, что $K < K_j$. Как можно изменить алгоритм, чтобы ключ K не сравнивался с K_j в основном цикле вычислений?
19. [21] Предложите алгоритм исключения данного элемента из пирамиды размера N , порождающий пирамиду размера $N - 1$.
20. [M20] Покажите, что формулы (14) задают размеры особых поддеревьев пирамиды.
21. [M24] Докажите, что формулы (15) задают размеры неособых поддеревьев пирамиды.
- >22. [20] Какие перестановки множества $\{1, 2, 3, 4, 5\}$ фаза построения пирамиды в алгоритме H преобразует в 5 3 4 1 2?
23. [M28] (а) Докажите, что длина пути B в алгоритме протаскивания никогда не превышает $\lceil \log_2(r/l) \rceil$.
(б) Согласно неравенствам (8), ни при каком конкретном применении алгоритма H величина B не может превзойти $N \lceil \log_2 N \rceil$. Найдите максимальное значение B по всевозможным исходным файлам как функцию от N . (Вы должны доказать, что существует исходный файл, на котором B принимает это максимальное значение.)
24. [M24] Выведите точную формулу стандартного отклонения величины B'_N (суммарная длина пути, пройденного по дереву во время фазы построения пирамиды в алгоритме H).
25. [M20] Чему равен средний вклад в значение величины C за время первой операции протаскивания, когда $l = 1$, а $r = N$, если $N = 2^{n+1} - 1$.
26. [M30] Решите упр. 25: (а) для $N = 26$, (б) для произвольного N .
27. [M25] (Дж. У. Ренч мл.) Докажите, что $\sum_{n \geq 1} x^n / (1 - x^n) = \sum_{n \geq 1} x^{n^2} (1 + x^n) / (1 - x^n)$. [Положив $x = \frac{1}{2}$, получите очень быстро сходящийся ряд для вычисления (19).]
28. [35] Продумайте идею *тернарных пирамид*, основанных на полных тернарных, а не бинарных деревьях. Будет ли тернарная пирамидальная сортировка быстрее бинарной?
29. [26] (У. С. Браун.) Постройте алгоритм умножения многочленов или степенных рядов $(a_1 x^{i_1} + a_2 x^{i_2} + \dots)(b_1 x^{j_1} + b_2 x^{j_2} + \dots)$, который бы порождал коэффициенты произведения $c_1 x^{i_1+j_1} + \dots$ по порядку, по мере того как перемножаются коэффициенты исходных многочленов. [Указание: воспользуйтесь подходящей приоритетной очередью.]
30. [M48] Может ли величина C превзойти $\frac{1}{2} N \log_2 N$ при пирамидальной сортировке файла? Чему равно максимальное значение C ? Чему равно минимальное значение?
31. [37] (Дж. У. Дж. Уильяме.) Покажите, что если две пирамиды подходящим образом совместить "основание к основанию", то это даст возможность поддерживать структуру, в которой в любой момент можно за $O(\log n)$ шагов исключить либо наибольший, либо наименьший элемент. (Такую структуру можно назвать *приоритетным деком*.)

32. [21] Разработайте алгоритм слияния двух непересекающихся приоритетных очередей, представленных в виде левосторонних деревьев, в одну. (В частности, если одна из данных очередей содержит всего один элемент, то ваш алгоритм будет вставлять его в другую очередь.)
- >33. [15] Почему в приоритетной очереди, представленной в виде левостороннего дерева, операция удаления корня выполняется путем слияния двух поддеревьев, а не "продвижения" узлов по направлению к вершине, как в пирамиде?
34. [M47] Сколько можно построить левосторонних деревьев из N узлов, если игнорировать значения поля KEY? [Эта последовательность начинается с чисел 1, 1, 2, 4, 8, 17, 38, ...; существует ли какая-нибудь простая асимптотическая формула?]
35. [26] Если в левостороннее дерево с N узлами добавить связи UP (ср. с обсуждением деревьев с тремя связями в п. 6.2.3), то это даст возможность исключать из приоритетной очереди произвольный узел P следующим образом: слить LEFT(P) и RIGHT(P) и поместить полученное поддерево на место P , затем исправлять поля DIST у предков узла P до тех пор, пока не будет достигнут либо корень, либо узел, у которого поле DIST не меняется.
Докажите, что при этом никогда не потребуется изменить более чем $O(\log N)$ полей DIST, несмотря даже на то, что дерево может содержать очень длинные восходящие пути.
36. [18] (*Замещение наиболее давно использованной страницы.*) Многие операционные системы используют алгоритм следующего типа: над набором узлов допустимы две операции — "использование" узла и замещение наиболее давно "использованного" узла новым узлом. Какая структура данных облегчает нахождение наиболее давно "использованного" узла?

5.2.4. Сортировка слиянием

Слияние означает объединение двух или более упорядоченных файлов в один упорядоченный файл. Можно, например, слить два подфайла—503 703 765 и 087 512 677, получив 087 503 512 677 703 765. Простой способ сделать это—сравнить два наименьших элемента, вывести наименьший из них и повторить эту процедуру.

Начав с

$$\begin{cases} 503 & 703 & 765 \\ 087 & 512 & 677 \end{cases},$$

получим

$$087 \begin{cases} 503 & 703 & 765 \\ 512 & 677 \end{cases}$$

затем

$$087 \ 503 \begin{cases} 703 & 765 \\ 512 & 677 \end{cases}$$

и т. д. Необходимо позаботиться о действиях на случай, когда исчерпается один из файлов. Весь процесс подробно описан в следующем алгоритме.

Алгоритм М. (*Двухпутевое слияние.*) Этот алгоритм осуществляет слияние двух упорядоченных файлов $x_1 \leq x_2 \leq \dots \leq x_m$ и $y_1 \leq y_2 \leq \dots \leq y_n$ в один файл $z_1 \leq z_2 \leq \dots \leq z_{m+n}$.

M1 [Начальная установка.] Установить $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$.

M2 [Найти наименьший элемент.] Если $x_i \leq y_j$, то перейти к шагу **M3**; в противном случае перейти к **M5**.

M3 [Вывести x_i .] Установить $z_k \leftarrow x_i, k \leftarrow k + 1, i \leftarrow i + 1$. Если $i \leq m$, то возвратиться к **M2**.

M4 [Вывести y_j, \dots, y_n .] Установить $(z_k, \dots, z_{m+n}) \leftarrow (y_j, \dots, y_n)$ и завершить работу алгоритма.

M5 [Вывести y_j .] Установить $z_k \leftarrow y_j, k \leftarrow k + 1, j \leftarrow j + 1$. Если $j \leq n$, то возвратиться к **M2**.

M6 [Вывести x_i, \dots, x_m .] Установить $(z_k, \dots, z_{m+n}) \leftarrow (x_i, \dots, x_m)$ и завершить работу алгоритма. ■

Picture: Рис. 29. Слияние $x_1 \leq \dots \leq x_m$ с $y_1 \leq \dots \leq y_n$.

В п. 5.3.2 мы увидим, что эта простая процедура, по существу, "наилучший из возможных" способов слияния на традиционной ЭВМ, если $m \approx n$. (Но, если m гораздо меньше n , можно разработать более эффективные алгоритмы сортировки, хотя в общем случае они довольно сложны.) Алгоритм М без особой потери эффективности можно немного упростить, добавив в конец исходных файлов искусственных "стражей" $x_{m+1} = y_{n+1} = \infty$ и останавливаясь перед выводом ∞ . Анализ алгоритма М см. в упр. 2.

Общий объем работы, выполняемой алгоритмом М, по существу, пропорционален $m + n$, поэтому ясно, что слияние—более простая задача, чем сортировка. Однако задачу сортировки можно свести к

слияниям, сливая все более длинные подфайлы до тех пор, пока не будет отсортирован весь файл. Такой подход можно рассматривать как развитие идеи сортировки вставками: вставка нового элемента в упорядоченный файл—частный случай слияния при $n = 1$! Если нужно ускорить процесс вставок, то можно рассмотреть вставку нескольких элементов за раз, ”группируя” их, а это естественным образом приводит к общей идее сортировки слиянием. С исторической точки зрения метод слияний—один из самых первых методов, предназначенных для сортировки

Picture: Таблица 1. Сортировка естественным двухпутевым слиянием

на ЭВМ; он был предложен Джоном фон Нейманом еще в 1945 г. (см. § 5.5).

Мы довольно подробно изучим слияния в § 5.4 в связи с алгоритмами внешней сортировки, а в настоящем пункте сосредоточим свое внимание на сортировке в быстрой памяти с произвольным доступом.

Таблица 1 иллюстрирует сортировку слиянием, когда ”свечка сжигается с обоих концов”, подобно тем процедурам просмотра элементов файла, которые применялись при быстрой сортировке, поразрядной обменной сортировке и т. д. Мы анализируем исходный файл слева и справа, двигаясь к середине. Пропустим пока первую строку и рассмотрим переход от второй строки к третьей. Слева мы видим возрастающий отрезок 503 703 765, а справа, если читать справа налево, имеем отрезок 087 512 677. Слияние этих двух последовательностей дает подфайл 087 503 512 677 703 765, который помещается слева в строке 3. Затем ключи 061 612 908 в строке 2 сливаются с 170 509 897, и результат (061 170 509 612 897 908) записывается *справа* в строке 3. Наконец, 154 275 426 653 сливается с 653 (перекрытие обнаруживается прежде, чем оно может привести к вредным последствиям), и результат записывается слева. Точно так же строка 2 получилась из исходного файла в строке 1.

Вертикальными линиями в табл. 1 отмечены границы между отрезками. Это так называемые ”ступеньки вниз”, где меньший элемент следует за большим. В середине файла обычно возникает двусмысленная ситуация, когда при движении с обоих концов мы прочитываем один и тот же ключ; это не приведет к осложнениям, если проявить чуточку осторожности, как в следующем алгоритме. Такой метод по традиции называется ”естественным” слиянием, потому что он использует отрезки, которые ”естественно” образуются в исходном файле.

Алгоритм N. (Сортировка естественным двухпутевым слиянием.) При сортировке записей R_1, \dots, R_N используются две области памяти, каждая из которых может содержать N записей. Для удобства обозначим записи, находящиеся во второй области, через R_{N+1}, \dots, R_{2N} , хотя в действительности запись R_{N+1} может и не примыкать непосредственно к R_N . Начальное содержимое записей R_{N+1}, \dots, R_{2N} не имеет значения. После завершения сортировки ключи будут упорядочены: $K_1 \leq \dots \leq K_N$.

Picture: Рис. 30. Сортировка слиянием.

- N1** [Начальная установка.] Установить $s \leftarrow 0$. (При $s = 0$ мы будем пересылать записи из области (R_1, \dots, R_N) в область (R_{N+1}, \dots, R_{2N}) ; при $s = 1$ области по отношению к пересылкам поменяются ролями.)
- N2** [Подготовка к просмотру.] Если $s = 0$, то установить $i \leftarrow 1, j \leftarrow N, k \leftarrow N + 1, l \leftarrow 2N$; если $s = 1$, то установить $i \leftarrow N + 1, j \leftarrow 2N, k \leftarrow 1, l \leftarrow N$. (Переменные i, j, k, l указывают текущие позиции во входных ”файлах”, откуда идет чтение, и в выходных ”файлах”, куда идет запись.) Установить $d \leftarrow 1, f \leftarrow 1$. (Переменная d определяет текущее направление вывода, f устанавливается равной 0, если необходимы дальнейшие просмотры.)
- N3** [Сравнение $K_i : K_j$] Если $K_i > K_j$, перейти к шагу **N8**. Если $i = j$, установить $P_k \leftarrow R_i$ и перейти к шагу **N13**.
- N4** [Пересылка R_i .] (Шаги **N4–N7** аналогичны шагам **M3–M4** алгоритма **M**.) Установить $R_k \leftarrow R_i, k \leftarrow k + d$.
- N5** [Ступенька вниз?] Увеличить i на 1. Затем, если $K_{i-1} \leq K_i$, возвратиться к шагу **N3**.
- N6** [Пересылка R_j .] Установить $R_k \leftarrow R_j, k \leftarrow k + d$.
- N7** [Ступенька вниз?] Уменьшить j на 1. Затем, если $K_{j+1} \leq K_j$, возвратиться к шагу **N6**; в противном случае перейти к шагу **N12**.
- N8** [Пересылка R_j .] (Шаги **N8–N11** двойственны по отношению к шагам **N4–N7**.) Установить $R_k \leftarrow R_j, k \leftarrow k + d$.
- N9** [Ступенька вниз?] Уменьшить j на 1. Затем, если $K_{j+1} \leq K_j$, возвратиться к шагу **N3**.
- N10** [Пересылка R_i .] Установить $R_k \leftarrow R_i, k \leftarrow k + d$.
- N11** [Ступенька вниз?] Увеличить i на 1. Затем, если $K_{i-1} \leq K_i$, возвратиться к шагу **N10**.
- N12** [Переключение направления.] Установить $f \leftarrow 0, d \leftarrow -d$ и взаимозаменить $k \leftrightarrow l$. Возвратиться к шагу **N3**.

N13 [Переключение областей.] Если $f = 0$, то установить $s \leftarrow 1 - s$ и возвратиться к **N2**. В противном случае сортировка завершена; если $s = 0$, то установить $(R_1, \dots, R_N) \leftarrow (R_{N+1}, \dots, R_{2N})$. (Если результат можно оставить в области (R_{N+1}, \dots, R_{2N}) , то последнее копирование необязательно.)

■

В этом алгоритме есть одна небольшая тонкость, которая объясняется в упр. 5.

Запрограммировать алгоритм **N** для машины MIX нетрудно, но основные сведения о его поведении можно получить и без построения всей программы. Если файл случаен, то в нем около $\frac{1}{2}N$ возрастающих отрезков, так как $K_i > K_{i+1}$ с вероятностью $\frac{1}{2}$; подробная информация о числе отрезков при несколько отличных предположениях была получена в п. 5.1.3. При каждом просмотре число отрезков сокращается вдвое (за исключением необычных случаев, таких, как ситуация, описанная в упр. 6). Таким образом, число просмотров, как правило, составляет около $\log_2 N$. При каждом просмотре мы должны переписать все N записей, и, как показано в упр. 2, большая часть времени затрачивается в шагах **N3**, **N4**, **N5**, **N8**, **N9**. Если считать, что равные ключи встречаются с малой вероятностью, то время, затрачиваемое во внутреннем цикле, можно охарактеризовать следующим образом:

	Шаг	Операции	Время
	N3	CMPA, JG, JE	$3.5u$
Либо	N4	STA, INC	$3u$
	N5	INC, LDA, CMPA, JGE	$6u$
Либо	N8	STX, INC	$3u$
	N9	DEC, LDX, CMPX, JGE	$6u$

Таким образом, при каждом просмотре на каждую запись затрачивается 12.5 единиц времени, и общее время работы асимптотически приближается к $12.5N \log_2 N$ как в среднем, так и в наихудшем случае. Это медленнее быстрой сортировки и не настолько лучше времени работы пирамидальной сортировки, чтобы оправдать вдвое больший расход памяти, так как асимптотическое время работы программы 5.2.3N равно $16N \log_2 N$.

В алгоритме **N** границы между отрезками полностью определяются "ступеньками вниз". Такой подход обладает тем возможным преимуществом, что исходные файлы с преобладанием возрастающего или убывающего расположения элементов могут обрабатываться очень быстро, но при этом замедляется основной цикл вычислений. Вместо проверок ступенек вниз можно принудительно установить длину отрезков, считая, что все отрезки исходного файла имеют длину 1, после первого просмотра все отрезки (кроме, возможно, последнего) имеют длину 2, ..., после k -го просмотра все отрезки (кроме, возможно, последнего) имеют длину 2^k . В отличие от "естественного" слияния в алгоритме **N** такой способ называется *простым* двухпутевым слиянием.

Алгоритм простого двухпутевого слияния очень напоминает алгоритм **N**—он описывается, по существу, той же блок-схемой; тем не менее методы достаточно отличаются друг от друга, и поэтому стоит записать весь алгоритм целиком.

Алгоритм S. (Сортировка простым двухпутевым слиянием.) Как и в алгоритме **N**, при сортировке записей R_1, \dots, R_N используются две области памяти.

S1 [Начальная установка.] Установить $s \leftarrow 0$, $p \leftarrow 1$. (Смысл переменных s, i, j, k, l, d см. в алгоритме **N**. Здесь p —размер возрастающих отрезков, которые будут сливаться во время текущего просмотра; q и r —количества неслитых элементов в отрезках.)

S2 [Подготовка к просмотру.] Если $s = 0$, то установить $i \leftarrow 1$, $j \leftarrow N$, $k \leftarrow N$, $l \leftarrow 2N + 1$; если $s = 1$, то установить $i \leftarrow N + 1$, $j \leftarrow 2N$, $k \leftarrow 0$, $l \leftarrow N + 1$. Затем установить $d \leftarrow 1$, $q \leftarrow p$, $r \leftarrow p$.

S3 [Сравнение $K_i : K_j$.] Если $K_i > K_j$, то перейти к шагу **S8**.

S4 [Пересылка R_i .] Установить $k \leftarrow k + d$, $R_k \leftarrow R_i$.

S5 [Конец отрезка?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, то возвратиться к шагу **S3**.

S6 [Пересылка R_j .] Установить $k \leftarrow k + d$. Затем, если $k = l$, перейти к шагу **S13**; в противном случае установить $R_k \leftarrow R_j$.

S7 [Конец отрезка?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, возвратиться к шагу **S6**; в противном случае перейти к шагу **S12**.

S8 [Пересылка R_j .] Установить $k \leftarrow k + d$, $R_k \leftarrow R_j$.

S9 [Конец отрезка?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, то возвратиться к шагу **S3**.

S10 [Пересылка R_i .] Установить $k \leftarrow k + d$. Затем, если $k = l$, перейти к шагу **S13**; в противном случае установить $R_k \leftarrow R_i$.

S11 [Конец отрезка?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, то возвратиться к шагу **S10**.

S12 [Переключение направления.] Установить $q \leftarrow p$, $r \leftarrow p$, $d \leftarrow -d$ и взаимозаменить $k \leftrightarrow l$. Возвратиться к шагу **S3**.

S13 [Переключение областей.] Установить $p \leftarrow p+p$. Если $p < N$, то установить $s \leftarrow 1-s$ и возвратиться к S2. В противном случае сортировка завершена; если $s = 0$, то установить

$$(R_1, \dots, R_n) \leftarrow (R_{N+1}, \dots, R_{2N}).$$

(Независимо от распределения исходного файла последнее копирование будет выполнено тогда и только тогда, когда значение $\lceil \log_2 N \rceil$ нечетно. Так что можно заранее предсказать положение отсортированного файла, и копирование, как правило, не требуется.) ■

Таблица 2

Сортировка простым двухпутевым слиянием															
503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Пример работы алгоритма см. в табл. 2. Довольно удивительно, что этот метод справедлив и тогда, когда N не является степенью 2; сливаемые отрезки не все имеют длину 2^k , тем не менее никаких явных мер предосторожности на случай таких исключений не предусмотрено! (См. упр. 8.) Проверки ступенек вниз заменены уменьшением переменных q и r и проверкой на равенство нулю. Благодаря этому время работы на машине MIX асимптотически приближается к $11N \log_2 N$ единицам, что несколько лучше значения, которого нам удалось добиться в алгоритме N.

На практике имеет смысл комбинировать алгоритм S с простыми вставками; вместо первых четырех просмотров алгоритма S можно простыми вставками отсортировать группы, скажем, из 16 элементов, исключив таким образом довольно расточительные вспомогательные операции, связанные со слиянием коротких файлов. Как мы уже видели в случае быстрой сортировки, такое комбинирование методов не влияет/на асимптотическое время работы, но дает тем не менее, немалую выгоду.

Рассмотрим теперь алгоритмы N и S с точки зрения структур данных. Почему нам необходима память под $2N$, а не под N записей? Причина относительно проста: мы работаем с четырьмя списками переменного размера (два "входных списка" и два "выходных списка" в каждом просмотре); при этом для каждой пары последовательно распределенных списков мы пользуемся стандартным понятием "встречного роста", обсуждавшимся в (п. 2.2.2). Но в любой момент времени половина памяти не используется, и после некоторого размышления становится ясно, что в действительности, для наших четырех списков следовало бы воспользоваться *связанным* распределением памяти. Если к каждой из N записей добавить поле связи, то все необходимое можно проделать, пользуясь алгоритмами слияния, которые производят простые манипуляции со связями и совсем не перемещают сами записи. Добавление N полей связи, как правило, выгоднее, чем добавление пространства памяти еще под N записей, а отказавшись от перемещения записей, мы можем также сэкономить и время. Итак, нам нужно рассмотреть алгоритм, подобный следующему.

Алгоритм L. (Сортировка посредством слияния списков.) Предполагается, что записи R_1, \dots, R_N содержат ключи K_1, \dots, K_N и "поля связи" L_1, \dots, L_N , в которых могут храниться числа от $-(N+1)$ до $(N+1)$. В начале и в конце файла имеются искусственные записи R_0 и R_{N+1} с полями связи L_0 и L_{N+1} . Этот алгоритм сортировки списков устанавливает поля связи таким образом, что записи оказываются связанными в возрастающем порядке. После завершения сортировки L_0 указывает на запись с наименьшим ключом; при $1 \leq k \leq N$ связь L_k указывает на запись, следующую за R_k , а если R_k — запись с наибольшим ключом, то $L_k = 0$. (См. формулы (5.2.1–9).)

В процессе выполнения этого алгоритма записи R_0 и R_{N+1} служат "головами" двух линейных списков, подписки которых в данный момент сливаются. Отрицательная связь означает конец подсписка, о котором известно, что он упорядочен; нулевая связь означает конец всего списка. Предполагается, что $N \geq 2$.

Через " $|L_s| \leftarrow p$ " обозначена операция "присвоить L_s значение p или $-p$, сохранив прежний знак L_s ". Такая операция легко реализуется на машине MIX, но, к сожалению, это не так для большинства ЭВМ. Нетрудно изменить алгоритм, чтобы получить столь же эффективный метод и для большинства других машин.

L1 [Подготовить два списка.] Установить $L_0 \leftarrow 1$, $L_{N+1} \leftarrow 2$, $L_i \leftarrow -(i+2)$ при $l \leq i \leq N-2$ и $L_{N-1} \leftarrow L_N \leftarrow 0$. (Мы создали два списка, содержащие соответственно записи R_1, R_3, R_5, \dots и R_2, R_4, R_6, \dots ; отрицательные связи говорят о том, что каждый упорядоченный "подсписок"

состоит всего лишь из одного элемента. Другой способ выполнить этот шаг, извлекая пользу из упорядоченности, которая могла присутствовать в исходных данных, см. в упр. 12.)

- L2 [Начать новый просмотр.] Установить $s \leftarrow 0, t \leftarrow N + 1, p \leftarrow L_s, q \leftarrow L_t$. Если $q = 0$, то работа алгоритма завершена. (При каждом просмотре p и q пробегают по спискам, которые подвергаются слиянию; s обычно указывает на последнюю обработанную запись текущего подсписка, а t — на конец только что выведенного подсписка.)
- L3 [Сравнить $K_p : K_q$.] Если $K_p > K_q$, то перейти к L6.
- L4 [Продвинуть p .] Установить $|L_s| \leftarrow p, s \leftarrow p, p \leftarrow L_p$. Если $p > 0$, то возвратиться к L3.
- L5 [Закончить подсписок.] Установить $L_s \leftarrow q, s \leftarrow t$. Затем установить $t \leftarrow q$ и $q \leftarrow L_q$ один или более раз, пока не станет $q \leq 0$, после чего перейти к L8.
- L6 [Продвинуть q .] (Шаги L6 и L7 двойственны по отношению к L4 и L5.) Установить $|L_s| \leftarrow q, s \leftarrow q, q \leftarrow L_q$. Если $q > 0$, то возвратиться к L3.
- L7 [Закончить подсписок.] Установить $L_s \leftarrow p, s \leftarrow t$. Затем установить $t \leftarrow p$ и $p \leftarrow L_p$ один или более раз, пока не станет $p > 0$.
- L8 [Конец просмотра?] (К этому моменту $p \leq 0$ и $q \leq 0$, так как оба указателя продвинулись до конца соответствующих подсписков.) Установить $p \leftarrow -p, q \leftarrow -q$. Если $q = 0$, то установить $|L_s| \leftarrow p, |L_t| \leftarrow 0$, и возвратиться к L2; в противном случае возвратиться к L3. ■

Пример работы этого алгоритма приведен в табл. 3, в которой показаны связи к моменту выполнения шага L2. По окончании работы алгоритма можно, пользуясь методом из упр. 5.2–12, переместить записи так, чтобы их ключи были упорядочены. Можно заметить интересную аналогию между слиянием списков и сложением разреженных многочленов (см. алгоритм 2.2.4А).

Таблица 3

Сортировка посредством слияния списков																		
j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K_i	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	—
L_j	1	—3	—4	—5	—6	—7	—8	—9	—10	—11	—12	—13	—14	—15	—16	0	0	2
L_j	2	—6	1	—8	3	—10	5	—11	7	—13	9	12	—16	14	0	0	15	4
L_j	4	3	1	—11	2	—13	8	5	7	0	12	10	9	14	16	0	15	6
L_j	4	3	6	7	2	0	8	5	1	14	12	10	13	9	16	0	15	11
L_j	4	12	11	13	2	0	8	5	10	14	1	6	3	9	16	7	15	0

Напишем теперь MIX-программу для алгоритма L, чтобы выяснить, столь ли выгодно оперировать списками с точки зрения времени, как и с точки зрения пространства?

Программа L. (Сортировка посредством слияния списков.) Для удобства предполагается, что записи занимают одно слово, причем L_j хранится в поле (0 : 2), а K_j — в поле (3 : 5) ячейки INPUT + j ; значения регистров: rI1 $\equiv p$, rI2 $\equiv q$, rI3 $\equiv s$, rI4 $\equiv t$, rA $\equiv R_q$; $N \geq 2$.

L	EQU	0 : 2		Определение имен полей.
ABSL	EQU	1 : 2		
KEY	EQU	3 : 5		
START	ENT1	N-2	1	L1. Подготовить два списка.
	ENNA	2, 1	N-2	
	STA	INPUT, 1 (L)	N-2	$L_i \leftarrow -(i+2)$.
	DEC1	1	N-2	
	J1P	*-3	N-2	$N-2 \geq i > 0$.
	ENTA	1	1	
	STA	INPUT (L)	1	$L_0 \leftarrow 1$.
	ENTA	2	1	
	STA	INPUT+N+1 (L)	1	$L_{N+1} \leftarrow 2$.
	STZ	INPUT+ N-1 (L)	1	$L_{N-1} \leftarrow 0$.
	STZ	INPUT +N (L)	1	$L_N \leftarrow 0$.
	JMP	L2	1	K L2.
L3Q	LDA	INPUT, 2	$C'' + B'$	L3. Сравнить $K_p : K_q$.
L3P	CMPA	INPUT, 1 (KEY)	C	
	JL	L6	C	K L6, если $K_q < K_p$.
L4	ST1	INPUT, 3 (ABSL)	C'	L4. Продвинуть p . $ L_s \leftarrow p$.
	ENT3	0, 1	C'	$s - p$.

	LD1	INPUT, 1 (L)	C'	$p \leftarrow L_p$
	J1P	L3P	C'	К L3, если $p > 0$.
L5	ST2	INPUT, 3 (L)	B'	L5. Закончить подписание. $L_s \leftarrow q$.
	ENT3	0, 4	B'	$s \leftarrow t$.
	ENT4	0, 2	D'	$t \leftarrow q$.
	LD2	INPUT, 2 (L)	D'	$q \leftarrow L_q$.
	J2P	*-2	D'	Повторить, если $q > 0$.
	JMP	L8	B'	К L8
L6	ST2	INPUT, 3 (ABSL)	C''	L6. Продвинуть q . $ L_s \leftarrow q$.
	ENT3	0, 2	C''	$s \leftarrow q$.
	LD2	INPUT, 2 (L)	C''	$q \leftarrow L_q$.
	J2P	L3Q	C''	К L3, если $q > 0$.
L7	ST1	INPUT, 3 (L)	B''	L7. Закончить подписание. $L_s \leftarrow p$.
	ENT3	0, 4	B''	$s \leftarrow t$.
	ENT4	0, 1	D''	$t \leftarrow p$.
	LD1	INPUT, 1 (L)	D''	$p \leftarrow L_p$.
	J1P	*-2	D''	Повторить, если $p > 0$.
L8	ENN1	0, 1	B	L8. Конец просмотра? $p \leftarrow -p$.
	ENN2	0, 2	B	$q \leftarrow -q$.
	J2NZ	L3Q	B	К L3, если $q \neq 0$.
	ST1	INPUT, 3 (ABSL)	A	$ L_s \leftarrow p$.
	STZ	INPUT, 4 (ABSL)	A	$ L_t \leftarrow 0$.
L2	ENT3	0	$A + 1$	L2. Начать новый просмотр, $s \leftarrow 0$.
	ENT4	N+1	$A + 1$	$t \leftarrow N + 1$.
	LD1	INPUT (L)	$A + 1$	$p \leftarrow L_s$.
	LD2	INPUT+N+1 (L)	$A + 1$	$q \leftarrow L_t$.
	J2NZ	L3Q	$A + 1$	К L3, если $q \neq 0$.

Время работы этой программы можно оценить при помощи методов, которыми мы уже не раз пользовались (см. упр. 13, 14); в среднем оно равно приблизительно $(10N \log_2 N + 4.92N)$ единиц с небольшим стандартным отклонением порядка \sqrt{N} . В упр. 15 показано, что за счет некоторого удлинения программы можно сократить время примерно до $9N \log_2 N$.

Итак, в случае внутреннего слияния связанное распределение памяти имеет бесспорные преимущества перед последовательным распределением: требуется меньше памяти, и программа работает на 10–20% быстрее. Аналогичные алгоритмы опубликованы Л. Дж. Вудрамом [IBM Systems J., 8 (1969), 189–203] и А. Д. Вудаллом [Сотр. Ж., 13 (1970), 110–111].

Упражнения

- [20] Обобщите алгоритм М на k -путевое слияние исходных файлов $x_{i1} \leq \dots \leq x_{im_i}$ при $i = 1, 2, \dots, k$.
- [M24] Считая, что все $\binom{m+n}{m}$ возможных расположений m элементов x среди n элементов y равновероятны, найдите математическое ожидание и стандартное отклонение числа выполнений шага М2 в алгоритме М. Чему равны максимальное и минимальное значения этой величины?
- >3. [20] (Изменение.) Даны записи R_1, \dots, R_M и R'_1, \dots, R'_N , ключи которых различны и упорядочены, т. е. $K_1 < \dots < K_M$ и $K'_1 < \dots < K'_N$. Как нужно изменить алгоритм М, чтобы в результате слияния получался файл, в котором отсутствуют записи R_i первого файла, если во втором файле тоже есть запись с таким же ключом?
- [21] В тексте отмечено, что сортировку слиянием можно рассматривать как обобщение сортировки вставками. Покажите, что метод слияний имеет непосредственное отношение и к выбору из дерева (воспользуйтесь в качестве иллюстрации рис. 23).
- >5. [21] Докажите, что в шагах N6 и N10 переменные i и j не могут быть равны. (Поэтому в этих шагах проверка на случай возможного перехода к N13 необязательна.)
- [22] Найдите такую перестановку $K_1 K_2 \dots K_{16}$ множества $\{1, 2, \dots, 16\}$, что

$$K_2 > K_3, K_4 > K_5, K_6 > K_7, K_8 > K_9,$$

$$K_{10} < K_{11}, K_{12} < K_{13}, K_{14} < K_{15},$$

которая тем не менее будет отсортирована при помощи алгоритма N всего за два просмотра. (Так как в искомой перестановке имеется не менее восьми отрезков, то мы могли бы ожидать, что

после первого просмотра останутся по меньшей мере четыре отрезка, после второго просмотра — два отрезка, и сортировка, как правило, не завершится раньше окончания третьего просмотра. Каким же образом можно обойтись всего двумя просмотрами?)

7. [16] Найдите точную формулу, выражающую число просмотров алгоритма S в виде функции от N.
8. [22] Предполагается, что во время работы алгоритма переменные q и r представляют длины неслитых частей отрезков, обрабатываемых в данный момент; в начале работы как q , так и r устанавливаются равными p , в то время как отрезки не всегда имеют такую длину. Почему же алгоритм тем не менее работает правильно?
9. [24] Напишите MIX-программу для алгоритма S. Выразите частоту выполнения каждой команды через величины, подобные A, B', B'', C', \dots в программе L.
10. [25] (Д. А. Белл.) Покажите, что простое двухпутевое слияние файлов с последовательно расположенными элементами можно выполнить, имея всего $\frac{3}{2}N$ ячеек памяти, а не $2N$, как в алгоритме S.
11. [21] Является ли алгоритм L алгоритмом "устойчивой" сортировки?
- >12. [22] Измените шаг L1 алгоритма L так, чтобы двухпутевое слияние стало "естественным", извлекающая пользу из наличия возрастающих отрезков в исходном файле. (В частности, если исходные данные уже упорядочены, то шаг L2 завершит работу алгоритма сразу же после выполнения измененного вами шага L1.)
- >13. [M34] Проанализируйте среднее время работы программы L подобно тому, как мы анализировали другие алгоритмы в этой главе. Дайте толкование величинам A, B, B', \dots и объясните, как вычислить их точные средние значения. Сколько времени затратит программа L на сортировку 16 чисел в табл. 3?
14. [M24] Пусть двоичное представление числа N — это $2^{e_1} + 2^{e_2} + \dots + 2^{e_t}$, где $e_1 > e_2 > \dots > e_t \geq 0$, $t \geq 1$. Докажите, что максимальное число сравнений ключей, выполняемых алгоритмом L, равно $1 - 2^{e_t} + \sum_{1 \leq k \leq t} (e_k + k - 1)2^{e_k}$.
15. [20] Если промоделировать ручную работу алгоритма L, то обнаружится, что в нем иногда выполняются лишние операции; примерно в половине случаев не нужны присваивания $|L_s| \leftarrow p$, $|L_s| \leftarrow q$ в шагах L4 и L6, поскольку мы имеем $L_s = p$ (или q) всякий раз, когда возвращаемся из шага L4 (или L6) к L3. Как улучшить программу L, избавившись от этих лишних присваиваний?
16. [28] Разработайте алгоритм слияния списков, подобный алгоритму L, но основанный на трехпутевом слиянии.
17. [20] (Дж. Мак-Карти.) Пусть двоичное представление числа N такое же, как в упр. 14, и предположим, что дано N записей, организованных в t упорядоченных подфайлов, имеющих размеры соответственно $2^{e_1}, 2^{e_2}, \dots, 2^{e_t}$. Покажите, как можно сохранить такое состояние при добавлении $(N+1)$ -й записи и $N \leftarrow N+1$. (Полученный алгоритм можно назвать "оперативной" сортировкой слиянием.)
18. [40] (М. А. Кронрод.) Можно ли отсортировать файл из N записей, содержащий всего два отрезка:

$$K_1 \leq \dots \leq K_M \quad \text{и} \quad K_{M+1} \leq \dots \leq K_N,$$

За $O(N)$ операций в памяти с произвольным доступом, используя лишь небольшое дополнительное пространство памяти фиксированного размера, не зависящего от M и N ? (Все алгоритмы слияния, описанные в этом пункте, используют дополнительное пространство памяти, пропорциональное N .)

19. [26] Рассмотрим железнодорожный разъезд с n "стеками", как показано на рис. 31 при $n = 5$; такой разъезд имеет некоторое отношение к алгоритмам сортировки с n просмотрами. В упр. с 2.2.1–2 по 2.2.1–5 мы рассмотрели разъезды с одним стеком. Вы видели, что если с правого конца поступает N вагонов, то слева может появиться сравнительно небольшое количество из N всевозможных перестановок этих вагонов.

Предположим, что в разъезд с n стеками справа поступает 2^n вагонов. Докажите, что при помощи подходящей последовательности операций слева можно получить любую из $2^n!$ всевозможных перестановок этих вагонов. (Каждый стек достаточно велик, и при необходимости в него можно поместить все вагоны.)

20. [47] В обозначениях упр. 2.2.1–4 при помощи разъездов с n стеками можно получить не более a_N^n перестановок N элементов; следовательно, для

Picture: Рис. 31. Железнодорожный разъезд с пятью "стеками".

получения всех $N!$ перестановок требуется не менее $\log N! / \log a_N \approx \log_4 N$ стеков. В упр. 19 показано, что нужно не более $\lceil \log_2 N \rceil$ стеков. Какова истинная скорость роста необходимого числа стеков при $N \rightarrow \infty$?

21. [23] (Э. Дж. Смит.) Объясните, как можно обобщить алгоритм L, чтобы он, помимо сортировки, вычислял также число *инверсий* в исходной перестановке.

5.2.5. Распределяющая сортировка

Мы подходим теперь к интересному классу методов сортировки, который, как показано в п. 5.4.7, по существу, прямо *противоположен* слиянию. Читателям, знакомым с перфокарточным оборудованием, хорошо известна эффективная процедура, применяемая в машинах для сортировки карт и основанная на сравнении цифр ключей; ту же идею можно приспособить и для программирования. Она общеизвестна под названиями "поразрядная сортировка", "цифровая сортировка" или "карманная сортировка".

Предположим, нам нужно отсортировать колоду из 52 игральные карты. Определим упорядочение по старшинству (достоинству) карт в масти

$$T < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < B < D < K,$$

а также по масти

$$\clubsuit < \diamond < \heartsuit < \spadesuit$$

Одна карта предшествует другой, если либо (i) она младше по масти, либо (ii) масти обеих карт одинаковы, но она младше по достоинству. (Это частный случай *лексикографического* упорядочения на множестве упорядоченных пар предметов; ср. с упр. 5-2.) Таким образом,

$$T\clubsuit < 2\clubsuit < \dots < K\clubsuit < T\diamond < \dots < D\spadesuit < K\spadesuit$$

Мы могли бы отсортировать карты одним из обсуждавшихся ранее методов; люди, как правило, пользуются способом, по сути аналогичным обменной поразрядной сортировке. Естественно отсортировать карты сначала по их масти, разложив их в четыре стопки, а затем перекладывать карты внутри каждой стопки до тех пор, пока они не будут упорядочены по достоинству.

Но существует более быстрый способ! Сначала разложить карты в 13 стопок лицевой стороной вверх по их достоинству. Затем собрать все стопки вместе: снизу тузы, затем двойки, тройки и т. д. и сверху короли (лицевой стороной вверх). Перевернуть колоду рубашками вверх и снова разложить, на этот раз в четыре стопки по масти. Сложив вместе полученные стопки так, чтобы внизу были трефы, затем бубны, черви и пики, получим упорядоченную колоду карт.

Та же идея годится и для сортировки числовых и буквенных данных. Почему же она правильна? Потому что (в нашем примере с игральными картами), если две карты при последнем раскладе попали в разные стопки, то они имеют разные масти, так что карта с меньшей мастью младше. Если же карты одной масти, то они уже находятся в нужном порядке благодаря предварительной сортировке. Иначе говоря, при втором раскладе в каждой из четырех стопок карты будут расположены в возрастающем порядке. Это доказательство можно обобщить и показать, что таким способом можно отсортировать любое множество с лексикографическим упорядочением; подробности см. в упр. 5-2 (в начале главы).

Только что описанный метод сортировки сразу не очевиден, и не ясно, кто же первый обнаружил, что он так удобен. В брошюре на 19 страницах под названием "The Inventory Simplified", опубликованной отделением фирмы IBM Tabulating Machines Company в 1923 г., представлен интересный цифровой метод вычисления сумм произведений на сортировальной машине. Пусть, например, нужно перемножить два числа, пробитых соответственно в колонках 1-10 и в колонках 23-25, и вычислить сумму таких произведений для большого числа карт. Тогда сначала можно отсортировать карты по колонке 25 и найти при помощи счетно-аналитической машины величины a_1, a_2, \dots, a_9 , где a_k — сумма чисел из колонок 1-10 по всем карточкам, на которых в колонке 25 пробита цифра k . Затем можно отсортировать карты по колонке 24 и найти аналогичные суммы b_1, b_2, \dots, b_9 , а потом по колонке 23, получив величины c_1, c_2, \dots, c_9 . Легко видеть, что искомая сумма произведений равна

$$a_1 + 2a_2 + \dots + 9a_9 + 10b_1 + 20b_2 + \dots + 90b_9 + 100c_1 + 200c_2 + \dots + 900c_9.$$

Такой перфокарточный метод табулирования естественным образом приводит к идее поразрядной сортировки "сначала-по-младшей-цифре", так что, по-видимому, она впервые стала известна операторам этих машин. Первая опубликованная ссылка на этот метод содержится в ранней работе Л. Дж. Комри, посвященной обсуждению перфокарточного оборудования [Transactions of the Office Machinery Users' Assoc., Ltd. (1929), 25-37, особенно стр. 28].

Чтобы выполнить поразрядную сортировку с помощью ЭВМ, необходимо решить, как представлять стопки. Пусть имеется M стопок; можно было бы выделить M областей памяти и пересылать каждую исходную запись в соответствующую область. Но это решение нас не удовлетворяет, потому

что в каждой области должно быть достаточно места для хранения N элементов, и тогда потребуется пространство под $(M+1)N$ записей. Такая чрезмерная потребность в памяти заставляла большинство программистов отказываться от применения поразрядной сортировки на вычислительных машинах, пока Х. Сьюворт [дипломная работа, M.I.T. Digital Computer Laboratory Report R-232 (Cambridge Mass: 1954), 25–28] не показал, что того же эффекта можно добиться, имея в распоряжении пространство всего под $2N$ записей и M счетчиков. Сделав один предварительный просмотр данных, можно просто посчитать, сколько элементов попадет в каждую область; это даст нам возможность точно распределить память под стопки. Мы уже применяли эту идею при распределяющей сортировке (алгоритм 5.2D).

Итак, поразрядную сортировку можно выполнить следующим образом: сначала произвести распределяющую сортировку *по младшим цифрам ключей* (в системе счисления с основанием M), переместив записи из области ввода во вспомогательную область, затем произвести еще одну распределяющую сортировку по следующей цифре, переместив записи обратно в исходную область и т. д., до тех пор, пока после завершающего просмотра (сортировка по старшей цифре) все ключи не окажутся расположенными в нужном порядке.

Если у нас имеется десятичная машина, а ключи—12-разрядные числа и если N весьма велико, то можно выбрать $M = 1000$ (считая три десятичные цифры за одну в системе счисления с основанием 1000); независимо от величины N сортировка будет выполнена за четыре просмотра. Аналогично, если бы имелась двоичная машина, а ключи—40-битовые двоичные числа, то можно положить $M = 1024$ и также завершить сортировку за четыре просмотра. Фактически каждый просмотр состоит из трех частей (подсчет, распределение памяти, перемещение); Фрэнд [JACM, 3 (1956), 151] предложил комбинировать два из этих трех действий, добавив еще M ячеек: накапливать значения счетчиков для $(k+1)$ -го просмотра одновременно с перемещением во время k -го просмотра.

В табл. 1 показано применение поразрядной сортировки к нашим 16 ключам при $M = 10$. При таких малых N поразрядная сортировка, как правило, не особенно полезна, так что этот маленький пример предназначен главным образом для того, чтобы продемонстрировать достаточность метода, а не его эффективность.

Таблица 1

Поразрядная сортировка	
Область ввода:	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
Счетчики для младших цифр:	1 1 2 3 1 2 1 3 1 1
Соответствующее распределение памяти:	1 2 4 7 8 10 11 14 15 16
Вспомогательная область:	170 061 512 612 503 653 703 154 275 765 426 087 897 677 908 509
Счетчики для средних цифр:	4 2 1 0 0 2 2 3 1 1
Соответствующее распределение памяти:	4 6 7 7 7 9 11 14 15 16
Область ввода:	503 703 908 509 512 612 426 653 154 061 765 170 275 677 087 897
Счетчики для старших цифр:	2 2 1 0 1 3 3 2 1 1
Соответствующее распределение памяти:	2 4 5 5 6 9 12 14 15 16
Вспомогательная область:	061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

Искушенный "современный" читатель заметит, однако, что идея счетчиков для распределения памяти привязана к "старомодным" понятиям о последовательном представлении данных; нам же известно, что специально для работы с множеством таблиц переменной длины придумано *связанное* распределение. Поэтому для поразрядной сортировки естественно будет воспользоваться связанными структурами данных. Так как каждая стопка просматривается последовательно, то все, что нам нужно,— иметь при каждом элементе одну-единственную ссылку на следующий элемент. Кроме того, никогда не придется перемещать записи: достаточно скорректировать связи—и можно смело двигаться дальше по спискам. Объем необходимой памяти равен $(1 + \epsilon)N + 2\epsilon M$ записей, где ϵ —пространство, занимаемое одним полем связи. Довольно интересны формальные подробности этой процедуры, поскольку они дают прекрасный пример типичных манипуляций со структурами данных, соединяющих в себе последовательное и связанное распределение памяти.

Picture: Рис. 32. Поразрядная сортировка списка.

Алгоритм R. (Поразрядная сортировка списка.) Предполагается, что каждая из записей R_1, \dots, R_N содержит поле связи LINK, а ключи представляют собой последовательность из p элементов

$$(a_p, \dots, a_2, a_1), \quad 0 \leq a_i < M,$$

и отношение порядка—лексикографическое, т. е.

$$(a_p, \dots, a_2, a_1) < (b_p, \dots, b_2, b_1)$$

тогда и только тогда, когда существует такой индекс j , $1 \leq j \leq p$, что

$$a_i = b_i \quad \text{при } i > j, \text{ но } a_j < b_j.$$

Ключи можно представлять себе, в частности, как числа, записанные в системе счисления с основанием M :

$$a_p M^{p-1} + \dots + a_2 M + a_1,$$

и в этом случае лексикографическое отношение порядка соответствует обычному упорядочению множества неотрицательных чисел. Ключи также могут быть цепочками букв алфавита и т. д.

Во время сортировки формируются M ”стопок” подобно тому, как это делается в сортировальной машине для перфокарт. Стопки фактически представляют собой очереди в смысле гл. 2, поскольку мы связываем их вместе таким образом, что они всегда просматриваются по принципу ”первым включается—первым исключается”. Для каждой стопки имеются две переменные-указатели: $\text{TOP}[i]$ и $\text{BOTM}[i]$, $0 \leq i < M$, и, как и в гл. 2, предполагается, что

$$\text{LINK}(\text{LOC}(\text{BOTM}([i]))) \equiv \text{BOTM}[i].$$

- R1** [Цикл по k .] Вначале установить $P \leftarrow \text{LOC}(R_N)$, указатель на последнюю запись. Затем выполнить шаги с **R2** по **R6** при $k = 1, 2, \dots, p$ (шаги с **R2** по **R6** составляют один ”просмотр”) и завершить работу алгоритма. Переменная P будет указывать на запись с наименьшим ключом, $\text{LINK}(P)$ —на запись со следующим по величине ключом, $\text{LINK}(\text{LINK}(P))$ —на следующую и т. д.; поле LINK последней записи будет равно Λ .
- R2** [Опустошить стопки.] При $0 \leq i < M$ установить $\text{TOP}[i] \leftarrow \text{LOC}(\text{BOTM}[i])$ и $\text{BOTM}[i] \leftarrow \Lambda$.
- R3** [Выделить k -ю цифру ключа.] Пусть $\text{KEY}(P)$ — ключ записи, на которую указывает P , — равен (a_p, \dots, a_2, a_1) ; установить $i \leftarrow a_k$, k -я младшая цифра этого ключа.
- R4** [Скорректировать связи.] Установить $\text{LINK}(\text{TOP}[i]) \leftarrow P$, затем $\text{TOP}[i] \leftarrow P$.
- R5** [Перейти к следующей записи.] Если $k = 1$ (первый просмотр) и если $P = \text{LOC}(R_j)$ при некотором $j \neq 1$, то установить $P \leftarrow \text{LOC}(R_{j-1})$ и возвратиться к шагу **R3**. Если $k > 1$ (не первый просмотр), то установить $P \leftarrow \text{LINK}(P)$ и возвратиться к **R3**, если $P \neq \Lambda$.
- R6** [Выполнить алгоритм Н.] (Теперь мы уже распределили все элементы по стопкам.) Выполнить приведенный ниже алгоритм Н, который сцепляет отдельные ”стопки” в один список, подготавливая их к следующему просмотру. Затем установить $P \leftarrow \text{BOTM}[0]$, указатель на первый элемент объединенного списка. (См. упр. 3.) ■

Алгоритм Н. (Сцепление очередей.) Из M данных очередей со связями, удовлетворяющими соглашениям алгоритма **R**, данный алгоритм создает одну очередь, меняя при этом не более M связей. В результате $\text{BOTM}[0]$ указывает на первый элемент, и стопка 0 предшествует стопке 1, ..., предшествует стопке $(M - 1)$.

- H1** [Начальная установка.] Установить $i \leftarrow 0$.
- H2** [Указатель на вершину стопки.] Установить $P \leftarrow \text{TOP}[i]$.
- H3** [Следующая стопка.] Увеличить i на 1. Если $i = M$, то установить $\text{LINK}(P) \leftarrow \Lambda$ и завершить работу алгоритма.
- H4** [Стопка пуста?] Если $\text{BOTM}[i] = \Lambda$, то возвратиться к **H3**.
- H5** [Сцепить стопки.] Установить $\text{LINK}(P) \leftarrow \text{BOTM}[i]$. Возвратиться к **H2**. ■

На рис. 33 показано содержимое стопок после каждого из трех просмотров, выполняемых при сортировке наших 16 чисел с $M = 10$. Алгоритм **R** очень просто запрограммировать для машины **MIX**, если только найти удобный способ изменять от просмотра к просмотру действия в шагах **R3** и **R5**. В следующей программе этого удалось добиться, не жертвуя скоростью внутреннего цикла, путем предварительной записи двух команд в тело программы. Заметим, что $\text{TOP}[i]$ и $\text{BOTM}[i]$ можно упаковать в одно слово.

Picture: Рис. 33. Поразрядная сортировка с использованием связанного распределения памяти (показано содержимое всех десяти стопок после каждого просмотра).

Программа R. (Поразрядная сортировка списков.) Предполагается, что исходные ключи в ячейках от $\text{INPUT} + 1$ до $\text{INPUT} + N$ содержат $p = 3$ компоненты (a_3, a_2, a_1) , хранящиеся соответственно в полях $(1 : 1)$, $(2 : 2)$ и $(3 : 3)$. (Таким образом, считается, что значение M меньше или равно размеру байта машины MIX.) В поле $(4 : 5)$ записи хранится связь LINK. Пусть $\text{TOP}[i] \equiv \text{PILES} + i(1 : 2)$ и $\text{BOM}[i] \equiv \text{PILES} + i(4 : 5)$ при $0 \leq i < M$. Удобно указывать в связи положение относительно ячейки INPUT, так что $\text{LOC}(\text{BOM}[i]) = \text{PILES} + i - \text{INPUT}$; чтобы избежать появления отрицательных связей, нужно расположить таблицу PILES после таблицы INPUT. Значения индексных регистров: $\text{rI1} \equiv P$, $\text{rI2} \equiv i$, $\text{rI3} \equiv 3 - k$, $\text{rI4} \equiv \text{TOP}[i]$; во время работы алгоритма $\text{H rI2} \equiv i - M$.

LINK	EQU	4:5		
TOP	EQU	1:2		
START	ENT1	N	1	R1. Цикл по k . $P \leftarrow \text{LOC}(R_N)$.
	ENT3	2	1	$k \leftarrow 1$.
2H	ENT2	$M-1$	3	R2. Опустошить стопки.
	ENTA	$\text{PILES} - \text{INPUT}, 2$	$3M$	$\text{LOC}(\text{BOM}[i])$
	STA	$\text{PILES}, 2 (\text{TOP})$	$3M$	$\rightarrow \text{TOP}[i]$
	STZ	$\text{PILES}, 2 (\text{LINK})$	$3M$	$\text{BOM}[i] \leftarrow \Lambda$.
	DEC2	1	$3M$	
	J2NN	$*-4$	$3M$	$M > i \geq 0$.
	LDA	$R3\text{SW}, 3$	3	
	STA	3F	3	Изменить команды
	LDA	$R5\text{SW}, 3$	3	для k -го просмотра.
	STA	5F	3	
3H	[LD2	$\text{INPUT}, 1(3:3)$		R3. Выделить k -ю цифру ключа.
4H	LD4	$\text{PILES}, 2 (\text{TOP})$	$3N$	R4. Скорректировать связи.
	ST1	$\text{INPUT}, 4 (\text{LINK})$	$3N$	$\text{LINK}(\text{TOP}[i]) \leftarrow P$.
	ST1	$\text{PILES}, 2 (\text{TOP})$	$3N$	$\text{TOP}[i] \leftarrow P$.
5H	[DEC1	1]		R5. Перейти к следующей записи.
	J1NZ	3B	$3N$	К R3, если просмотр закончен.
6H	ENN2	M	3	R6. Выполнить алгоритм H.
	JMP	7F	3	К H2 с $i \leftarrow 0$.
R3SW	LD2	$\text{INPUT}, 1(1:1)$	N	Команда для R3 при $k = 3$.
	LD2	$\text{INPUT}, 1(2:2)$	N	Команда для R3 при $k = 2$.
	LD2	$\text{INPUT}, 1(3:3)$	N	Команда для R3 при $k = 1$.
R5SW	LD1	$\text{INPUT}, 1(\text{LINK})$	N	Команда для R5 при $k = 3$.
	LD1	$\text{INPUT}, 1(\text{LINK})$	N	Команда для R5 при $k = 2$.
	DEC1	1	N	Команда для R5 при $k = 1$.
9H	LDA	$\text{PILES} + M, 2 (\text{LINK})$	$3M - 3$	H4. Стопка пуста?
	JAZ	8F	$3M - 3$	К H3, если $\text{BOM}[i] = \Lambda$
	STA	$\text{INPUT}, 1 (\text{LINK})$	$3M - 3 - E$	H5. Сцепить стопки $\text{LINK}(P) \leftarrow \text{BOM}[i]$.
7H	LD1	$\text{PILES} + M, 2 (\text{TOP})$	$3M - E$	H2. Указатель на вершину стопки.
8H	INC2	1	$3M$	H3. Следующая стопка, $i \leftarrow i + 1$.
	J2NZ	9B	$3M$	К H4, если $i \neq M$.
	STZ	$\text{INPUT}, 1 (\text{LINK})$	3	$\text{LINK}(P) \leftarrow \Lambda$.
	LD1	$\text{PILES} (\text{LINK})$	3	$P \leftarrow \text{BOM}[0]$.
	DEC3	1	3	
	J3NN	2B	3	$1 \leq k \leq 3$

Время работы программы R равно $32N + 48M + 38 - 4E$, где N —число исходных записей, M —основание системы счисления (число стопок), а E —число встретившихся пустых стопок. Сравнение с другими программами, построенными на основе аналогичных предположений (программы 5.2.1M, 5.2.4L), говорит явно в пользу программы R. Время работы p -проходного варианта программы R равно $(11p - 1)N + O(pM)$ единиц; критический фактор, влияющий на время работы,—внутренний цикл, который содержит пять обращений к памяти и один переход. Для типичной вычислительной машины $M = b^r$ и $p = \lceil t/r \rceil$, где t —число цифр в ключах, представленных в системе счисления с основанием b ; с ростом r убывает p , так что можно воспользоваться нашими формулами для определения "наилучшего" значения r .

Единственная переменная величина в формуле времени работы—это E —число пустых стопок, обнаруженных в шаге H4. Предположим, что все M^N последовательностей цифр M -ичной системы счисления равновероятны. Из изучения "покер-теста" в п. 3.3.2D мы умеем вычислять вероятность

того, что в каждом просмотре встретится ровно $M - r$ пустых стопок; она равна

$$\frac{M(M-1)\dots(M-r+1)}{M^N} \left\{ \begin{matrix} N \\ r \end{matrix} \right\},$$

где $\left\{ \begin{matrix} N \\ r \end{matrix} \right\}$ — число Стирлинга второго рода. Согласно упр. 5,

$$E = (\min \max(M - N, 0)p, \text{ave } M(1 - \frac{1}{M})^N p, \max(M - 1)p).$$

В последние годы появляется все больше ”трубопроводных”, или ”магистральных” вычислительных машин. Эти машины имеют несколько арифметических устройств и схему ”опережения”, так что обращения к памяти и вычисления могут в значительной степени совмещаться во времени; но эффективность таких машин заметно понижается при наличии условных переходов, если только эти переходы не происходят почти всегда в одном и том же направлении. Внутренний цикл поразрядной сортировки хорошо приспособлен для таких машин, поскольку это простое итеративное вычисление, типичное ”пережевывание чисел”. Поэтому для магистральных машин поразрядная сортировка обычно бывает наиболее эффективным методом из всех известных методов внутренней сортировки, при условии что N не слишком мало и ключи не слишком длинные.

Разумеется, если ключи уж очень длинные, поразрядная сортировка не так эффективна. Представьте себе, например, что нужно отсортировать перфокарты по ключу из 80 колонок; как правило, встретится очень мало пар карт, у которых бы совпали первые пять колонок, так что первые 75 просмотров выполняются почти впустую. При анализе обменной поразрядной сортировки мы обнаружили, что вовсе не обязательно проверять много битов ключей, если просматривать их не справа налево, а слева направо. Поэтому давайте возвратимся к идее поразрядной сортировки, в которой ключи просматриваются, начиная со старших цифр (СЦ), а не с младших цифр (МЦ).

Мы уже отмечали, что СЦ-поразрядная сортировка естественным образом приходит на ум. В самом деле, нетрудно понять, почему при сортировке почты в отделениях связи пользуются именно этим методом. Большое количество писем можно отсортировать по отдельным мешкам, соответствующим географическим областям; теперь каждый мешок содержит уже меньшее количество писем, которые можно независимо сортировать по другим мешкам, соответствующим все меньшим и меньшим географическим районам. (Разумеется, прежде чем подвергать письма дальнейшей сортировке, их можно переправить поближе к месту назначения.) Этот принцип ”разделяй и властвуй” весьма привлекателен, и единственная причина его непригодности для сортировки перфокарт в том, что большое количество стопок приводит к путанице. Этим же явлением объясняется относительная эффективность алгоритма R (хотя здесь сначала рассматриваются МЦ), потому что нам никогда не приходится работать более чем с M стопками и стопки приходится сцеплять всего p раз. С другой стороны, нетрудно построить СЦ-поразрядный метод с использованием связанного распределения памяти с отрицательными связями для обозначение границ между стопками, как в алгоритме 5.2.4L. (См. упр. 10.)

Пожалуй, наилучший компромиссный выход указал М. Д. Макларен [JACM, 13 (1966), 404–411], который предложил использовать МЦ-сортировку, как в алгоритме R, но лишь в применении к старшим цифрам. Это не будет полной сортировкой файла, но в результате файл становится почти упорядоченным, т. е. в нем остается очень мало инверсий, так что для завершения сортировки можно воспользоваться методом простых вставок. Наш анализ алгоритма 5.2.1M применим и к этой ситуации; если ключи распределены равномерно, то после сортировки файла по старшим p цифрам в нем останется в среднем

$$\frac{1}{4}N(N-1)M^{-p}$$

инверсий. [См. формулу (5.2.1–14) и упр. 5.2.1–38.] Макларен вычислил среднее число обращений к памяти, приходящееся на один обрабатываемый элемент, и оказалось, что оптимальный выбор значений M и p (в предположении, что M — степень двойки, ключи равномерно распределены и $N/M^p \leq 0.1$, так что отклонения от равномерного распределения приемлемы) описывается следующей таблицей:

$N =$	100	1000	5000	10000	50000	100000
Наилучшее $M =$	32	128	256	512	1024	1024
Наилучшее $p =$	2	2	2	2	2	2
$\bar{\beta}(N) =$	19.3	18.5	18.2	18.2	18.1	18.0

Здесь $\bar{\beta}(N)$ — среднее число обращений к памяти на один сортируемый элемент; эта величина ограничена при $N \rightarrow \infty$, если взять $p = 2$ и $M > \sqrt{N}$, так что среднее время сортировки есть $O(N)$, а

не $O(N \log N)$. Этот метод является усовершенствованием метода вставок в несколько списков (алгоритм 5.2.1M), который, по существу, представляет собой случай $p = 1$. В упр. 12 приводится интересная процедура Макларена для окончательного перераспределения после частичной сортировки файла с использованием списков.

Если воспользоваться методами алгоритма 5.2D и упр. 5.2-13, то можно обойтись без полей связи; при этом в дополнение к памяти, занятой самими записями, потребуется всего $O(\sqrt{N})$ ячеек. Если исходные данные распределены равномерно, то среднее время сортировки пропорционально N .

Упражнения

- >1. [20] Алгоритм из упр. 5.2–13 показывает, как можно выполнить распределяющую сортировку, имея пространство памяти всего под N записей (и M полей счетчиков), а не под $2N$ записей. Приводит ли эта идея к усовершенствованию алгоритма поразрядной сортировки, проиллюстрированного в табл. 1?
2. [13] Является ли алгоритм R алгоритмом "устойчивой" сортировки?
3. [15] Объясните, почему в алгоритме H переменной $\text{BOTM}[0]$ присваивается значение указателя на первую запись в "объединенной" очереди, *несмотря на то что строка 0 могла быть пустой*.
- >4. [23] Во время работы алгоритма R все M стопок хранятся в виде связанных очередей (первым включается—первым исключается). Исследуйте идею связывания элементов стопок как в *стеке*. (На рис. 33 стрелки пойдут не вверх, а вниз, и таблица BOTM станет не нужна.) Покажите, что если сцеплять стопки в соответствующем порядке, то может получиться правильный метод сортировки. Будет ли этот алгоритм более простым или более быстрым?
5. [M24] Пусть $g_{MN}(z) = \sum p_{MNk} z^k$, где p_{MNk} — вероятность того, что после случайного просмотра поразрядной сортировки, разложившего N элементов на M стопок, получилось ровно k пустых стопок. (а) Покажите, что $g_{M,N+1}(z) = g_{MN}(z) + ((1-z)/M)g'_{MN}(z)$. (б) Найдите при помощи указанного соотношения простые выражения для математического ожидания и дисперсии этого распределения вероятностей как функций от M и N .
6. [20] Какие изменения необходимо внести в программу R, чтобы она сортировала не трехбайтовые ключи, а восьмибайтовые? Считается, что старшие байты ключа K_i хранятся в ячейке $\text{KEY} + i(1:5)$, а три младших байта, как и раньше, — в ячейке $\text{INPUT} + i(1:3)$. Каково время работы программы с этими изменениями?
7. [20] Обсудите, в чем состоит сходство и отличие алгоритма R и алгоритма обменной поразрядной сортировки (алгоритм 5.2.2R).
- >8. [20] В алгоритмах поразрядной сортировки, обсуждавшихся в тексте, предполагалось, что все сортируемые ключи неотрицательны. Какие изменения следует внести в эти алгоритмы в том случае, когда ключами могут быть и отрицательные числа, представленные в *дополнительном или обратном* коде?
9. [20] (Продолжение упр. 8.) Какие изменения нужно внести в эти алгоритмы в случае, когда ключами являются числа, представленные в виде абсолютной величины со знаком?
10. [30] Сконструируйте алгоритм поразрядной сортировки "сначала-по-старшей-цифре", использующий связанное распределение. (Так как размер подфайлов все уменьшается, то разумно уменьшить M , а для сортировки коротких файлов применить не поразрядную сортировку.)
11. [16] Перестановка шестнадцати исходных чисел, показанная в табл. 1, содержит вначале 41 инверсию. После завершения сортировки инверсий, разумеется, нет совсем. Сколько инверсий осталось бы в файле, если бы мы пропустили первый просмотр, а выполнили бы поразрядную сортировку лишь по цифрам десятков и сотен? Сколько инверсий останется, если пропустить как первый, так и второй просмотры?
12. [24] (М. Д. Макларен.) Предположим, алгоритм R применили только к p старшим цифрам реальных ключей; тогда файл, если читать его по порядку, указанному связями, почти отсортирован, но ключи, у которых старшие p цифр совпадают, могут быть неупорядочены. Придумайте алгоритм перераспределения записей на том же месте так, чтобы ключи расположились по порядку: $K_1 \leq K_2 \leq \dots \leq K_N$. [Указание: частный случай, когда файл полностью отсортирован, можно найти в ответе к упр. 5.2–12, его можно скомбинировать с простыми вставками без потери эффективности, так как в файле осталось мало инверсий.]
13. [40] Реализуйте метод внутренней сортировки, предложенный в тексте в конце этого пункта, получив программу сортировки случайных данных за $O(N)$ единиц времени, требующую всего $O(N)$ дополнительных ячеек памяти.
14. [22] Последовательность игральных карт можно отсортировать в возрастающем порядке: T 2 ... В Д К от верхней карты к нижней, за два просмотра, раскладывая карты каждый раз лишь в две стопки: разложите карты лицевой стороной вниз в две стопки, содержащие соответственно T 2 9

3 10 и 4 В 5 6 Д К 7 8 (от нижней карты к верхней); затем положите вторую стопку поверх первой, поверните колоду лицевой стороной вверх и разложите в две стопки Т 2 3 4 5 6 7 8 и 9 10 В Д К. Соедините эти две стопки и поверните их лицевой стороной вверх. Колода отсортирована.

Докажите, что приведенную выше последовательность карт нельзя отсортировать в *убывающем* порядке: К Д В . . . 2 Т, от верхней карты к нижней, за два просмотра, даже если разрешено раскладывать карты в три стопки. (Сдавать карты, нужно всегда сверху колоды, поворачивая их при раздаче рубашкой вверх. На рисунке верхняя карта колоды изображена справа, а нижняя — слева.)

15. [M25] Рассмотрите задачу из упр. 14 в случае, когда карты раздаются лицевой стороной вверх, а не вниз. Таким образом, один просмотр можно потратить на преобразование возрастающего порядка в убывающий. Сколько нужно просмотров?

Как только появится аналитическая машина, она, безусловно, определит дальнейший путь развития науки. Всякий раз, когда с ее помощью будет найден какой-либо результат, тут же возникнет вопрос: нельзя ли тот же результат получить на этой машине за кратчайшее время?

Чарльз Бэббидж (1864)

5.3. ОПТИМАЛЬНАЯ СОРТИРОВКА

Теперь, когда мы проанализировали такое множество методов внутренней сортировки, пришло время обратиться к более общему вопросу: *какой метод внутренней сортировки наилучший?* Существует ли такой верхний предел скорости сортировки, которого бы не мог достичь ни один программист, как бы искусен он ни был?

Разумеется, наилучшего возможного способа сортировки *нет*; мы должны точно определить, что понимать под словом "наилучший", но не существует наилучшего возможного способа определить слово "наилучший". Аналогичные вопросы об оптимальности алгоритмов мы обсуждали в п. 4.3.3, 4.6.3 и 4.6.4, где рассматривалось умножение с высокой точностью и вычисление полиномов. В каждом случае, для того чтобы выполнялись условия "достаточности", т. е. чтобы задача стала разрешимой, необходимо было сформулировать довольно простое определение алгоритма, "наилучшего из возможных". И в каждом случае перед нами вставали интереснейшие задачи, настолько сложные, что они до сих пор полностью не решены. Так же обстоит дело и с сортировкой: были получены некоторые интересные результаты, но осталось еще много интригующих вопросов, на которые до сих пор нет ответов.

Изучение внутреннего механизма методов сортировки обычно было направлено на минимизацию числа сравнений ключей при сортировке n элементов, или слиянии m элементов с n элементами, или выборе t -го наибольшего элемента из неупорядоченного набора n элементов. В п. 5.3.1, 5.3.2 и 5.3.3 эти вопросы обсуждаются в общем случае; в п. 5.3.4 рассматриваются аналогичные вопросы с интересным ограничением: последовательность сравнений должна быть, по существу, заранее фиксирована. Некоторые другие типы интересных теоретических вопросов, связанных с оптимальной сортировкой, можно найти в упражнениях к п. 5.3.4 и в обсуждении внешней сортировки в п. 5.4.4.

5.3.1. Сортировка с минимальным числом сравнений

Очевидно, минимальное число сравнений ключей, необходимое для сортировки n элементов, равно *нулю*, поскольку, как мы видели, существуют методы поразрядной сортировки, в которых вообще не выполняется сравнений. В самом деле, можно написать MIX-программы, способные сортировать и не содержащие тем не менее ни одной команды условного перехода! (См. упр. 5-6 в начале этой главы.) Мы также встречались с несколькими методами сортировки, которые, по существу, были основаны на сравнении ключей, но время работы которых на деле определялось другими факторами, такими, как перемещение данных, вспомогательные операции и т. д.

Поэтому ясно, что подсчет числа сравнений — не единственный способ измерить эффективность метода сортировки. Однако в любом случае небезынтересно провести тщательное исследование числа сравнений, поскольку теоретическое изучение этого вопроса позволит нам с пользой для дела проникнуть во внутреннюю природу процессов сортировки, а также поможет отточить мастерство для решения более практических задач, которые могут встать перед нами в будущем.

Чтобы исключить поразрядную сортировку, где совсем не выполняется сравнений, ограничимся обсуждением методов сортировки, основанных только на абстрактном линейном отношении порядка " $<$ " между ключами, рассмотренном в начале этой главы. Для простоты мы также ограничим свое обсуждение случаем *различных* ключей, а это значит, что при любом сравнении ключей K_i и K_j

возможны лишь два исхода: либо $K_i < K_j$, либо $K_i > K_j$. (Распространение этой теории на общий случай, когда допускаются равные ключи, см. в упр. от 3 до 12.)

Задачу сортировки посредством сравнений можно сформулировать также другими эквивалентными способами. Если есть n грузов и весы с двумя чашами, то каково минимальное число взвешиваний, необходимое для того, чтобы расположить грузы по порядку в соответствии с весом, если в каждой чаше весов помещается только один груз? Или же, если в некотором турнире участвуют n игроков, то каково наименьшее число игр, достаточное для того, чтобы распределить места между соревнующимися в предположении, что силы игроков можно линейно упорядочить (ничейные результаты не допускаются).

Методы сортировки n элементов, удовлетворяющие указанным ограничениям, можно представить посредством структуры расширенного бинарного дерева, такого, как показано на рис. 34. Каждый *внутренний узел* (изображенный в виде кружочка) содержит два индекса " $i : j$ " и означает сравнение ключей K_i и K_j . Левое поддерево этого узла соответствует последующим сравнениям, которые необходимо выполнить, если $K_i < K_j$, а правое поддерево — тем действиям, которые необходимо предпринять в случае $K_i > K_j$. Каждый *внешний узел* дерева (изображенный в виде прямоугольника) содержит перестановку $a_1 a_2 \dots a_n$.

Picture: Рис. 34. Дерево сравнений для сортировки трех элементов.

множества $\{1, 2, \dots, n\}$, обозначающую тот факт, что было установлено упорядочение

$$K_{a_1} < K_{a_2} < \dots < K_{a_n}.$$

(Если взглянуть на путь от корня к этому внешнему узлу, то каждое из $n - 1$ соотношений $K_{a_i} < K_{a_{i+1}}$, где $1 \leq i < n$, будет результатом некоторого сравнения $a_i : a_{i+1}$ или $a_{i+1} : a_i$ на этом пути.)

Так, на рис. 34 представлен метод сортировки, согласно которому нужно сначала сравнить K_1 с K_2 ; если $K_1 > K_2$, то продолжать (двигаясь по правому поддереву) сравнивать K_2 с K_3 , а затем, если $K_2 < K_3$, сравнить K_1 с K_3 ; наконец, если $K_1 > K_3$, становится ясно, что $K_2 < K_3 < K_1$. Реальный алгоритм сортировки обычно будет также перемещать ключи по файлу, но нас интересуют только сравнения, поэтому мы игнорируем все перемещения данных. При сравнении K_i с K_j в этом дереве всегда имеются в виду *исходные* ключи K_i и K_j , а не те ключи, которые могли занять i -ю и j -ю позиции в файле в результате перемещения записей.

Возможны и избыточные сравнения; например, на рис. 35 нет необходимости сравнивать $3 : 1$, поскольку из неравенств $K_1 < K_2$ и $K_2 < K_3$ следует $K_1 < K_3$. Никакая перестановка не может соответствовать левому поддереву узла $< 3 : 1 >$ на рис. 35, так что эта часть алгоритма никогда не будет выполняться! Поскольку нас интересует минимальное число сравнений, то можно считать, что избыточных сравнений не производится; с этого момента мы будем иметь дело со структурой расширенного бинарного дерева, в котором каждому внешнему узлу соответствует некоторая перестановка. Все перестановки исходных ключей возможны, и каждая перестановка определяет единственный путь от корня к внешнему узлу; отсюда вытекает, что *в дереве сравнений для сортировки n элементов без избыточных сравнений имеется ровно $n!$ внешних узлов.*

Picture: Рис. 35. Пример избыточного сравнения.

Оптимизация в наихудшем случае. Первая естественным образом возникающая задача — найти деревья сравнений, минимизирующие *максимальное* число выполняемых сравнений. (Позже мы рассмотрим *среднее* число сравнений.)

Пусть $S(n)$ — минимальное число сравнений, достаточное для сортировки n элементов. Если все внутренние узлы в дереве сравнений располагаются на уровнях $< k$, то очевидно, что в дереве не может быть более 2^k узлов. Следовательно, полагая $k = S(n)$, имеем

$$n! \leq 2^{S(n)}.$$

Поскольку $S(n)$ — целое число, то можно записать эту формулу иначе, получив нижнюю оценку:

$$S(n) \geq \lceil \log_2 n! \rceil. \quad (1)$$

Заметим, что по формуле Стирлинга

$$\lceil \log_2 n! \rceil = n \log_2 n - n/(\ln 2) + \frac{1}{2} \log_2 n + O(1); \quad (2)$$

следовательно, необходимо выполнить около $n \log_2 n$ сравнений.

Соотношение (1) часто называют "теоретико-информационной нижней оценкой", поскольку специалист в области теории информации сказал бы, что в процессе сортировки проверяется $(\log_2 n!)$ "битов информации"; каждое сравнение дает не более одного "бита информации". Такие деревья, как на рис. 34, называют также "вопросниками" ("questionnaires"), а их математические свойства исследованы в книге Клода Пикара *Théorie des questionnaires* (Paris: Gauthier-Villars, 1965). Из всех рассмотренных нами методов сортировки три метода требуют меньше всего сравнений: бинарные вставки (ср. с п. 5.2.1), выбор из дерева (ср. с п. 5.2.3) и простое двухпутевое слияние, как оно описано в алгоритме 5.2.4L. Нетрудно видеть, что максимальное число сравнений для метода бинарных вставок равно

$$B(n) = \sum_{1 \leq k \leq n} \lceil \log_2 k \rceil = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1 \quad (3)$$

(ср. с упр. 1.2.4-42), а максимальное число сравнений для алгоритма 5.2.4L приведено в упр. 5.2.4-14. Оказывается (см. п. 5.3.3), что для выбора из дерева верхняя оценка числа сравнений либо такая же, как для бинарных вставок, либо такая же, как для двухпутевого слияния, в зависимости от того, как строится дерево. Во всех трех случаях имеем асимптотическое значение $n \log_2 n$; объединяя верхнюю и нижнюю оценки для $S(n)$, докажем, что

$$\lim_{n \rightarrow \infty} \frac{S(n)}{n \log_2 n} = 1. \quad (4)$$

Таким образом, мы получили приближенную формулу для $S(n)$, однако желательно иметь более точную информацию. В следующей таблице приведены значения указанных выше величин при малых n :

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \log_2 n! \rceil$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n)$	0	1	3	6	8	11	14	17	21	25	29	33	37	41	45	49	54
$L(n)$	0	1	3	5	9	11	14	17	25	27	30	33	38	41	45	49	65

Здесь $B(n)$ и $L(n)$ относятся соответственно к бинарным вставкам и слиянию списков. Можно показать, что $B(n) \leq L(n)$ при любом n (см. упр. 2).

Как видно из приведенной выше таблицы, $S(4) = 5$, но $S(5)$ может равняться либо 7, либо 8. В результате снова приходим к задаче, поставленной в начале § 5.2. Каков наилучший способ сортировки пяти элементов? Возможна ли сортировка пяти элементов при помощи всего семи сравнений?

Такая сортировка возможна, но сам способ найти не так просто. Начинаем так же, как при сортировке четырех элементов посредством слияний, сравнивая сперва $K_1 : K_2$, затем $K_3 : K_4$, а затем наибольшие элементы обеих пар. Эти сравнения порождают конфигурацию, которую можно изобразить диаграммой

Picture: Рис. стр. 222

показывающей, что $a < b < d$ и $c < d$. (Для представления известных отношений порядка между элементами удобно воспользоваться направленными графами, такими, как этот, где неравенство $x < y$ считается известным тогда и только тогда, когда на графе есть путь от x к y .) Теперь вставляем пятый элемент $K_5 = e$ в соответствующее место среди $\{a, b, d\}$; для этого требуются всего два сравнения, поскольку можно сравнить его сначала с b , а затем с a или d . Таким образом, остается одна из четырех возможностей:

Picture: p.223

и в каждом случае достаточно еще двух сравнений, чтобы вставить c в цепочку остальных элементов, меньших d . Такой способ сортировки пяти элементов впервые обнаружил Г. Б. Демут [Ph. D. thesis, Stanford University (Oct., 1956). 41–43].

Сортировка вставками и слиянием. Изящное обобщение изложенного выше метода принадлежит Лестеру Форду мл. и Селмеру М. Джонсону. Поскольку оно объединяет некоторые особенности двух способов сортировки: посредством слияний и посредством вставок, то мы назовем этот метод *сортировкой вставками и слиянием*. Рассмотрим, например, задачу сортировки 21 элемента. Начать можно со сравнений десяти пар ключей $K_1 : K_2, K_3 : K_4, \dots, K_{19} : K_{20}$, затем следует отсортировать вставками и слиянием большие элементы пар. В результате получим конфигурацию

Picture: 223.2

аналогичную (5). Следующий шаг состоит в том, чтобы вставить элемент b_3 в последовательность $\{b_1, a_1, a_2\}$, а затем b_2 —в последовательность остальных элементов, меньших a_2 , приходим к конфигурации

Picture: 223.3

Назовем верхние элементы *главной цепочкой*. Элемент b_5 можно вставить в главную цепочку за три сравнения (сравнив его сначала с c_4 , затем с c_2 или c_6 и т. д.); затем еще за три сравнения можно переместить в главную цепочку b_4 , что приводит к конфигурации

Picture: 224.1

Следующий шаг решающий; ясно ли вам, что делать дальше? При помощи всего четырех сравнений вставляем b_{11} (а не b_7) в главную цепочку. После этого элементы $b_{10}, b_9, b_8, b_7, b_6$ (именно в таком порядке) можно вставить в нужное место в главной цепочке не более чем за четыре сравнения каждый.

Аккуратный подсчет числа требуемых сравнений показывает, что 21 элемент можно отсортировать не более чем за $10 + 22 + 2 + 2 + 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 = 66$ шагов. Поскольку

$$2^{65} < 21! < 2^{66},$$

ясно также, что и в любом другом случае необходимо не менее 66 сравнений; следовательно,

$$S(21) = 66. \quad (10)$$

(При сортировке бинарными вставками понадобилось бы 74 сравнения.)

Вобщем случае сортировка вставками и слиянием для n элементов выглядит следующим образом:

- i) Произвести сравнения $\lfloor n/2 \rfloor$ непересекающихся пар элементов. (Если n нечетно, то один элемент не участвует в сравнениях.)
- ii) Отсортировать $\lfloor n/2 \rfloor$ больших элементов пар, найденных в шаге (i), вставками и слиянием.
- iii) Для элементов введем обозначения $a_1, a_2, \dots, a_{\lfloor n/2 \rfloor}, b_1, b_2, \dots, b_{\lceil n/2 \rceil}$, как в (7), где $a_1 \leq a_2 \leq \dots \leq a_{\lfloor n/2 \rfloor}$ и $b_i \leq a_i$ при $1 \leq i \leq \lfloor n/2 \rfloor$; назовем b_1 и все элементы a "главной цепочкой". Не трогая элементов b_j при $j > \lceil n/2 \rceil$, вставить бинарными вставками в главную цепочку остальные элементы b в следующем порядке:

$$b_3, b_2; b_5, b_4; b_{11}, b_{10}, \dots, b_6; b_{t_k}, b_{t_k-1}, \dots, b_{t_{k-1}+1}; \dots \quad (11)$$

Нам хотелось бы определить последовательность $(t_1, t_2, t_3, t_4, \dots) = (1, 3, 5, 11, \dots)$, участвующую в (11), таким образом, чтобы каждый из элементов $b_{t_k}, b_{t_k-1}, \dots, b_{t_{k-1}+1}$ можно было вставить в главную цепочку не более, чем за k сравнений. Обобщая (7), (8) и (9), получим диаграмму

Picture: 224.2

где главная цепочка до $a_{t_{k-1}}$ включительно содержит $2t_{k-1} + (t_k - t_{k-1} - 1)$ элементов. Это число должно быть меньше 2^k ; для нас лучше всего положить его равным $2^k - 1$, и тогда

$$t_{k-1} + t_k = 2^k. \quad (12)$$

Поскольку $t_1 = 1$, то для удобства можно положить $t_0 = 1$; тогда, суммируя геометрическую прогрессию, найдем

$$\begin{aligned} t_k &= 2^k - t_{k-1} = 2^k - 2^{k-1} + t_{k-2} = \dots \\ &\dots = 2^k - 2^{k-1} + \dots + (-1)^k 2^0 = (2^{k+1} + (-1)^k)/3. \end{aligned} \quad (13)$$

(Любопытно, что точно такая же последовательность возникла при изучении алгоритма вычисления наибольшего общего делителя двух целых чисел; ср. с упр. 4.5.2-27.)

Пусть $F(n)$ —число сравнений, необходимых для сортировки n элементов вставками и слиянием.

Ясно, что

$$F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G(\lceil n/2 \rceil), \quad (14)$$

где функция G описывает количество работы, выполняемой в шаге (iii). Если $t_{k-1} \leq m \leq t_k$, то, суммируя по частям, получаем

$$\begin{aligned} G(m) &= \sum_{1 \leq j < k} j(t_j - t_{j-1}) + k(m - t_{k-1}) = \\ &= km - (t_0 + t_1 + \dots + t_{k-1}). \end{aligned} \quad (15)$$

Положим

$$w_k = t_0 + t_1 + \dots + t_{k-1} = \lfloor 2^{k+1}/3 \rfloor, \quad (16)$$

и тогда $(w_0, w_1, w_2, w_3, w_4, \dots) = (0, 1, 2, 5, 10, 21, \dots)$. В упр. 13 показано, что

$$F(n) - F(n-1) = k \quad \text{тогда и только тогда, когда } w - k < n \leq w_{k+1}, \quad (17)$$

а последнее условие эквивалентно неравенствам

$$\begin{aligned} \frac{2^{k+1}}{3} < n \leq \frac{2^{k+2}}{3}, \\ k + 1 < \log_2(3n) \leq k + 2; \end{aligned}$$

следовательно,

$$F(n) - F(n-1) = \left\lceil \log_2 \left(\frac{3}{4}n \right) \right\rceil. \quad (18)$$

(Этой формулой мы обязаны А. Адьяну [Ph. D. thesis, Univ. of Minnesota (1969), 38–42].) Отсюда вытекает, что функция F выражается удивительно простой формулой:

$$F(n) = \sum_{1 \leq k \leq n} \left\lceil \log_2 \left(\frac{3}{4}k \right) \right\rceil, \quad (19)$$

которая очень похожа на соответствующую формулу (3) для бинарных вставок. В ”замкнутом виде” эту сумму можно найти в упр. 14.

Воспользовавшись (19), нетрудно построить таблицу значений функции $F(n)$; имеем

$n = 1$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \log_2 n! \rceil = 0$	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$F(n) = 0$	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50
$n = 18$	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
$\lceil \log_2 n! \rceil = 53$	57	62	66	70	75	80	84	89	94	98	103	108	113	118	123	
$F(n) = 54$	58	62	66	71	76	81	86	91	96	101	106	111	116	121	126	

Заметим, что при $1 \leq n \leq 11$ и при $20 \leq n \leq 21$ $F(n) = \lceil \log_2 n! \rceil$; таким образом, при этих значениях сортировка вставками и слиянием оптимальна:

$$S(n) = \lceil \log_2 n! \rceil = F(n) \quad \text{при } 1 \leq n \leq 11 \text{ и } 20 \leq n \leq 21. \quad (20)$$

Задачу нахождения функции $S(n)$ поставил Гуго Штейнгауз во втором издании своей классической книги *Mathematical Snapshots* (Oxford University Press, 1950, 38–39⁹). Он описал бинарные вставки, которые являются наилучшим способом сортировки n элементов при условии, что n -й элемент не рассматривается до тех пор, пока не отсортированы первые $n - 1$ элементов; и он сделал предположение о том, что метод бинарных вставок оптимален и в общем случае. Несколько лет спустя [*Calcutta Math. Soc. Golden Jubilee Commemoration*, 2 (1959), 323–327] он сообщил, что двое его коллег, С. Трибула и С. Пин, ”недавно” опровергли его предположение и нашли значения $S(n)$ при $n \leq 11$. Вероятно, Трибула и Пин независимо пришли к сортировке вставками и слиянием, по которой вскоре появилась публикация Л. Р. Форда и С. М. Джонсона [*АММ*, 66 (1950), 387–389].

После изобретения сортировки вставками и слиянием первым неизвестным значением функции $S(n)$ стало $S(12)$. Из табл. 1 видно, что число $12!$ довольно близко к 2^{29} , поэтому существование 29-шаговой процедуры сортировки 12 элементов весьма маловероятно. Для решения этого вопроса

⁹ Есть перевод первого издания этой книги: Математический калейдоскоп, М., Гостехиздат, 1949.– Прим. перев.

Марком Уэлсом был предпринят исчерпывающий поиск (заявший на вычислительной машине Маниак II около 60 ч.), который показал, что $S(12) = 30$ [Proc. IFIP Congress 65, 2 (1965), 497–498]. Итак, процедура вставок и слияний оказывается оптимальной и при $n = 12$.

Таблица 1
Значения факториалов в двоичной системе счисления

1	=	1!
10	=	2!
110	=	3!
11000	=	4!
1111000	=	5!
1011010000	=	6!
1001110110000	=	7!
1001110110000000	=	8!
1011000100110000000	=	9!
1101110101111100000000	=	10!
10011000010001010100000000	=	11!
1110010001100111111000000000	=	12!
10111001100101000110011000000000	=	13!
101000100110111111011001010000000000	=	14!
1001100000111011101101110101100000000000	=	15!
10011000001110111011101110101100000000000000	=	16!

***Более подробный анализ.** Чтобы более тщательно исследовать функцию $S(n)$, внимательно изучим диаграммы частичного упорядочения, такие, как (5). Полученные после нескольких сравнений сведения можно представить в виде направленного графа. Этот направленный граф в силу транзитивности отношения $<$ не содержит циклов; следовательно, его всегда можно изобразить таким образом, чтобы все дуги шли слева направо; поэтому удобнее удалить из диаграммы стрелки. В результате диаграмма (5) преобразуется в

Picture: p.227

Пусть G —такой направленный граф; обозначим через $T(G)$ число перестановок, согласующихся с G , т. е. число способов пометить вершины графа G целыми числами $\{1, 2, \dots, n\}$ так, чтобы число в вершине x было меньше числа в вершине y , если дуга $x \rightarrow y$ принадлежит G . Пример перестановки, согласующейся с (21): $a = 1, b = 4, c = 2, d = 5, e = 3$. Мы изучили функцию $T(G)$ для различных G в п. 5.1.4, где было замечено, что $T(G)$ есть число способов "топологически отсортировать" граф G .

Пусть G —граф из n элементов, который можно получить после k сравнений; определим *эффективность* графа G функцией

$$E(G) = \frac{n!}{2^k T(G)}. \quad (22)$$

(Эта идея принадлежит Франку Хуану и Шень Линю.) Строго говоря, эффективность не есть функция лишь самого графа G —она зависит от того пути, которым мы пришли к G в процессе сортировки, однако нам удобно допустить эту маленькую неточность. Выполнив еще одно сравнение над элементами i и j , получим два графа G_1 , и G_2 : один—для случая $K_i < K_j$ и другой—для случая $K_i > K_j$. Ясно, что

$$T(G) = T(G_1) + T(G_2).$$

Если $T(G_1) \geq T(G_2)$, то имеем

$$T(G) \leq 2T(G_1),$$

$$E(G_1) = \frac{n!}{2^{k+1}T(G_1)} = \frac{E(G)T(G)}{2T(G_1)} \leq E(G). \quad (23)$$

Следовательно, каждое сравнение приводит к графу меньшей или равной эффективности; нельзя увеличить эффективность за счет дополнительных сравнений.

Заметим, что если G совсем не содержит дуг, то $k = 0$ и $T(G) = n!$, т. е. начальная эффективность равна 1. Если же граф G представляет окончательный результат сортировки, то G выглядит как отрезок прямой, и $T(G) = 1$. Так, например, если нам нужно построить процедуру сортировки, которая бы сортировала пять элементов за семь или менее сравнений, то необходимо получить линейный граф

Picture: 228.1

эффективность которого равна $5!/(2^7 \times 1) = 120/128 = 15/16$. Отсюда следует, что все графы, возникающие в процессе сортировки, должны иметь эффективность $\geq \frac{15}{16}$; если бы появился какой-нибудь граф меньшей эффективности, то по крайней мере один из его потомков тоже имел бы меньшую эффективность, и мы бы в конце концов пришли к линейному графу с эффективностью $< \frac{15}{16}$. В общем случае это рассуждение показывает, что все графы, соответствующие узлам дерева для некоторой процедуры сортировки n элементов, должны иметь эффективность $\geq n!/2^l$, где $l + 1$ — число уровней в дереве. Это еще один способ доказательства неравенства $S(n) \geq \lceil \log_2 n! \rceil$, хотя такое рассуждение на самом деле не сильно отличается от приведенного выше.

Граф (21) имеет эффективность 1, поскольку $T(G) = 15$ и граф G был получен за три сравнения. Чтобы выяснить, какие вершины должны участвовать в следующем сравнении, можно построить матрицу сравнений

$$C(G) = \begin{matrix} & a & b & c & d & e \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{pmatrix} 0 & 15 & 10 & 15 & 11 \\ 0 & 0 & 5 & 15 & 7 \\ 5 & 10 & 0 & 15 & 9 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 8 & 6 & 12 & 0 \end{pmatrix} \end{matrix}, \quad (24)$$

где C_{ij} есть $T(G_1)$ для графа G_1 , полученного из G путем добавления дуги $i \rightarrow j$. Если мы, например, сравним K_c с K_e , то 15 перестановок, согласующихся с G , распадутся на две группы: $C_{ec} = 6$, в которых $K_e < K_c$, и $C_{ce} = 9$, в которых $K_c < K_e$. Последний граф имел бы эффективность $15/(2 \times 9) = \frac{5}{6} < \frac{15}{16}$, так что это сравнение не может привести к семишаговой процедуре сортировки. Следующим сравнением, для того чтобы сохранить эффективность $\geq \frac{15}{16}$, *обязано* быть $K_b : K_e$.

Понятие "эффективность" особенно полезно при рассмотрении "связных компонент" графов. Возьмем, например, граф

Picture: p.229.1

он состоит из двух компонент

Picture: p.229.2

где ни одна дуга не соединяет G' и G'' ; следовательно, он был образован путем нескольких сравнений вершин только G' и независимо нескольких сравнений вершин только G'' . В общем случае предположим, что граф $G = G' + G''$ не содержит дуг, связывающих G' и G'' , где G' и G'' имеют соответственно n' и n'' вершин; легко видеть, что

$$T(G) = \binom{n' + n''}{n'} T(G') T(G''), \quad (25)$$

поскольку каждая перестановка, согласующаяся с G , получается путем выбора n' элементов, которые считаются принадлежащими графу G' , и последующего составления перестановки, согласующейся с G' , и независимо, перестановки, согласующейся с G'' . Пусть внутри G' выполнено k' сравнений, а внутри G'' — соответственно k'' сравнений; получаем основной результат

$$E(G) = \frac{(n' + n'')!}{2^{k' + k''} T(G)} = \frac{n'!}{2^{k'} T(G')} \cdot \frac{n''!}{2^{k''} T(G'')} = E(G') E(G''), \quad (26)$$

показывающий, что между эффективностью графа и эффективностями его компонент существует простая связь. Поэтому мы можем ограничить наше рассмотрение графами, имеющими только одну компоненту.

Теперь предположим, что G' и G'' — однокомпонентные графы, и мы хотим связать их вместе, сравнив вершину x графа G' с вершиной y графа G'' . Нужно выяснить, насколько эффективным получится граф. Для этой цели нам необходима функция, которую можно обозначить через

$$\binom{p}{m} < \binom{q}{n}, \quad (27)$$

равная по определению числу перестановок, согласующихся с графом

Picture: p.230

Таким образом, $\binom{p}{m} < \binom{q}{n}$ есть произведение $\binom{m+n}{m}$ на вероятность того, что p -й наименьший элемент из множества m чисел меньше q -го наименьшего элемента из независимо выбранного множества n чисел.

Вупр. 17 показано, что функцию $\binom{p < q}{m < n}$ можно выразить через биномиальные коэффициенты двумя способами:

$$\begin{aligned} \binom{p < q}{m < n} &= \sum_{0 \leq k < q} \binom{m - p + n - k}{m - p} \binom{p - 1 + 1}{p - 1} = \\ &= \sum_{p \leq j \leq m} \binom{n - q + m - j}{n - q} \binom{q - 1 + j}{q - 1}. \end{aligned} \tag{29}$$

(Между прочим, алгебраически никоим образом не очевидно, что эти две суммы произведений биномиальных коэффициентов должны быть равны.) Имеем также формулы

$$\binom{p < q}{m < n} + \binom{q < p}{n < m} = \binom{m + n}{m}, \tag{30}$$

$$\binom{q < p}{n < m} = \binom{m + 1 - p < n + 1 - q}{m < n}. \tag{31}$$

Для ясности рассмотрим два графа

Picture: p.231

Нетрудно показать простым пересчетом, что $T(G') = 42$ и $T(G'') = 5$; так что если G —граф с 11 вершинами, содержащий G' и G'' в качестве своих компонент, то по формуле (25) $T(G) = \binom{11}{4} \cdot 42 \cdot 5 = 69\,300$. Это число перестановок слишком внушительно, чтобы их можно было выписать и, таким образом, выяснить, в скольких из них $x_i < y_j$ для всех i и j . Однако это вычисление менее чем за час можно проделать вручную следующим образом. Построим матрицы $A(G')$ и $A(G'')$, где A_{ik} —число перестановок, согласующихся с G' (или G''), в которых x_i (или y_i) равно k . Тогда число перестановок, согласующихся с G , в которых x_i меньше y_j , есть сумма по всем p и q , $1 \leq p \leq 7$, $1 \leq q \leq 4$, произведений (ip) -го элемента матрицы $A(G')$ на $\binom{p < q}{7 < 4}$ и на (jq) -й элемент матрицы $A(G'')$. Иначе говоря, нужно вычислить произведение матриц $A(G') \cdot L \cdot A(G'')^T$, где $L_{pq} = \binom{p < q}{7 < 4}$. Оно равно

$$\begin{pmatrix} 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 18 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \end{pmatrix} \begin{pmatrix} 210 & 294 & 322 & 329 \\ 126 & 238 & 301 & 325 \\ 70 & 175 & 265 & 315 \\ 35 & 115 & 215 & 295 \\ 15 & 65 & 155 & 260 \\ 5 & 29 & 92 & 204 \\ 1 & 8 & 36 & 120 \end{pmatrix} \begin{pmatrix} 2 & 3 & 0 & 0 \\ 2 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 48169 & 42042 & 66858 & 64031 \\ 22825 & 16005 & 53295 & 46475 \\ 48169 & 42042 & 66858 & 64031 \\ 22110 & 14850 & 54450 & 47190 \\ 5269 & 2442 & 27258 & 21131 \\ 22825 & 16005 & 53295 & 46475 \\ 5269 & 2442 & 27258 & 21131 \end{pmatrix}.$$

Таким образом, "наилучший" способ соединить G' с G'' —это сравнить x_1 с y_2 ; в 42 042 случаях получим $x_1 < y_2$ и в 69 300 – 42 042 = 27 258 случаях— $x_1 > y_2$. (В силу симметрии, по существу, к тем же результатам привели бы сравнения x_3 с y_2 , x_5 с y_3 или x_7 с y_3 .) Эффективность полученного графа для $x_1 < y_2$ равна

$$\frac{69\,300}{84\,084} E(G')E(G''),$$

т. е. не особенно высока; следовательно, по-видимому, вообще не стоит ни в одном методе сортировки применять соединение G' с G'' ! Смысл этого примера в том, что мы смогли принять такое решение, не производя непомерно больших вычислений. Этими идеями можно воспользоваться также для независимого подтверждения принадлежащего Марку Уэлсу доказательства того, что $S(12) = 30$. Начав с графа, содержащего одну вершину, мы можем повторять попытки добавления сравнений к одному из наших графов G или к паре компонент графа G' и G'' с таким расчетом, чтобы оба полученных графа содержали 12

Picture: Рис. 36. Некоторые графы и их эффективности, полученные в начале длинного доказательства неравенства $S(12) > 29$.

или менее вершин и обладали эффективностью $\geq 12!/2^{29} \approx 0.89221$. Всякий раз, как это оказывается возможным, мы выбираем граф с меньшей эффективностью и добавляем его к нашему множеству, если только он не является изоморфным одному из уже включенных в множество графов (или двойственным к нему, т. е. получается обращением отношения порядка). Если оба полученных графа имеют одинаковую эффективность, то произвольным образом выбирается один из них. Первые 24 графа, полученные таким способом, изображены на рис. 36, где приведены также их эффективности.

При помощи вычислительной машины было построено ровно 1594 графа, прежде чем этот процесс завершился. Поскольку граф не был получен, можно сделать вывод о том, что $S(12) > 29$. Весьма правдоподобно, что и для доказательства неравенства $S(22) > 70$ можно произвести аналогичный эксперимент за вполне разумное время, поскольку $22!/2^{70} \approx 0.952$ —это чрезвычайно высокая эффективность для сортировки за 70 шагов. (Из 1594 найденных графов с 12 или менее вершинами всего 92 имеют столь высокую эффективность.)

Промежуточные результаты дают веские основания предположить, что $S(13) = 33$, и, следовательно, сортировка вставками и слиянием не оптимальна при $n = 13$. Но до сих пор никому не удалось обнаружить *ни одного* такого значения n , что $S(n) < F(n)$.

Наверняка можно доказать, что $S(16) < F(16)$, поскольку $F(16)$ —это как раз такое число сравнений, какое требуется, чтобы сначала отсортировать 10 элементов за $S(10)$ шагов, а затем посредством бинарных вставок вставить по одному остальные шесть элементов. Непременен должен существовать более хороший способ!

Среднее число сравнений. До сих пор мы рассматривали процедуры, наилучшие в том смысле, что они не плохи в наихудшем случае; мы искали "минимаксные" процедуры, минимизирующие *максимальное* число сравнений. Поищем теперь "минисредние" процедуры, минимизирующие *среднее* число сравнений в предположении, что входные данные случайны, т. е. все перестановки равновероятны.

Рассмотрим еще раз изображенное на рис. 34 представление процедуры сортировки в виде дерева. Среднее число сравнений по всем перестановкам для этого дерева равно

$$\frac{2 + 3 + 3 + 3 + 3 + 2}{6} = 2\frac{2}{3}.$$

В общем случае среднее число сравнений для метода сортировки есть *длина внешнего пути дерева*, деленная на n . (Напомним, что длина внешнего пути—это сумма всех расстояний от корня до каждого из внешних узлов; см. п. 2.3.4.5.) Из обсуждения в п. 2.3.4.5 легко видеть, что минимум длины внешнего пути достигается на таком бинарном дереве с N внешними узлами, у которого имеется $2^q - N$ внешних узлов на уровне $q - 1$ и $2N - 2^q$ на уровне q , где $q = \lceil \log_2 N \rceil$. (Корень находится на нулевом уровне.) Таким образом, минимальная длина внешнего пути равна

$$(q - 1)(2^q - N) + q(2N - 2^q) = (q + 1)N - 2^q. \quad (33)$$

Имеется еще один интересный способ охарактеризовать минимальную длину пути: *расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l , что все внешние узлы находятся на уровнях l и $l + 1$.* (См. упр. 20.)

Если положить $q = \log_2 N + \theta$, где $0 \leq \theta < 1$, то формула минимальной длины внешнего пути примет вид

$$N(\log_2 N + 1 + \theta - 2^\theta). \quad (34)$$

График функции $1 + \theta - 2^\theta$ изображен на рис. 37; при $0 < \theta < 1$ она принимает положительные, но очень малые значения, не превышающие

$$1 - (1 + \ln \ln 2)/(\ln 2) = 0.08607\ 13320\ 55934 +. \quad (35)$$

Picture: Рис. 37. Функция $1 + \theta - 2^\theta$.

Таким образом, минимальная возможная средняя длина пути, которая получается в результате деления (34) на N , не может быть меньше $\log_2 N$ и больше $\log_2 N + 0.0861$. (Этот результат впервые получил Э. Глисон в неопубликованной заметке (1956).)

Если теперь положим $N = n!$, то получим нижнюю оценку среднего числа сравнений по всем схемам сортировки. Заметим, что оценка равна $\log_2 n! + O(1) = n \log_2 n - n/(\ln 2) + O(\log n)$.

Пусть $\bar{F}(n)$ —среднее число сравнений, выполняемых алгоритмом сортировки вставками и слиянием; имеем

	$n = 1$	2	3	4	5	6	7	8
Нижняя оценка (33)	$= 0$	2	16	112	832	6896	62368	619904
$n!\bar{F}(n)$	$= 0$	2	16	112	832	6912	62784	623232

Итак, сортировка вставками и слиянием оптимальна в обоих смыслах при $n \geq 5$, однако при $n = 6$ при таком методе выполняется в среднем $6912/720 = 9.6$ сравнения вместо возможных, согласно нижней оценке, $6896/720 = 9.577777\dots$ сравнения. Немного подумав, нетрудно понять, почему это так: некоторые "удачные" перестановки шести элементов сортируются методом вставок и слияний

всего за восемь сравнений, и тогда дерево сравнений имеет внешние узлы на трех, а не на двух уровнях. Из-за этого увеличивается суммарная длина пути. В упр. 24 показано, что можно построить процедуру сортировки шести элементов, требующую всегда девять или десять сравнений; следовательно, этот метод превосходит метод вставок и слияний в среднем и не хуже него в худшем случае.

И. Сезари [thesis (Paris, 1968), p. 37] доказал, что при $n = 7$ не существует метода сортировки, при котором бы достигалась нижняя оценка 62368 длины внешнего пути. (Используя результат упр. 22, этот факт можно доказать вручную.) С другой стороны, он построил процедуры, для которых достигается нижняя оценка (33) при $n = 9$ или 10. Вообще же задача минимизации среднего числа сравнений гораздо сложнее нахождения функции $S(n)$. Вполне даже возможно, что при некоторых n все методы, минимизирующие *среднее* число сравнений, в худшем случае требуют *более* $S(n)$ сравнений.

Упражнения

- [20] Нарисуйте деревья сравнений для сортировки четырех элементов методами (а) бинарных вставок; (б) простого двухпутевого слияния. Каковы длины внешних путей для этих деревьев?
- [M24] Докажите, что $B(n) \leq L(n)$, и найдите все значения n , при которых имеет место равенство.
- [M22] Если допускаются равные ключи, то при сортировке трех элементов возможны 13 исходов:

$$\begin{aligned} K_1 = K_2 = K_3, & \quad K_1 = K_2 < K_3, & \quad K_1 = K_3 < K_2, \\ K_2 = K_3 < K_1, & \quad K_1 < K_2 = K_3, & \quad K_2 < K_1 = K_3, \\ K_3 < K_1 = K_2, & \quad K_1 < K_2 < K_3, & \quad K_1 < K_3 < K_2, \\ K_2 < K_1 < K_3, & \quad K_2 < K_3 < K_1, & \quad K_3 < K_1 < K_2, & \quad K_3 < K_2 < K_1. \end{aligned}$$

Обозначим через P_n число возможных исходов при сортировке n элементов, если допускаются равные ключи, так что $(P_0, P_1, P_2, P_3, P_4, P_5, \dots) = (1, 1, 3, 13, 75, 541, \dots)$. Докажите, что производящая функция $P(z) = \sum_{n \geq 0} P_n z^n / n!$ равна $1/(2 - e^z)$. *Указание:* покажите, что

$$P_n = \sum_{k > 0} \binom{n}{k} P_{n-k} \quad \text{при } n > 0.$$

- [BM27] (О. А. Гросс.) Найдите предел последовательности чисел P_n из упр. 3 при $n \rightarrow \infty$. [*Возможное указание:* рассмотрите частичное разложение в дробь $\operatorname{ctg} z$.]
- [16] Если допускаются равные ключи, то каждое сравнение может иметь не два, а три результата: $K_i < K_j$, $K_i = K_j$, $K_i > K_j$. В этой общей ситуации алгоритмы сортировки можно представлять в виде расширенных *тернарных* деревьев, в которых каждый внутренний узел $i : j$ имеет три поддерева: левое, среднее и правое, соответствующие трем возможным исходам сравнения. Нарисуйте расширенное тернарное дерево, определяющее алгоритм сортировки для $n = 3$, если допускаются равные ключи. В вашем дереве должно быть 13 внешних узлов, соответствующих 13 возможным исходам, перечисленным в упр. 3.
- [M22] Пусть $S'(n)$ — минимальное число сравнений, необходимых для сортировки n элементов и выявления всех равенств между ключами, если каждое сравнение имеет три возможных результата, как в упр. 5. Нетрудно обобщить "теоретико-информационное" рассуждение, приведенное в тексте, и показать, что $S'(n) \geq \lceil \log_3 P_n \rceil$, где P_n — функция, изученная в упр. 3 и 4; докажите, что на самом деле $S'(n) = S(n)$.
- [20] Нарисуйте расширенное тернарное дерево в смысле упр. 5 для сортировки четырех элементов, если известно, что все ключи равны либо 0, либо 1. (Так, например, если $K_1 < K_2$ и $K_3 < K_4$, то понятно, что $K_1 = K_3$ и $K_2 = K_4$!) Добейтесь минимального числа сравнений в среднем, считая, что все 2^4 возможных исходных файлов равновероятны.
- [26] Нарисуйте расширенное тернарное дерево, как в упр. 7, для сортировки четырех элементов, если известно, что все ключи равны либо -1 , либо 0, либо $+1$. Добейтесь минимального числа сравнений в среднем, считая, что все 3^4 возможных исходных файлов равновероятны.
- [M20] Каково минимальное число сравнений в наихудшем случае при сортировке n элементов, как в упр. 7, когда известно, что все элементы равны либо 0, либо 1?
- [M25] Каково минимальное *среднее* число сравнений при сортировке n элементов, как в упр. 7, когда известно, что все ключи равны либо 0, либо 1? Результат представьте в виде функции от n .
- [BM25] Пусть $S_m(n)$ — минимальное число сравнений, необходимое в наихудшем случае для сортировки n элементов, как в упр. 5, если известно, что все ключи принадлежат множеству $\{1, 2, \dots, m\}$. [Таким образом, согласно упр. 6, $S_n(n) = S(n)$.] Докажите, что $S_m(n)$ стремится к $n \log_2 m$ при фиксированном m и $n \rightarrow \infty$.

- >12. [M25] (У. Г. Бурисиус, около 1954 г.) Предположим, что равные ключи могут встречаться, но мы хотим просто отсортировать элементы $\{K_1, K_2, \dots, K_n\}$ так, чтобы определить перестановку $a_1 a_2 \dots a_n$, такую, что $K_{a_1} \leq K_{a_2} \leq \dots \leq K_{a_n}$; нам не важно, имеет ли место равенство между элементами K_{a_i} и $K_{a_{i+1}}$.
- Будем говорить, что дерево сравнений *сильно* сортирует последовательность ключей, если оно сортирует эту последовательность в вышеуказанном смысле, независимо от того, какой путь выбирать в узлах $i : j$, для которых $K_i = K_j$. (Дерево бинарное, а не тернарное.)
- Докажите, что дерево, не содержащее избыточных сравнений, сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сортирует любую последовательность различных ключей.
 - Докажите, что дерево сравнений сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сильно сортирует любую последовательность нулей и единиц.
13. [M28] Докажите утверждение (17).
14. [M24] Выразите сумму (19) в "замкнутом виде".
15. [M21] Определите асимптотическое поведение функции $B(n)$ и $F(n)$ с точностью до $O(\log n)$. [Указание: покажите, что в обоих случаях коэффициент при n содержит функцию, изображенную на рис. 37.]
16. [BM26] (Ф. Хуан и Ш. Линь.) Докажите, что при $n \geq 22$ выполняется неравенство $F(n) > \lceil \log_2 n! \rceil$.
17. [M20] Докажите тождество (29).
18. [20] Если бы процедура, начало которой изображено на рис. 36, породила граф

Picture: p.236

с эффективностью $12!/2^{29}$, то было ли бы тем самым доказано, что $S(12) = 29$?

19. [40] Проведите эксперименты со следующим эвристическим правилом решения относительно того, какую пару ключей сравнивать следующей при конструировании дерева сравнений. Пусть на каждой стадии сортировки ключей $\{K_1, \dots, K_n\}$ число ключей, о которых на основании выполненных до сих пор сравнений известно, что они $\leq K_i$, обозначается через u_i , а число ключей, о которых известно, что они $\geq K_i$, обозначается через v_i , $1 \leq i \leq n$.
- Перенумеруем ключи так, чтобы последовательность u_i/v_i стала возрастающей: $u_1/v_1 \leq u_2/v_2 \leq \dots \leq u_n/v_n$. Теперь сравним $K_i : K_{i+1}$, где i —индекс, минимизирующий выражение $|u_i v_{i+1} - u_{i+1} v_i|$. (Хотя этот метод использует гораздо меньше информации, чем содержится в полной матрице сравнений, подобной (24), он, как оказывается, во многих случаях дает оптимальные результаты.)
- >20. [M26] Докажите, что расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l , что все внешние узлы находятся на уровнях l и $l + 1$ (или, быть может, только на уровне l).
21. [M21] *Высотой* расширенного бинарного дерева называется максимальный номер уровня, на котором есть внешние узлы. Пусть x —внутренний узел расширенного бинарного дерева; обозначим через $t(x)$ число внешних узлов-потомков узла x , а через $l(x)$ корень левого поддерева узла x . Если x —внешний узел, то положим $t(x) = 1$. Докажите, что расширенное бинарное дерево имеет минимальную высоту среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняется неравенство

$$|t(x) - 2t(l(x))| \leq 2^{\lceil \log_2 t(x) \rceil} - t(x).$$

22. [M24] Продолжение упр. 21. Докажите, что бинарное дерево имеет минимальную длину внешнего пути среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняются неравенства

$$|t(x) - 2t(l(x))| \leq 2^{\lceil \log_2 t(x) \rceil} - t(x) \text{ и } |t(x) - 2t(l(x))| \leq t(x) - 2^{\lfloor \log_2 t(x) \rfloor}.$$

[Так, например, если $t(x) = 67$, то должно быть $t(l(x)) = 32, 33, 34$ или 35 . Если нужно просто минимизировать высоту дерева, то, согласно предыдущему упражнению, достаточно, чтобы $3 \leq t(l(x)) \leq 64$.]

23. [10] В тексте доказано [см. формулу (34)], что среднее число сравнений, выполняемых любым методом сортировки n элементов, не может быть меньше $\lceil \log_2 n! \rceil \approx n \log_2 n$. Однако при сортировке вставками в несколько списков (алгоритм 5.2.1M) затрачивается в среднем всего $O(n)$ единиц времени. Чем это объясняется?
24. [27] (К. Пикар.) Постройте такое дерево сортировки для шести элементов, чтобы все его внешние узлы располагались на уровнях 10 и 11.

25. [11] Если бы существовала процедура сортировки семи элементов, на которой достигался минимум среднего числа сравнений, вычисляемый при помощи формулы (34), то сколько внешних узлов было бы на уровне 13 соответствующего дерева?
26. [M42] Найдите процедуру сортировки для семи элементов, минимизирующую среднее число выполняемых сравнений.
- >27. [20] Пусть известно, что конфигурации $(K_1 < K_2 < K_3, K_1 < K_3 < K_2, K_2 < K_1 < K_3, K_2 < K_3 < K_1, K_3 < K_1 < K_2, K_3 < K_2 < K_1)$ встречаются с вероятностями соответственно (.01, .25, .01, .24, .25, .24). Найдите дерево сравнений, которое бы сортировало такие три элемента с наименьшим средним числом сравнений.
28. [40] Напишите MIX-программу, которая сортирует 5 однословных ключей за минимально возможное время, после чего останавливается. (См. основные правила в начале § 5.2.)
29. [M25] (С. М. Чэйз.) Пусть $a_1 a_2 \dots a_n$ — перестановка множества $\{1, 2, \dots, n\}$. Докажите, что любой алгоритм, который распознает, является ли данная перестановка четной или нечетной (т. е. содержит ли она четное или нечетное число инверсий), и основанный исключительно на сравнениях элементов a , должен выполнить не менее $n \log_2 n$ сравнений, хотя он имеет всего два возможных исхода.
30. [M23] (Оптимальная обменная сортировка.) Любой алгоритм обменной сортировки в смысле определения, данного в п. 5.2.2, можно представить в виде *дерева сравнений-обменов*, а именно в виде структуры бинарного дерева, внутренние узлы которого имеют вид $\langle i : j \rangle$, где $i < j$, и интерпретируются следующим образом: "если $K_i \leq K_j$, то продвинуться по левой ветви дерева; если $K_i > K_j$, то поменять местами записи i и j и продвинуться по правой ветви дерева" По достижении внешнего узла должны выполняться условия $K_1 \leq K_2 \leq \dots \leq K_n$. Таким образом, дерево сравнений-обменов отличается от дерева сравнений тем, что оно описывает не только операции сравнения, но и операции перемещения данных.
- Обозначим через $S_e(n)$ минимальное число сравнений-обменов, необходимых в наихудшем случае для сортировки элементов при помощи дерева сравнений-обменов. Докажите, что $S_e(n) \leq S(n) + n - 1$.
31. [M38] Продолжение упр. 30. Докажите, что $S_e(5) = 8$.
32. [M42] Продолжение упр. 31. Исследуйте значения функции $S_e(n)$ при малых $n > 5$.
33. [M30] (Т. Н. Хиббард.) *Вещественнозначным деревом поиска* порядка x с разрешением δ называется расширенное бинарное дерево, каждый узел которого содержит неотрицательное действительное значение, такое, что (i) значение в любом внешнем узле $\leq \delta$; (ii) значение в любом внутреннем узле \leq суммы значений двух его сыновей; (iii) значение в корне равно x . *Длина взвешенного пути* такого дерева определяется как сумма по всем внешним узлам номеров уровней этих узлов, умноженных на содержащиеся в них значения.

Докажите, что вещественнозначное дерево поиска порядка x с разрешением 1 имеет минимальную среди всех таких деревьев того же порядка и с тем же разрешением длину взвешенного пути тогда и только тогда, когда в (ii) имеет место равенство и для всех пар значений x_0 и x_1 , принадлежащих узлам-братьям, выполняются следующие условия: (iv) не существует целого числа $k \geq 0$, такого, что $x_0 < 2^k < x_1$ или $x_1 < 2^k < x_0$; (v) $\lceil x_0 \rceil - x_0 + \lceil x_1 \rceil - x_1 < 1$. (В частности, если x — целое число, то из условия (v) следует, что все значения в дереве — целые, а условие (iv) эквивалентно результату упр. 22.)

Докажите также, что соответствующая длина взвешенного пути равна $x \lceil \log_2 x \rceil + \lceil x \rceil - 2^{\lceil \log_2 x \rceil}$.

34. [M50] Определите точные значения функции $S(n)$ для бесконечного множества аргументов n .

5.3.2. *Слияние с минимальным числом сравнений

Рассмотрим теперь вопрос, имеющий отношение к предыдущему пункту: каков наилучший способ слияния упорядоченного множества m элементов с упорядоченным множеством n элементов? Обозначим элементы сливаемых множеств через

$$\begin{aligned} A_1 < A_2 < \dots < A_m, \\ B_1 < B_2 < \dots < B_n. \end{aligned} \quad (1)$$

Как и в п. 5.3.1, будем предполагать, что все $m+n$ элементов различны. Элементы A среди элементов B могут располагаться $\binom{m+n}{m}$ способами; таким образом, из рассуждения, которым мы воспользовались в задаче о сортировке, следует, что необходимо выполнить по крайней мере

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil \quad (2)$$

сравнений. Если положить $m = \alpha n$ и устремить n к ∞ , оставив α неизменным, то по формуле Стирлинга

$$\log_2 \binom{\alpha n + n}{\alpha n} = n((1 + \alpha) \log_2(1 + \alpha) - \alpha \log_2 \alpha) - \frac{1}{2} \log_2 n + O(1). \quad (3)$$

Обычная процедура слияния (алгоритм 5.2.4М) выполняет в худшем случае $m + n - 1$ сравнений.

Обозначим через $M(m, n)$ функцию, аналогичную $S(n)$, а именно минимальное число сравнений, заведомо достаточное для слияния m элементов с n элементами. Из только что сделанного наблюдения следует, что

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil \leq M(m, n) \leq m + n - 1 \quad \text{при всех } m, n \geq 1. \quad (4)$$

Формула (3) показывает, насколько далеко могут отстоять друг от друга нижняя и верхняя оценки. При $\alpha = 1$ (т. е. $m = n$) нижняя оценка равна $2n - \frac{1}{2} \log_2 n + O(1)$, так что обе оценки — величины одного порядка, но разность между ними может быть сколь угодно велика. При $\alpha = 0.5$ (т. е. $m = \frac{1}{2}n$) нижняя оценка равна

$$\frac{3}{2}n \left(\log_2 3 - \frac{2}{3} \right) + O(\log n),$$

что составляет примерно $\log_2 3 - \frac{2}{3} \approx 0.918$ от верхней оценки. С убыванием α разница между верхней и нижней оценками все увеличивается, поскольку стандартный алгоритм слияния разработан главным образом для файлов с $m \approx n$.

При $m = n$ задача о слиянии имеет весьма простое решение; неверной оказывается не верхняя, а нижняя оценка (4). Следующую теорему независимо доказали Р. Л. Грэхем и Р. М. Карп примерно в 1968 г.

Теорема М. При $m \geq 1$ справедливо равенство $M(m, m) = 2m - 1$.

Доказательство. Рассмотрим какой-нибудь алгоритм, который осуществляет слияние элементов $A_1 < \dots < A_m$ с $B_1 < \dots < B_m$. При сравнении элементов $A_i : B_j$ выберем ветвь $A_i < B_j$, если $i < j$, и ветвь $A_i > B_j$, если $i \geq j$. Слияние должно завершиться конфигурацией

$$B_1 < A_1 < B_2 < A_2 < \dots < B_m < A_m, \quad (5)$$

поскольку она согласуется со всеми выбранными ветвями. И каждое из $2m - 1$ сравнений $B_1 : A_1$, $A_1 : B_2$, $B_2 : A_2$, \dots , $B_m : A_m$ должно быть выполнено явно, иначе нашлись бы по меньшей мере две конфигурации, не противоречащие известным фактам. Если бы мы, например, не сравнили A_1 с B_2 , то конфигурация

$$B_1 < B_2 < A_1 < A_2 < \dots < B_m < A_m$$

была бы неотличима от (5). ■

Простая модификация этого доказательства дает аналогичную формулу

$$M(m, m+1) = 2m \quad \text{при } m \geq 0. \quad (6)$$

Нахождение нижних оценок. Теорема М показывает, что "теоретико-информационная" нижняя оценка (2) может сколь угодно далеко отстоять от истинной нижней границы; таким образом, метод доказательства теоремы М дает нам еще один способ нахождения нижних оценок. Такой метод доказательства часто рассматривается как порождение *дьявола*, некоего злого духа, который пытается принудить алгоритм работать как можно медленнее. Когда алгоритм слияния решает сравнить элементы $A_i : B_j$, дьявол так определяет судьбу сравнения, что вынуждает алгоритм избрать наиболее трудный путь. Если бы мы смогли придумать подходящего дьявола, то смогли бы убедиться в том, что всякий правильный алгоритм слияния должен выполнить довольно много сравнений. (Некоторые называют его не дьяволом, а "оракулом" или "демоном"; однако желательно избегать таких терминов в этом контексте, поскольку слово "оракул" имеет совсем другое значение в теории рекурсивных функций, а слово "демон" употребляется в другом смысле в терминологии, связанной с искусственным интеллектом.)

Мы будем использовать *дьяволов с ограниченными возможностями*, произвол которых лимитирован заранее заданными результатами некоторых сравнений. В методе слияния, находящемся под воздействием дьявола с ограниченными возможностями, ограничения считаются неизвестными и поэтому выполняются все необходимые сравнения даже в том случае, когда их результаты предопределены. Например, в доказательстве теоремы М мы ограничили все результаты сравнений условием (5), тем не менее в алгоритме слияния нельзя воспользоваться этим обстоятельством, чтобы избежать хотя бы одного сравнения.

Ограничения, которые мы будем использовать в следующем обсуждении, относятся к левому и правому концам файлов. Левые ограничения обозначаются символами

- (нет ограничения слева),
- \ (результаты всех сравнений не должны противоречить условию $A_1 < B_1$),
- / (результаты всех сравнений не должны противоречить условию $A_1 > B_1$).

Правые ограничения обозначаются символами

- (нет ограничения справа),
- \ (результаты всех сравнений не должны противоречить условию $A_m < B_n$),
- / (результаты всех сравнений не должны противоречить условию $A_m > B_n$).

Существует девять типов дьяволов, обозначаемых символами $\nabla M\phi$, где ∇ — левое ограничение, а ϕ — правое. Например, дьявол " $\backslash M \backslash$ " должен говорить, что $A_1 < B_j$ и $A_i < B_n$; дьявол ".M." не подчиняется никаким ограничениям. При некоторых малых значениях m и n дьяволы с ограниченными возможностями некоторых типов могут не существовать; при $m = 1$, очевидно, не может быть дьявола " $\backslash M$ ".

Займемся теперь построением весьма сложного, но чрезвычайно коварного дьявола для слияний. Он не всегда порождает оптимальные результаты, но дает нижние оценки, которые охватывают множество интересных случаев. Предположим, заданы m и n , а также левые и правые ограничения ∇ и ϕ , и пусть дьявола спрашивают, который из двух элементов A_i и B_j больше. Дьявол может, вообще говоря, применить шесть стратегий сведения задачи к случаю меньшего значения $m + n$:

Стратегия $A(k, l)$ для $i \leq k \leq m$ и $1 \leq l \leq j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_k\}$ с $\{B_1, \dots, B_{l-1}\}$ и $\{A_{k+1}, \dots, A_m\}$ с $\{B_l, \dots, B_n\}$. Тогда последующие сравнения $A_p : B_q$ дадут результаты: $A_p < B_q$, если $p \leq k$ и $q \geq l$, и $A_p > B_q$, если $p > k$ и $q < l$; они будут управляться дьяволом $(k, l - 1, \nabla, \cdot)$, если $p \leq k$ и $q < l$, и дьяволом $(m - k, n + 1 - l, \cdot, \phi)$, если $p > k$ и $q \geq l$.

Стратегия $B(k, l)$ для $i \leq k \leq m$ и $1 \leq l < j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_k\}$ с $\{B_1, \dots, B_l\}$ и $\{A_{k+1}, \dots, A_m\}$ с $\{B_l, \dots, B_n\}$ при условии $A_k < B_l < A_{k+1}$. (Заметим, что B_l участвует в обоих списках, подлежащих слиянию. Условие $A_k < B_l < A_{k+1}$ обеспечивает такое положение, при котором слияние одной пары файлов не может дать никакой информации, которая бы помогла при слиянии другой пары.) Тогда последующие сравнения $A_p : B_q$ дадут результаты: $A_p < B_q$, если $p \leq k$ и $q \geq l$, и $A_p > B_q$, если $p > k$ и $q \leq l$; они будут управляться дьяволом $(k, l, \nabla, \backslash)$, если $p \leq k$ и $q \leq l$, и дьяволом $(m - k, n + 1 - l, /, \phi)$, если $p > k$ и $q \geq l$.

Стратегия $C(k, l)$ для $i < k \leq m$ и $1 \leq l \leq j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_k\}$ с $\{B_1, \dots, B_{l-1}\}$ и $\{A_k, \dots, A_m\}$ с $\{B_l, \dots, B_n\}$ при условии $B_{l-1} < A_k < B_l$. (Аналогично стратегии B, но файлы A и B меняются ролями.)

Стратегия $A'(k, l)$ для $1 \leq k \leq i$ и $j \leq l \leq n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_{k-1}\}$ с $\{B_1, \dots, B_l\}$ и $\{A_k, \dots, A_m\}$ с $\{B_{l+1}, \dots, B_n\}$. (Аналогично стратегии A.)

Стратегия $B'(k, l)$ для $1 \leq k \leq i$ и $j < l \leq n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_{k-1}\}$ с $\{B_1, \dots, B_l\}$ и $\{A_k, \dots, A_m\}$ с $\{B_l, \dots, B_n\}$ при условии $A_{k-1} < B_l < A_k$. (Аналогично стратегии B.)

Стратегия $C'(k, l)$ для $1 \leq k \leq i$ и $j \leq l \leq n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \dots, A_k\}$ с $\{B_1, \dots, B_l\}$ и $\{A_k, \dots, A_m\}$ с $\{B_{l+1}, \dots, B_n\}$ при условии $B_l < A_k < B_{l+1}$. (Аналогично стратегии C.)

Из-за налагаемых ограничений приведенные выше стратегии не могут применяться в некоторых случаях, которые мы здесь приводим:

Стратегия	Не должна применяться, если
$A(k, 1), B(k, 1), C(k, 1)$	$\nabla = /$
$A'(1, l), B'(1, l), C'(1, l)$	$\nabla = \backslash$
$A(m, l), B(m, l), C(m, l)$	$\phi = /$
$A'(k, n), B'(k, n), C'(k, n)$	$\phi = \backslash$

Обозначим через $\nabla M\phi(m, n)$ максимальную нижнюю оценку, которую можно получить при помощи дьявола из описанного выше класса. Если первое сравнение есть $A_i : B_j$, то каждая стратегия,

если она применима, дает неравенства, связывающие эти девять функций, а именно

$$\begin{aligned} A(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M.(k, l - 1) + .M\phi(m - k, n + 1 - l); \\ B(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M\backslash(k, l) + /M\phi(m - k, n + 1 - l); \\ C(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M/(k, l - 1) + \backslash M\phi(m + 1 - k, n + 1 - l); \\ A'(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M.(k - 1, l) + .M\phi(m + 1 - k, n - l); \\ B'(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M\backslash(k - 1, l) + /M\phi(m + 1 - k, n + 1 - l); \\ C'(k, l) : \nabla M\phi(m, n) &\geq 1 + \nabla M/(k, l) + \backslash M\phi(m + 1 - k, n - l). \end{aligned}$$

При фиксированных i и j дьявол примет ту стратегию, которая максимизирует нижнюю оценку, задаваемую правыми частями неравенств; таким образом, $\nabla M_\phi(m, n)$ есть минимум этих нижних оценок по всем $1 \leq i \leq m$ и $1 \leq j \leq n$. Если m или n равны нулю, то и значение функции $\nabla M_\phi(m, n)$ равно 0.

Пусть, например, $m = 2$ и $n = 3$, а наш дьявол не ограничен. Если первым выполняется сравнение $A_1 : B_1$, то дьявол может принять стратегию $A'(1, 1)$, в результате чего потребуются еще $.M.(0, 1) + .M.(2, 2) = 3$ сравнения. Если первым выполняется сравнение $A_1 : B_3$, то он может выбрать стратегию $B(1, 2)$, тогда потребуются еще $.M\backslash(1, 2) + /M.(1, 2) = 4$ сравнения. Независимо от того, каково было первое сравнение, дьявол гарантирует выполнение еще по крайней мере 3 сравнений. Следовательно, $.M.(2, 3) = 4$.

Не так просто выполнить эти вычисления вручную, но при помощи ЭВМ можно довольно быстро получить таблицы функций $\nabla M\phi$. Эти функции обладают некоторыми очевидными свойствами симметрии

$$/M.(m, n) = .M\backslash(m, n) = \backslash M.(n, m) = .M/(n, m), \quad (7)$$

позволяющими свести наши девять функций всего лишь к четырем:

$$.M.(m, n), \quad /M.(m, n), \quad /M\backslash(m, n) \text{ и } /M/(m, n).$$

В табл. 1 приведены значения для всех $m, n \leq 10$. Наш дьявол для слияний определен таким образом, что

$$.M.(m, n) \leq M(m, n) \quad \text{при всех } m, n \geq 0. \quad (8)$$

Это соотношение содержит в качестве частного случая теорему М, поскольку при $|m - n| \leq 1$ наш дьявол изберет простую стратегию этой теоремы.

Рассмотрим теперь несколько простых соотношений, которым удовлетворяет функция M :

$$M(m, n) = M(n, m), \quad (9)$$

$$M(m, n) \leq M(m, n + 1), \quad (10)$$

$$M(k + m, n) \leq M(k, n) + M(m, n), \quad (11)$$

$$M(m, n) \leq \max(M(m, n - 1) + 1, M(m - 1, n) + 1), \quad (12)$$

$$M(m, ni) \leq \max(M(m, n - 2) + 1, M(m - 1, n) + 2) \quad \text{при } m \geq 1, n \leq 2. \quad (13)$$

Соотношение (12) следует из обычной процедуры слияния, если начать со сравнения элементов $A_1 : B_1$. Соотношение (13) выводится аналогично, только в первую очередь сравниваются $A_1 : B_2$; если $A_1 > B_2$, то нужно еще $M(m, n - 2)$ сравнений, если же $A_1 < B_2$, то можно вставить A_1 в соответствующее место и слить $\{A_2, \dots, A_m\}$ с $\{B_1, \dots, B_n\}$. Нетрудно видеть, что неравенства (12) и (13) можно обобщить:

$$M(m, n) \leq \max(M(m, n - k) + 1, \quad M(m - 1, n) + 1 + \lceil \log_2 k \rceil) \quad \text{при } m \geq 1, n \geq k, \quad (14)$$

выполнив сначала сравнение $A_1 : B_k$, и применив бинарный поиск в случае $A_1 < B_k$.

Оказывается, $M(m, n) = .M.(m, n)$ при всех $m, n \leq 10$, так что в табл. 1 в действительности приведены оптимальные значения для слияний. Это можно доказать, применяя соотношения (9)–(14), а также специальные построения для $(m, n) = (2, 8)$, $(3, 6)$ и $(5, 9)$, которые приводятся в упр. 8, 9 и 10.

Таблица 1

Нижние оценки для слияний, выполненных при участии "дьявола"

$.M.(m, n)$											$/M.(m, n)$										
	1	2	3	4	5	6	7	8	9	10	n	1	2	3	4	5	6	7	8	9	10
1	1	2	2	3	3	3	3	4	4	4	1	2	2	3	3	3	3	4	4	4	1
2	2	3	4	5	5	6	6	6	7	7	1	3	4	4	5	5	6	6	7	7	2
3	2	4	5	6	7	7	8	8	9	9	1	3	5	6	7	7	8	8	9	9	3
4	3	5	6	7	8	9	10	10	11	11	1	4	5	7	8	9	9	10	10	11	4
5	3	5	7	8	9	10	11	12	12	13	1	4	6	8	9	10	11	12	12	13	5
6	3	6	7	9	10	11	12	13	14	15	1	4	6	8	10	11	12	13	14	14	6
7	3	6	8	10	11	12	13	14	15	16	1	4	7	9	10	12	13	14	15	16	7
8	4	6	8	10	12	13	14	15	16	17	1	5	7	9	11	13	14	15	16	17	8
9	4	7	9	11	12	14	15	16	17	18	1	5	8	10	11	13	15	16	17	18	9
10	4	7	9	11	13	15	16	17	18	19	1	5	8	10	12	14	15	17	18	19	10
m																					m
$/M \setminus (m, n)$											$/M / (m, n)$										
1	$-\infty$	2	2	3	3	3	3	4	4	4	1	1	1	1	1	1	1	1	1	1	1
2	$-\infty$	2	4	4	5	5	6	6	7	7	1	3	3	4	4	4	4	5	5	5	2
3	$-\infty$	2	4	6	6	7	8	8	8	9	1	3	5	5	6	6	7	7	8	8	3
4	$-\infty$	2	5	6	8	8	9	10	10	11	1	4	5	7	7	8	9	9	9	10	4
5	$-\infty$	2	5	7	8	10	10	11	12	13	1	4	6	7	9	9	10	11	11	12	5
6	$-\infty$	2	5	7	9	10	12	13	14	14	1	4	6	8	9	11	11	12	13	14	6
7	$-\infty$	2	5	8	10	11	12	14	15	16	1	4	7	9	10	11	13	14	15	15	7
8	$-\infty$	2	6	8	10	12	13	15	16	17	1	5	7	9	11	12	14	15	16	17	8
9	$-\infty$	2	6	9	10	12	14	16	17	18	1	5	8	9	11	13	15	16	17	18	9
10	$-\infty$	2	6	9	11	13	15	16	18	19	1	5	8	10	12	14	15	17	18	19	10
	1	2	3	4	5	6	7	8	9	10	n	1	2	3	4	5	6	7	8	9	10

С другой стороны, такой дьявол не всегда дает наилучшую возможную нижнюю оценку; простейший пример: $m = 3, n = 11$, когда $.M.(3, 11) = 9$, а $M(3, 11) = 10$. Чтобы понять, где же в этом случае наш дьявол "промахнулся", нужно изучить мотивы, на которых основаны его решения; при дальнейшем тщательном исследовании обнаруживается, что если $(i, j) \neq (2, 6)$, то дьявол может отыскать стратегии, требующие 10 сравнений; если же $(i, j) = (2, 6)$, то ни одна стратегия не превосходит стратегии $A(2, 4)$, которая приводит к нижней оценке $1 + .M.(2, 3) + .M.(1, 8) = 9$. Необходимо, но не достаточно, чтобы процесс заканчивался слиянием $\{A_1, A_2\}$ с $\{B_1, B_2, B_3\}$ и $\{A_3\}$ с $\{B_4, \dots, B_{11}\}$; таким образом, нижняя оценка в этом случае оказывается не точной.

Аналогично можно показать, что $.M.(2, 38) = 10$, в то время как $M(2, 38) = 11$; значит, наш дьявол не достаточно искусен, чтобы справиться со случаем $m = 2$. Однако существует бесконечный класс значений, для которых он работает безупречно.

Теорема К.

$$\begin{aligned}
 M(m, m + 2) &= 2m + 1 && \text{при } m \geq 2, \\
 M(m, m + 3) &= 2m + 2 && \text{при } m \geq 4, \\
 M(m, m + 4) &= 2m + 3 && \text{при } m \geq 6.
 \end{aligned}$$

Доказательство. На самом деле мы можем доказывать эти соотношения, заменив M на $.M.$; при малых m эти результаты были получены с помощью ЭВМ, поэтому можно предполагать, что m достаточно велико. Можно также считать, что первое сравнение есть $A_i : B_j$, где $i \leq \lceil m/2 \rceil$. Если $j \leq i$, то воспользуемся стратегией $A'(i, i)$; тогда получим

$$.M.(m, m + d) \geq 1 + .M.(i - 1, i) + .M.(m + 1 - i, m + d - i) = 2m + d - 1,$$

применив индукцию по $d, d \leq 4$. Если $j > i$, то воспользуемся стратегией $A(i, i + 1)$; применив индукцию по m , получим

$$.M.(m, m + d) \geq 1 + .M.(i, i) + .M.(m - i, m + d - i) = 2m + d - 1.$$



Первые два утверждения теоремы К получили Ф. Хуан и Ш. Линь в 1969 г. Это доказательство дает основания предположить, что $M(m, m + d) = 2m + d - 1$ при всех достаточно больших m , где d фиксировано. (Ср. с упр. 6.)

Верхние оценки. Рассмотрим теперь *верхние* оценки функции $M(m, n)$; хорошие верхние оценки соответствуют эффективным алгоритмам слияния.

При $m = 1$ задача слияния эквивалентна задаче вставки, когда имеется $n + 1$ мест между элементами B_1, \dots, B_n , куда может попасть элемент A_1 . В этом случае нетрудно видеть, что *любое* расширенное бинарное дерево с $n + 1$ внешними узлами есть дерево для некоторого метода слияния! (См. упр. 2.) Следовательно, можно выбрать оптимальное бинарное дерево, реализовав теоретико-информационную нижнюю оценку

$$1 + \lceil \log_2 n \rceil = M(1, n) = \lceil \log_2(n + 1) \rceil. \tag{15}$$

Разумеется, бинарный поиск (п. 6.2.1)—простейший способ, позволяющий достичь этого значения.

Случай $m = 2$ чрезвычайно интересен, но он гораздо сложнее. Его полностью исследовали Р. Л. Грэхем, Ф. К. Хуан и Ш. Линь (см. упр. 11, 12, 13); имеем

$$M(2, n) = \left\lceil \log_2 \frac{7}{12}(n + 1) \right\rceil + \left\lceil \log_2 \frac{14}{17}(n + 1) \right\rceil. \tag{16}$$

Мы видели, что при $m = n$ оптимальна обычная процедура слияния, а при $m = 1$ оптимальна довольно сильно отличающаяся от нее процедура бинарного поиска. Нам же нужен некоторый промежуточный метод, объединяющий в себе лучшие черты алгоритмов обычного слияния и бинарного поиска. Формула (14) наводит на мысль о следующем алгоритме, которым мы обязаны Ф. К. Хуану и Ш. Линю [SIAM J. Computing, 1 (1972), 31–39].

Алгоритм Н. (Бинарное слияние.)

Н1 Если m или n равно 0, то остановиться.

Если $m \leq n$, то установить $t \leftarrow \lfloor \log_2(n/m) \rfloor$.

Если $m > n$, установить $t \leftarrow \lfloor \log_2(m/n) \rfloor$ и перейти к **Н4**.

Н2 Сравнить $A_m : B_{n+1-2^t}$. Если A_m меньше, то установить $n \leftarrow n - 2^t$ и возвратиться к шагу **Н1**.

Н3 Воспользовавшись методом бинарного поиска (который требует еще ровно t сравнений), вставить A_m в соответствующее место среди $\{B_{n+1-2^t}, \dots, B_n\}$. Если k —максимальный индекс, такой, что $B_k < A_m$, то установить $m \leftarrow m - 1$ и $n \leftarrow k$. Возвратиться к **Н1**.

Н4 (Шаги **Н4** и **Н5** подобны шагам **Н2** и **Н3**, но переменные m, n, A, B меняются ролями.). Если $B_n < A_{m+1-2^t}$, то установить $m \leftarrow m - 2^t$ и возвратиться к шагу **Н1**.

Н5 Вставить B_n в соответствующее место среди элементов A . Если k —максимальный индекс, такой, что $A_k < B_n$, то установить $m \leftarrow k$ и $n \leftarrow n - 1$. Возвратиться к шагу **Н1**. ■

В качестве примера работы этого алгоритма в табл. 2 показан процесс, слияния трех ключей $\{087, 503, 512\}$ с тринадцатью ключами $\{061, 154, \dots, 908\}$; в этом примере выполняется восемь сравнений.

Таблица 2

Пример применения метода бинарного слияния																		
А (сравниваются выделенные элементы)				В										Вывод				
{087 503 512 }	{061 154 170 275 426 509 612 653 677 703 765 897 908}																	
{087 603 512 }	{061 151 170 275 426 509 612 653 677}													{703 765 897 908}				
{087 503 512 }	{061 154 170 275 426 509 612}													{653 677 703 765 897 908}				
{087 503 512 }	{061 154 170 275 426 509 612 }													{653 677 703 765 897 908}				
{087 503 }	{061 154 170 275 426 509}													{512 612 653 677 703 765 897 908}				
{087 503 }	{061 154 170 275 426 509 }													{512 612 653 677 793 765 897 908}				
{ 087 }	{061 154 170 275 426}													{503 509 512 612 653 677 703 765 897 908}				
{ 087 }	{ 061 }													{154 170 275 426 503 509 512 612 653 677 703 765 897 908}				
	{061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908}																	

Пусть $H(m, n)$ —максимальное число сравнений, выполняемых алгоритмом Хуана и Линя. Чтобы вычислить $H(m, n)$, можно предположить, что $k = n$ в шаге **Н3** и $k = m$ в шаге **Н5**, поскольку при помощи индукции по n нетрудно доказать, что $H(m, n) \leq H(m, n + 1)$ при всех $n \geq m$. Таким образом, при $m \leq n$ имеем

$$H(m, n) = \max(H(m, n - 2^t) + 1, H(m - 1, n) + t + 1) \quad \text{при } 2^t m \leq n < 2^{t+1} m. \tag{17}$$

Заменим n на $2n + \varepsilon$ ($\varepsilon = 0$ или 1) и получим

$$H(m, 2n + \varepsilon) = \max(H(m, 2n + \varepsilon - 2^{t+1}) + 1, H(m - 1, 2n + \varepsilon) + t + 2) \quad \text{при } 2^t m \leq n < 2^{t+1} m.$$

Отсюда вытекает, если применить индукцию по n , что

$$H(m, 2n + \varepsilon) = H(m, n) + m \quad \text{при } m \leq n, \varepsilon = 0 \text{ или } 1. \quad (18)$$

Легко видеть также, что $H(m, n) = m + n - 1$, если $m \leq n < 2m$. Следовательно, многократное применение формулы (18) даст нам общую формулу

$$H(m, n) = m + \lceil n/2^t \rceil - 1 + tm \quad \text{при } m \leq n, t = \lceil \log_2(n/m) \rceil. \quad (19)$$

Полагая $m = \alpha n$ и $\theta = \log_2(n/m) - t$, найдем

$$H(\alpha n, n) = \alpha n(1 + 2^\theta - \theta - \log_2 \alpha) + O(1) \quad (20)$$

при $n \rightarrow \infty$. Из формул (5.3.1-35) известно, что $1.9139 < 1 + 2^\theta - \theta \leq 2$. Следовательно, (20) можно сравнить с теоретико-информационной нижней оценкой (3). Хуан и Линь доказали (см. упр. 17), что

$$H(m, n) < \left\lceil \log_2 \binom{m+n}{m} \right\rceil + \min(m, n). \quad (21)$$

Алгоритм слияния Хуана—Линя, который можно было бы назвать алгоритмом "бинарного слияния", не всегда оптимален, но обладает тем неопределимым достоинством, что его довольно легко запрограммировать. Он сводится к "децентрированному бинарному поиску" при $m = 1$ и к обычной процедуре слияния при $m \approx n$, так что это золотая середина между двумя указанными методами. Кроме того, во многих случаях он является оптимальным (см. упр. 16).

Формула (18) наводит на мысль о том, что и сама функция M , быть может, удовлетворяет неравенству

$$M(m, n) \leq M(m, \lfloor n/2 \rfloor) + m. \quad (22)$$

Это и в самом деле так (см. упр. 19). Таблицы значений функции $M(m, n)$ позволяют предположить, что, возможно, имеют место и другие соотношения, такие, как

$$M(m+1, n) \geq 1 + M(m, n) \geq M(m, n+1) \quad \text{при } m \leq n; \quad (23)$$

$$M(m+1, n+1) \geq 2 + M(m, n). \quad (24)$$

но в настоящее время не известно никаких доказательств этих неравенств.

Упражнения

1. [15] Найдите одно любопытное соотношение, которое связывает функцию $M(m, n)$ и функцию S , определенную в п. 5.3.1. [Указание: рассмотрите $S(m+n)$.]
- >2. [22] При $m = l$ любой алгоритм слияния, не содержащий избыточных сравнений, определяет расширенное бинарное дерево с $\binom{m+n}{m} = n+1$ внешними узлами. Докажите, что верно и обратное, т. е. каждому расширенному бинарному дереву соответствует некоторый алгоритм слияния с $m = 1$.
3. [M24] Докажите, что $M(1, n) = M(1, n)$ при всех n .
4. [M44] Справедливо ли неравенство $M(m, n) \geq \lceil \log_2 \binom{m+n}{m} \rceil$ при всех m и n ?
5. [M30] Докажите, что $M(m, n) \leq M(m, n+1)$.
6. [M26] Сформулированное выше доказательство теоремы К требует проверки большого числа случаев с привлечением ЭВМ. Каким образом можно резко сократить число таких случаев?
7. [21] Докажите неравенство (11).
- >8. [24] Докажите, что $M(2, 8) \leq 6$. Для этого придумайте такой алгоритм слияния двух элементов с восемью другими, который бы выполнял не более шести сравнений.
9. [27] Докажите, что три элемента можно слить с шестью элементами не более чем за семь шагов.
10. [33] Докажите, что пять элементов можно слить с девятью не более чем за двенадцать шагов. [Указание: опыт введения дьявола подсказывает, что начать нужно со сравнения $A_1 : B_2$, затем, если $A_1 < B_2$, попытаться сравнить $A_5 : B_8$.]
11. [M40] (Ф. Хуан, Ш. Линь.) Пусть $g_{2k} = \lfloor 2^k \cdot \frac{17}{14} \rfloor$, $g_{2k+1} = \lfloor 2^k \cdot \frac{12}{7} \rfloor$ при $k \geq 0$, так что $(g_0, g_1, g_2, \dots) = (1, 1, 2, 3, 4, 6, 9, 13, 19, 27, 38, 54, 77, \dots)$. Докажите, что для слияния двух элементов с g_t элементами требуется в худшем случае более чем t сравнений, однако слить два элемента с $g_t - 1$ можно не более чем за t шагов. [Указание: покажите, что если $n \geq g_{t-1}$, и нужно слить $\{A_1, A_2\}$ с $\{B_1, B_2, \dots, B_n\}$ за t сравнений, то наилучшее первое сравнение—это $A_2 : B_{g_{t-1}}$.]

12. [M21] Пусть $R_n(i, j)$ — наименьшее число сравнений, необходимое для сортировки различных объектов $\{\alpha, \beta, X_1, X_2, \dots, X_n\}$, если заданы соотношения

$$\alpha < \beta, \quad X_1 < X_2 < \dots < X_n, \quad \alpha < X_{i+1}, \quad \beta > X_{n-j}.$$

[Условие $\alpha < X_{i+1}$ или $\beta > X_{n-j}$ теряет смысл, если $i \geq n$ или $j \geq n$. Поэтому $R_n(n, n) = M(2, n)$.] Ясно, что $R_n(0, 0) = 0$. Докажите, что

$$R_n(i, j) = 1 + \min \left(\min_{1 \leq k \leq i} \max(R_n(k-1, j), R_{n-k}(i-k, j)), \min_{1 \leq k \leq j} \max(R_n(i, k-1), R_{n-k}(i, j-k)) \right)$$

при $0 \leq i \leq n, 0 \leq j \leq n, i+j > 0$.

13. [M42] (Р. Л. Грэхем). Покажите, что решение рекуррентного соотношения из упр. 12 можно выразить следующим образом. Определим функцию $G(x)$ при $0 < x < \infty$ такими правилами:

$$G(x) = \begin{cases} 1, & 0 < x \leq \frac{5}{7}; \\ \frac{1}{2} + \frac{1}{8}G(8x-5), & \frac{5}{7} < x \leq \frac{3}{4}; \\ \frac{1}{2}G(2x-1), & \frac{3}{4} < x \leq 1; \\ 0, & 1 \leq x \leq \infty. \end{cases}$$

(См. рис. 38.) Поскольку $R_n(i, j) = R_n(j, i)$ и так как $R_n(0, j) = M(1, j)$, то можно считать, что $1 \leq i \leq j \leq n$. Пусть $p = \lfloor \log_2 i \rfloor, q = \lfloor \log_2 j \rfloor, r = \lfloor \log_2 n \rfloor$ и $t = n - 2^r + 1$. Тогда

$$R_n(i, j) = p + q + S_n(i, j) + T_n(i, j),$$

где функции S_n и T_n принимают значения 0 или 1:

$$S_n(i, j) = 1 \text{ тогда и только тогда, когда } q < r \text{ или } (i - 2^p \geq u \text{ и } j - 2^r \geq u);$$

$$T_n(i, j) = 1 \text{ тогда и только тогда, когда } p < r \text{ или } (t > \frac{6}{7}2^{r-2} \text{ и } i - 2^r \geq v),$$

где $u = 2^p G(t/2^p)$ и $v = 2^{r-2} G(t/2^{r-2})$.

(Это, быть может, самое сложное рекуррентное соотношение из всех, которые когда-либо будут решены!)

14. [46] (Хуан и Линь.) Пусть $h_{3k} = 2^k + 2^{k-1} - 1, h_{3k+1} = g_{2k} + g_{2k-3} + 2^{k-2}, h_{3k+2} = 2g_{2k}$ при $k \geq 2$, за исключением $h_8 = 9$, и начальные значения подобраны так, что $(h_0, h_1, h_2, \dots) = (1, 1, 2, 2, 3, 4, 5, 7, 9, 11, 14, 18, 23, 29, 38, 47, 59, 76, \dots)$. Здесь g_k — функция, которая была определена в упр. 11. Докажите (или опровергните), что $M(3, h_t) > t, M(3, h_t - 1) \leq t$ при всех t .
15. [12] В шаге N1 алгоритма бинарного слияния может потребоваться вычисление значения $\lfloor \log_2(n/m) \rfloor$. Как можно легко вычислить это значение, не применяя операций деления и взятия логарифма?

Picture: Рис. 38. Функция Грэхема (см. упр. 13).

16. [18] При каких m и $n, 1 \leq m \leq n \leq 10$, оптимален алгоритм бинарного слияния Хуана и Линя?
17. [M25] Докажите неравенство (21). [Указание: это неравенство не очень жесткое.]
18. [M40] Исследуйте среднее число сравнений, выполняемых алгоритмом бинарного слияния.
- >19. [23] Докажите, что функция M удовлетворяет неравенству (22).
20. [20] Покажите, что если $M(m, n+1) \leq M(m+1, n)$ при всех $m \leq n$, то $M(m, n+1) \leq 1 + M(m, n)$ при всех $m \leq n$.
21. [M47] Докажите или опровергните соотношения (23), (24).
22. [M50] Исследуйте минимальное среднее число сравнений, необходимых для слияния m элементов с n элементами.
23. [BM30] (Э. Рейнгольд.) Пусть $\{A_1, \dots, A_n\}$ и $\{B_1, \dots, B_n\}$ — множества, содержащие по n элементов каждое. Рассмотрите алгоритм, который пытается проверить наличие равенства между множествами исключительно путем сравнений на равенство элементов этих множеств. Таким образом, алгоритм задает вопросы типа " $A_i = B_j$?" при некоторых i и j и выбирает дальнейший путь вычислений в зависимости от того, был ли ответ положительным или отрицательным. Определив подходящего дьявола, докажите, что любой такой алгоритм в наихудшем для себя случае вынужден выполнить не менее $\frac{1}{2}n(n+1)$ сравнений.

5.3.3. * Выбор с минимальным числом сравнений

При поиске наилучших возможных процедур для выбора t -го элемента в порядке убывания из n элементов мы встречаемся с классом задач, подобных рассмотренным в предыдущем пункте. История этого вопроса восходит к занимательному (хотя и серьезному) очерку преподобного Ч. Л. Доджсона о турнирах по теннису, появившемся в *St. James's Gazette* 1 августа 1883 г. (стр. 5–6). Доджсон, который, разумеется, более известен как Льюис Кэррол, рассматривал несправедливые правила, по которым присуждались (и до сих пор присуждаются) призы в турнирах по теннису. Рассмотрим, например, рис. 39, где показан типичный турнир "с выбыванием" между 32 игроками, помеченными 01, 02, ..., 32. В финале игрок 01 одерживает победу над игроком 05, поэтому ясно, что игрок 01 — чемпион и заслужил первый приз. Неправедливость проявляется в том, что игрок 05 обычно получает второй приз, хотя он может и не быть вторым игроком. Выиграть второй приз можно, даже если играешь хуже половины игроков турнира. В самом деле, как заметил Доджсон, второй игрок выигрывает второй приз в том и только том случае, если первоначально он и чемпион находились в противоположных половинах турнира; для 2^n игроков это происходит с вероятностью $2^{n-1}/(2^n - 1)$, так что почти в половине случаев второй приз получает не тот игрок! Если проигравшие в полуфинале (игроки 25 и 17 на рис. 39) соревнуются за третий приз, то весьма маловероятно, что третий игрок получит третий приз.

Поэтому Доджсон решил найти такой турнир, который правильно определяет второго и третьего игроков в предположении транзитивности. (Иначе говоря, если игрок A побеждает игрока B , а B побеждает C , то можно считать, что игрок A победит C). Он придумал процедуру, в которой проигравшим дают сыграть еще несколько игр, пока не станет определенно известно, что они хуже других трех игроков. Пример схемы Доджсона приводится на рис. 40, изображающем дополнительный турнир, который следует провести вместе с турниром, показанным на рис. 39. Делается попытка организовать встречи игроков, у которых до сих пор были равные результаты, и исключить матчи между игроками, побежденными одним и тем же человеком. Например, игрок 16 проигрывает 11, а игрок 13 проигрывает 12 в первом туре; после того как игрок 16 проигрывает 13 во втором туре, 16 исключается, так как теперь известно, что он хуже, чем 11, 12 и 13. В третьем туре мы не позволяем номеру 19 играть с 21, так как они оба были побеждены

Picture: Рис. 39. Турнир 32 игроков с выбыванием.

Picture: Рис. 40. Теннисный турнир Льюиса Кэррола (в дополнение к турниру рис. 39).

игроком 18 и мы не могли бы автоматически исключить проигравшего во встрече 19 с 21.

Было бы приятно сообщить, что турнир Льюиса Кэррола оказался оптимальным, но, к сожалению, это не так. Из записи в его дневнике от 23 июля 1883 г. явствует, что он составил этот очерк примерно за шесть часов, и чувствовал, что, "поскольку теннисный сезон приближается к концу, очерк следует написать побыстрее, не слишком увлекаясь качеством". В его процедуре делается больше сравнений, чем необходимо, и она не сформулирована достаточно четко, чтобы квалифицировать ее как алгоритм. С другой стороны, в ней имеются некоторые очень интересные аспекты, если судить с точки зрения параллельных вычислений. Она также представляется отличным расписанием теннисного турнира, поскольку Кэррол включил в нее несколько драматических эффектов; например, он определил, что два финалиста должны пропустить пятый тур и сыграть "длинный" матч в турах 6 и 7. Однако организаторы турниров, по-видимому, сочли это предложение излишне логичным, и потому система Кэррола, скорее всего, никогда не испытывалась. Вместо этого практикуется метод "рассеивания" более сильных игроков, чтобы они попали в разные части дерева.

На математическом семинаре в 1929–1930 г. Гуго Штейнгауз поставил задачу нахождения минимального числа теннисных матчей, требуемых для определения первого и второго игроков в турнире, если имеется $n > 2$ игроков. Ю. Шрейер [*Mathesis Polska*, 7 (1932), 154–160] привел процедуру, требующую самое большее $n - 2 + \lceil \log_2 n \rceil$ матчей, используя, по существу, тот же метод, что и первые две стадии процесса, который мы назвали сортировкой посредством выбора из дерева (см. п. 5.2.3, рис. 23), однако не выполняя дополнительных сравнений, содержащих $-\infty$. Шрейер также утверждал, что $n - 2 + \lceil \log_2 n \rceil$ — наилучшее возможное значение; но его доказательство было ошибочным, как и еще одна попытка доказательства, предпринятая Е. Слупецки [*Colloquium Mathematician*, 2 (1951), 286–290]. Прошло 32 года, прежде чем С. С. Кислицыным было опубликовано правильное, хотя и очень сложное доказательство [*Сибирский математический журнал*, 5 (1964), 557–564]. Пусть $V_t(n)$ для $1 \leq t \leq n$ обозначает минимальное число сравнений, требуемых для определения t -го в порядке убывания элемента из n элементов, и пусть $W_t(n)$ равно наименьшему числу сравнений, необходимых для определения наибольшего, второго, ..., t -го элементов всех сразу. Из соображений симметрии имеем

$$V_t(n) = V_{n+1-t}(n); \quad (1)$$

очевидно также, что

$$V_1(n) = W_1(n), \quad (2)$$

$$V_t(n) \leq W_t(n), \quad (3)$$

$$W_n(n) = W_{n-1}(n) = S(n). \quad (4)$$

В п. 5.2.3 мы видели, что

$$V_1(n) = n - 1. \quad (5)$$

Есть удивительно простое доказательство этого факта, поскольку каждый участник турнира, кроме чемпиона, должен проиграть по крайней мере одну игру! Обобщая эту идею и используя "дьявола", мы можем без особого труда доказать теорему Шрейера—Кислицына.

Теорема S. При $n \geq 2$ справедливо равенство $V_2(n) = W_2(n) = n - 2 + \lceil \log_2 n \rceil$.

Доказательство. Предположим, что в турнире, где с помощью некоторой данной процедуры должен определиться второй игрок, участвуют n игроков, и пусть a_j —число игроков, проигравших j или больше матчей. Общее число сыгранных матчей будет тогда равно $a_1 + a_2 + a_3 + \dots$. Мы не можем определить второго игрока, не выявив заодно и чемпиона (см. упр. 2), поэтому из предыдущих рассуждений $a_1 = n - 1$. Для завершения доказательства покажем, что всегда существует последовательность результатов матчей, которая приводит к $a_2 \geq \lceil \log_2 n \rceil - 1$.

Предположим, что к концу турнира чемпион сыграл p игр и победил p игроков; одним из них был второй игрок, а остальные должны проиграть по крайней мере еще по одному разу, поэтому $a_2 \geq p - 1$. Итак, мы можем закончить доказательство, построив дьявола, предопределяющего результаты игр таким образом, чтобы чемпиону пришлось сыграть по крайней мере еще с $\lceil \log_2 n \rceil$ другими участниками турнира.

Пусть дьявол считает, что игрок A лучше, чем B , если A ранее не проигрывал, а B хотя бы однажды проиграл, или если оба не проигрывали, но B выиграл к этому моменту меньше матчей, чем A . При других обстоятельствах дьявол может принимать произвольное решение, не противоречащее некоторому частичному упорядочению.

Рассмотрим результаты завершеного турнира, матчи которого предопределялись таким дьяволом. Мы скажем, что " A превосходит B " тогда и только тогда, когда $A = B$ или A превосходит игрока, который первым победил B . (Только первое поражение игрока существенно в этом отношении, последующие его игры игнорируются. В соответствии с устройством дьявола любой игрок, *первым* победивший какого-то, ни в одной из предыдущих встреч не должен иметь поражений. Отсюда следует, что игрок, который выиграл свои первые p матчей, превосходит на основании этих p игр не более 2^p игроков. (Если $p = 0$, это очевидно, если же $p > 0$, то p -й матч был сыгран против игрока, который либо ранее потерпел поражение, либо превосходит не более 2^{p-1} игроков.) Чемпион превосходит всех, поэтому он должен был сыграть не менее $\lceil \log_2 n \rceil$ матчей. ■

Таким образом, задача нахождения второго в порядке убывания элемента полностью решена в минимаксном смысле. В упр. 6 показано, что можно дать простую формулу для минимального числа сравнений, необходимых для выявления второго элемента множества, если известно произвольное частичное упорядочение элементов.

А если $t > 2$? В упомянутой статье Кислицын пошел дальше. Он рассмотрел большие значения t , доказав, что

$$W_t(n) \leq n - t + \sum_{n+1-t < j \leq n} \lceil \log_2 j \rceil \quad \text{при } n \geq t. \quad (6)$$

Мы видели, что при $t = 1$ и $t = 2$ эта формула представляет собой равенство; при $t = 3$ она может быть слегка улучшена (см. упр. 21).

Мы докажем теорему Кислицына, показав, что первые t стадий *выбора из дерева* требуют не более $n - t + \sum_{n+1-t < j \leq n} \lceil \log_2 j \rceil$ сравнений (исключая все сравнения, содержащие $-\infty$). Интересно, что правая часть (6) равна $B(n)$, когда $t = n - 1$ и $t = n$ [см. формулу (5.3.1-3)]; следовательно, выбор из дерева и бинарные вставки приводят к одной и той же верхней оценке для задачи сортировки, хотя это совершенно различные методы.

Пусть α —расширенное бинарное дерево с n внешними узлами и π —перестановка множества $\{1, 2, \dots, n\}$. Поместим элементы перестановки π во внешние узлы слева направо в симметричном порядке и заполним внутренние узлы в соответствии с правилами турнира с выбыванием как при выборе из дерева. Повторное применение операции выбора к результирующему дереву определяет

последовательность $c_{n-1}c_{n-2} \dots c_1$, где c_j есть число сравнений, требуемых, чтобы перенести элемент j в корень дерева, после того как элемент $j + 1$ был заменен на $-\infty$. Например, если α —дерево

Picture: p.256

а $\pi = 5\ 3\ 1\ 4\ 2$, то мы получаем последовательные деревья

Picture: p. 257.1

Если же $\pi = 3\ 1\ 5\ 4\ 2$, то последовательность $c_4c_3c_2c_1$ будет иной, именно $2\ 1\ 1\ 0$. Легко видеть, что c_1 всегда есть 0.

Пусть $\mu(\alpha, \pi)$ —мультимножество $\{c_{n-1}, c_{n-2}, \dots, c_1\}$, определяемое α и π . Если

Picture: p.257.2

и если элементы 1 и 2 не содержатся оба либо в α' , либо в α'' , то легко видеть, что

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + 1) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{0\} \quad (8)$$

для подходящих перестановок π' и π'' , где $(\mu + 1)$ обозначает мультимножество, получаемое прибавлением 1 к каждому элементу μ . С другой стороны, если и элемент 1, и элемент 2 находятся в α' , имеем

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + \varepsilon) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{0\},$$

где $(\mu + \varepsilon)$ обозначает какое-нибудь мультимножество, получаемое прибавлением 1 к некоторым элементам μ и 0 к остальным. Аналогичная формула справедлива, если элементы 1 и 2 находятся в α'' . Будем говорить, что мультимножество μ_1 мажорирует μ_2 , если μ_1 и μ_2 содержат равное число элементов и k -й в порядке убывания элемент μ_1 больше или равен k -го в порядке убывания элемента μ_2 для всех k . Определим $\mu(\alpha)$ как мажоранту $\mu(\alpha, \pi)$ по всем перестановкам π в том смысле, что $\mu(\alpha)$ мажорирует $\mu(\alpha, \pi)$ при всех π и $\mu(\alpha) = \mu(\alpha, \pi)$ при некотором π . Приведенные выше формулы показывают, что

Picture: p.257.3

следовательно, $\mu(\alpha)$ есть мультимножество всех расстояний от корня до внутренних узлов α .

Если читатель уследил за ходом наших рассуждений, ему должно быть ясно, что теперь мы готовы доказать теорему Кислицына (6), поскольку $W_t(n)$ меньше или равно $n - 1$ плюс $t - 1$ наибольших элементов $\mu(\alpha)$, где α —любое дерево, используемое при сортировке посредством выбора из дерева. Можно выбрать в качестве α полное бинарное дерево с n внешними узлами (см. п. 2.3.4.5); в этом случае

$$\begin{aligned} \mu(\alpha) &= \{ \lfloor \log_2 1 \rfloor, \lfloor \log_2 2 \rfloor, \dots, \lfloor \log_2(n-1) \rfloor \} = \\ &= \{ \lceil \log_2 2 \rceil - 1, \lceil \log_2 3 \rceil - 1, \dots, \lceil \log_2 n \rceil - 1 \}. \end{aligned} \quad (10)$$

Мы получим формулу (6), если рассмотрим $t - 1$ наибольших элементов этого мультимножества.

Теорема Кислицына дает хорошую верхнюю оценку для $W_t(n)$; Кислицын отметил, что $V_3(5) = 6 < W_3(5) = 7$, но не смог найти в общем случае лучшую оценку для $V_t(n)$. Это было сделано А. Адьяном и М. Собедем, которые использовали *выбор с замещением* вместо выбора из дерева (см. п. 5.4.1). Выведенная ими формула [Univ. of Minnesota, Dept. of statistics, report 121 (May, 1969)]

$$V_t(n) \leq n - t + (t - 1)\lceil \log_2(n + 2 - t) \rceil, \quad n \geq t, \quad (11)$$

отличается от (6) тем, что каждый элемент суммы в (6) заменен на наименьший элемент.

Теорему Адьяна и Собеда можно доказать, воспользовавшись следующим построением. Сначала образуем бинарное дерево для турнира с выбыванием $n - t + 2$ элементов. (Это требует $n - t + 1$ сравнений.) Наибольший элемент превосходит $n - t + 1$ других элементов, поэтому он не может быть t -м в порядке убывания. Заменяем его во внешнем узле дерева на один из $t - 2$ элементов, оставшихся в резерве, и найдем наибольший из получившихся $n - t + 2$ элементов; это требует не более $\lceil \log_2(n + 2 - t) \rceil$ сравнений, поскольку придется заново вычислить только один путь в дереве. Повторим эту операцию всего $t - 2$ раз, по одному разу для каждого элемента из резерва. Наконец, заменим текущий наибольший элемент на $-\infty$ и определим наибольший из оставшихся $n + 1 - t$ элементов; для этого потребуется не

более $\lceil \log_2(n + 2 - t) \rceil - 1$ сравнений, и t -й в порядке убывания элемент исходного множества попадет в корень дерева. Суммирование сравнений дает (11).

Разумеется, мы должны заменить t на $n + 1 - t$ в правой части соотношения (11), если $n + 1 - t$ дает лучшее значение (как при $n = 6, t = 3$). Как ни странно, но эта формула дает для $V_7(13)$ меньшую оценку, чем для $V_6(13)$. Верхняя оценка в (11) точна для $n \leq 6$, но когда n и t становятся большими, можно получить значительно лучшие оценки для $V_t(n)$.

Например, можно использовать следующий изящный метод (принадлежащий Дэвиду Дорену), чтобы показать, что $V_4(8) \leq 12$. Обозначим элементы через X_1, \dots, X_8 ; сначала сравним $X_1 : X_2, X_3 : X_4$ и двух победителей, проделаем то же для $X_5 : X_6, X_7 : X_8$, и их победителей. Переименуем элементы так, чтобы получить $X_1 < X_2 < X_4 > X_3, X_5 < X_6 < X_8 > X_7$, затем сравним $X_2 : X_6$; в силу симметрии положим $X < 2 < X_6$, поэтому имеем конфигурацию

Picture: p.259.1

(Теперь X_1 и X_8 вышли из игры, и мы должны найти третий в порядке убывания элемент из $\{X_2, \dots, X_7\}$.) Сравним $X_2 : X_7$, и отбросим меньший; в худшем случае получим $X_2 < X_7$, и нам надо найти третий в порядке убывания элемент из

Picture: p.259.2

Это можно сделать еще за $V_3(5) - 2 = 4$ шага, так как процедура для $t = 3$ и $n = 5$ в (11) начинается со сравнения двух непересекающихся пар элементов.

Можно использовать другие трюки подобного вида, чтобы получить результаты, показанные в табл. 1; до сих пор не видно никакого общего метода, и значения в таблице для $n \geq 8$ могут претерпеть изменения. Заметим, что $V_3(10) \leq 14$, поэтому (11) не всегда дает равенство для $t = 3$. Тот факт, что $V_4(7) = 10$, показывает, что соотношение (11) приводит к ошибке на 2 уже при $n = 7$.

Неплохую нижнюю оценку в задаче выбора получил Дэвид Киркпатрик (Ph. D. thesis, Univ. of Toronto, 1974). Построенный им дьявол доказывает, что

$$V_t(n) > n + t - 3 + \sum_{0 \leq j \leq t-2} \lceil \log_2((n + 2 - t)/(t + j)) \rceil, \quad n \geq 2t - 1. \tag{12}$$

Киркпатрик точно установил поведение функции $V_t(n)$ при $t = 3$, доказав, что

$$V_3(n) = n + \lceil \log_2((n - 1)/2.5) \rceil + \lceil \log_2((n - 1)/4) \rceil$$

при всех $n \geq 50$ (ср. с упр. 22).

Линейный метод. Если n нечетно и $t = \lceil n/2 \rceil$, то t -й в порядке убывания (и t -й в порядке возрастания) элемент называется медианой. В соответствии с (11) мы можем найти медиану n элементов за $\approx \frac{1}{2}n \log_2 n$ сравнений, но это лишь приблизительно вдвое быстрее сортировки, хотя нам нужна значительно меньшая информация. В течение нескольких лет объединенные усилия ряда исследователей были направлены на улучшение формулы (11) для больших значений t и n ; наконец, в 1971 г. Мануэль Блум открыл метод, требующий только $O(n \log \log n)$ шагов. Подход Блума к этой задаче дал толчок к развитию нового класса методов, который привел к следующему построению, принадлежащему Р. Райвесту и Р. Тарьяну.

Таблица 1

Наилучшие из известных верхних оценок для $V_t(n)$										
n	$V_1(n)$	$V_2(n)$	$V_3(n)$	$V_4(n)$	$V_5(n)$	$V_6(n)$	$V_7(n)$	$V_8(n)$	$V_9(n)$	$V_{10}(n)$
1	0									
2	1	1								
3	2	3	2							
4	3	4	4	3						
5	4	6	6	6	4					
6	5	7	8	8	7	5				
7	6	8	10	10*	10	8	6			
8	7	9	11	12	12	11	9	7		
9	8	11	12	14	15*	14	12	11	8	
10	9	12	14*	15	17	17	15	14*	12	9

*) В этих случаях упр. 10–12 дают построения, позволяющие улучшить (11).

Теорема Л. Если $n > 32$, то $V_t(n) \leq 15n - 163$ при $1 \leq t \leq n$.

Доказательство. Когда n мало, теорема тривиальна, так как $V_t(n) \leq S(n) \leq 10n \leq 15n - 163$ для $32 < n \leq 2^{10}$. Добавив самое большее 13 фиктивных элементов " $-\infty$ ", можно считать, что $n = 7(2q + 1)$ при некотором целом $q \geq 73$. Теперь для выбора t -го в порядке убывания элемента воспользуемся следующим методом.

Шаг 1. Разобьем элементы на $2q + 1$ групп по 7 элементов в каждой и отсортируем каждую группу. Это потребует не более $13(2q + 1)$ сравнений.

Шаг 2. Найдем медиану из $2q + 1$ медиан, полученных на шаге 1, и обозначим ее x . Проведя индукцию по q , замечаем, что это требует не более $V_{q+1}(2q + 1) \leq 30q - 148$ сравнений.

Шаг 3. Теперь $n - 1$ элементов, отличных от x , разбиваются на три множества (рис. 41):

- $4q + 3$ элементов, о которых известно, что они больше x (область В);
- $4q + 3$ элементов, о которых известно, что они меньше x (область С);
- $6q$ элементов, отношение которых к x неизвестно (области А, D).

Выполнив дополнительно $4q$ сравнений, мы можем в точности сказать, какие элементы из областей А и D меньше x . (Сначала мы сравниваем x со средним элементом каждой тройки.)

Picture: Рис. 41. Алгоритм выбора Райвеста и Тарьяна ($q = 4$).

Шаг 4. Теперь при некотором r мы нашли r элементов, больших x , и $n - 1 - r$ элементов, меньших x . Если $t = r + 1$, то x и будет ответом; если $t < r + 1$, то нам нужно найти t -й элемент в порядке убывания из r больших элементов; и если $t > r + 1$, то нам нужно найти $(t - 1 - r)$ -й элемент в порядке убывания из $n - 1 - r$ меньших элементов. Суть дела в том, что r и $n - 1 - r$ оба меньше или равны $10q + 3$ (размер областей А и D плюс В или С). Индукцией по q выводим, что этот шаг требует не более $15(10q + 3) - 163$ сравнений.

Общее число сравнений оказывается не больше

$$13(2q + 1) + 30q - 148 + 4q + 15(10q + 3) - 163 = 15(14q - 6) - 163.$$

Так как мы начали с не менее $14q - 6$ элементов, доказательство завершено. ■

Метод, использованный в этом доказательстве, не вполне совершенный, поскольку на шаге 4 теряется значительная информация. Тщательные улучшения, проделанные В. Праттом, Р. Райвестом и Р. Тарьяном, показывают, что константу 15 можно уменьшить до 5.43.

Среднее число. Вместо минимизации *максимального* числа сравнений можно искать алгоритм, который минимизирует *среднее* число сравнений, предполагая, что порядок случаен. Как обычно, эта задача значительно труднее, и она все еще не решена даже в случае $t = 2$. Клод Пикар упомянул эту задачу в своей

Picture: Рис. 42. Процедура, которая выбирает второй элемент из $\{X_1, X_2, X_3, X_4, X_5, X_6\}$, используя в среднем $6\frac{1}{2}$ сравнений. Каждая "симметричная" ветвь идентична своему брату, однако имена переставлены соответствующим образом. Во внешних узлах записано $j k$, если известно, что X_j — второй, а X_k — наибольший элемент; число перестановок, приводящих к этому узлу, записано непосредственно под ним.

книге *Théorie des Questionnaires* (1965), широкое исследование было предпринято Милтоном Собелем [Univ. of Minnesota, Dept. of Statistics, report 113 (November, 1968)].

Собель построил процедуру, изображенную на рис. 42, которая находит второй в порядке убывания элемент из шести элементов, в среднем используя только $6\frac{1}{2}$ сравнений. В худшем случае требуется 8 сравнений, и это хуже, чем $V_2(6) = 7$; но все известные процедуры для этой задачи, требующие не более 7 сравнений, используют в среднем по крайней мере $6\frac{2}{3}$ сравнений. Таким образом, вероятно, никакая процедура нахождения второго из шести элементов не будет оптимальной одновременно и как минимаксная, и как минимизирующая среднее число сравнений.

Пусть $\bar{V}(n)$ обозначает минимальное среднее число сравнений, необходимых для определения t -го элемента в порядке убывания из n элементов. В следующей таблице показаны наилучшие известные верхние оценки для $\bar{V}_2(n)$, вычисленные Собелем:

$$\bar{V}_2(n) \leq \begin{matrix} n = 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2\frac{2}{2} & 4 & 5\frac{4}{4} & 6\frac{1}{1} & 7\frac{17}{17} & 9 & 10\frac{1}{1} & 11\frac{4}{4} \end{matrix} \quad (13)$$

Собель предположил, что

$$\bar{V}_2(n) \geq n - 2 + \lfloor 2 \log_2 n \rfloor / 2. \quad (14)$$

Р. У. Флойд в 1970 г. обнаружил, что медиана n элементов в среднем может быть найдена всего за $\frac{3}{2}n + O(n^{\frac{2}{3}} \log n)$ сравнений. (См. упр. 13.) Фактически он доказал, что

$$\bar{V}_t(n) \leq n + t + f(n), \quad \text{где } \lim_{n \rightarrow \infty} f(n)/n = 0. \quad (15)$$

Предполагается, что этот результат является наилучшей асимптотической формулой, однако никакой удовлетворительной нижней оценки все еще не найдено.

Упражнения

1. [15] Почему в турнире Льюиса Кэррола (рис. 39 и 40) игрок 13 выбывает, несмотря на то что он выиграл свой матч в третьем туре?
- >2. [M25] Докажите, что после того, как мы нашли с помощью последовательности сравнений t -й элемент в порядке убывания из n элементов, мы также знаем, какие $t - 1$ элементов больше него и какие $n - t$ элементов меньше.
3. [M21] Докажите, что $V_t(n) \geq V_t(n - 1) + 1$ при $1 \leq t \leq n$.
4. [M20] Докажите, что $W_t(n) \geq \lceil \log_2 n^t \rceil$, где $n^t = n(n - 1) \dots (n + 1 - t)$.
5. [10] Докажите, что $W_3(n) \leq V_3(n) + 1$.
- >6. [M26] (Р. У. Флойд.) Дано n различных элементов $\{X_1, \dots, X_n\}$ и отношения $X_i < X_j$ для некоторых пар (i, j) . Мы хотим найти второй в порядке убывания элемент. Если известно, что $X_i < X_j$ и $X_i < X_k$ при $j \neq k$, то X_i не может быть вторым элементом, поэтому его можно исключить. В результате отношения будут иметь вид

Picture: p.264

а именно образуется m групп элементов, которые можно представить вектором (l_1, l_2, \dots, l_m) ; j -я группа содержит $l_j + 1$ элементов, про один из которых известно, что он больше остальных. Например, изображенная конфигурация может быть описана вектором $(3, 2, 4, 0, 1)$; если ни одного отношения не известно, то имеем вектор из n нулей.

Пусть $f(l_1, l_2, \dots, l_m)$ — минимальное число сравнений, нужных для определения второго элемента такого частично упорядоченного множества. Докажите, что

$$f(l_1, l_2, \dots, l_m) = m - 2 + \lceil \log_2(2^{l_1} + 2^{l_2} + \dots + 2^{l_m}) \rceil.$$

[Указание. Покажите, что наилучшая стратегий всегда состоит в том, чтобы сравнивать наибольшие элементы двух самых маленьких групп, пока не сведем от к единице; используйте индукцию по $l_1 + l_2 + \dots + l_m + 2m$.]

7. [M20] Докажите (8).
8. [M21] Формула Кислицына (6) основана на сортировке посредством выбора из дерева, использующей полное бинарное дерево с n внешними узлами. Может ли выбор из дерева, основанный на некотором другом дереве, дать лучшую оценку для каких-нибудь t и n ?
- >9. [20] Нарисуйте дерево сравнений для нахождения медианы пяти элементов не более чем за шесть шагов, используя метод выбора с замещением Адьана и Собеля [см. (11)].
10. [35] Покажите, что медиана семи элементов может быть найдена не более чем за 10 шагов.
11. [28] (Адьан и Собель.) Расширив метод Дорена, сформулированный в тексте, покажите, что медиана девяти элементов может быть найдена не более чем за 15 шагов.
12. [21] (Адьан и Собель.) Докажите, что $V_3(n) \leq V_3(n - 1) + 2$. [Указание: начните с удаления наименьшего из $\{X_1, X_2, X_3, X_4\}$.]
- >13. [BM28] (Р. У. Флойд.) Покажите, что если начать с нахождения медианы $\{X_1, \dots, X_{n^{2/3}}\}$, используя рекурсивно определенный метод, то можно найти медиану $\{X_1, \dots, X_n\}$, выполнив в среднем $\frac{3}{2}n + O(n^{2/3} \log n)$ сравнений.
- >14. [20] (М. Собель.) Покажите, что, используя не более пяти сравнений, можно найти два наибольших элемента из $\{X_1, X_2, X_3, X_4, X_5\}$, если не важен их взаимный порядок.
15. [22] (И. Пол.) Предположим, что нас интересует минимизация пространства, а не времени. Какое минимальное число слов памяти требуется для вычисления t -го из n элементов, если каждый элемент занимает одно слово и элементы вводятся в особый регистр по одному?
- >16. [25] (И. Пол.) Покажите, что мы можем найти одновременно максимум и минимум множества из n элементов, используя не более $\lceil \frac{3}{2}n \rceil - 2$ сравнений, и это число не может быть уменьшено.

[Указание. Любая стадия такого алгоритма может быть представлена четверкой (a, b, c, d) , где a элементов вообще не сравнивались, b элементов выигрывали, но никогда не проигрывали, c проигрывали, но никогда не выигрывали, d как выигрывали, так и проигрывали. Постройте подходящего дьявола.]

17. [20] (Р. У. Флойд.) Покажите, что можно выбрать k наибольших и l наименьших элементов множества из n элементов, используя не более $\lceil \frac{3}{2}n \rceil - k - l + \sum_{n+1-k < j \leq n} \lceil \log_2 j \rceil + \sum_{n+1-l < j \leq n} \lceil \log_2 j \rceil$ сравнений.
18. [M20] Если бы в доказательстве теоремы L были использованы группы размера 5, а не 7, то какая бы получилась теорема?
19. [M44] Найдите точное значение $\bar{V}_2(6)$. Может ли оно достигаться в процедуре, никогда не выполняющей более семи сравнений?
20. [M47] Докажите (или опровергните) предположение Собея (13).
21. [25] (С. Лин.) Докажите, что $W_3(2^k + 2) \leq 2^k + 2k$, если $k \geq 3$.
22. [24] (Дэвид Г. Киркпатрик.) Покажите, что в случае $4 \cdot 2^k < n - 1 < 5 \cdot 2^k$ верхняя оценка (11) для $V_3(n)$ может быть следующим образом уменьшена на 1: (i) Образуйте четыре "дерева с выбыванием" размера 2^k . (ii) Найдите минимальный из четырех максимумов и удалите все 2^k элементов соответствующего дерева. (iii) Используя накопленную информацию, постройте одно дерево с выбыванием размера $n - 1 - 2^k$. (iv) Продолжайте, как при доказательстве (11).
23. [M49] Каково асимптотическое значение $V_{\lceil n/2 \rceil}(n)$ при $n \rightarrow \infty$?
24. [M48] Каково асимптотическое значение $\bar{V}_{\lceil n/2 \rceil}(n)$ при $n \rightarrow \infty$?

5.3.4. Сети сортировки

В настоящем пункте мы будем изучать класс методов сортировки, удовлетворяющих некоторому ограничению. Интерес к таким методам объясняется в основном приложениями и солидной теоретической основой. Это новое ограничение требует, чтобы *последовательность сравнений не зависела от предыстории*¹⁰ в том смысле, что если мы сравниваем K_i и K_j , то последующие сравнения для случая $K_i < K_j$ в точности те же, что и для случая $K_i > K_j$, однако i и j меняются ролями. На рис. 43(a) изображено дерево сравнений, в котором это условие выполнено. (Заметим, что на каждом уровне производится одинаковое число сравнений, поэтому после m сравнений имеется 2^m результатов; так как $n!$ не является степенью 2, то некоторые сравнения будут излишними в том смысле, что одно из их поддеревьев никогда не встречается на практике. Иными словами, на некоторых ветвях дерева приходится выполнять больше сравнений, чем необходимо, чтобы сортировка была правильной на всех соответствующих ветвях.)

Так как каждый путь такого дерева сверху донизу определяет все дерево, то подобную схему сортировки проще изображать в виде *сети*, как на рис. 43(b). Прямоугольники в такой сети представляют "компараторные модули", имеющие два входа (изображенные линиями, входящими в модуль сверху)

Picture: Рис. 43. Дерево сравнений, в котором не учитывается предыстория, (a) и соответствующая сеть (b).

и два выхода (изображенные линиями, выходящими вниз); левый выход есть меньший из двух входов, а правый выход — больший из них. Элемент K'_1 в нижней части сети есть наименьший из $\{K_1, K_2, K_3, K_4\}$, K'_2 — второй в порядке возрастания и т. д. Нетрудно доказать, что любая сеть сортировки соответствует дереву сравнений, обладающему свойством независимости от предыстории (в указанном выше смысле), и что любое такое дерево соответствует сети компараторных модулей.

Между прочим заметим, что с инженерной точки зрения компараторный модуль довольно легко изготовить. Предположим, например, что по линиям связи в модуль поступают двоичные числа по одному биту в единицу времени, начиная со старшего. Каждый компараторный модуль имеет три состояния и функционирует следующим образом:

Момент t			Момент $(t + 1)$		
Состояние	Входы		Состояние	Выходы	
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	2	0	1
0	1	1	0	1	1
1	x	y	1	x	y
2	x	y	2	y	x

¹⁰ В первой редакции книги автор называл такую последовательность однородной. — Прим. ред.

Первоначально все модули находятся в состоянии 0 и выдают 00. Модуль переходит в состояние 1 или 2, как только его входы станут различными. Числа, которые в момент времени t начали поступать сверху в сеть, соответствующую рис. 43(b), начнут в момент $t + 3$ выводиться снизу в отсортированном

Picture: Рис. 44. Еще один способ представления сортировки последовательности $\langle 4, 1, 3, 2 \rangle$ посредством сети, изображенной на рис. 43.

порядке, если включить соответствующий задерживающий элемент в линиях K'_1 и K'_4 .

Для разработки теории сетей сортировки удобно изображать их несколько иным способом (рис. 44). На этом рисунке числа поступают слева, а компараторные модули изображены вертикальными соединениями между двумя прямыми; каждый компаратор вызывает, если необходимо, перестановку своих входов таким образом, что после прохождения компаратора большее число оказывается на нижней линии. В правой части диаграммы все числа упорядочены сверху вниз.

Ранее, изучая оптимальную сортировку, мы уделяли основное внимание минимизации числа сравнений, почти (или совсем)

Picture: Рис. 45. Получение $(n + 1)$ -элементного сортировщика из n -элементного: (a)—вставка; (b)—выбор.

не учитывая перемещение данных или сложность структуры решений метода сортировки. В этом отношении сети сортировки имеют некоторое преимущество, так как данные могут храниться в n ячейках, а структура решений "прямолинейна"; нет необходимости запоминать результаты предыдущих сравнений—план неизменен и фиксирован заранее. Еще одним важным

Picture: Рис. 46. Сетевые аналоги элементарных схем внутренней сортировки, полученные многократным применением операции, представленной на рис. 45: (a)—простая вставка; (b)—метод пузырька.

преимуществом сетей сортировки является то, что часть операций можно совместить, если выполнять их одновременно (на подходящей машине). Например, пять шагов на рис. 43 и 44 сокращаются до трех, если допустить одновременные неперекрывающиеся сравнения, так как можно объединить первые два и следующие два шага; позднее в данном пункте мы используем это свойство сетей сортировки. Таким образом, сети сортировки могут быть очень полезны, хотя возможность построения эффективной сети сортировки n элементов при больших n вовсе не очевидна; возможно, мы обнаружим, что для поддержания однородной структуры решений требуется много дополнительных сравнений.

Имеется два простых способа построения сети сортировки для $n + 1$ элементов, если дана сеть для n элементов: с использованием либо принципа *вставки*, либо принципа *выбора*. На

Picture: Рис. 47. При параллельном выполнении операций простая вставка совпадает с методом пузырька!

рис. 45(a) показано, как $(n + 1)$ -й элемент может быть вставлен на нужное место после того, как первые n элементов отсортированы, а на рис. 45(b) показано, как можно выбрать наибольший элемент, прежде чем перейти к сортировке остальных. Многократное применение рис. 45(a) дает сетевой аналог простых вставок (алгоритм 5.2.1S), а многократное применение рис. 45(b) приводит к сетевому аналогу метода пузырька (алгоритм 5.2.2B). На рис. 46 изображены соответствующие сети для шести элементов. Интересно заметить, что если сжать каждую сеть, чтобы обеспечить одновременные операции, то оба метода сведутся к одной и той же "треугольной" процедуре с $(2n - 3)$ стадиями (рис. 47).

Легко доказать, что сети, представленные на рис. 43 и 44, будут сортировать любое множество из четырех чисел, поскольку первые четыре компаратора направляют наименьший и наибольший элементы на положенные им места, а последний компаратор располагает в требуемом порядке остальные два элемента. Однако не всегда так легко сказать, будет ли данная сеть сортировать все возможные входные последовательности; например, сети

Picture: p.269

являются правильными четырехэлементными сетями сортировки, но доказательство их правильности нетривиально. Было бы достаточно проверить каждую n -элементную сеть на всех $n!$ перестановках n различных чисел, но фактически мы можем обойтись значительно меньшим количеством проверок.

Теорема Z. (Принцип нулей и единиц.) Если сеть с n входами сортирует в неубывающем порядке все 2^n последовательностей из 0 и 1, то она будет сортировать в неубывающем порядке любую последовательность n чисел.

Доказательство. (Это частный случай теоремы Бурисиуса, упр. 5.3.1-12.) Если $f(x)$ —любая монотонная функция, для которой $f(x) \leq f(y)$ при $x \leq y$, и если данная сеть преобразует $\langle x_1, \dots, x_n \rangle$ в $\langle y_1, \dots, y_n \rangle$, то, как нетрудно видеть, эта сеть преобразует $\langle f(x_1), \dots, f(x_n) \rangle$ в $\langle f(y_1), \dots, f(y_n) \rangle$. Если $y_i > y_{i+1}$ при некотором i , то рассмотрим монотонную функцию f , которая для всех чисел $< y_i$ принимает значение 0, а для всех чисел $\leq y_1$ —значение 1. Эта функция определяет последовательность нулей и единиц $\langle f(x_1), \dots, f(x_n) \rangle$, которая не сортируется данной сетью. Следовательно, если все последовательности 0 и 1 поддаются сортировке, то будем иметь $y_i \leq y_{i+1}$ для всех $1 \leq i < n$. ■

Принцип нулей и единиц довольно полезен для построения сетей сортировки. В качестве нетривиального примера получим обобщенный вариант "обменной сортировки со слиянием" Бэтчера (алгоритм 5.2.2М). Идея состоит в том, чтобы сортировать $m + n$ элементов, сортируя первые m и последние n элементов независимо и затем применяя к результату (m, n) -сеть слияния. Построить (m, n) -сеть слияния можно по индукции:

- Если $m = 0$ или $n = 0$, то сеть пустая. Если $m = n = 1$, то сеть состоит из единственного компараторного модуля.
- Если $mn > 1$, то обозначим сливаемые последовательности $\langle x_1, \dots, x_m \rangle$ и $\langle y_1, \dots, y_n \rangle$. Сольем "нечетные последовательности" $\langle x_1, x_3, \dots, x_{2\lceil m/2 \rceil - 1} \rangle$ и $\langle y_1, y_3, \dots, y_{2\lceil n/2 \rceil - 1} \rangle$ и получим отсортированный результат $\langle v_1, v_2, \dots, v_{\lceil m/2 \rceil + \lceil n/2 \rceil} \rangle$; сольем "четные последовательности" $\langle x_2, x_4, \dots, x_{2\lfloor m/2 \rfloor} \rangle$ и $\langle y_2, y_4, \dots, y_{2\lfloor n/2 \rfloor} \rangle$ и получим отсортированный результат $\langle w_1, w_2, \dots, w_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor} \rangle$. И наконец, применим операции сравнения-обмена

$$w_1 : v_2, w_2 : v_3, w_3 : v_4, w_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor} : v^* \quad (1)$$

к последовательности

$$\langle v_1, w_1, v_2, w_2, v_3, w_3, \dots, v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor}, w_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor}, v^*, v^{**} \rangle; \quad (2)$$

результат будет отсортирован. (!) (Здесь $v^* = v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor + 1}$ не существует, если m и n оба четные, и $v^{**} = v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor + 2}$ существует, лишь если m и n оба нечетные; общее число компараторных модулей, указанных в (1), равно $\lfloor (m + n) / 2 \rfloor$.)

Назовем (m, n) -сеть слияния Бэтчера *четно-нечетным слиянием*. Построенное в соответствии с этими принципами (4, 7)-слияние показано на рис. 48.

Picture: Рис. 48. Четно-нечетное слияние для $m = 4$ и $n = 7$.

Чтобы доказать, что эта очень странная процедура действительно работает при $mn > 1$, воспользуемся принципом нулей и единиц и проверим ее на всех последовательностях 0 и 1. После начальных m -сортировки и n -сортировки последовательность $\langle x_1, \dots, x_m \rangle$ будет состоять из k нулей, за которыми следуют $m - k$ единиц, а последовательность $\langle y_1, \dots, y_n \rangle$ —из l нулей с последующими $n - l$ единицами при некоторых k и l . Следовательно, последовательность $\langle v_1, v_2, \dots \rangle$ будет состоять из $\lceil k/2 \rceil + \lceil l/2 \rceil$ нулей с последующими единицами, а $\langle w_1, w_2, \dots \rangle$ —из $\lfloor k/2 \rfloor + \lfloor l/2 \rfloor$ нулей с последующими единицами. Решающим моментом доказательства является то, что

$$(\lceil k/2 \rceil + \lceil l/2 \rceil) - (\lfloor k/2 \rfloor + \lfloor l/2 \rfloor) = 0, 1 \text{ или } 2. \quad (3)$$

Если эта разность равна 0 или 1, то последовательность (2) уже упорядочена, а если она равна 2, то одна из операций сравнения-обмена в (1) ставит все на свои места. Доказательство завершено. (Заметим, что принцип нулей и единиц сводит $\binom{m+n}{m}$ случаев в задаче слияния всего лишь к $(m + 1)(n + 1)$, каждый из которых изображается двумя параметрами k и l .) Пусть $C(m, n)$ —число компараторных модулей, используемых при четно-нечетном слиянии m и n элементов, не считая начальных m - и n -сортировок; имеем

$$C(m, n) = \begin{cases} mn, & \text{если } mn \leq 1; \\ C(\lceil m/2 \rceil, \lceil n/2 \rceil) + C(\lfloor m/2 \rfloor, \lfloor n/2 \rfloor) + \lfloor (m + n - 1) / 2 \rfloor, & \text{если } mn > 1. \end{cases} \quad (4)$$

В общем случае это не слишком простая функция от m и n , однако, заметив, что $C(1, n) = n$ и что

$$C(m+1, n+1) - C(m, n) = 1 + C(\lceil m/2 \rceil + 1, \lfloor n/2 \rfloor + 1) - C(\lceil m/2 \rceil, \lfloor n/2 \rfloor), \quad \text{если } mn \geq 1,$$

мы можем вывести соотношение

$$C(m+1, n+1) - C(m, n) = t + 2 + \lfloor n/2^{t+1} \rfloor, \quad \text{если } n \geq m \geq 1 \text{ и } t = \lfloor \log_2 m \rfloor. \quad (5)$$

Следовательно,

$$C(m, m+r) = B(m) + m + R_m(r) \quad \text{при } m \geq 0, r \geq 0, \quad (6)$$

где $B(m)$ есть функция "бинарной вставки" $\sum_{1 \leq k \leq m} \lfloor \log_2 k \rfloor$ из соотношения (5.3.1-3), а $R_m(r)$ обозначает сумму первых r членов ряда

$$\left\lfloor \frac{r+0}{1} \right\rfloor + \left\lfloor \frac{r+1}{2} \right\rfloor + \left\lfloor \frac{r+2}{4} \right\rfloor + \left\lfloor \frac{r+3}{4} \right\rfloor + \left\lfloor \frac{r+4}{8} \right\rfloor + \dots + \left\lfloor \frac{r+j}{2^{\lfloor \log_2 j \rfloor + 1}} \right\rfloor + \dots \quad (7)$$

Если же $r = 0$, получаем важный частный случай

$$C(m, m) = B(m) + m. \quad (8)$$

Кроме того, если $t = \lfloor \log_2 m \rfloor$, то

$$\begin{aligned} R_m(r + 2^t) &= R_m(r) + 1 \cdot 2^{t-1} + 2 \cdot 2^{t-2} + \dots + 2^{t-1} \cdot 2^0 + m = \\ &= R_m(r) + m + t \cdot 2^{t-1}. \end{aligned}$$

Следовательно, $C(m, n + 2^t) - C(m, n)$ имеет простой вид и

$$C(m, n) = \left(\frac{t}{2} + \frac{m}{2^t} \right) n + O(1) \quad \text{при фиксированном } m, n \rightarrow \infty, t = \lfloor \log_2 m \rfloor; \quad (9)$$

член $O(1)$ становится в конце концов периодической функцией от n с длиной периода 2^t . Асимптотически при $n \rightarrow \infty$ величина $C(n, n)$ равна $n \log_2 n$ из (8) и упр. 5.3.1-15.

Сети с минимальным числом сравнений. Пусть $\hat{S}(n)$ — минимальное число сравнений, требуемых в сети сортировки для n элементов; ясно, что $\hat{S}(n) \geq S(n)$, где $S(n)$ — минимальное число сравнений, необходимое для сортировки без всяких ограничений (см. п. 5.3.1). Мы видели, что $\hat{S}(4) = 5 = S(4)$, поэтому новое ограничение не вызывает потери эффективности при $n = 4$; но уже при $n = 5$ оказывается, что $\hat{S}(5) = 9$, в то время как $S(5) = 7$. Задача определения $\hat{S}(n)$ кажется еще более трудной, чем задача определения $S(n)$; до сих пор неизвестно даже асимптотическое поведение $\hat{S}(n)$.

Интересно проследить историю этой задачи, так как на каждый новый шаг приходилось затрачивать определенные усилия. Сети сортировки были впервые исследованы П. Н. Армстронгом, Р. Дж. Нельсоном и Д. Дж. О'Коннором около 1954 г. [см. U. S. Patent 3029413]; они показали, что $\hat{S}(n+1) \leq \hat{S}(n) + n$. Как сказано в их патентной заявке, "приложив старания, можно сконструировать экономичные n -входные сортирующие переключатели, используя уменьшенное число двухвходных сортирующих переключателей"; они привели примеры конструкций для $4 \leq n \leq 8$, используя соответственно 5, 9, 12, 18 и 19 компараторов. Работая далее над этой задачей, Нельсон совместно с Р. Ч. Бозе еще до 1960 г. разработали общую процедуру для построения сетей сортировки, показывающую, что $\hat{S}(2^n) \leq 3^n - 2^n$ при всех n , поэтому $\hat{S}(n) = O(n^{\log_2 3}) = O(n^{1.585})$. Бозе и Нельсон опубликовали свой интересный метод в *ЖАСМ*, 9 (1962), 282–296, высказав предположение, что это наилучший возможный результат; Т. Н. Хиббард [*ЖАСМ*, 10 (1963), 142–150] нашел аналогичный, но несколько более простой метод, в котором используется такое же число сравнений, подкрепив тем самым это предположение.

В 1964 г. Р. У. Флloyd и Д. Э. Кнут использовали новый подход к этой задаче, приведший их к асимптотической оценке вида $\hat{S}(n) = O(n^{1+c/\sqrt{\log n}})$. Работая независимо, К. Э. Бэтчер открыл описанную выше общую стратегию слияния; используя число компараторов, определяемое как

$$c(1) = 0, c(n) = c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil, \lfloor n/2 \rfloor) \quad \text{при } n \geq 2, \quad (10)$$

он доказал, что (см. упр. 5.2.2-14)

$$c(2^t) = (t^2 - t + 4)2^{t-2} - 1,$$

и отсюда вывел, что $\hat{S}(n) = O(n(\log n)^2)$. Как Бэтчер, так и Флойд с Кнудом опубликовали свои конструкции лишь через некоторое время [Notices of the Amer. Math. Soc., 14 (1967), 283; Proc. AFIPS Spring Joint Computer Conference, 32 (1968), 307–314].

Кое-кому удалось сократить число компараторов, используемых в конструкции слияния с обменами, предложенной Бэтчером; в следующей таблице показаны наилучшие из известных в настоящее время верхних оценок для $\hat{S}(n)$:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
$c(n) =$	0	1	3	5	9	12	16	19	26	31	37	41	48	53	59	63		(11)
$\hat{S}(n) \leq$	0	1	3	5	9	12	16	19	25	29	35	39	46	51	56	60		

Так как $\hat{S}(n) < c(n)$ при $8 < n \leq 16$, то обменная сортировка со слиянием неоптимальна при всех $n > 8$. Если $n \leq 8$, то такая сортировка эквивалентна по количеству компараторов конструкции Бозе и Нельсона. Флойд и Кнут доказали в 1964–1966 гг., что указанные значения $\hat{S}(n)$ точны при $n < 8$ [см. A Survey of Combinatorial Theory (North-Holland, 1973), 163–172]; значения $\hat{S}(n)$ при $n > 8$ до сих пор неизвестны.

Конструкции, приводящие к указанным выше значениям для $\hat{S}(n)$, изображены на рис. 49. Сеть при $n = 9$, основанная на интересном трехпутевом слиянии, была найдена Р. У. Флойдом в 1964 г.; установить ее справедливость можно при помощи общего принципа, описанного в упр. 27. Сеть при $n = 10$ в 1969 г. построил А. Ваксман; он рассматривал входы как перестановки множества $\{1, 2, \dots, 10\}$ и пытался, сохраняя некоторую симметрию, насколько возможно уменьшить число значений, которые могут появляться в каждой строке на данной стадии.

В 1969 г. Г. Шапиро нашел сеть сортировки 16 элементов с 62 компараторами, и это было весьма неожиданно, поскольку метод Бэтчера (63 сравнения), казалось, использует все возможности, если n является степенью 2. М. У. Грин вскоре после того, как он ознакомился с конструкцией Шапиро, поверг всех в еще большее изумление, найдя сортировку с 60 сравнениями, показанную на рис. 49. Первая часть конструкции Грина довольно проста для понимания; после того как выполнены 32 операции сравнения-обмена слева от пунктирной линии, все прямые можно так пометить 16 подмножествами $\{a, b, c, d\}$, чтобы про прямую, помеченную s , было известно, что она содержит числа, меньшие или равные содержимому прямой, помеченной t , всякий раз, когда s есть подмножество t . Состояние сортировки в этот момент обсуждается более подробно в упр. 32. Однако сравнения, выполняемые на последующих уровнях, становятся совершенно загадочными, и до сих пор никто не знает, как обобщить эту конструкцию, чтобы получить столь же эффективные сети для больших значений n .

Шапиро и Грин открыли также изображенную на рис. 49 сеть для $n = 12$. Хорошие сети для $n = 11, 13, 14$ и 15 можно получить, удалив нижнюю прямую сети для $n + 1$ вместе со всеми компараторами, подсоединенными к этой линии.

Picture: Рис. 49. Эффективные сети сортировки.

Наилучшие известные к настоящему моменту сети для $n \rightarrow \infty$ см. в докторской диссертации Д. Ван Вориса (Stanford University, 1971); его сети требуют асимптотически $\frac{1}{4}n(\log_2 n)^2 - \alpha n \log_2 n$ компараторов, где $\alpha = \frac{1}{4} + \frac{1}{6} \sum_{k \geq 0} 2^{-2^k - k} \approx 0.356852$.

Сети с минимальным временем. В физических реализациях сетей сортировки и на параллельных ЭВМ можно выполнять непересекающиеся операции сравнения-обмена одновременно, поэтому кажется естественным попытаться минимизировать время задержки. По некотором размышлении заключаем, что время задержки сети сортировки равно максимальному числу компараторов, прилегающих к какому-либо "пути" через сеть, если определить путь как траекторию любого движения слева направо,

Picture: Рис. 50. Выполнение каждого сравнения в наиболее ранний из возможных моментов.

возможно, с переходом с одной прямой на другую через компараторы. У каждого компаратора мы можем поставить порядковый номер, указывающий самый ранний момент, когда может быть выполнено сравнение; этот номер на единицу больше, чем максимальный номер у компараторов, предшествующих данному. (См. рис. 50(a); в части (b) этого рисунка показана та же сеть, перерисованная так, чтобы каждое сравнение выполнялось в наиболее ранний возможный момент.)

В описанной выше сети Бэтчера для четно-нечетного слияния затрачивается $T_b(m, n)$ единиц времени, где $T_b(m, 0) = T_b(0, n) = 0$, $T_b(1, 1) = 1$ и

$$T_B(m, n) = 1 + \max(T_B(\lfloor m/2 \rfloor, \lfloor n/2 \rfloor), T_B(\lceil m/2 \rceil, \lceil n/2 \rceil)) \quad \text{для } mn \geq 2.$$

Используя эти соотношения, можно доказать по индукции, что $T_B(m, n+1) \geq T_B(m, n)$; следовательно, $T_B(m, n) = 1 + T_B(\lceil m/2 \rceil, \lceil n/2 \rceil)$ для $mn \geq 2$, а отсюда заключаем, что

$$T_B(m, n) = 1 + \lceil \log_2 \max(m, n) \rceil \quad \text{для } mn \geq 1. \quad (12)$$

Таким образом, как показано в упр. 5, метод сортировки Бэтчера имеет время задержки

$$\binom{1 + \lceil \log_2 n \rceil}{2}. \quad (13)$$

Пусть $\hat{T}(n)$ —минимальное время задержки, достижимое в любой сети сортировки n элементов. Некоторые из описанных выше сетей можно улучшить, не используя дополнительных компараторов, так, чтобы они имели меньшее время задержки, как

Picture: Рис. 51. Сети сортировки, которые необыкновенно быстры, если сравнения выполняются параллельно.

показано на рис. 51 для $n = 6$ и $n = 9$, а в упр. 7—для $n = 10$. Можно получить еще меньшее время задержки, если добавить один или два дополнительных модуля, как показывают сети для $n = 10, 12$ и 16 на рис. 51. Эти построения приводят к следующим верхним оценкам для $T(n)$ при умеренных значениях n :

$$\hat{T}(n) \leq \begin{matrix} n = 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \leq 0 & 1 & 3 & 3 & 5 & 5 & 6 & 6 & 7 & 7 & 8 & 8 & 9 & 9 & 9 & 9 \end{matrix} \quad (14)$$

Известно, что приведенные здесь значения точны при $n \leq 8$ (см. упр. 4). Сети на рис. 51 заслуживают тщательного изучения, поскольку вовсе не очевидно, что они годятся для сортировки; эти сети были открыты в 1969–1971 гг. Г. Шапиро ($n = 6, 9, 12$) и Д. Ван Ворисом ($n = 10, 16$).

Сети слияния. Пусть $\hat{M}(m, n)$ обозначает минимальное число компараторов, необходимых для сети, которая сливает m элементов $x_1 \leq \dots \leq x_m$ с n элементами $y_1 \leq \dots \leq y_n$, образуя отсортированную последовательность $z_1 \leq \dots \leq z_{m+n}$. К настоящему времени не открыто ни одной сети слияния, которая была бы лучше четно-нечетного слияния, описанного выше; следовательно, функция $C(m, n)$ в (6) представляет наилучшую известную верхнюю оценку для $\hat{M}(m, n)$.

Р. У. Флойд обнаружил интересный способ, позволяющий определить *нижние* оценки в этой задаче слияния.

Теорема F. При всех $n \geq 1$ справедливо неравенство $\hat{M}(2n, 2n) \geq 2\hat{M}(n, n) + n$.

Доказательство. Рассмотрим сеть с $\hat{M}(2n, 2n)$ компараторными модулями, способную сортировать все входные последовательности $\langle z_1, \dots, z_{4n} \rangle$, такие, что $z_1 \leq z_3 \leq \dots \leq z_{4n-1}$ и $z_2 \leq z_4 \leq \dots \leq z_{4n}$. Мы можем считать, что каждый модуль заменяет (z_i, z_j) на $(\min(z_i, z_j), \max(z_i, z_j))$ при некоторых $i < j$ (упр. 16). Итак, компараторы можно разделить на три класса:

- $i \leq 2n$ и $j \leq 2n$;
- $i > 2n$ и $j > 2n$;
- $i \leq 2n$ и $j > 2n$.

Класс (а) должен содержать по крайней мере $\hat{M}(n, n)$ компараторов, так как $z_{2n+1}, z_{2n+2}, \dots, z_{4n}$ могут уже находиться на своих местах, когда слияние начинается; аналогично в классе (b) должно быть по крайней мере $\hat{M}(n, n)$ компараторов. Кроме того, как показывает входная последовательность $\langle 0, 1, 0, 1, \dots, 0, 1 \rangle$, класс (с) содержит не менее n компараторов, так как n нулей должны переместиться из $\{z_{2n+1}, \dots, z_{4n}\}$ в $\{z_1, \dots, z_{2n}\}$. ■

Многokратное применение теоремы F доказывает, что $\hat{M}(2^m, 2^m) \geq \frac{1}{2}(m+2)2^m$; следовательно, $\hat{M}(n, n) \geq \frac{1}{2}n \log_2 n + O(n)$. Слияние без сетевого ограничения требует лишь $M(n, n) = 2n - 1$ сравнений; таким образом, мы доказали, что сетевое слияние сложнее по существу, чем слияние вообще. Четно-нечетное слияние показывает, что $\hat{M}(n, n) \leq C(n, n) = n \log_2 n + O(n)$, поэтому асимптотическое поведение $\hat{M}(n, n)$ известно с точностью до множителя 2. (Точные значения $\hat{M}(n, n)$ известны для $n \leq 5$; см. упр. 9.) А. К. Яо и Ф. Яо доказали, что $\hat{M}(2, n) = C(2, n) = \lceil \frac{3}{2}n \rceil$ и $\hat{M}(m, n) \geq \frac{1}{2}n \log_2(m+1)$ при $m < n$ [JACM, будет опубликовано].

Битонная сортировка. Если допустимы одновременные сравнения, то, как видно из формулы 12, при четно-нечетном слиянии для $1 \leq m \leq n$ возникает задержка на $\lceil \log_2(2n) \rceil$ единиц времени. Бэтчер нашел другой тип сети слияния, называемой *битонным сортировщиком*, для которого время задержки снижается до $\lceil \log_2(m+n) \rceil$, но он требует больше компараторных модулей.

Последовательность $\langle z_1, \dots, z_p \rangle$ из p чисел будем называть *битонной*, если $z_1 \geq \dots \geq z_k \leq \dots \leq z_p$ для некоторого k , $1 \leq k \leq p$ (сравните это с обычным определением "монотонных" последовательностей). Битонный сортировщик порядка p —это компараторная сеть, способная сортировать в неубывающем порядке любую битонную последовательность длины p . Задача слияния $x_1 \leq \dots \leq x_m$ с $y_1 \leq \dots \leq y_n$ является частным случаем задачи битонной сортировки, так как слияние можно осуществить, применив к последовательности $\langle x_m, \dots, x_1, y_1, \dots, y_n \rangle$ битонный сортировщик порядка $m + n$.

Заметим, что если последовательность $\langle z_1, \dots, z_p \rangle$ битонная, то таковыми же являются и все ее подпоследовательности. Вскоре после того, как Бэтчер открыл сети четно-нечетного слияния, он обнаружил, что аналогичным образом можно построить битонный сортировщик порядка p , сначала независимо сортируя битонные подпоследовательности $\langle z_1, z_3, z_5, \dots \rangle$ и $\langle z_2, z_4, z_6, \dots \rangle$, а затем выполняя сравнения-обмены $z_1 : z_2, z_3 : z_4, \dots$ (Доказательство см. в упр. 10.) Если соответствующее число компараторных модулей обозначить через $C'(p)$, то будем иметь

$$C'(p) = C'(\lceil p/2 \rceil) + C'(\lfloor p/2 \rfloor) + \lfloor p/2 \rfloor \quad \text{при } p \geq 2. \quad (15)$$

а время задержки, очевидно, равно $\lceil \log_2 p \rceil$. На рис. 52 показан битонный сортировщик порядка 7, построенный этим способом; он может быть использован и как (3, 4), и как (2, 5)-сеть слияния с задержкой в три единицы; четно-нечетное слияние для $m = 2$ и $n = 5$ имеет на один компаратор меньше, но на один уровень задержки больше.

Битонный сортировщик Бэтчера порядка 2^k особенно интересен; он состоит из k уровней по 2^{k-1} компараторов в каждом. Занумеруем входные прямые $z_0, z_1, \dots, z_{2^k-1}$; при этом элемент 2, сравнивается с z_j на уровне l тогда и только тогда, когда двоичные представления i и j различаются только в l -м бите слева. Эта простая структура приводит к параллельной сети сортировки, которая так же быстра, как обменная сортировка

Picture: Рис. 52. Битонный сортировщик Бэтчера порядка 7.

со слиянием (алгоритм 5.2.2М), но значительно проще для реализации. (См. упр. 11 и 13.)

Если $m = n$, то нетрудно видеть, что и четно-нечетное слияние, и битонный сортировщик Бэтчера обеспечивают абсолютный минимум времени задержки, достижимого в любой сети слияния.

Picture: Рис. 53. Один элемент сливается с шестью другими с разветвлением, чтобы достичь минимально возможного времени задержки.

В (n, n) -сети слияния n -й по величине выход (считая от наименьшего) должен зависеть от всех $2n$ входов, и если его можно вычислить за l шагов, то он будет зависеть не более, чем от 2^l входов; поэтому $2^l \geq 2n$, $l \geq \lceil \log_2(2n) \rceil$.

Если $m < n$, то n -й выход (m, n) -сети слияния зависит от $2m + 1$ входов (ср. с упр. 29), поэтому те же рассуждения дают в этом случае минимальное время задержки для слияния $\lceil \log_2(2m + 1) \rceil$. Бэтчер показал [Report GER-14122 (Akron, Ohio: Goodyear Aerospace Corporation, 1968)], что это минимальное время задержки достигается, если в сети допускается разветвление, т. е. такое разбиение линий, что одно и то же число в одно и то же время используется несколькими модулями. В качестве примера на рис. 53 изображена (для $n = 6$) сеть, которая сливает один элемент с n другими всего с двумя уровнями задержки. Конечно, сети с разветвлением не соответствуют нашим соглашениям; довольно легко понять, что любая $(1, n)$ -сеть слияния без разветвления должна иметь время задержки $\log_2(n + 1)$ или более. (См. упр. 14.)

Сети выбора. Сети можно применить также и к задаче п. 5.3.3. Пусть $\hat{U}_t(n)$ обозначает минимальное число компараторов, необходимых в сети, которая перемещает t наибольших из n различных входов на t определенных выходных прямых, они могут располагаться на этих выходных прямых в произвольном порядке. Пусть $\hat{V}_t(n)$ обозначает минимальное количество компараторов, нужное для перемещения t -го в порядке убывания из n различных входов на определенную выходную прямую, и пусть $\hat{W}_t(n)$ обозначает минимальное число компараторов, требуемых для перемещения t наибольших из n различных входов на определенные t выходных прямых в неубывающем порядке. Нетрудно видеть (см. упр. 17), что

$$\hat{U}_t(n) \leq \hat{V}_t(n) \leq \hat{W}_t(n). \quad (16)$$

Сначала предположим, что имеется $2t$ элементов $\langle x_1, \dots, x_{2t} \rangle$ и мы хотим выбрать t наибольших; В. Е. Алексеев [Кибернетика, 5, 5 (1969), 99–103] заметил, что это может быть выполнено, если сначала отсортировать $\langle x_1, \dots, x_t \rangle$ и $\langle x_{t+1}, \dots, x_{2t} \rangle$, а затем сравнить и поменять местами

$$x_1 : x_{2t}, x_2 : x_{2t-1}, \dots, x_t : x_{t+1}. \quad (17)$$

Так как ни в одной из этих пар не может содержаться более одного из наибольших t элементов (почему?), то процедура Алексеева должна выбрать t наибольших элементов.

Если нам нужно выбрать t наибольших из nt элементов, то мы можем применить эту процедуру $n - 1$ раз (исключая каждый раз t элементов); следовательно,

$$\hat{U}_t(nt) \leq (n - 1)(2\hat{S}(t) + t). \tag{18}$$

Алексеев также получил интересную *нижнюю* оценку для задачи выбора.

Теорема А. $\hat{U}_t(n) \geq (n - t)\lceil \log_2(t + 1) \rceil$.

Доказательство. Удобнее рассматривать эквивалентную задачу выбора *наименьших* t элементов. Около каждой прямой компараторной сети можно выписать числа (l, u) , как показано на рис. 54, где l и u обозначают соответственно минимальное и

Picture: Рис. 54. Отделение четырех наибольших от четырех наименьших. (Числа над прямыми используются в доказательстве теоремы А.)

максимальное значения, которые могут появиться в этом месте, если входом служит перестановка $\{1, 2, \dots, n\}$. Пусть l_i и l_j —нижние оценки на прямых i и j перед сравнением $x_i : x_j$, и пусть l'_i и l'_j —соответствующие нижние оценки после этого

Picture: Рис. 55. Иная интерпретация сети, изображенной на рис. 54.

сравнения. Очевидно, что $l'_i = \min(l_i, l_j)$, а в упр. 24 доказывается (неочевидное) соотношение

$$l'_j \leq l_i + l_j. \tag{19}$$

Теперь дадим другую интерпретацию действия сети (рис. 55); предположим, что на всех входных прямых содержится ноль, а каждый "компаратор" помещает теперь на верхнюю прямую меньший из его входов, а на нижнюю прямую—большой вход *плюс один*. Получающиеся числа $\langle m_1, m_2, \dots, m_n \rangle$ обладают свойством

$$2^{m_i} \geq l_i \tag{20}$$

в любом месте сети, так как это свойство первоначально справедливо и сохраняется каждым компаратором в силу (19). Кроме того, окончательное значение

$$m_1 + m_2 + \dots + m_n$$

равно общему числу компараторов в сети, так как каждый компаратор добавляет к этой сумме единицу.

Если сеть выбирает наименьшие t чисел, то $n - t$ из чисел l_i больше или равны $t + 1$; следовательно, $n - t$ из чисел m_i должны быть $\geq \lceil \log_2(t + 1) \rceil$. ■

Нижняя оценка в теореме А оказывается точной, если $t = 1$ или $t = 2$ (см. упр. 19). В табл. 1 даны значения $\hat{U}_t(n)$, $\hat{V}_t(n)$ и $\hat{W}_t(n)$ для небольших t и n .

Сравнения, необходимые для сетей выбора ($\hat{U}_t(n)$, $\hat{V}_t(n)$, $\hat{W}_t(n)$)	Таблица 1					
	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$
$n = 1$	(0, 0, 0)					
$n = 2$	(1, 1, 1)	(0, 1, 1)				
$n = 3$	(2, 2, 2)	(2, 3, 3)	(0, 2, 3)			
$n = 4$	(3, 3, 3)	(4, 5, 5)	(3, 5, 5)	(0, 3, 5)		
$n = 5$	(4, 4, 4)	(6, 7, 7)	(6, 7, 8)	(4, 7, 9)	(0, 4, 9)	
$n = 6$	(5, 5, 5)	(8, 9, 9)	(8, 10, 10)	(8, 10, 12)	(5, 9, 12)	(0, 5, 12)

Упражнения

(ПЕРВАЯ ЧАСТЬ) Далее в нескольких упражнениях дано более глубокое развитие теории сетей сортировки, поэтому будет удобно ввести некоторые обозначения. Вместо модуля сравнения-обмена будем писать $[i : j]$. Сеть с n входами и r компараторными модулями запишем как $[i_1 : j_1][i_2 : j_2] \dots [i_r : j_r]$, где все i и j меньше или равны n ; для краткости будем называть ее n -сетью. Сеть называется

стандартной, если $i_q < j_q$ для $1 \leq q \leq r$. Так, например, рис. 44 описывает стандартную 4-сеть, обозначаемую последовательностью компараторов $[1 : 2][3 : 4][1 : 3][2 : 4][2 : 3]$.

Наши соглашения в тексте об изображении диаграмм сетей позволяют рисовать только стандартные сети; все компараторы $[i : j]$ изображаются прямой от i к j , где $i < j$. Если нужно нарисовать нестандартную сеть, то можно использовать *стрелку* от i к j , указывающую, что большее число направляется к острию стрелки. Например, на рис. 56 изображена нестандартная сеть для 16 элементов с компараторами $[1 : 2][4 : 3][5 : 6][8 : 7]$ и т. д. В упр. 11 доказывается, что это сеть сортировки. Если $x = \langle x_1, \dots, x_n \rangle$ есть n -вектор, а α есть n -сеть, то используем обозначение $x\alpha$ для вектора чисел $\langle (x\alpha)_1, \dots, (x\alpha)_n \rangle$, порожденных сетью. Положим также для краткости $a \vee b = \max(a, b)$, $a \wedge b = \min(a, b)$, $\bar{a} = 1 - a$; тогда $(x[i : j])_i = x_i \wedge x_j$, $(x[i : j])_j = x_i \vee x_j$ и $(x[i : j])_k = x_k$ для $i \neq k \neq j$. Будем говорить, что α является *сетью сортировки*, тогда и только тогда, когда $(x\alpha)_i \leq (x\alpha)_{i+1}$ для $1 \leq i < n$ и всех x .

Символ $e^{(i)}$ обозначает вектор, у которого в позиции i находится 1, а в остальных местах 0; таким образом, $(e^{(i)})_j = \delta_{ij}$. Символ D_n обозначает множество всех 2^n n -местных векторов из 0 и 1, а P_n обозначает множество всех $n!$ векторов, являющихся перестановками $\{1, 2, \dots, n\}$. Мы будем использовать обозначения $x \wedge y$ и $x \vee y$ для векторов $\langle x_1 \wedge y_1, \dots, x_n \wedge y_n \rangle$ и $\langle x_1 \vee y_1, \dots, x_n \vee y_n \rangle$ и будем писать $x \leq y$, если $x_i \leq y_i$ при всех i . Таким образом, $x \leq y$ тогда и только тогда, когда $x \vee y = y$, и тогда и только тогда, когда

Picture: Рис. 56. Нестандартная сеть, основанная на битонной сортировке.

$x \wedge y = x$. Если x и y лежат в D_n , то будем говорить, что x *покрывает* y , если $x = y \vee e^{(i)} \neq y$ при некотором i . Наконец, для всех x в D_n пусть $\nu(x)$ будет числом единиц в x , а $\zeta(x)$ — числом нулей; таким образом, $\nu(x) + \zeta(x) = n$.

1. [20] Нарисуйте сеть четно-нечетного слияния для $m = 3$ и $n = 5$.
2. [22] Покажите, что алгоритму сортировки В. Пратта (см. упр. 5.2.1-30) соответствует сеть сортировки n элементов, имеющая приблизительно $(\log_2 n) \times (\log_3 n)$ уровней задержки. Нарисуйте такую сеть для $n = 12$.
3. [M20] (К. Э. Бэтчер.) Найдите простое соотношение между $C(m, m-1)$ и $C(m, m)$.
- >4. [M23] Докажите, что $\hat{T}(6) = 5$.
5. [M21] Докажите, что выражение (13) действительно определяет время задержки для сети сортировки, описанной соотношениями (10).
6. [28] Пусть $T(n)$ будет минимальным числом стадий, требуемых для сортировки с *одновременным выполнением непересекающихся сравнений* (без сетевого ограничения); каждое такое множество сравнений может быть представлено узлом, содержащим множество пар $i_1 : j_1, i_2 : j_2, \dots, i_r : j_r$, где все $i_1, j_1, i_2, j_2, \dots, i_r, j_r$ различны; от этого узла отходит вниз 2^r ветвей, соответствующих случаям

$$\langle K_{i_1} < K_{j_1}, K_{i_2} < K_{j_2}, \dots, K_{i_r} < K_{j_r} \rangle;$$

$$\langle K_{i_1} > K_{j_1}, K_{i_2} < K_{j_2}, \dots, K_{i_r} < K_{j_r} \rangle \text{ и т. д.}$$

Докажите, что $T(5) = T(6) = 5$.

7. [25] Покажите, что если последний компаратор сети для $n = 10$ на рис. 49 поместить непосредственно перед вторым и третьим с конца компараторами, то сеть по-прежнему будет сортировать.
8. [M20] Докажите, что $\hat{M}(m_1 + m_2, n_1 + n_2) \geq \hat{M}(m_1, n_1) + \hat{M}(m_2, n_2) + \min(m_1, n_2)$ при $m_1, m_2, n_1, n_2 \geq 0$.
9. [M25] (Р. У. Флойд.) Докажите, что $\hat{M}(3, 3) = 6$, $\hat{M}(4, 4) = 9$, $\hat{M}(5, 5) = 13$.
10. [M22] Докажите, что битонный сортировщик Бэтчера, как он определен в тексте перед (15), действительно работает. [Указание. Достаточно доказать, что будут сортироваться все последовательности, состоящие из k единиц, за которыми следуют l нулей, за которыми следуют $n - k - l$ единиц.]
11. [M23] Докажите, что битонный сортировщик Бэтчера порядка 2^p будет сортировать не только последовательности $\langle z_0, z_1, \dots, z_{2^p-1} \rangle$, для которых $z_0 \geq \dots \geq z_k \leq \dots \leq z_{2^p-1}$, но также и все последовательности, для которых $z_0 \leq \dots \leq z_k \geq \dots \geq z_{2^p-1}$. [Как следствие этого, сеть на рис. 56 будет сортировать 16 элементов, так как каждая стадия состоит из битонных сортировщиков или обращенных битонных сортировщиков, применяемых к последовательностям, которые были отсортированы в противоположных направлениях.]
12. [M20] Докажите или опровергните следующее утверждение: если x и y — битонные последовательности равной длины, то последовательности $x \vee y$ и $x \wedge y$ также битонные.
- >13. [24] (Х. С. Стоун). Покажите, что сеть сортировки для 2^t элементов можно построить по схеме, проиллюстрированной для $t = 4$ на рис. 57. Каждый из t^2 шагов этой схемы состоит из "идеального

тасования” первых 2^{t-1} элементов с последними 2^{t-1} , за которым следуют операции, выполняемые одновременно над 2^{t-1} парами соседних элементов. Каждая из этих операций обозначена либо ”0” (нет операции), либо ”+” (стандартный компараторный модуль), либо ”-” (обращенный компараторный модуль). Сортировка протекает в t стадий по t шагов каждая; на последней стадии все операции суть ”+”. В течение стадии s при $s < t$ мы выполняем $t - s$ шагов, где все операции суть ”0”, а затем выполняем s шагов, где на каждом $q - m$ шаге поочередно выполняются 2^{q-1} операций ”+” и затем 2^{q-1} операций ”-” при $q = 1, 2, \dots, s$.

[Заметим, что эта схема сортировки может быть выполнена весьма простым устройством, реализующим один шаг ”тасования и операций” и передающим выход обратно на вход. Первые три шага на рис. 57 можно, конечно, устранить, они оставлены, лишь чтобы сделать схему понятнее. Стоун замечает, что тот же принцип ”тасования/операции” встречается в некоторых других алгоритмах, таких, как быстрое преобразование Фурье (см. п. 4.6.4).]

14. [M20] Докажите, что любая $(1, n)$ -сеть слияния без разветвления должна иметь не менее $\lceil \log_2(n+1) \rceil$ уровней задержки.
15. [20] Найдите нестандартную сеть сортировки четырех элементов, содержащую только пять компараторных модулей.
16. [M22] Докажите, что следующий алгоритм преобразует любую сеть сортировки $[i_1 : j_1] \cdots [i_r : j_r]$ в стандартную сеть сортировки:

T1. Пусть q — наименьший индекс, такой, что $i_q > j_q$. Если таких индексов нет, то остановиться.

T2. Заменить все вхождения i_q на j_q и все вхождения j_q на i_q во всех компараторах $[i_s : j_s]$ для $q \leq s \leq r$. Вернуться к шагу T1. ■

Например, сеть $[4 : 1] [3 : 2] [1 : 3] [2 : 4] [1 : 2] [3 : 4]$ преобразуется сначала в $[1 : 4] [3 : 2] [4 : 3] [2 : 1] [4 : 2] [3 : 1]$, затем в $[1 : 4] [2 : 3] [4 : 2] [3 : 1] [4 : 3] [2 : 1]$, затем в $[1 : 4] [2 : 3] [2 : 4] [3 : 1] [2 : 3] [4 : 1]$ и т. д., пока не получится стандартная сеть $[1 : 4] [2 : 3] [2 : 4] [1 : 3] [1 : 2] [3 : 4]$.

Picture: Рис. 57. Сортировка 16 элементов с ”идеальным тасованием”.

17. [M25] Пусть D_{tn} будет множеством всех $\binom{n}{t}$ последовательностей $\langle x_1, \dots, x_n \rangle$ из нулей и единиц, имеющих ровно t единиц. Докажите, что $\hat{U}_t(n)$ равно минимальному числу компараторов, которые необходимы в сети, сортирующей все элементы D_{tn} ; что $\hat{V}_t(n)$ равно минимальному числу компараторов, нужных для сортировки $D_{tn} \cup D_{(t-1)n}$; и что $\hat{W}_t(n)$ равно минимальному числу компараторов, нужных для сортировки $\bigcup_{0 \leq k \leq t} D_{kn}$.
- >18. [M20] Докажите, что сеть, которая определяет медиану $2t - 1$ элементов, требует не менее $(t - 1)\lceil \log_2(t + 1) \rceil + \lceil \log_2 t \rceil$ компараторных модулей. [Указание: см. доказательство теоремы А.]
19. [M22] Докажите, что $\hat{U}_2(n) = 2n - 4$ и $\hat{V}_2(n) = 2n - 3$ для всех $n \geq 2$.
20. [24] Докажите, что $\hat{V}_3(5) = 7$.
21. [M15] Пусть α — любая n -сеть, а x и y — два n -вектора. Докажите, что из $x \leq y$ следует $x\alpha \leq y\alpha$.
22. [M15] Докажите, что если x и y суть n -векторы действительных чисел, то $x \cdots y \leq (x\alpha) \cdot (y\alpha)$. (Здесь $x \cdot y$ — скалярное произведение $x_1y_1 + \dots + x_ny_n$.)
23. [M17]. Пусть α есть n -сеть. Докажите, что существует перестановка $p \in P_n$, такая, что $(p\alpha)_i = j$ тогда и только тогда, когда в D_n найдутся векторы x, y , такие, что x покрывает y , $(x\alpha)_i = 1$, $(y\alpha)_i = 0$ и $\zeta(y) = j$.
- >24. [M21] (В. Е. Алексеев.) Пусть α есть n -сеть; введем обозначения $l_k = \min\{(p\alpha)_k \mid p \in P_n\}$, $u_k = \max\{(p\alpha)_k \mid p \in P_n\}$ при $1 \leq k \leq n$ для нижней и верхней границ диапазона значений, которые могут появляться на прямой k выхода. Пусть l'_i и u'_i — аналогично определенные величины для сети $\alpha' = \alpha[i : j]$. Докажите, что $l'_i = l_i \wedge l_j$, $l'_j \leq l_i + l_j$, $u'_i \geq u_i + u_j - (n + 1)$, $u'_j = u_i \vee u_j$. [Указание: для данных векторов x и y из D_n , таких, что $(x\alpha)_i = (y\alpha)_j = 0$, $\zeta(x) = l_i$, $\zeta(y) = l_j$, найдите вектор z из D_n , такой, что $(z\alpha')_j = 0$, $\zeta(z) \leq l_i + l_j$.]
25. [M30] Пусть l_k и u_k определены, как в упр. 24. Докажите, что множество $\{(p\alpha)_k \mid p \in P_n\}$ содержит все целые числа между l_k и u_k включительно.
26. [M24] (Р. У. Флойд.) Пусть α есть n -сеть. Докажите, что множество $D_n\alpha = \{x\alpha \mid x \in D_n\}$ может быть определено, из множества $P_n\alpha = \{p\alpha \mid p \in P_n\}$ и, обратно, $P_n\alpha$ может быть определено из $D_n\alpha$.
- >27. [M20] Пусть x и y — векторы, и пусть $x\alpha$ и $y\alpha$ — отсортированные векторы. Докажите, что $(x\alpha)_i \leq (y\alpha)_j$ тогда и только тогда, когда для любой совокупности j элементов из y можно найти совокупность i элементов из x , такую, что любой элемент, взятый из x , меньше некоторого элемента,

взятого из y , или равен ему. Используйте этот принцип для доказательства того, что *если отсортировать строки любой матрицы, а затем отсортировать столбцы, то строки останутся упорядоченными.*

- >28. [M20] Следующая диаграмма показывает, как записать формулы для содержимого всех линий сети сортировки через ее входы:

Picture: p.287

Используя законы коммутативности $x \wedge y = y \wedge x$, $x \vee y = y \vee x$, законы ассоциативности $x \wedge (y \wedge z) = (x \wedge y) \wedge z$, $x \vee (y \vee z) = (x \vee y) \vee z$, законы дистрибутивности $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$, $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, законы поглощения $x \wedge (x \vee y) = x$ и законы идемпотентности $x \wedge x = x \vee x = x$, мы можем свести формулы в правой части этой сети соответственно к $(a \wedge b \wedge c \wedge d)$, $(a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \vee (b \wedge c \wedge d)$, $(a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$, $a \vee b \vee c \vee d$. Докажите, что в общем случае t -й в порядке убывания элемент из $\{x_1, \dots, x_n\}$ дается "элементарной симметрической функцией"

$$\sigma_t(x_1, \dots, x_n) = \bigvee \{x_{i_1} \wedge x_{i_2} \wedge \dots \wedge x_{i_t} \mid 1 \leq i_1 < i_2 < \dots < i_t \leq n\}.$$

[Здесь $\binom{n}{t}$ членов объединяются операцией \vee вместе. Таким образом, задача нахождения сети сортировки минимальной стоимости эквивалентна задаче вычисления элементарных симметрических функций с минимальным числом схем "и/или", где на каждом шаге две величины ϕ и ψ заменяются на $\phi \wedge \psi$ и $\phi \vee \psi$.]

29. [M20] Дано, что $x_1 \leq x_2 \leq x_3$ и $y_1 \leq y_2 \leq y_3 \leq y_4 \leq y_5$ и что $z_1 \leq z_2 \leq \dots \leq z_8$ — результат слияния x с y . Найдите выражения для каждого z через x и y , используя операторы \wedge и \vee .
30. [BM24] Докажите, что любая формула, содержащая \wedge , \vee и независимые переменные $\{x_1, \dots, x_n\}$, может быть приведена с использованием тождеств из упр. 28 к "канонической" форме $\tau_1 \vee \tau_2 \vee \dots \vee \tau_k$, здесь $k \geq 1$ и каждый τ_i имеет вид $\wedge \{x_j \mid j \in S_i\}$, где S_i — подмножество $\{1, 2, \dots, n\}$ и никакое множество S_i не включается в S_j , если $i \neq j$. Докажите также, что две такие канонические формы равны для всех x_1, \dots, x_n тогда и только тогда, когда они идентичны (с точностью до порядка).
31. [M24] (Р. Дедекинд, 1897.) Пусть δ_n — число различных канонических форм от x_1, \dots, x_n в смысле упр. 30. Так, $\delta_1 = 1$, $\delta_2 = 4$ и $\delta_3 = 18$. Чему равно δ_4 ?
32. [M28] (М. У. Грин.) Пусть $G_1 = \{00, 01, 11\}$; определим G_{n+1} как множество всех цепочек $\theta\phi\psi\omega$, таких, что $\theta, \phi, \psi, \omega$ имеют длину 2^{n+1} и $\theta\phi, \psi\omega, \theta\psi$ и $\phi\omega$ принадлежат G_n . Пусть α — сеть, состоящая из четырех первых уровней 16-сортировщика, изображенного на рис. 48. Покажите, что $D_{16}\alpha = G_4$, и докажите, что это множество имеет в точности $\delta_4 + 2$ элементов. (См. упр. 31.)
- >33. [M22] Не все δ_n функций от $\langle x_1, \dots, x_n \rangle$ из упр. 31 могут встретиться в компараторных сетях. А именно докажите, что функция $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$ не может быть результатом никакой компараторной сети от $\langle x_1, \dots, x_n \rangle$.
34. [23] Является ли следующая сеть сетью сортировки?

Picture: p.288

34. [20] Докажите, что в любой *стандартной сети* сортировки должен по крайней мере один раз встретиться каждый из компараторов $[i : i + 1]$ при $1 \leq i < n$.
- >36. [22] Сеть на рис. 47 содержит только *кратчайшие* сравнения $[i : i + 1]$; будем называть такие сети *примитивными*, (а) Докажите, что примитивная сеть сортировки для n элементов должна иметь не менее $\binom{n}{2}$ компараторов. [Указание: рассмотрите инверсии перестановки.] (б) (Р. У. Флойд, 1964.) Пусть α — примитивная сеть для n элементов, а x — вектор, такой, что $(x\alpha)_i > (x\alpha)_j$ при некоторых $i < j$. Докажите, что $(y\alpha)_i > (y\alpha)_j$, где y — вектор $\langle n, n - 1, \dots, 1 \rangle$. (с) В качестве следствия (б) докажите, что примитивная сеть является сетью сортировки тогда и только тогда, когда она сортирует единственный вектор $\langle n, n - 1, \dots, 1 \rangle$!
37. [M22] Четно-нечетная сортировка с транспозициями для n чисел, $n \geq 3$, это n -уровневая сеть с $\frac{1}{2}n(n - 1)$ компараторами, напоминающая кирпичную кладку (рис. 58). (Если n четно, имеются две возможности.) Такую сортировку особенно легко реализовать аппаратно, так как попеременно выполняются только два вида действий. Докажите, что такая сеть действительно будет правильной сетью сортировки. [Указание: см. упр. 36.]

Picture: Рис. 58. Четно-нечетная сортировка с транспозициями.

38. [29] Можно дать другую интерпретацию сетям сортировки, считая, что на каждой линии находится мультимножество из m чисел, а не одно число; при этой интерпретации операция $[i : j]$

заменяет x_i и x_j соответственно на x_i и x_j и $x_i \text{ dndn } x_j$ — наименьшие m и наибольшие m из $2m$ чисел $x_i \uplus x_j$. (Рис. 59 иллюстрирует это при $m = 2$.) Если a и b суть мультимножества, содержащие m чисел каждое, то будем говорить, что $a \text{ lfff } b$ тогда

Picture: Рис. 59. Другая интерпретация сети сортировки, представленной на рис. 44: каждый компараторный модуль выполняет операцию слияния.

и только тогда, когда $a \text{ urup } b = a$ (или, эквивалентно, $a \text{ dndn } b = b$; наибольший элемент a меньше или равен наименьшему элементу b). Таким образом, $a \text{ urup } b \text{ lfff } a \text{ dndn } b$.

Пусть α есть n -сеть, а $x = \langle x_1, \dots, x_n \rangle$ — вектор, в котором каждая компонента x_i — мультимножество из m элементов. Докажите, что если $(x\alpha)_i$ не $\text{lfff } (x\alpha)_j$ в описанной интерпретации, то в D_n найдется вектор y , такой, что $(y\alpha)_i = 1$ и $(y\alpha)_j = 0$. [Следовательно, сеть сортировки n элементов превращается в сеть сортировки mn элементов, если заменить сравнения m -путевыми слияниями. На рис. 60 изображен восьмизэлементный сортировщик, построенный из четырехэлементного с использованием этого наблюдения.]

- >39. [M23] Покажите, что в обозначениях упр 38 ($x \text{ urup } y$) $\text{urup } z = x \text{ urup } (y \text{ urup } z)$ и $(x \text{ dndn } y) \text{ dndn } z = x \text{ dndn } (y \text{ dndn } z)$, однако $(x \text{ dndn } y) \text{ urup } z$ не всегда равно $(x \text{ urup } z) \text{ dndn } (y \text{ urup } z)$ и $(x \text{ urup } y) \text{ dndn } (x \text{ urup } z) \text{ dndn } (y \text{ urup } z)$ не всегда равно средним m элементам $x \uplus y \uplus z$. Найдите правильную формулу для этих средних элементов, используя в ней x, y, z , а также операции urup и dndn .
40. [M25] (Р. Л. Грэхем.) Компаратор $[i : j]$ называется избыточным в сети $\alpha_1 [i : j] \alpha_2$, если либо $(x\alpha_1)_i \leq (x\alpha_1)_j$ для всех векторов x , либо $(x\alpha_1)_i \geq (x\alpha_1)_j$ для всех векторов x . Докажите, что если α является сетью с r неизбыточными компараторами, то найдутся по крайней мере r различных упорядоченных

Picture: Рис. 60. 8-сортировщик, построенный из 4-сортировщика с использованием слияния.

пар (i, j) различных индексов, таких, что $(x\alpha)_i \leq (x\alpha)_j$ для всех векторов x . (Следовательно, сеть без избыточных компараторов содержит не более $\binom{n}{2}$ модулей.)

- >41. [M27] (В. Е. Алексеев.) Пусть $\alpha = [i_1 : j_1] \dots [i_r : j_r]$ есть n -сеть; для $1 \leq s \leq r$ определим $\alpha^s = [i'_1 : j'_1] \dots [i'_{s-1} : j'_{s-1}] [i_s : j_s] \dots [i_r : j_r]$, где i'_k и j'_k получены из i_k и j_k заменой i_s на j_s и j_s на i_s везде, где они встречаются. Например, если $\alpha = [1 : 2] [3 : 4] [1 : 3] [2 : 4] [2 : 3]$, то $\alpha^4 = [1 : 4] [3 : 2] [1 : 3] [2 : 4] [2 : 3]$.
- a) Докажите, что $D_n \alpha = D_n (\alpha^s)$.
- b) Докажите, что $(\alpha^s)^t = (\alpha^t)^s$.
- c) Сопряжением α является любая сеть вида $(\dots ((\alpha^{s_1})^{s_2}) \dots)^{s_k}$. Докажите, что α имеет не более 2^{r-1} сопряжений.
- d) Пусть $g_\alpha(x) = 1$, если $x \in D_n \alpha$, и $g_\alpha(x) = 0$, если $x \notin D_n \alpha$, и пусть

$$f_\alpha(x) = (\bar{x}_{i_1} \vee x_{j_1}) \wedge \dots \wedge (\bar{x}_{i_r} \vee x_{j_r}).$$

Докажите, что $g_\alpha(x) = \bigvee \{ f_{\alpha'}(x) \mid \alpha' \text{ есть сопряжение } \alpha \}$.

- e) Пусть G_α — направленный граф с вершинами $\{1, \dots, n\}$ и дугами $i_s \rightarrow j_s$ для $1 \leq s \leq r$. Докажите, что α является сетью сортировки тогда и только тогда, когда для всех ее сопряжений α' в $G_{\alpha'}$ имеется ориентированный путь от i до $i+1$ для $1 \leq i < n$. [Это довольно интересное условие, поскольку G_α не зависит от порядка компараторов в α .]
- >42. [25] (Д. Ван Ворис.) Докажите, что $\hat{S}(n) \geq \hat{S}(n-1) + \lceil \log_2 n \rceil$.
43. [23] Перестановочной сетью называется последовательность модулей $[i_1 : j_1] \dots [i_r : j_r]$, где каждый модуль $[i : j]$ может устанавливаться извне в одно из двух состояний: либо он передает свои входы без изменений, либо меняет местами x_i и x_j (независимо от значений x_i и x_j), и последовательность модулей должна быть такой, что на выходе можно получить любую перестановку входов при соответствующей установке модулей. Любая сеть сортировки является, очевидно, перестановочной сетью, но обратное неверно. Найдите перестановочную сеть для пяти элементов, имеющую только восемь модулей.
44. [46] Изучите свойства сетей сортировки, построенных из m -сортировщиков вместо 2-сортировщиков. (Например, Г. Шапиро построил сеть для сортировки 16 элементов, используя четырнадцать 4-сортировщиков. Наилучшее ли это решение? Существует ли для всех m эффективный способ сортировки m^2 элементов с помощью модулей, выполняющих m -сортировку?)
45. [48] Найдите, (m, n) -сеть слияния с числом компараторов, меньшим $C(m, n)$, или докажите, что такой сети не существует.

46. [48] Найдите (m, n) -сеть слияния меньше, чем с $\lceil \log_2(m+n) \rceil$ уровнями задержки, или докажите, что ее не существует.
47. [48] Изучите класс схем сортировки, которые могут быть реализованы в виде схем с идеальным тасованием, как на рис. 57, но с другим расположением операций "0", "+" и "-".
48. [BM49] Исследуйте свойства операций urur и dndn , определенных в упр. 38. Можно ли охарактеризовать все тождества в этой алгебре каким-либо изящным способом или вывести все их из конечного набора тождеств? В этом отношении такие тождества, как $x \text{urur} x \text{urur} x = x \text{urur} x$ или $x \text{urur}(x \text{dndn}(x \text{urur}(x \text{dndn} y))) = x \text{urur}(x \text{dndn} y)$, которые имеют место только для $m \leq 2$, представляют относительно небольшой интерес; рассматривайте лишь тождества, справедливые при *всех* m .
49. [M49] Каково асимптотическое поведение функции $T(n)$, определенной в упр. б? Может ли быть $T(n) < \hat{T}(n)$ при каком-нибудь n ?
50. [50] Найдите точное значение $\hat{S}(n)$ для какого-либо $n > 8$.
51. [M50] Докажите, что асимптотическое значение $\hat{S}(n)$ не есть $O(n \log n)$.

УПРАЖНЕНИЯ, (ВТОРАЯ ЧАСТЬ)

Следующие упражнения имеют дело с различными типами оптимальных задач, касающихся сортировки. Первые несколько задач основаны на "интересном" многоголовочном обобщении метода пузырька, предложенном Ф. Н. Армстронгом и Р. Дж. Нельсоном еще в 1954 г. [См. U. S. Patents 3029413, 3034102.] Пусть $1 = h_1 < h_2 < \dots < h_m = n$ — возрастающая последовательность целых чисел; будем называть ее "последовательностью головок" длины m с диапазоном n . Она будет использоваться при определении методов сортировки специального вида. Сортировка записей $R_1 \dots R_N$ осуществляется за несколько проходов, а каждый проход состоит из $N + n - 1$ шагов. На шаге j при $j = 1 - n, 2 - n, \dots, N - 1$ рассматриваются записи $R_{j+h[1]}, R_{j+h[2]}, \dots, R_{j+h[m]}$, которые в случае необходимости переставляются так, чтобы их ключи оказались упорядоченными. (Мы говорим, что $R_{j+h[1]} \dots R_{j+h[m]}$ находятся "под головками чтения-записи". Если $j + h[k]$ либо < 1 , либо $> N$, то запись $R_{j+h[k]}$ не рассматривается, иначе говоря, ключи $K_0, K_{-1}, K_{-2}, \dots$ считаются равными $-\infty$, а K_{N+1}, K_{N+2}, \dots — равными $+\infty$. Поэтому при $j \leq -h[m-1]$ или $j > N - h[2]$ шаг j тривиален.)

Например, в следующей таблице показан один проход сортировки при $m = 3, N = 9$ и $h_1 = 1, h_2 = 2, h_3 = 4$:

	K_{-2}	K_{-1}	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}
$j = -3$	—	—	—	<u>3</u>	1	4	5	9	2	6	8	7			
$j = -2$		—	—	3	<u>1</u>	4	5	9	2	6	8	7			
$j = -1$			—	<u>3</u>	1	<u>4</u>	5	9	2	6	8	7			
$j = 0$				<u>1</u>	<u>3</u>	4	<u>5</u>	9	2	6	8	7			
$j = 1$				1	<u>3</u>	<u>4</u>	5	<u>9</u>	2	6	8	7			
$j = 2$				1	3	<u>2</u>	<u>4</u>	9	<u>5</u>	6	8	7			
$j = 3$				1	3	2	<u>4</u>	<u>6</u>	5	<u>9</u>	8	7			
$j = 4$				1	3	2	4	<u>5</u>	<u>6</u>	9	<u>8</u>	7			
$j = 5$				1	3	2	4	5	<u>6</u>	<u>7</u>	8	<u>9</u>			
$j = 6$				1	3	2	4	5	6	<u>7</u>	<u>8</u>	9	—		
$j = 7$				1	3	2	4	5	6	7	<u>8</u>	<u>9</u>		—	
$j = 8$				1	3	2	4	5	6	7	8	<u>9</u>	—		—

Заметим, что, если $m = 2, h_1 = 1$ и $h_2 = 2$, этот "многоголовочный" метод сводится к методу пузырька (алгоритм 5.2.2B).

52. [21] (Джеймс Дугунди.) Докажите, что если $h[k+1] = h[k] + 1$ при некотором $k, 1 \leq k < m$, то многоголовочный сортировщик, определенный выше, отсортирует любой входной файл за конечное число проходов. Но если $h[k+1] \geq h[k] + 2$ при $1 \leq k < m$, то может случиться, что входная последовательность *никогда* не станет упорядоченной.
- >53. [50] (Армстронг и Нельсон.) Пусть $h[k+1] \leq h[k] + k$ при $1 \leq k \leq m$ и $N \geq n - 1$. Докажите, что в течение первого прохода наибольшие $n - 1$ элементов всегда займут свои окончательные места. [Указание: используйте принцип нулей и единиц; докажите, что если сортируется последовательность из нулей и единиц, причем единиц меньше n , то все головки могут читать 1 лишь в том случае, когда все нули лежат слева от головок.]

Докажите, что если головки удовлетворяют сформулированным условиям, то сортировка будет закончена не более, чем за $\lceil (N - 1)/(n - 1) \rceil$ проходов. Существует ли входной файл, для которого необходимо ровно столько проходов?

54. [26] Докажите, что при $n = N$ первый проход поместит наименьший ключ в позицию R_1 тогда и только тогда, когда $h[k+1] \leq 2h[k], 1 \leq k < m$.

55. [34](Дж. Хопкрофт.) "Совершенным сортировщиком" N элементов называется многоголовочный сортировщик, который всегда заканчивает работу за один проход. Упражнение 53 доказывает, что последовательность

$$\langle h_1, h_2, h_3, h_4, \dots, h_m \rangle = \left\langle 1, 2, 4, 7, \dots, 1 + \binom{m}{2} \right\rangle.$$

образует совершенный сортировщик для $N = \binom{m}{2}$ элементов, используя $m = (\sqrt{8N - 7} + 1)/2$ головок. Например, последовательность головок $\langle 1, 2, 4, 7, 11, 16, 22 \rangle$ является совершенным сортировщиком для 22 элементов.

Докажите, что последовательность головок $\langle 1, 2, 4, 7, 11, 16, 23 \rangle$ на самом деле будет совершенным сортировщиком для 23 элементов.

56. [49] Определите при заданном m наибольшее N , для которого существует совершенный сортировщик с m головками. Верно ли, что $N = O(m^2)$?

57. [23] (В. Пратт.) Если каждая головка h_k находится в положении 2^{k-1} для $1 \leq k \leq m$, то сколько проходов потребуется для сортировки последовательности нулей и единиц $z_1 z_2 \dots z_{2^m-1}$, где $z_j = 0$ тогда и только тогда, когда j является степенью 2?

58. [24] (Однородная сортировка.) В дереве на рис. 34 в п. 5.3.1 сравнение 2 : 3 выполняется в обоих ветвях уровня 1; а в каждой ветви уровня 2 выполняется сравнение 1 : 3, если только оно не является избыточным. В общем случае мы можем рассмотреть класс алгоритмов сортировки, однородных именно в этом смысле, предполагая, что $M = \binom{N}{2}$ пар $\{(a, b) \mid 1 \leq a < b \leq N\}$ выстроены в последовательность

$$(a_1, b_2) (a_2, b_2) \dots (a_M, b_M);$$

мы можем последовательно выполнять те из сравнений $K_{a_1} : K_{b_1}, K_{a_2} : K_{b_2}, \dots$, результат которых еще не известен. Каждая из $M!$ расстановок пар (a, b) определяет алгоритм однородной сортировки. Идея однородной сортировки принадлежит Х. Л. Бьюсу [JACM, 17 (1970), 482–495], в работе которого были предложены ближайшие несколько упражнений.

Для формального определения однородной сортировки удобно воспользоваться теорией графов. Пусть G — направленный граф с вершинами $\{1, 2, \dots, N\}$ и без дуг. Для $i = 1, 2, \dots, M$ мы добавляем дуги к G следующим образом:

Случай 1. В G имеется путь от a_i к b_i . Добавить к G дугу $a_i \rightarrow b_i$.

Случай 2. В G имеется путь от b_i к a_i . Добавить к G дугу $b_i \rightarrow a_i$.

Случай 3. В G нет пути ни от a_i к b_i , ни от b_i к a_i . Сравнить $K_{a_i} : K_{b_i}$; затем, если $K_{a_i} \leq K_{b_i}$, добавить к G дугу $a_i \rightarrow b_i$, если же $K_{a_i} > K_{b_i}$, то добавить дугу $b_i \rightarrow a_i$.

Нас интересует главным образом число сравнений ключей, выполняемых алгоритмом однородной сортировки, а не механизм, с помощью которого действительно устраняются избыточные сравнения; граф G не обязательно строить в явном виде — здесь он используется только для определения однородной сортировки.

Будем также рассматривать *ограниченную однородную сортировку*, при которой в указанных выше случаях 1–3 учитываются только пути длины 2. (Алгоритм ограниченной сортировки может выполнять некоторые избыточные сравнения, но, как показывает упр. 59, анализ ограниченного случая несколько проще.)

Докажите, что алгоритм ограниченной однородной сортировки совпадает с алгоритмом однородной сортировки, когда последовательность пар лексикографически упорядочена:

$$(1, 2) (1, 3) (1, 4) \dots (1, N) (2, 3) (2, 4) \dots (2, N) \dots (N - 1, N).$$

Покажите, что на самом деле оба алгоритма эквивалентны "быстрой сортировке" (алгоритм 5.2.2Q), если все ключи различны и избыточные сравнения быстрой сортировки устранены, как в упр. 5.2.2-24. (Не обращайте внимания на порядок, в котором действительно выполняются сравнения в быстрой сортировке; учитывайте только, какие пары ключей сравниваются.)

59. [M38] Для заданной, как в упр. 58, последовательности пар $(a_1, b_1) \dots (a_M, b_M)$ пусть c_i будет числом пар (j, k) , таких, что $j < k < i$ и $(a_i, b_i), (a_j, b_j), (a_k, b_k)$ образуют треугольник. (а) Докажите, что среднее число сравнений, выполняемых алгоритмом ограниченной однородной сортировки, равно $\sum_{1 \leq i \leq M} 2/(c_i + 2)$. (б) Используйте результат (а) и упр. 58, чтобы определить среднее число неизбыточных сравнений, выполняемых быстрой сортировкой. (с) Следующая последовательность пар навеяна сортировкой слиянием (но не эквивалентна ей):

$$(1, 2) (3, 4) (5, 6) \dots (1, 3) (1, 4) (2, 3) (2, 4) (5, 7) \dots (1, 5) (1, 6) (1, 7) (1, 8) (2, 5) \dots$$

При однородном методе, основанном на этой последовательности, будет выполняться в среднем больше или меньше сравнений, чем при быстрой сортировке?

60. [M29] Быстрая сортировка производит в наихудшем случае $\binom{N}{2}$ сравнений. Верно ли, что все алгоритмы ограниченной однородной сортировки (в смысле упр. 57) выполняют $\binom{N}{2}$ сравнений в наихудшем случае?

Picture: Рис. 61. Устройство, для которого стратегия метода пузырька является оптимальной.

61. [M48] (Х. Л. Бьюс.) Верно ли, что быстрая сортировка имеет минимальное среднее число сравнений среди всех алгоритмов (ограниченной) однородной сортировки?
62. [25] Докторская диссертация Говарда Б. Демута "Electronic Data Sorting" (Stanford University: October 1956) была, вероятно, первой работой, в которой сколько-нибудь детально рассматривались вопросы сложности вычислений. Демут рассмотрел несколько абстрактных моделей устройств для сортировки и нашел нижние и верхние оценки среднего и максимального времени выполнения, достижимого в каждой модели. Простейшая его модель—"циклическая нереверсивная память" (рис. 61)—станет темой этого упражнения.

Рассмотрим машину, которая сортирует $R_1 R_2 \dots R_N$ за ряд проходов, где каждый проход состоит из следующих $N + 1$ шагов:

Шаг 1. Установить $R \leftarrow R_1$. (R —это внутренний регистр машины.)

Шаг i . ($1 < i \leq N$): либо (а) установить $R_{i-1} \leftarrow R$, $R \leftarrow R_i$;

либо (б) установить $R_{i-1} \leftarrow R_i$, оставляя R неизменным.

Шаг $N + 1$. Установить $R_N \leftarrow R$.

Задача состоит в том, чтобы найти такой метод выбора между альтернативами (а) и (б), чтобы минимизировать число проходов, нужных для сортировки.

Докажите, что метод "пузырька" оптимален для этой модели. Другими словами, покажите, что при стратегии, которая выбирает альтернативу (а), если $R \leq R_i$, и альтернативу (б), если $R > R_i$, достигается минимальное число проходов.

5.4. ВНЕШНЯЯ СОРТИРОВКА

Пришло время заняться интересными задачами, возникающими в том случае, когда число сортируемых записей превышает объем быстродействующего оперативного запоминающего устройства. Внешняя сортировка в корне отлична от внутренней (хотя в обоих случаях необходимо расположить записи данного файла в неубывающем порядке), и объясняется это тем, что время доступа к файлам на внешних носителях нас жесточайшим образом лимитирует. Структура данных должна быть такой, чтобы сравнительно медленные периферийные запоминающие устройства (ленты, диски, барабаны и т. д.) могли справиться с потребностями алгоритма сортировки. Поэтому большинство изученных до сих пор методов внутренней сортировки (вставка, обмен, выбор) фактически бесполезно для внешней сортировки; необходимо рассмотреть всю проблему заново.

Предположим, например, что предназначенный для сортировки файл состоит из 5000 записей $R_1 R_2 \dots R_{5000}$ длиной по 20 слов (хотя ключи K_i не обязательно такой длины). Как быть, если во внутренней памяти данной машины помещается одновременно только 1000 из этих записей?

Сразу напрашивается такое решение: начать с сортировки каждого из пяти подфайлов $R_1 \dots R_{1000}$, $R_{1001} \dots R_{2000}$, ..., $R_{4001} \dots R_{5000}$ по отдельности и затем слить полученные подфайлы. К счастью, слияние оперирует только очень простыми структурами данных, именно линейными списками, пройти которые можно последовательным образом, как стеки или очереди. Поэтому для слияния годятся самые дешевые внешние запоминающие устройства.

Только что описанный процесс—внутренняя сортировка с последующим "внешним слиянием"—весьма популярен, и наше изучение внешней сортировки сведется в основном к вариациям на эту тему.

Возрастающие последовательности записей, получаемые на начальной фазе внутренней сортировки, в литературе о сортировке часто называются *цепочками*; эта терминология довольно широко распространена, но, к сожалению, она противоречит еще более распространенному использованию термина "цепочка" в других разделах вычислительной науки, где он означает *произвольную* последовательность символов. При изучении перестановок уже было дано вполне подходящее название для упорядоченных сегментов файла, которые мы договорились называть возрастающими отрезками или просто *отрезками*. В соответствии с этим будем использовать слово "отрезки" для обозначения упорядоченных частей файла. Таким образом, использование понятий "цепочки отрезков" и "отрезки цепочек" не приведет ни к каким недоразумениям.

Рассмотрим сначала процесс внешней сортировки, использующей в качестве вспомогательной памяти *магнитные ленты*. Вероятно, простейшим и наиболее привлекательным способом слияния с применением лент служит сбалансированное двухпутевое слияние, в основе которого лежит идея, использовавшаяся ранее в алгоритмах 5.2.4N, S и L. В процессе слияния нам потребуются четыре "рабочие ленты". На протяжении первой фазы возрастающие отрезки, получаемые при внутренней сортировке, помещаются поочередно на ленты 1 и 2 до тех пор, пока не исчерпаются исходные данные. Затем ленты 1 и 2 перематываем к началу и сливаем отрезки, находящиеся на этих лентах, получая новые отрезки, вдвое длиннее исходных. Эти новые отрезки записываются по мере их формирования попеременно на ленты 3 и 4. (Если на ленте 1 на один отрезок больше, чем на ленте 2, то предполагается, что лента 2 содержит дополнительный "фиктивный" отрезок длины 0.) Затем все ленты перематываются к началу и содержимое лент 3 и 4 сливается в удвоенные по длине отрезки, записываемые поочередно на ленты 1 и 2. Процесс продолжается (при этом длина отрезков каждый раз удваивается) до тех пор, пока не останется один отрезок (а именно весь упорядоченный файл). Если после внутренней сортировки было получено S отрезков, причем $2^{k-1} < S \leq 2^k$, то процедура сбалансированного двухпутевого слияния произведет ровно $k = \lceil \log_2 S \rceil$ проходов по всем данным.

Например, в рассмотренной выше ситуации, когда требуется упорядочить 5000 записей, а объем внутренней памяти составляет 1000 записей, мы имеем $S = 5$. Начальная распределительная фаза процесса сортировки поместит пять отрезков на ленты следующим образом:

$$\begin{aligned} \text{Лента 1} & R_1 \dots R_{1000}; R_{2001} \dots R_{3000}; R_{4001} \dots R_{5000}. \\ \text{Лента 2} & R_{1001} \dots R_{2000}; R_{3001} \dots R_{4000}. \\ \text{Лента 3} & \text{(пустая)} \\ \text{Лента 4} & \text{(пустая)} \end{aligned} \quad (1)$$

После первого прохода слияния на лентах 3 и 4 получатся более длинные отрезки, чем на лентах 1 и 2:

$$\begin{aligned} \text{Лента 3} & R_1 \dots R_{2000}; R_{4001} \dots R_{5000}. \\ \text{Лента 4} & R_{2001} \dots R_{4000}. \end{aligned} \quad (2)$$

В конец ленты 2 неявно добавляется фиктивный отрезок, так что отрезок $R_{4001} \dots R_{5000}$ просто копируется на ленту 3. После перематки всех лент к началу следующий проход по данным приведет к такому результату:

$$\begin{aligned} \text{Лента 1} & R_1 \dots R_{4000}. \\ \text{Лента 2} & R_{4001} \dots R_{5000}. \end{aligned} \quad (3)$$

(Отрезок $R_{4001} \dots R_{5000}$ снова копируется, но если бы мы начали с 8000 записей, то в этот момент лента 2 содержала бы $R_{4001} \dots R_{8000}$.) Наконец, после еще одной перематки на ленте 3 окажется отрезок $R_1 \dots R_{5000}$, и сортировка закончится.

Сбалансированное слияние легко обобщается на случай T лент для любого $T \geq 3$. Выберем произвольное число P , такое, что $1 \leq P < T$, и разделим T лент на два "банка": P лент в левом банке и $T - P$ лент в правом банке. Распределим исходные отрезки как можно равномернее по P лентам левого "банка", затем выполним P -путевое слияние слева направо, после этого — $(T - P)$ -путевое слияние справа налево и т. д., пока сортировка не завершится. Обычно значение P лучше всего выбирать равным $\lceil T/2 \rceil$ (см. упр. 3, 4).

При $T = 4$, $P = 2$ имеем частный случай — сбалансированное двухпутевое слияние. Вновь рассмотрим предыдущий пример, используя большее количество лент; положим $T = 6$ и $P = 3$. Начальное распределение теперь будет таким:

$$\begin{aligned} \text{Лента 1} & R_1 \dots R_{1000}; R_{3001} \dots R_{4000}. \\ \text{Лента 2} & R_{1001} \dots R_{2000}; R_{4001} \dots R_{5000}. \\ \text{Лента 3} & R_{2001} \dots R_{3000}. \end{aligned} \quad (4)$$

Первый проход слияния приведет к

$$\begin{aligned} \text{Лента 4} & R_1 \dots R_{3000}. \\ \text{Лента 5} & R_{3001} \dots R_{5000}. \\ \text{Лента 6} & \text{(пустая)} \end{aligned} \quad (5)$$

(Предполагается, что на ленте 3 помещен фиктивный отрезок.) На втором проходе слияния работа завершается и отрезки $R_1 \dots R_{5000}$ помещаются на ленту 1. Этот частный случай $T = 6$ эквивалентен $T = 5$, так как шестая лента используется лишь при $S \geq 7$.

Трехпутевое слияние затрачивает фактически несколько больше времени центрального процессора, чем двухпутевое, но оно обычно пренебрежимо мало по сравнению с временем, необходимым для чтения, записи и перемотки ленты; мы довольно хорошо оценим время выполнения сортировки, если примем во внимание только суммарную величину перемещений лент. В предыдущем примере ((4) и (5)) требуются только два прохода по данным в сравнении с тремя проходами при $T = 4$. Таким образом, слияние при $T = 6$ займет около двух третей времени по отношению к предыдущему случаю.

Сбалансированное слияние кажется очень простым и естественным. Но если приглядеться внимательнее, то сразу видно, что это не наилучший способ в разобранных выше частных случаях. Вместо того чтобы переходить от (1) к (2) и перематывать все ленты, нам следовало остановить первое слияние, когда ленты 3 и 4 содержали соответственно $R_1 \dots R_{2000}$ и $R_{2001} \dots R_{4000}$, а лента 1 была готова к считыванию $R_{4001} \dots R_{5000}$. Затем ленты 2, 3, 4 могли быть перемотаны к началу, и сортировка завершилась бы трехпутевым слиянием на ленту 2. Общее число записей, прочитанных с ленты в ходе этой процедуры, составило бы $4000 + 5000 = 9000$ против $5000 + 5000 + 5000 = 15000$ в сбалансированной схеме. Сообразительная машина могла бы постичь и это!

Имея пять отрезков и четыре ленты, можно поступить еще лучше, распределив отрезки следующим образом:

Лента 1 $R_1 \dots R_{1000}; R_{3001} \dots R_{4000}$.
 Лента 2 $R_{1001} \dots R_{2000}; R_{4001} \dots R_{5000}$.
 Лента 3 $R_{2001} \dots R_{3000}$.
 Лента 4 (пустая).

Теперь, выполнив трехпутевое слияние на ленту 4, затем перемотку лент 3 и 4 с последующим трехпутевым слиянием на ленту 3, можно было бы завершить сортировку, прочитав всего $3000 + 5000 = 8000$ записей.

Наконец, если бы мы имели шесть лент, то могли бы, конечно, записать исходные отрезки на ленты 1–5 и закончить сортировку за один проход, выполнив пятипутевое слияние на ленту 6. Рассмотрение этих случаев показывает, что простое сбалансированное слияние не является наилучшим, и было бы интересно поискать улучшенные схемы слияния.

В последующих пунктах этой главы внешняя сортировка исследуется более глубоко. В п. 5.4.1 рассматривается фаза внутренней сортировки, порождающая начальные отрезки; особый интерес представляет техника "выбора с замещением", которая использует порядок, присутствующий в большинстве данных, чтобы породить длинные отрезки, значительно превосходящие емкость внутренней памяти. В п. 5.4.1 обсуждаются также, структуры данных, удобные для целей многопутевого слияния.

Важнейшие схемы слияния обсуждаются в п. 5.4.2–5.4.5. Пока мы не вступим в единоборство с грубой действительностью настоящих лент и реальных сортируемых данных, для нас лучше, изучая характеристики этих схем, иметь весьма наивное представление о ленточной сортировке. Например, можно с легкой душой полагать (как мы делали до сих пор), что первоначальные исходные записи появляются волшебным образом в течение первой распределительной фазы; на самом деле они вероятно, будут занимать одну из наших лент и, быть может, даже целиком заполнят несколько бобин, так как лента не бесконечна! Лучше всего пренебречь подобными техническими деталями до тех пор, пока не будет достигнуто "академическое" понимание классических схем слияния. Затем в п. 5.4.6 мы "вернемся на землю", рассмотрев практические ограничения, которые сильно влияют на выбор схемы слияния. В п. 5.4.6 сравниваются основные схемы слияния из п. 5.4.2–5.4.5 с использованием множества разнообразных предположений, которые встречаются на практике.

Иные подходы к проблеме внешней сортировки, не основанные на слиянии, обсуждаются в п. 5.4.7 и 5.4.8. Наш обзор внешней сортировки заканчивается в п. 5.4.9, где рассматривается важная проблема сортировки на такой памяти, как диски и барабаны.

Упражнения

- [15] В тексте внешняя сортировка рассматривается после внутренней. Почему нельзя вообще покончить с фазой внутренней сортировки, просто сливая записи во все более и более длинные отрезки прямо с самого начала?
- [10] Каким будет содержимое лент (аналогичное (1)–(3)), если записи $R_1 \dots R_{5000}$ сортируются с помощью 3-ленточного сбалансированного метода при $P = 2$? Сравните этот случай со слиянием на 4 лентах; сколько проходов по всем данным будет сделано после первоначального распределения отрезков?

3. [20] Покажите, что сбалансированное $(P, T - P)$ -путевое слияние, примененное к S начальным отрезкам, производит $2k$ проходов, если $P^k(T - P)^{k-1} < S \leq P^k(T - P)^k$, и производит $2k + l$ проходов, если $P^k(T - P)^k < S \leq P^{k+1}(T - P)^k$.

Дайте простые формулы для вычисления (а) точного числа проходов как функции от S при $T = 2P$, (б) приближенного числа проходов при $S \rightarrow \infty$ для любых P и T .

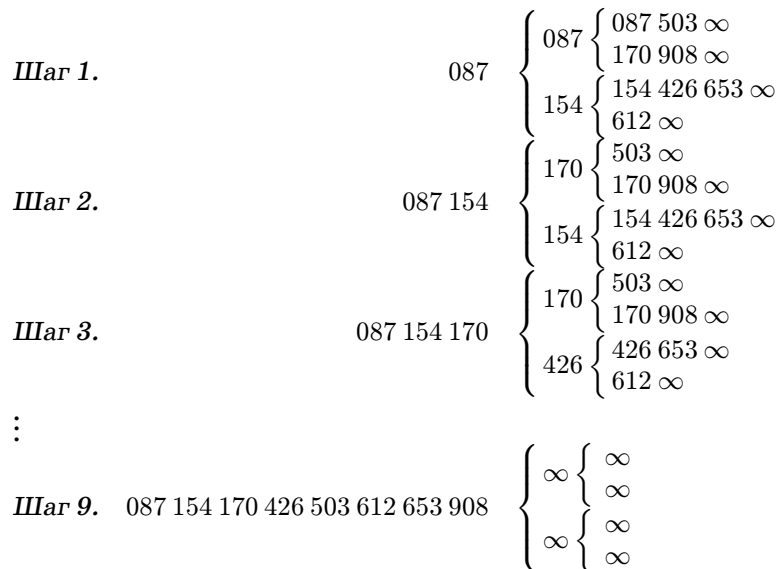
4. [BM15] При каком значении P , $1 \leq P < T$, значение $P(T - P)$ максимально?

5.4.1. Многопутевое слияние и выбор с замещением

В п. 5.2.4 мы изучали методы внутренней сортировки, основанные на двухпутевом слиянии— процессе объединения двух упорядоченных последовательностей в одну. Нетрудно расширить это до понятия P -путевого слияния, когда P входных отрезков объединяются в один выходной.

Пусть имеется P возрастающих отрезков, т. е. последовательностей записей, ключи которых расположены в неубывающем порядке. Очевидным способом их слияния будет следующий: посмотреть на первые записи каждого отрезка и выбрать из них ту, которой соответствует минимальный ключ; эта запись передается на выход и исключается из входных данных, затем процесс повторяется.

Пока P не слишком велико, этот выбор удобно осуществлять, просто выполняя $P - 1$ сравнений для нахождения наименьшего из текущих ключей. Но если, скажем, $P \geq 8$, то можно сократить работу, используя *дерево выбора*, как описано в п. 5.2.3; тогда каждый раз требуется только $\approx \log_2 P$ сравнений (после начального формирования дерева). Рассмотрим, например, четырехпутевое слияние с двухуровневым деревом выбора:



В этом примере в конце каждого отрезка помещен добавочный ключ ∞ , чтобы слияние заканчивалось естественно. Так как внешнее слияние обычно имеет дело с очень длинными отрезками, то эта добавочная запись с ключом ∞ не увеличит существенно длину данных или объем работы при слиянии; кроме того; такая "концевая" запись часто служит удобным способом разделения записей файла.

В рассматриваемом процессе каждый шаг, кроме первого, состоит из замещения наименьшего элемента следующим элементом из этого же отрезка и изменения соответствующего пути в дереве выбора. Так, на шаге 2 изменяется 3 узла, которые содержали 087 на шаге 1; на шаге 3 изменяется 3 узла, содержавшие 154 на шаге 2, и т. д. Процесс замещения в дереве выбора одного ключа другим называется *выбором с замещением*.

Мы можем по-разному рассматривать приведенное четырехпутевое слияние. С одной точки зрения оно эквивалентно трем двухпутевым слияниям, выполняемым совместно, как сопрограммы; каждый узел дерева выбора изображает одну из последовательностей, используемых в процессах слияния. Дерево выбора, по существу, используется как приоритетная очередь с дисциплиной "наименьший из включенных первым исключается".

Picture: Рис. 62. Турнир, при котором выбирается наименьший ключ; используется полное бинарное дерево, узлы которого занумерованы номерами от 1 до 23.

Так же как в п. 5.2.3, мы могли бы для реализации приоритетной очереди использовать не дерево выбора, а пирамиду. (Пирамиду, конечно, надо было бы организовать таким образом, чтобы на

вершине появлялся *наименьший* элемент, а не наибольший, обратив для этого порядок соотношения (5.2.3-3.) Так как пирамида имеет переменный размер, можно избежать использования ключа ∞ ; слияние заканчивается, когда пирамида становится пустой. С другой стороны, приложения внешней сортировки обычно имеют дело со сравнительно длинными записями и ключами, так что в пирамиду будут записаны вместо самих ключей указатели на них; мы увидим ниже, что деревья выбора могут настолько удобно изображаться с помощью указателей, что они, вероятно, в данной ситуации лучше пирамид.

Дерево "проигравших". На рис. 62 изображено полное бинарное дерево с 12 внешними (квадратными) узлами и 11 внутренними (круглыми); во внешних узлах записаны ключи, во внутренних — "победители", если дерево рассматривать как турнир для выбора

Picture: Рис. 63. Тот же турнир, что и на рис. 62, но показаны проигравшие, а не победители; чемпион находится на самом верху.

наименьшего ключа. Числа меньшего размера над каждым узлом показывают традиционный способ распределения последовательных позиций памяти для полного бинарного дерева.

Чтобы определить новое состояние дерева выбора, изображенного на рис. 62, когда наименьший ключ 061 будет заменен другим ключом, нужно рассмотреть только ключи 512, 087 и 154 и никакие другие. Если интерпретировать дерево как турнир, последние три ключа представляют собой проигравших в матчах с игроком 061. Это наводит на мысль, что мы в действительности должны записать во внутренние узлы *проигравшего* в каждом матче, а не победителя, тогда информация, необходимая для изменения дерева, будет легкодоступной.

На рис. 63 изображено то же дерево, что и на рис. 62, но вместо победителей в нем представлены проигравшие. Дополнительный узел с номером "0" добавлен на вершине дерева для указания чемпиона турнира. Заметим, что каждый ключ, кроме чемпиона, является проигравшим ровно один раз (см. п. 5.3.3); таким образом, каждый ключ появляется один раз во внешнем узле и один раз во внутреннем.

На практике внешним узлам в нижней части рис. 63 будут соответствовать весьма длинные записи, расположенные в памяти ЭВМ, а внутренним узлам — указатели на эти записи. Заметим, что P -путевое слияние требует ровно P внешних и P внутренних узлов по одному в соседних группах, что наводит на мысль о ряде эффективных методов распределения памяти. Нетрудно видеть, каким образом можно использовать дерево проигравших для выбора с замещением. Более детально мы обсудим этот алгоритм в настоящем пункте чуть позже.

Получение начальных отрезков посредством выбора с замещением. Техника выбора с замещением может использоваться также на первой фазе внешней сортировки, если фактически выполнить P -путевое слияние входных данных с самими собой. В этом случае P выбирается достаточно большим, чтобы заполнить, по существу, всю внутреннюю память. Каждая запись при выводе замещается очередной записью из исходных данных. Если у этой новой записи ключ меньше, чем у выведенной записи, то мы не включаем ее в текущий отрезок; в противном случае мы обычным образом включаем ее в дерево выбора, так что она образует часть отрезка, порождаемого в данный момент. Таким образом, каждый отрезок может содержать больше P записей, хотя в любой момент в дереве выбора находится не более P записей. Таблица 1 иллюстрирует этот процесс для $P = 4$ (числа, заключенные в скобки, ожидают включения в следующий отрезок).

Таблица 1
Пример четырехпутевого выбора с замещением

Содержимое памяти				Вывод
503	087	512	061	061
603	087	512	908	087
503	170	512	908	170
503	897	512	908	503
(275)	897	512	908	512
(275)	897	653	908	653
(275)	897	(426)	908	897
(275)	(154)	(426)	908	908
(275)	(154)	(426)	(509)	(конец отрезка)
275	154	426	509	154
275	612	426	509	275

и т.д.

Этот важный метод формирования начальных отрезков был впервые описан Гарольдом К. Сьювордом [Master's Thesis, Digital Computer Laboratory Report R-232 (Mass. Inst. of Technology, 1954), 29–30], который привел соображения в пользу того, что при применении к случайным данным отрезки, видимо, будут содержать более $1.5P$ записей. Ту же идею предложил примерно в 1950 г. А. И. Думи в связи со специальным устройством для сортировки, разрабатываемым Engineering Research Associates, но не опубликовал ее. Само название "выбор с замещением" (replacement selecting) было придумано Э. Х. Фрэндом [JACM, 3 (1956), 154], который заметил, что "ожидаемая длина порождаемой последовательности не поддается точной формулировке, но эксперименты позволяют предположить, что разумной оценкой будет $2P$ ".

Э. Ф. Мур предложил изящный способ доказательства того, что ожидаемая длина отрезка равна $2P$, проведя аналогию со

Picture: Рис. 64. Вечный снегоочиститель в своем нескончаемом цикле.

снегоочистителем, движущимся по кругу [U. S. Patent 2983904 (1961), cols. 3–4]. Рассмотрим ситуацию, изображенную на рис. 64: на кольцевую дорогу равномерно падают снежинки, а один снегоочиститель непрерывно убирает снег. Снег исчезает из системы, как только он сбрасывается за пределы дороги. Точки дороги обозначаются вещественными числами x , $0 \leq x < 1$; снежинка, падающая в точку x , соответствует входной записи с ключом x , а снегоочиститель представляет собой вывод процесса выбора с замещением.

Скорость снегоочистителя обратно пропорциональна весу снега, который встречается на его пути, и ситуация вполне уравновешена, так что общее количество снежинок на дороге в точности равно P . Каждый раз, когда снегоочиститель проходит точку 0, на выходе формируется новый отрезок.

Picture: Рис. 65. Поперечное сечение, показывающее переменный вес снега перед снегоочистителем, когда система находится в установившемся режиме.

Интуитивно ясно, что система, поработав некоторое время, выйдет на установившийся режим, при котором снегоочиститель будет двигаться с постоянной скоростью (в силу круговой симметрии дороги). Это означает, что в точке, где находится снегоочиститель, снег имеет постоянный вес, а впереди снегоочистителя этот вес линейно уменьшается, как показано на рис. 65. Отсюда следует, что количество снега, удаляемого за один оборот (а именно длина отрезка), вдвое превосходит количество снега P , присутствующего в любой момент.

Во многих коммерческих приложениях входные данные *нельзя* считать полностью случайными — в них уже существует определенный порядок; следовательно, отрезки, порождаемые выбором с замещением, имеют тенденцию содержать даже больше, чем $2P$ записей. В дальнейшем мы увидим, что время, нужное для внешней сортировки слиянием, в значительной степени зависит от количества отрезков, порождаемых начальной распределительной фазой, так что выбор с замещением становится особенно привлекательным; другие способы внутренней сортировки порождали бы примерно вдвое больше начальных отрезков, поскольку размеры памяти ограничены.

Теперь детально рассмотрим процесс создания начальных отрезков выбором с замещением. Следующий алгоритм принадлежит Джону Р. Уолтерсу, Джеймсу Пэйнтнеру и Мартину Залку, которые использовали его в программе сортировки посредством слияния, для ЭВМ Philco 2000 в 1958 г. Он включает в себя прекрасный способ начального формирования дерева выбора и разделения записей, принадлежащих разным отрезкам, а также вывода последнего отрезка по единообразной, сравнительно простой логике. (Надлежащая обработка последнего отрезка, порожденного выбором с замещением, оказывается довольно сложной; блок, осуществляющий эту обработку, часто бывает камнем преткновения для программиста.) Основная идея заключается в том, чтобы рассматривать каждый ключ как пару (S, K) , где K — первоначальный ключ, а S — номер отрезка, которому принадлежит данная запись. Мы получим выходную последовательность, порожденную выбором с замещением, если лексикографически упорядочим эти расширенные ключи (S считается старше K).

Приводимый ниже алгоритм использует для представления дерева выбора структуру данных, состоящую из P узлов; предполагается, что j -й узел $X[j]$ содержит c слов, начинающихся с $\text{LOC}(X[j]) = L_0 + cj$ при $0 \leq j < P$, он представляет как внутренний узел с номером j , так и внешний узел с номером $P + j$ (рис. 63). В каждом узле имеется несколько полей:

KEY = ключ, находящийся в данном внешнем узле;

RECORD = запись, находящаяся в данном внешнем узле (включая KEY как подполе);

LOSER = указатель на "проигравшего" в данном внутреннем узле;

RN = номер отрезка, куда включена запись, на которую указывает LOSER;

FE = указатель на внутренний узел, расположенный в дереве выбора выше данного внешнего узла;

FI = указатель на внутренний узел, расположенный в дереве выбора выше данного внутреннего узла.

Например, при $P = 12$ внутренний узел с номером 5 и внешний узел с номером 17 на рис. 63 оба будут представлены узлом $X[5]$ с полями $KEY = 170$, $LOSER = L_0 + 9c$ (адрес внешнего узла с номером 21), $FE = L_0 + 8c$, $FI = L_0 + 2c$.

Значения в полях FE и FI являются константами; таким образом, строго говоря, нет необходимости хранить их в явном виде. Однако иногда на начальной фазе внешней сортировки для быстрой работы с устройствами ввода/вывода выгоднее хранить эту избыточную информацию, чем вычислять ее каждый раз заново.

Алгоритм R. (Выбор с замещением.) Этот алгоритм читает записи последовательно из входного файла и записывает их последовательно в выходной файл, производя RMAX отрезков длины P или больше (за исключением последнего отрезка). Имеется $P \geq 2$ узлов $X[0], \dots, X[P-1]$ с полями, описанными выше.

Picture: Рис. 66. Получение начальных отрезков посредством выбора с замещением.

R1 [Начальная установка.] Установить $RMAX \leftarrow 0$, $RC \leftarrow 0$, $LASTKEY \leftarrow \infty$, $Q \leftarrow LOC(X[0])$ и $RQ \leftarrow 0$. (RC —номер текущего отрезка, а $LASTKEY$ —ключ последней выведенной записи. Начальное значение $LASTKEY$ должно быть больше любого возможного ключа; ср. с упр. 8.) Для $0 \leq j < P$ начальное содержимое $X[j]$ установить следующим образом (где $J = LOC(X[j])$):

$$\begin{aligned} LOSER(J) &\leftarrow J; RN(J) \leftarrow 0; \\ FE(J) &\leftarrow LOC(X[\lfloor (P+j)/2 \rfloor]); FI(J) \leftarrow LOC(X[\lfloor j/2 \rfloor]). \end{aligned}$$

(Установка $LOSER(J)$ и $RN(J)$ есть искусственный способ образовать начальное дерево, рассматривая фиктивный отрезок с номером 0, который никогда не выводится. Это—некий трюк; см. упр. 10.)

R2 [Конец отрезка?] Если $RQ = RC$, то перейти к шагу **R3**. (В противном случае $RQ = RC + 1$, и мы только что закончили отрезок с номером RC ; в этом месте следует выполнить те специальные действия, которые нужны в соответствии со схемой слияния для последующих этапов сортировки.) Если $RQ > RMAX$, то алгоритм завершен; в противном случае установить $RC \leftarrow RQ$.

R3 [Вывод вершины дерева.] (Сейчас Q указывает на "чемпиона", и RQ —номер его отрезка.) Если $RQ \neq 0$, то вывести $RECORD(Q)$ и установить $LASTKEY \leftarrow KEY(Q)$.

R4 [Ввод новой записи.] Если входной файл исчерпан, установить $RQ \leftarrow RMAX + 1$ и перейти к шагу **R5**. В противном случае поместить новую запись из входного файла в $RECORD(Q)$. Если $KEY(Q) < LASTKEY$ (т. е. эта запись не принадлежит текущему отрезку), то $RQ \leftarrow RQ + 1$, и теперь, если $RQ > RMAX$, установить $RMAX \leftarrow RQ$.

R5 [Подготовка к изменению.] (Сейчас Q указывает на новую запись с номером отрезка RQ .) Установить $T \leftarrow FE(Q)$. (T —переменный указатель, который будет двигаться по дереву.)

R6 [Установка нового проигравшего.] Если $RN(T) < RQ$ или если $RN(T) = RQ$ и $KEY(LOSER(T)) < KEY(Q)$, то поменять местами $LOSER(T) \leftrightarrow Q$, $RN(T) \leftrightarrow RQ$. (В переменных Q и RQ запоминается текущий победитель и номер его отрезка.)

R7 [Сдвиг вверх.] Если $T = LOC(X[1])$, то вернуться к шагу **R2**, в противном случае $T \leftarrow FI(T)$ и вернуться к **R6**. ■

В алгоритме R говорится о вводе и выводе записей по одной, тогда как практически оказывается лучше читать и записывать относительно большие блоки записей. Следовательно, на самом деле за кулисами прячутся буферы ввода и вывода; их присутствие в памяти приводит к уменьшению значения P . Это будет пояснено в п. 5.4.6.

Э. Х. Фрэнд [JACM, bf 3 (1956), 154] предложил следующее обобщение метода выбора с замещением. В тех случаях, когда вводимый ключ меньше, чем $LASTKEY$ (так что он не попадет в текущий отрезок), но больше или равен последнему ключу, действительно записанному на ленту (так что его фактически можно было бы поместить в текущий отрезок), вставлять этот ключ внутрь буфера вывода. Кроме того, некоторые ЭВМ умеют выполнять "чтение вразброс" и "запись со сборкой", т. е. вводить информацию во внутреннюю память не обязательно в последовательные ячейки, а "вразброс" и выводить ее, собирая из разных мест. Это позволяет совмещать память для буферов с памятью для дерева выбора.

***Преобразование отрезков с задержкой.** Р. Дж. Динсмор [САСМ, 8 (1965), 48] предложил интересное усовершенствование выбора с замещением, использующее понятие, которое будем называть *степенью свободы*. Как мы видели, каждый блок записей, находящийся на ленте в составе отрезка, содержит записи в неубывающем порядке, так что первый элемент наименьший, а последний наибольший. В обычном процессе выбора с замещением наименьший элемент каждого блока в некотором отрезке всегда не меньше, чем наибольший элемент в предыдущем блоке этого отрезка; это соответствует "1 степени свободы". Динсмор предложил ослабить это условие до " m степеней свободы"; новое условие не требует, чтобы наименьший элемент каждого блока был не меньше, чем наибольший элемент предыдущего блока, но он *не должен быть меньше, чем наибольшие элементы каких-то m предыдущих блоков того же отрезка*. Записи в отдельном блоке упорядочены, как и ранее, но соседние блоки не обязаны быть взаимно упорядоченными.

Предположим, например, что блоки содержат только по две записи; следующая последовательность блоков является отрезком с тремя степенями свободы:

$$|08\ 50|06\ 90|17\ 27|42\ 67|51\ 89| \quad (1)$$

Следующий блок, который может быть частью этого отрезка, должен начинаться с элемента, не меньшего, чем третий по порядку элемент множества $\{50, 90, 27, 67, 89\}$, считая от наибольшего, т. е. не меньше 67. Последовательность (1) не является отрезком с двумя степенями свободы, так как 17 меньше, чем 50 и 90.

Отрезок с m степенями свободы в процессе чтения в следующей фазе сортировки может быть преобразован таким образом, что для всех практических целей он будет отрезком в обычном смысле. Начнем с чтения первых m блоков в m буферов и будем производить m -путевое слияние их; когда один из буферов исчерпается, поместим в него $(m + 1)$ -й блок и т. д. Таким образом, мы можем восстановить отрезок в виде одной упорядоченной последовательности, так как первое слово каждого вновь считываемого блока должно быть больше или равно последнему слову только что исчерпанного блока (если оно не было меньше, чем наибольшие элементы каких-либо m блоков, предшествующих ему). Этот метод преобразования отрезка, в сущности, является m -путевым слиянием, использующим только одно ленточное устройство для всех входных блоков!

Процедура преобразования действует как сопрограмма, к которой обращаются каждый раз, когда нужно получить одну очередную запись отрезка. Мы можем преобразовывать различные отрезки с различных ленточных устройств и с различными степенями свободы и сливать получающиеся отрезки—все в одно и то же время. Это, по существу, подобно тому, как если бы мы четырехпутевое слияние, рассмотренное в начале этого пункта, представили себе как несколько двухпутевых слияний, происходящих одновременно.

Эта остроумная идея до сих пор не проанализирована до конца. Имеются некоторые предварительные результаты, показывающие, что, когда P велико по сравнению с размером блока, длина отрезка при $m = 2$ приблизительно равна $2.1P$, она равна $2.3P$ при $m = 4$ и $2.5P$ при $m = 8$. Такое увеличение, быть может, недостаточно, чтобы оправдать усложнение алгоритма. С другой стороны, метод может оказаться выгодным, если на протяжении второго этапа сортировки есть место для довольно большого числа буферов.

***Натуральный выбор.** Другой путь увеличения длины отрезков, порождаемых выбором с замещением, был исследован У. Д. Фрэйзером и Ч. К. Уоном. Их идея состоит в том, чтобы следовать алгоритму R, но, когда на шаге $R4\ KEY(Q) < LASTKEY$, новая запись $RECORD(Q)$ не остается в дереве, а выводится в некоторый внешний *резервуар* и читается новая запись. Этот процесс продолжается до тех пор, пока в резервуаре не окажется определенное количество записей P' ; тогда остаток текущего отрезка выводится из дерева, и элементы резервуара используются в качестве исходных данных для следующего отрезка.

Этот метод должен породить более длинные отрезки, чем выбор с замещением, поскольку он "обходит" вновь поступающие "мертвые" записи, вместо того чтобы позволять им заполнять дерево; но ему требуется дополнительное время на обмен с резервуаром. Когда $P' > P$, некоторые записи могут оказываться в резервуаре дважды, но при $P' \leq P$ такого случиться не может.

Фрэйзер и Уон, проведя обширные эмпирические испытания своего метода, заметили, что, когда P достаточно велико (скажем, $P \geq 32$) и $P' = P$, средняя длина отрезка для случайных данных оказывается равной eP , где $e \approx 2.718$ —основание натуральных логарифмов. Это явление, а также тот факт, что метод был получен как эволюционное развитие простого выбора с замещением, послужили для них непосредственным основанием назвать свой метод *натуральным выбором*.

Можно доказать "натуральный" закон для длины отрезка, вновь воспользовавшись аналогией со снегоочистителем на рис. 64 и применив элементарный математический анализ. Пусть L обозначает длину пути, а $x(t)$ —положение снегоочистителя в момент t при $0 \leq t \leq T$. Предположим, что в

момент T резервуар заполняется; в этот момент падение снега временно прекращается, пока снегоочиститель возвращается в исходное положение (счищая P снежинок, оставшихся на его пути). Ситуация такая же, как и ранее, только "условия равновесия" другие—вместо P снежинок на всей дороге в любой момент времени мы имеем P снежинок перед снегоочистителем и резервуар (за снегоочистителем), заполняющийся до уровня в P снежинок. В течение интервала времени dt снегоочиститель продвигается на dx , если выводятся $h(x, t)dx$ записей, где $h(x, t)$ —толщина слоя снега в момент времени t в точке $x = x(t)$, измеряемая в соответствующих единицах; следовательно, $h(x, t) = h(x, 0) + Kt$ для всех x . Так как число записей в памяти остается постоянным, то $h(x, t)dx$ есть также число записей, вводимых *перед* снегоочистителем, а именно $K dt(L - x)$, где K —скорость падения снега (рис. 67). Таким образом,

$$\frac{dx}{dt} = \frac{K(L - x)}{h(x, t)}. \tag{2}$$

К счастью, оказывается, что $h(x, t)$ —константа, и она равна KT при всех $x = x(t)$ и $0 \leq t \leq T$, так как снег падает равномерно в точку $x(t)$ в течение $T - t$ единиц времени после того, как снегоочиститель проходит эту точку, плюс t единиц времени перед тем, как он вернется. Иными словами, снегоочиститель видит перед собой все время одинаковый слой снега на протяжении всего пути, если допустить, что достигнут установившийся режим, когда этот путь все время один и тот же. Следовательно, общее количество счищаемого снега (длина отрезка) есть $KT L$,

Picture: Рис. 67. Вводится и выводится равное количество снега; за время dt снегоочиститель перемещается на dx .

а количество снега в памяти есть количество снега, счищаемого после момента T , а именно $KT(L - x(T))$.

Решением уравнения (2) при условии, что $x(0) = 0$, будет

$$x(t) = L(1 - e^{-t/T}). \tag{3}$$

Следовательно, $P = KTL e^{-1} = (\text{длина отрезка})/e$ —это как раз то, что мы и хотели доказать.

В упр. 21–23 показано, что этот анализ можно распространить на случай произвольного P' ; например, когда $P' = 2P$, средняя длина отрезка оказывается равной $e^\theta(e - \theta)P$, где $\theta = \frac{1}{2}(e - \sqrt{e^2 - 4})$,—результат, который вряд ли можно было предположить заранее! В табл. 2 приводится зависимость между длиной отрезка и размером резервуара; с помощью этой таблицы можно оценить полезность натурального выбора для конкретной машины в той или иной ситуации.

***Анализ выбора с замещением.** Вернемся теперь к случаю выбора с замещением без вспомогательного резервуара. Аналогия со снегоочистителем дает довольно хорошую оценку средней длины отрезков, получаемых при выборе с замещением "в пределе", тем не менее можно получить значительно более точную информацию об алгоритме R, применяя факты об отрезках в перестановках, изученных нами в п. 5.1.3. Для удобства будем считать, что входной файл является последовательностью (произвольной длины) независимых случайных действительных чисел, расположенных между 0 и 1.

Таблица 2

Размер резервуара	Длина отрезков при натуральном выборе		
	Длина отрезка	Параметр	Размер резервуара
1.00000P	2.71828P	1.00000	0.38629P
2.00000P	3.53487P	1.43867	1.30432P
3.00000P	4.16220P	1.74773	2.72294P
4.00000P	4.69445P	2.01212	4.63853P
5.00000P	5.16369P	2.24038	21.72222P
10.00000P	7.00877P	3.17122	5.29143P

"Параметр" $k + \theta$ определен в упр. 22

Пусть

$$g_P(z_1, z_2, \dots, z_k) = \sum_{l_1, l_2, \dots, l_k \geq 0} a_P(l_1, l_2, \dots, l_k) z_1^{l_1} z_2^{l_2} \dots z_k^{l_k}$$

—производящая функция для длины отрезка, полученного при P -путевом выборе с замещением, примененном к такому файлу, где $a_P(l_1, l_2, \dots, l_k)$ есть вероятность того, что первый отрезок имеет длину l_1 , второй—длину l_2 , ..., k -й имеет длину l_k . Будем основываться на следующей "теореме независимости", Так как она сводит наш анализ к случаю $P = 1$.

Теорема К. $g_P(z_1, z_2, \dots, z_k) = g_1(z_1, z_2, \dots, z_k)^P$.

Доказательство. Пусть исходные ключи суть X_1, X_2, X_3, \dots . Алгоритм **R** разделяет их на P подпоследовательностей в соответствии с тем, в какой внешний узел дерева они попадают; подпоследовательность, содержащая X_n , определяется значениями X_1, \dots, X_{n-1} . Таким образом, эти подпоследовательности являются независимыми последовательностями независимых случайных чисел, расположенных между 0 и 1. Кроме того, выход выбора с замещением в точности совпадает с результатом P -путевого слияния, если его произвести над этими подпоследовательностями; некоторый элемент принадлежит j -му отрезку подпоследовательности тогда и только тогда, когда он принадлежит j -му отрезку, полученному при выборе с замещением (так как на шаге **R4** ключи **LASTKEY** и **KEY(Q)** принадлежат одной подпоследовательности).

Иначе говоря, можно считать, что алгоритм **R** применяется к P случайным независимым исходным файлам и что шаг **R4** читает следующую запись из файла, соответствующего внешнему узлу Q ; в этом смысле рассматриваемый алгоритм эквивалентен P -путевому слиянию, где концы отрезков отмечаются убыванием элементов.

Таким образом, на выходе будут отрезки длин (l_1, \dots, l_k) тогда и только тогда, когда подпоследовательности состоят из отрезков длин $(l_{11}, \dots, l_{1k}), \dots, (l_{P1}, \dots, l_{Pk})$ соответственно; где l_{ij} — некоторые неотрицательные целые числа, удовлетворяющие соотношению $\sum_{1 \leq i \leq P} l_{ij} = l_j$ при $1 \leq j \leq k$. Отсюда следует, что

$$a_P(l_1, \dots, l_k) = \sum_{\substack{l_{11} + \dots + l_{P1} = l_1 \\ \vdots \\ l_{1k} + \dots + l_{Pk} = l_k}} a_1(l_{11}, \dots, l_{1k}) \dots a_1(l_{P1}, \dots, l_{Pk}),$$

что эквивалентно искомому результату. ■

В п. 5.1.3 мы изучили среднее значение L_k — длины k -го отрезка при $P = 1$ (эти значения приведены в табл. 5.1.3-2). Из теоремы К следует, что средняя длина k -го отрезка при любом P в P раз больше средней длины при $P = 1$, она равна $L_k P$; дисперсия также в P раз больше, так что стандартное отклонение длины отрезка пропорционально \sqrt{P} . Эти результаты были впервые получены Б. Дж. Гэсснер около 1958 г.

Таким образом, первый отрезок, полученный для случайных данных алгоритмом **R**, будет иметь длину, приближенно равную $(e - 1)P \approx 1.718P$ записей, второй — приближенно $(e^2 - 2e)P \approx 1.952P$, третий — $1.996P$; длина следующих отрезков будет очень близка к $2P$, пока мы не дойдем до последних двух отрезков (см. упр. 14). Стандартное отклонение длины большинства этих отрезков приближенно равно $\sqrt{(4e - 10)P} \approx 0.934\sqrt{P}$ [*SACM*, 6 (1963), 685–687].

Кроме этого, согласно упр. 5.1.3-10, суммарная длина первых k отрезков будет довольно близка к $(2k - \frac{1}{3})P$ со стандартным отклонением $((\frac{2}{3}k + \frac{2}{9})P)^{1/2}$. Производящие функции $g_1(z, z, \dots, z)$ и $g_1(1, \dots, 1, z)$ выводятся в упр. 5.1.3-9 и 11.

В приведенном выше анализе предполагалось, что исходный файл бесконечно длинный, но доказательство теоремы К показывает, что точно такая же вероятность $a_P(l_1, \dots, l_k)$ получилась бы в случае любой случайной исходной последовательности, содержащей по крайней мере $l_1 + \dots + l_k + P$ элементов. Следовательно, полученные результаты применимы для файла размера, скажем, $N > (2K + 1)P$ в силу малой величины стандартного отклонения.

Мы познакомимся с рядом применений, в которых схема слияния требует, чтобы некоторые отрезки были возрастающими, а некоторые — убывающими. Поскольку остаток, накапливающийся в памяти у конца возрастающего отрезка, имеет тенденцию содержать числа, в среднем меньшие, чем случайные данные, то изменение направления упорядочения уменьшает среднюю длину отрезков. Рассмотрим, например, снегоочиститель, который должен выполнять разворот каждый раз, как он достигает конца прямой дороги; он будет очень быстро передвигаться по только что очищенному участку. В случае изменяемого направления длина отрезков для случайных данных изменяется между $1.5P$ и $2P$ (см. упр. 24).

Упражнения

- [10] Каким будет шаг 4 в примере четырехпутевого слияния в начале этого пункта?
- [12] Какие изменения произошли бы в дереве рис. 63, если бы ключ 061 был заменен ключом 612?
- [16] (Э. Ф. Мур.) Что получится в результате применения четырехпутевого выбора с замещением к последовательным словам следующего предложения:¹¹

¹¹ Восемьдесят семь лет тому назад наши предки основали на этом континенте новую нацию, посвятившую себя делу свободы и убежденную в том, что все люди созданы равными. — Прим. перев.

fourscore and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicated to the proposition that all men are created equal.

(Используйте обычный алфавитный порядок, рассматривая каждое слово как один ключ.)

4. [16] Примените четырехпутевый *натуральный* выбор к предложению из упр. 3, используя резервуар емкости 4.
5. [00] Верно ли, что выбор с замещением, использующий дерево, работает, только если P есть степень двойки или сумма двух степеней двойки?
6. [15] В алгоритме R указывается, что P должно быть ≥ 2 ; какие относительно небольшие изменения надо сделать в этом алгоритме, чтобы он правильно работал для всех $P \geq 1$?
7. [17] Что делает алгоритм R в случае отсутствия исходной информации?
8. [20] Алгоритм R использует искусственный ключ " ∞ ", который должен быть больше любого возможного ключа. Покажите, что если бы какой-нибудь реальный ключ оказался равным ∞ , то алгоритм мог бы ошибиться, и объясните, как изменить алгоритм в случае, когда реализация "настоящей" бесконечности неудобна.
- >9. [23] Как вы изменили бы алгоритм R , чтобы он выводил некоторые заданные отрезки (определяемые RC) в возрастающем порядке, а другие в убывающем?
10. [26] Начальная установка указателей $LOSER$ на шаге $R1$ обычно не соответствует никакому действительному турниру, так как внешний узел $P + j$ может не лежать в поддереве с вершиной во внутреннем узле j . Объясните, почему алгоритм R все равно работает. [Указание. Будет ли работать алгоритм R , если множеству $\{LOSER(LOC(X[0])), \dots, LOSER(LOC(X[P-1]))\}$ присваивается на шаге $R1$ произвольная перестановка множества $\{LOC(X[0]), \dots, LOC(X[P-1])\}$?
11. [M25] Верно ли, что для случайных исходных данных вероятность того, что $KEY(Q) < LASTKEY$ на шаге $R4$, приближенно равна $1/2$?
12. [M46] Проведите детальное исследование того, сколько раз выполняется каждая часть алгоритма R ; например, как часто выполняется перестановка на шаге $R6$?
13. [13] Почему второй отрезок, полученный при выборе с замещением, обычно длиннее первого?
- >14. [BM25] Используйте аналогию со снегоочистителем, чтобы оценить среднюю длину двух *последних* отрезков, полученных при выборе с замещением, примененном к длинной последовательности исходных данных.
15. [20] Верно ли, что последний отрезок, полученный при выборе с замещением, никогда не содержит более P записей? Обсудите ваш ответ.
16. [M26] Найдите "простое" необходимое и достаточное условие того, что файл $R_1 R_2 \dots R_N$ будет полностью упорядочен за один проход P -путевого выбора с замещением. Какова вероятность этого события как функция P и N , если исходными данными служит случайная перестановка множества $\{1, 2, \dots, N\}$?
17. [20] Что получается в результате работы алгоритма R , когда исходные ключи представляют собой невозрастающую последовательность $K_1 \geq K_2 \geq \dots \geq K_N$?
- >18. [22] Что произойдет, если вновь применить алгоритм R к файлу, полученному в результате работы алгоритма R ?
19. [BM22] Используйте аналогию со снегоочистителем, чтобы доказать, что первый отрезок, полученный при выборе с замещением, имеет длину примерно $(e-1)P$ записей.
20. [BM24] Какую примерно длину имеет первый отрезок, полученный при натуральном выборе, когда $P = P'$?
- >21. [BM23] Определите приблизительную длину отрезков, полученных посредством натурального выбора при $P' < P$.
22. [BM40] Целью этого упражнения является определение средней длины отрезков, получаемых при натуральном выборе при $P' > P$. Пусть $\kappa = k + \theta$ — действительное число ≥ 1 , где $k = \lfloor \kappa \rfloor$, а $\theta = \kappa \bmod 1$, и рассмотрим функцию $F(\kappa) = F_k(\theta)$, где $F_k(\theta)$ — полиномы, определяемые производящей функцией

$$\sum_{k \geq 0} F_k(\theta) z^k = e^{-\theta z} / (1 - z e^{1-z}).$$

Таким образом, $F_0(\theta) = 1$, $F_1(\theta) = e - \theta$, $F_2(\theta) = e^2 - e - e\theta + \frac{1}{2}\theta^2$ и т. д.

Предположим, что в момент $t = 0$ снегоочиститель начинает моделировать процесс натурального выбора, и допустим, что за T единиц времени позади него упадут ровно P снежинок. В этот момент второй снегоочиститель начинает тот же путь, занимая в момент времени $t + T$ то же положение, что занимал первый снегоочиститель в момент t . В конце концов, к моменту κT позади первого снегоочистителя упадут ровно P' снежинок; он мгновенно очищает остаток дороги и исчезает.

Используя эту модель для интерпретации натурального выбора, покажите, что длина отрезка $e^\theta F(\kappa)P$ получается при

$$P'/P = k + 1 + e^\theta \left(\kappa F(\kappa) - \sum_{0 \leq j \leq \kappa} F(\kappa - j) \right).$$

23. [BM35] Предыдущее упражнение анализирует натуральный выбор в том случае, когда записи из резервуара всегда читаются в том же порядке, в котором они записывались: "первым включается—первым исключается". Оцените длину отрезков, которая получилась бы, если бы содержимое резервуара, оставшееся от предыдущего отрезка, читалось в совершенно случайном порядке, как если бы записи в резервуаре тщательно перемешивались между отрезками.
24. [BM39] Цель этого упражнения—анализ последствий, вызванных случайным изменением направления упорядочения отрезков в выборе с замещением.

- а) Пусть $g_P(z_1, z_2, \dots, z_k)$ —та же производящая функция, что и в теореме К, но для каждого из k отрезков определено, является ли он возрастающим или убывающим. Например, мы можем считать, что все отрезки с нечетными номерами возрастающие, а с четными убывающие. Покажите, что теорема К справедлива для каждой из 2^k производящих функций такого вида.
- б) В силу (а) можно считать $P = 1$. Можно также предположить, что исходной является равномерно распределенная последовательность независимых случайных величин, заключенных между 0 и 1. Пусть

$$a(x, y) = \begin{cases} e^{1-x} - e^{y-x}, & \text{если } x \leq y; \\ e^{1-x}, & \text{если } x > y. \end{cases}$$

Пусть $f(x) dx$ —вероятность того, что определенный возрастающий отрезок начинается с x . Докажите, что $\left(\int_0^1 a(x, y) f(x) dx \right) dy$ есть вероятность того, что следующий отрезок начинается с y . [Указание: рассмотрите для каждого $n \geq 0$ вероятность того, что $x \leq X_1 \leq \dots \leq X_n > y$ при данных x и y .]

- с) Рассмотрите отрезки, меняющие направление упорядочения с вероятностью p , другими словами, направление каждого отрезка, кроме первого, совпадает с направлением предыдущего отрезка с вероятностью $q = 1 - p$ и противоположно ему с вероятностью p . (Таким образом, если $p = 0$, то все отрезки имеют одинаковое направление; если $p = 1$, направление отрезков чередуется, а при $p = 1/2$ отрезки случайные и независимые) Пусть

$$f_1(x) = 1, \quad f_{n+1}(y) = p \int_0^1 a(x, y) f_n(1-x) dx + q \int_0^1 a(x, y) f_n(x) dx.$$

Покажите, что вероятность того, что n -й отрезок начинается с x , есть $f_n(x) dx$, если $(n-1)$ -й отрезок возрастающий, и $f_n(1-x) dx$, если $(n-1)$ -й отрезок убывающий.

- д) Найдите решение f для уравнения "установившегося режима"

$$f(y) = p \int_0^1 a(x, y) f(1-x) dx + q \int_0^1 a(x, y) f(x) dx, \quad \int_0^1 f(x) dx = 1.$$

[Указание: покажите, что $f''(x)$ не зависит от x .]

- е) Покажите, что последовательность $f''(x)$ части (с) весьма быстро сходится к функции $f(x)$ части (д).
- ф) Покажите, что средняя длина возрастающего отрезка, начинающегося с x , равна e^{1-x} .
- г) Наконец, объедините все предыдущие результаты и докажите следующую теорему. Если направления последовательных отрезков при выборе с замещением независимо изменяются на противоположные с вероятностью p , то средняя длина отрезка стремится к $(6/(3+p))P$. (Эта теорема при $p = 1$ впервые была доказана Кнуттом [САСМ, 6 (1963), 685–688]; при $p = 1/2$ ее доказал Э. Г. Конхейм в 1978 г.)

25. [BM40] Рассмотрите следующую процедуру.

- N1. Прочитать запись, поместив ее в резервуар емкостью в одно слово. Затем прочитать следующую запись R, и пусть K будет ее ключом.
- N2. Вывести содержимое резервуара, установить LASTKEY равным его ключу и опустошить резервуар.
- N3. Если $K < \text{LASTKEY}$, то вывести R, установить $\text{LASTKEY} \leftarrow K$ и перейти к N5.

№4. Если резервуар не пуст, вернуться к №2; в противном случае поместить R в резервуар.

№5. Прочитать новую запись R и установить K равным ее ключу. Перейти к №3. ■

Эта процедура, в сущности, эквивалентна натуральному выбору с $P = 1$ и $P' = 1$ или $P' = 2$ (в зависимости от того, в какой момент мы опустошаем резервуар—как только он заполнится или когда нам надо будет записать в заполненный резервуар новый элемент, переполняющий его), за исключением того, что эта процедура порождает *убывающие* отрезки и никогда не останавливается. Эти отклонения не приносят вреда, они удобны для нашей цели.

Действуя, как в упр. 24, обозначим через $f_n(x, y) dy dx$ вероятность того, что (x, y) суть значения (LASTKEY, K) соответственно сразу же после n -го выполнения шага №2. Докажите, что существует функция $g_n(x)$ от одной переменной, такая, что $f_n(x, y) = g_n(x)$, если $x < y$, и $f_n(x, y) = g_n(x) - e^{-y}(g_n(x) - g_n(y))$, если $x > y$. Функция $g_n(x)$ определяется соотношениями $g_1(x) = 1$,

$$g_{n+1}(x) = \int_0^x e^u g_n(u) du + \int_0^x dv (v+1) \int_v^1 du ((e^v - 1)g_n(u) + g_n(v)) + x \int_x^1 dv \int_v^1 du ((e^v - 1)g_n(u) + g_n(v)).$$

Покажите далее, что ожидаемая длина n -го отрезка равна

$$\int_0^1 dx \int_0^x dy (g_n(x)(e^y - 1) + g_n(y)) \left(2 - \frac{1}{2}y^2\right) + \int_0^1 dx (1-x)g_n(x)e^x.$$

[Замечание. Решение этого уравнения в установившемся режиме оказывается очень сложным; оно было численно найдено Дж. Мак-Кенной. Он показал, что длина отрезка стремится к предельному значению 2.61307209. Теорема K не применима к натуральному выбору, так что случай $P = 1$ нельзя распространить на другие P .]

26. [M33] Рассматривая алгоритм упр. 25 как определение натурального выбора для $P' = 1$, найдите среднюю длину *первого* отрезка для $P' = r$ при любом $r \geq 0$ по следующей схеме:

а) Покажите, что первый отрезок имеет длину n с вероятностью

$$(n+r) \binom{n+r}{n} / (n+r+1)!$$

б) Определим "числа Стирлинга второго порядка" $\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\}$ правилами

$$\left\{ \begin{smallmatrix} 0 \\ m \end{smallmatrix} \right\} = \delta_{m0}, \quad \left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = (n+m-1) \left(\left\{ \begin{smallmatrix} n-1 \\ m \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n-1 \\ m-1 \end{smallmatrix} \right\} \right) \quad \text{при } n > 0.$$

Докажите, что

$$\left[\begin{smallmatrix} n+r \\ n \end{smallmatrix} \right] = \sum_{0 \leq k \leq r} \binom{n+r}{k+r} \left\{ \begin{smallmatrix} r \\ k \end{smallmatrix} \right\}.$$

в) Докажите, что средняя длина первого отрезка будет, следовательно, $c_r e^{-r} - 1$, где

$$c_r = \sum_{0 \leq k \leq r} \left[\left[\begin{smallmatrix} r \\ k \end{smallmatrix} \right] \right] (r+k+1)/(r+k)!.$$

27. [25] В тексте рассматривается только случай сортировки записей фиксированного размера. Как разумным образом приспособить выбор с замещением к записям *переменной длины*?

5.4.2. Многофазное слияние

Теперь, после того как мы выяснили, как можно образовать начальные отрезки, рассмотрим различные методы распределения отрезков по лентам и слияния их до тех пор, пока не получится единственный отрезок.

Предположим сначала, что в нашем распоряжении имеются три ленточных устройства: $T1$, $T2$ и $T3$; можно воспользоваться сбалансированным слиянием, описанным в начале § 5.4, для $P = 2$ и $T = 3$. Оно принимает следующий вид:

V1. Распределить начальные отрезки попеременно на ленты $T1$ и $T2$.

V2. Слить отрезки с лент $T1$ и $T2$ на $T3$; затем остановиться, если $T3$ содержит только один отрезок.

V3. Скопировать отрезки с $T3$ попеременно на $T1$ и $T2$, затем вернуться к шагу V2. ■

Если начальное распределение дало 5 отрезков, то первый проход слияния приведет к $\lceil S/2 \rceil$ отрезкам на T_3 , второй—к $\lceil S/4 \rceil$ и т. д. Таким образом, если, скажем, $17 \leq S \leq 32$, то произойдет 1 проход распределения, 5 проходов слияния и 4 прохода копирования; в общем случае при $S > 1$ число проходов по всем данным будет равно $2\lceil \log_2 S \rceil$.

Проходы копирования в этой процедуре нежелательны, так как они не уменьшают числа отрезков. Можно обойтись половиной копирований, если использовать *двухфазную* процедуру:

- A1. Распределить начальные отрезки попеременно на ленты T_1 и T_2 .
- A2. Слить отрезки с лент T_1 и T_2 на T_3 ; остановиться, если T_3 содержит только один отрезок.
- A3. Скопировать *половину* отрезков с T_3 на T_1 .
- A4. Слить отрезки с лент T_1 и T_3 на T_2 ; остановиться, если T_2 содержит только один отрезок.
- A5. Скопировать *половину* отрезков с T_2 на T_1 . Вернуться к шагу A2. ■

Число проходов по всем данным сократилось до $\frac{3}{2}\lceil \log_2 S \rceil + \frac{1}{2}$, так как в шагах A3 и A5 выполняется только "половина прохода", т. е. сэкономлено около 25% времени.

В действительности можно даже полностью устранить копирование, если начать с F_n отрезков на ленте T_1 и с F_{n-1} отрезков на T_2 , где F_n и F_{n-1} —последовательные числа Фибоначчи. Рассмотрим, например, случай $n = 7$, $S = F_n + F_{n-1} = 13 + 8 = 21$:

	Содержимое T_1	Содержимое T_2	Содержимое T_3	Примечания
Фаза 1.	1,1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1, 1,1		Начальное распределение
Фаза 2.	1,1,1,1,1	—	2,2,2,2,2, 2,2,2	Слияние 8 отрезков на T_3
Фаза 3.	—	3, 3, 3, 3, 3	2,2,2	Слияние 5 отрезков на T_3
Фаза 4.	5,5,5	3, 3	—	Слияние 3 отрезков на T_1
Фаза 5.	5	—	8,8	Слияние 2 отрезков на T_3
Фаза 6.	—	13	8	Слияние 1 отрезка на T_2
Фаза 7.	21	—	—	Слияние 1 отрезка на T_1

Здесь "2, 2, 2, 2, 2, 2, 2, 2", например, обозначает восемь отрезков относительной длины 2, если считать относительную длину каждого начального отрезка равной 1. Всюду в этой таблице числа Фибоначчи!

Полный проход по данным осуществляют только фазы 1 и 7; фаза 2 обрабатывает лишь 16/21 общего числа начальных отрезков, фаза 3—лишь 15/21 и т. д.; таким образом, суммарное число "проходов" равно $(21 + 16 + 15 + 15 + 16 + 13 + 21)/21 = 5\frac{4}{7}$, если предположить, что начальные отрезки имеют примерно равную длину. Для сравнения заметим, что рассмотренная выше двухфазная процедура затратила бы 8 проходов на сортировку этих же начальных отрезков. Мы увидим, что в общем случае эта схема Фибоначчи требует приблизительно $1.04 \log_2 S + 0.99$ проходов, что делает ее сравнимой с *четырёхленточным* сбалансированным слиянием, хотя она использует только три ленты.

Эту идею можно обобщить на случай T лент при любом $T \geq 3$, используя $(T-1)$ -путевое слияние. Мы увидим, например, что в случае четырех лент требуется только около $0.703 \log_2 S + 0.96$ проходов по данным. Обобщенная схема использует обобщенные числа Фибоначчи. Рассмотрим следующий пример с шестью лентами:

	Число обрабатываемых начальных отрезков					
	T_1	T_2	T_3	T_4	T_5	T_6
Фаза 1.	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	—
Фаза 2.	1^{15}	1^{14}	1^{12}	1^8	—	5^{16}
Фаза 3.	1^7	1^6	1^4	—	9^8	5^8
Фаза 4.	1^3	1^2	—	17^4	9^4	5^4
Фаза 5.	1^1	—	33^2	17^2	9^2	5^2
Фаза 6.	—	65^1	33^1	17^1	9^1	5^1
Фаза 7.	129^1	—	—	—	—	—

Здесь 1^{31} обозначает 31 отрезок относительной длины 1 и т. д.; везде используется пятипутевое слияние. Эта общая схема была разработана Р. Л. Гилстэдом [Proc. AFIPS Eastern Jt. Computer Conf., 18 (1960), 143–148], который назвал ее многофазным слиянием. Случай трех лент был ранее открыт Б. К. Бетцем [неопубликованная заметка, Minneapolis-Honeywell Regulator Co. (1956)].

Чтобы заставить многофазное слияние работать, как в предыдущем примере, необходимо после каждой фазы иметь "точное фибоначчьево распределение" отрезков по лентам. Читая приведенную выше таблицу снизу вверх, можно заметить, что первые семь точных фибоначчьевых распределений при $T = 6$ суть $\{1, 0, 0, 0, 0\}$, $\{1, 1, 1, 1, 1\}$, $\{2, 2, 2, 2, 1\}$, $\{4, 4, 4, 3, 2\}$, $\{8, 8, 7, 6, 4\}$, $\{16, 15, 14, 12, 8\}$ и $\{31, 30, 28, 24, 16\}$. Теперь перед нами стоят следующие важные вопросы:

- 1) Какое правило скрыто за этими точными фибоначчиевыми распределениями?
- 2) Что делать, если S не соответствует точному фибоначчиевому распределению?
- 3) Как построить начальный проход распределения, чтобы он порождал нужное расположение отрезков на лентах?
- 4) Сколько "проходов" по данным потребует T -ленточное многофазное слияние (как функция от S —числа начальных отрезков)?

Мы обсудим эти четыре вопроса по очереди, при этом сначала дадим "простые ответы", а затем займемся более глубоким анализом.

Точные фибоначчиевы распределения можно получить, "прокручивая" рассмотренную схему в обратную сторону, циклически переставляя содержимое лент. Например, при $T = 6$ имеем следующее распределение отрезков:

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$	Сумма	Лента с окончательным результатом
0	1	0	0	0	0	1	$T1$
1	1	1	1	1	1	5	$T6$
2	2	2	2	2	1	9	$T5$
3	4	4	4	3	2	17	$T4$
4	8	8	7	6	4	33	$T3$
5	16	15	14	12	8	65	$T2$
6	31	30	28	24	16	129	$T1$
7	61	59	55	47	31	253	$T6$
8	120	116	108	92	61	497	$T5$

$$\begin{array}{cccccccc}
 \dots\dots\dots & & & & & & & \\
 n & a_n & b_n & c_n & d_n & e_n & t_n & T(k) \\
 n+1 & a_n + b_n & a_n + c_n & a_n + d_n & a_n + e_n & a_n & t_n + 4a_n & T(k-1) \\
 \dots\dots\dots & & & & & & &
 \end{array} \tag{1}$$

(После начального распределения лента $T6$ всегда будет пустой.)

Из правила перехода от уровня n к уровню $n + 1$ ясно, что условия

$$a_n \geq b_n \geq c_n \geq d_n \geq e_n \tag{2}$$

выполняются на любом уровне. В самом деле, легко видеть из (1), что

$$\begin{aligned}
 e_n &= a_{n-1}, \\
 d_n &= a_{n-1} + e_{n-1} = a_{n-1} + a_{n-2}, \\
 c_n &= a_{n-1} + d_{n-1} = a_{n-1} + a_{n-2} + a_{n-3}, \\
 b_n &= a_{n-1} + c_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4}, \\
 a_n &= a_{n-1} + b_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} + a_{n-5},
 \end{aligned} \tag{3}$$

где $a_0 = 1$ и где мы полагаем $a_n = 0$ при $n = -1, -2, -3, -4$. Числа Фибоначчи p -го порядка $F_n^{(p)}$ определяются правилами

$$\begin{aligned}
 F_n^{(p)} &= F_{n-1}^{(p)} + F_{n-2}^{(p)} + \dots + F_{n-p}^{(p)} && \text{при } n \geq p; \\
 F_n^{(p)} &= 0 && \text{при } 0 \leq n \leq p - 2; \\
 F_{p-1}^{(p)} &= 1.
 \end{aligned} \tag{4}$$

Другими словами, мы начинаем с $p - 1$ нулей, затем пишем 1, а каждое следующее число является суммой p предыдущих чисел. При $p = 2$ это обычная последовательность Фибоначчи; для больших значений p эту последовательность впервые изучил, по-видимому, В. Шлегель в [*El Progreso Matematico*, 4 (1894), 173–174]. Шлегель вывел производящую функцию

$$\sum_{n \geq 0} F_n^{(p)} z^n = \frac{z^{p-1}}{1 - z - z^2 - \dots - z^p} = \frac{z^{p-1} - z^p}{1 - 2z + z^{p+1}}. \tag{5}$$

Формула (3) показывает, что число отрезков на $T1$ в процессе шестиленточного многофазного слияния является числом Фибоначчи пятого порядка $a_n = F_{n+4}^{(5)}$.

В общем случае, если положить $P = T - 1$, распределения в многофазном слиянии для T лент будут аналогичным образом соответствовать числам Фибоначчи P -го порядка. В точном распределении n -го уровня на k -й ленте будет

$$F_{n+P-2}^{(P)} + F_{n+P-3}^{(P)} + \dots + F_{n+k-2}^{(P)}$$

начальных отрезков для $1 \leq k \leq P$, а общее количество начальных отрезков на всех лентах будет, следовательно, равно

$$t_n = PF_{n+P-2}^{(P)} + (P-1)F_{n+P-3}^{(P)} + \dots + F_{n-1}^{(P)}. \quad (6)$$

Это решает вопрос о "точном фибоначчиевом распределении". Но что мы должны делать, если S не равно в точности t_n ни при каком n ? Как первоначально поместить отрезки на ленты?

Если S не является точным числом Фибоначчи (а чисел Фибоначчи не так уж много), то можно действовать, как в сбалансированном P -путевом слиянии, добавляя "фиктивные

Picture: Рис. 68. Сортировка многофазным слиянием.

отрезки"; поэтому можно считать, что S , в конце концов, будет точным. Есть несколько способов добавления фиктивных отрезков; мы еще не знаем, как это сделать "наилучшим" способом. В первую очередь рассмотрим метод распределения отрезков и приписывания фиктивных отрезков, который хотя и не самый оптимальный, но зато достаточно простой и, по-видимому, лучше всех других методов такой же степени сложности.

Алгоритм D. (Сортировка многофазным слиянием с использованием "горизонтального" распределения.) Этот алгоритм берет начальные отрезки и распределяет их один за другим по лентам, пока запас начальных отрезков не исчерпается. Затем он определяет, как надо сливать ленты, используя P -путевое слияние, в предположении, что имеются $T = P + 1 \geq 3$ лентопротяжных устройств. Ленту T можно использовать для хранения ввода, так как на нее не записывается ни одного начального отрезка. В памяти хранятся следующие таблицы:

$A[j], 1 \leq j \leq T$: Точное фибоначчиевое распределение, к которому мы стремимся.
 $D[j], 1 \leq j \leq T$: Число фиктивных отрезков, которые считаются присутствующими в начале ленты на логическом устройстве с номером j .
 $TARE[j], 1 \leq j \leq T$: Номер физического лентопротяжного устройства, соответствующего логическому устройству с номером j .

(Оказалось, что удобно работать с "номерами логических лентопротяжных устройств", соответствие которых физическим устройствам меняется в процессе выполнения алгоритма.)

- D1** [Начальная установка.] Установить $A[j] \leftarrow D[j] \leftarrow 1$ и $TARE[j] \leftarrow j$ при $1 \leq j < T$. Установить $A[T] \leftarrow D[T] \leftarrow 0$ и $TARE[T] \leftarrow T$. Затем установить $l \leftarrow 1, j \leftarrow 1$.
- D2** [Ввод на ленту j .] Записать один отрезок на ленту j и уменьшить $D[j]$ на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу **D5**.
- D3** [Продвижение j .] Если $D[j] < D[j+1]$, то увеличить j на 1 и вернуться к шагу **D2**. В противном случае, если $D[j] = 0$, перейти к шагу **D4**, а если $D[j] \neq 0$, установить $j \leftarrow 1$ и вернуться к шагу **D2**.
- D4** [Подняться на один уровень.] Установить $l \leftarrow l + 1, a \leftarrow A[1]$, затем для $j = 1, 2, \dots, P$ (именно в этом порядке) установить $D[j] \leftarrow a + A[j+1] - A[j]$ и $A[j] \leftarrow a + A[j+1]$. (См. (1). Отметим, что $A[P+1]$ всегда 0. В этом месте будем иметь $D[1] > D[2] > \dots > D[T]$.) Теперь установить $j \leftarrow 1$ и вернуться к шагу **D2**.
- D5** [Слияние.] Если $l = 0$, то сортировка завершена, результат находится на $TARE[1]$. В противном случае сливать отрезки с лент $TARE[1], \dots, TARE[P]$ на $TARE[T]$ до тех пор, пока $TARE[P]$ не станет пустой и $D[P]$ не обратится в 0. Процесс слияния для каждого сливаемого отрезка должен протекать следующим образом. Если $D[j] > 0$ для всех $j, 1 \leq j \leq P$, то увеличить $D[T]$ на 1 и уменьшить каждое $D[j]$ на 1 для $1 \leq j \leq P$; в противном случае сливать по одному отрезку с каждой ленты $TARE[j]$, такой, что $D[j] = 0$, и уменьшить $D[j]$ на 1 для остальных j . (Таким образом, считается, что фиктивные отрезки находятся в начале ленты, а не в конце.)
- D6** [Опустить на один уровень.] Установить $l \leftarrow l - 1$. Перемотать ленты $TARE[P]$ и $TARE[T]$. (В действительности перемотка $TARE[P]$ могла быть начата на шаге **D5** после ввода с нее последнего блока.) Затем установить $(TARE[1], TARE[2], \dots, TARE[T]) \leftarrow (TARE[T], TARE[1], \dots, TARE[T-1])$, $(D[1], D[2], \dots, D[T]) \leftarrow (D[T], D[1], \dots, D[T-1])$ и вернуться к шагу **D5**. ■

Правило распределения, которое так лаконично сформулировано в шаге D3 этого алгоритма, стремится по возможности уравнивать числа фиктивных отрезков на каждой ленте. Рисунок 69 иллюстрирует порядок распределения, когда мы переходим от уровня 4 (33 отрезка) к уровню 5 (65 отрезков) в сортировке с шестью лентами; если было бы, скажем, только 53 начальных

Picture: Рис. 69. Порядок, в котором отрезки с 34-го по 65-й распределяются на ленты при переходе с уровня 4 на уровень 5. (См. таблицу точных распределений на стр. 320.) Заштрихованные области соответствуют первым 33 отрезкам, которые были распределены к моменту достижения уровня 4.

отрезка, то все отрезки с номерами 54 и выше рассматривались бы как фиктивные. (На самом деле отрезки записываются в конец ленты, но удобнее считать, что они записываются в начало, так как предполагается, что фиктивные отрезки находятся в начале ленты.)

Мы уже рассмотрели первые три из поставленных выше вопросов, осталось выяснить число "проходов" по данным. Сравнивая наш шестиленточный пример с таблицей (1), мы видим, что суммарное количество обработанных начальных отрезков при $S = t_6$ есть $a_5t_1 + a_4t_2 + a_3t_3 + a_2t_4 + a_1t_5 + a_0t_6$, если исключить начальный проход распределения. В упр. 4 выводятся производящие функции

$$\begin{aligned}
 a(z) &= \sum_{n \geq 0} a_n z^n = \frac{1}{1 - z - z^2 - z^3 - z^4 - z^5}, \\
 t(z) &= \sum_{n \geq 1} t_n z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{1 - z - z^2 - z^3 - z^4 - z^5}.
 \end{aligned}
 \tag{7}$$

Отсюда следует, что в общем случае число обрабатываемых начальных отрезков при $S = t_n$ равно коэффициенту при z^n в произведении $a(z) \cdot t(z)$ плюс t_n (это дает начальный проход распределения). Теперь мы можем вычислить асимптотическое поведение многофазного слияния, как показано в упр. 5-7, и получаем результаты, приведенные в табл. 1.

В табл. 1 "отношение роста" есть предел $\lim_{n \rightarrow \infty} t_{n+1}/t_n$, показывающий, во сколько приблизительно раз возрастает число

Таблица 1

Ленты	Аппроксимация поведения сортировки многофазным слиянием			
	Фазы	Проходы	Проходы/фазы,	Отношение
в процентах роста				
3	$2.078 \ln S + 0.678$	$1.504 \ln S + 0.992$	72	1.6180340
4	$1.641 \ln S + 0.364$	$1.015 \ln S + 0.965$	62	1.8392868
5	$1.524 \ln S + 0.078$	$0.863 \ln S + 0.921$	57	1.9275620
6	$1.479 \ln S - 0.185$	$0.795 \ln S + 0.864$	54	1.9659482
7	$1.460 \ln S - 0.424$	$0.762 \ln S + 0.797$	52	1.9835826
8	$1.451 \ln S - 0.642$	$0.744 \ln S + 0.723$	51	1.9919642
9	$1.447 \ln S - 0.838$	$0.734 \ln S + 0.646$	51	1.9960312
10	$1.445 \ln S - 1.017$	$0.728 \ln S + 0.568$	50	1.9980295
20	$1.443 \ln S - 2.170$	$0.721 \ln S - 0.030$	50	1.9999981

отрезков на каждом уровне. "Проходы" обозначают среднее количество обработок каждой записи, а именно $(1/S)$, умноженное на общее число начальных отрезков, обрабатываемых в течение фаз распределения и слияния. Установленные числа проходов и фаз справедливы в каждом случае с точностью до $O(S^{-\epsilon})$ при некотором $\epsilon > 0$ для точного распределения при $S \rightarrow \infty$.

На рис. 70 изображены в виде функций от S средние количества слияний каждой записи при использовании алгоритма D в случае неточных чисел. Заметим, что при использовании трех лент как раз после точных распределений появляются "пики" относительной неэффективности, но это явление в значительной степени исчезает при четырех или большем числе лент. Использование восьми или более лент дает сравнительно малое улучшение по сравнению с шестью или семью лентами.

***Более детальное рассмотрение.** В сбалансированном слиянии, требующем k проходов, каждая запись обрабатывается в ходе сортировки ровно k раз. Но многофазная процедура не является такой беспристрастной: некоторые записи могут обрабатываться

Picture: Рис. 70. Эффективность многофазного слияния, использующего алгоритм D.

много большее число раз, чем другие, и мы можем увеличить скорость, если условимся помещать фиктивные отрезки в часто обрабатываемые позиции.

По этой причине изучим более подробно многофазное распределение. Вместо того чтобы интересоваться только числом отрезков на каждой ленте, как в (1), припишем каждому отрезку его *число слияний*—сколько раз он обрабатывается в течение всего процесса сортировки. Вместо (1) получим следующую таблицу:

Уровень	T1	T2	T3	T4	T5
0	0	—	—	—	—
1	1	1	1	1	1
2	21	21	21	21	2
3	3221	3221	3221	322	32
4	43323221	43323221	4332322	433232	4332
5	5443433243323221	544343324332322	54434332433232	544343324332	54434332
.....					
n	A_n	B_n	C_n	D_n	E_n
$n + 1$	$(A_n + 1)B_n$	$(A_n + 1)C_n$	$(A_n + 1)D_n$	$(A_n + 1)E_n$	$A_n + 1$
.....					

Здесь A_n есть цепочка из a_n значений, представляющих числа слияний каждого отрезка на T1, если мы начинаем с распределения n -го уровня; B_n есть соответствующая цепочка для T2 и т. д. Обозначение " $(A_n + 1)B_n$ " читается: " A_n , все значения которой увеличены на 1, а за нею B_n ".

Рисунок 71 (а), на котором "снизу вверх" изображены A_5, B_5, C_5, D_5, E_5 , демонстрирует, каким образом числа слияний для каждого отрезка появляются на ленте; заметим, что отрезок в начале любой ленты будет обрабатываться 5 раз, в то время как отрезок в конце T1 будет обрабатываться лишь однажды.

Picture: Рис. 71. Анализ многофазного распределения пятого уровня на шести лентах: (а)—числа слияний; (б)—оптимальный порядок распределения.

Эта "дискриминация" при многофазном слиянии приводит к тому, что фиктивные отрезки лучше помещать в начало ленты, а не в конец. На рис. 71 (б) представлен оптимальный порядок распределения отрезков для случая пятиуровневого многофазного слияния; каждый новый отрезок помещается в позицию с наименьшим из оставшихся числом слияний. Заметим, что алгоритм D (рис. 69) несколько хуже, так как он заполняет некоторые позиции "4" до того, как заполнены все позиции "3". Рекуррентные соотношения (8) показывают, что все B_n, C_n, D_n, E_n являются начальными подцепочками A_n . В действительности, используя (8), можно вывести формулы

$$\begin{aligned}
 E_n &= (A_{n-1}) + 1, \\
 D_n &= (A_{n-1}A_{n-2}) + 1, \\
 C_n &= (A_{n-1}A_{n-2}A_{n-3}) + 1, \\
 B_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}) + 1, \\
 A_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}A_{n-5}) + 1,
 \end{aligned}
 \tag{9}$$

обобщающие соотношения (3), которые имеют дело только с длинами этих цепочек. Кроме того, из правил, определяющих цепочки A , следует, что структура в начале каждого уровня, в сущности, одна и та же; имеем

$$A_n = n - Q_n, \tag{10}$$

где Q_n есть цепочка из a_n значений, определяемая законом

$$\begin{aligned}
 Q_n &= Q_{n-1}(Q_{n-2} + 1)(Q_{n-3} + 2)(Q_{n-4} + 3)(Q_{n-5} + 4) \quad \text{при } n \geq 1, \\
 Q_0 &= '0'; Q_n = (\text{пусто}) \quad \text{при } n < 0.
 \end{aligned}
 \tag{11}$$

Так как Q_n начинается с Q_{n-1} , то можно рассмотреть *бесконечную* цепочку Q_∞ , первые a_n элементов которой совпадают с Q_n ; эта цепочка, по существу, описывает все числа слияний в многофазном распределении. В случае шести лент имеем

$$Q_\infty = 011212231223233412232334233434412232334233434452334344534454512232 \dots \tag{12}$$

В упр. 11 содержится интересная интерпретация этой цепочки.

При условии что A_n есть цепочка $m_1 m_2 \dots m_{a_n}$, обозначим через $A_n(x) = x^{m_1} + x^{m_2} + \dots + x^{m_{a_n}}$ соответствующую производящую функцию, описывающую, сколько раз появляется каждое число

слияний; аналогично введем $B_n(x), C_n(x), D_n(x), E_n(x)$. Например, $A_4(x) = x^4 + x^3 + x^3 + x^2 + x^3 + x^2 + x^2 + x = x^4 + 3x^3 + 3x^2 + x$. В силу соотношений (9) имеем при $n \geq 1$

$$\begin{aligned} E_n(x) &= x(A_{n-1}(x)), \\ D_n(x) &= x(A_{n-1}(x) + A_{n-2}(x)), \\ C_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x)), \\ B_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x)), \\ A_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x) + A_{n-5}(x)), \end{aligned} \tag{13}$$

где $A_0(x) = 1$ и $A_n(x) = 0$ при $n = -1, -2, -3, -4$. Следовательно,

$$\sum_{n \geq 0} A_n(x)z^n = \frac{1}{1 - x(z + z^2 + z^3 + z^4 + z^5)} = \sum_{k \geq 0} x^k (z + z^2 + z^3 + z^4 + z^5)^k. \tag{14}$$

Рассматривая отрезки на всех лентах, положим

$$T_n(x) = A_n(x) + B_n(x) + C_n(x) + D_n(x) + E_n(x), \quad n \geq 1; \tag{15}$$

из (13) немедленно получаем

$$T_n(x) = 5A_{n-1}(x) + 4A_{n-2}(x) + 3A_{n-3}(x) + 2A_{n-4}(x) + A_{n-5}(x),$$

а значит, и

$$\sum_{n \geq 1} T_n(x)z^n = \frac{x(5z + 4z^2 + 3z^3 + 2z^4 + z^5)}{1 - x(z + z^2 + z^3 + z^4 + z^5)}. \tag{16}$$

Соотношение (16) показывает, что легко вычислить коэффициенты $T_n(x)$:

	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}
x	5	4	3	2	1	0	0	0	0	0	0	0	0	0
x^2	0	5	9	12	14	15	10	6	3	1	0	0	0	0
x^3	0	0	5	14	26	40	55	60	57	48	35	20	10	4
x^4	0	0	0	5	19	45	85	140	195	238	260	255	220	170
x^5	0	0	0	0	5	24	69	154	294	484	703	918	1088	1168

Столбцы этой таблицы дают $T_n(x)$; например, $T_4(x) = 2x + 12x^2 + 14x^3 + 5x^4$. Каждый элемент этой таблицы (кроме элементов первой строки) является суммой пяти элементов, расположенных в предыдущей строке непосредственно левее него.

Число отрезков в точном распределении n -го уровня равно $T_n(1)$, а общее количество обрабатываемых отрезков в процессе их слияния равно производной $T'_n(1)$. Далее,

$$\sum_{n \geq 1} T'_n(x)z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{(1 - x(z + z^2 + z^3 + z^4 + z^5))^2}; \tag{18}$$

полагая $x = 1$ в (16) и (18), получаем, что число слияний для точного распределения n -го уровня есть коэффициент при z^n в $a(z)t(z)$ [ср. (7)]. Это согласуется с нашими предыдущими рассуждениями.

Функции $T_n(x)$ можно использовать для определения совершаемой работы, когда фиктивные отрезки добавляются оптимальным образом. Пусть $\sum_n(m)$ есть сумма наименьших m чисел слияний в распределении n -го уровня. Посмотрев на столбцы (17), мы без труда вычислим эти суммы $\sum_n(m)$:

$m =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$n = 1$	1	2	3	4	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n = 2$	1	2	3	4	6	8	10	12	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n = 3$	1	2	3	5	7	9	11	13	15	17	19	21	24	27	30	33	36	∞	∞	∞	∞
$n = 4$	1	2	4	6	8	10	12	14	16	18	20	22	24	26	29	32	35	38	41	44	47
$n = 5$	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	32	35	38	41	44	47
$n = 6$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	33	36	38	42	45	48
$n = 7$	2	4	6	8	10	12	14	16	18	20	23	26	29	32	35	38	41	44	47	50	53

Например, если мы хотим отсортировать 17 отрезков, используя распределение 3-го уровня, то общее количество операций есть $\sum_3(17) = 36$, но если использовать распределение 4-го или 5-го уровня, то

общее количество операций в процессе слияния будет только $\sum_4(17) = \sum_5(17) = 35$. Выгоднее использовать уровень 4, хотя число 17 соответствует точному распределению 3-го уровня! В самом деле, по мере возрастания S оптимальный номер уровня оказывается значительно больше, чем используемый в алгоритме D.

Упражнение 14 показывает, что существует неубывающая последовательность чисел M_n , такая, что уровень n оптимален для $M_n \leq S < M_{n+1}$, но не для $S \geq M_{n+1}$. В случае шести лент только что вычисленная таблица $\sum_n(m)$ дает

$$M_0 = 0, \quad M_1 = 2, \quad M_2 = 6, \quad M_3 = 10, \quad M_4 = 14.$$

Выше мы имели дело только со случаем шести лент, однако ясно, что те же идеи применимы к многофазной сортировке с T лентами для любого $T \geq 3$; просто в соответствующих местах надо заменить 5 на $P = T - 1$. В табл. 2 изображены последовательности M_n , полученные для различных значений T . Таблица 3 и рис. 72 дают представление об общем количестве обрабатываемых начальных отрезков после выполнения оптимального распределения фиктивных отрезков. (Формулы внизу табл. 3 следует принимать с осторожностью, так как это приближение по методу наименьших квадратов на области $1 \leq S \leq 5000$ ($1 \leq S \leq 10\,000$ при $T = 3$), что приводит к некоторому отклонению, поскольку данная область значений S не является одинаково подходящей для всех T . При $S \rightarrow \infty$ число обрабатываемых начальных отрезков после оптимального многофазного распределения асимптотически равно $S \log_p S$, но сходимость к этому асимптотическому пределу крайне медленная.)

При помощи табл. 4 можно сравнить метод распределения алгоритма D с результатами оптимального распределения, приведенными в табл. 3. Ясно, что алгоритм D не очень близок к оптимальному при больших S и T ; однако непонятно, можно ли поступить в этих случаях существенно лучше алгоритма D, не прибегая к значительным усложнениям, особенно если мы не знаем S заранее. К счастью, заботиться о больших S приходится довольно редко (см. п. 5.4.6), так что алгоритм D на практике не так уж плох, на самом деле — даже весьма неплох.

Математически многофазная сортировка впервые была проанализирована У. К. Картером [Proc. IFIP Congress (1962), 62–66].

Picture: Рис. 72. Эффективность многофазного слияния с оптимальным начальным распределением (ср. с рис. 70).

Многие из приведенных результатов относительно оптимального размещения фиктивных отрезков принадлежат Б. Сэкману и Т. Синглеру [A vector model for merge sort analysis, неопубликованный доклад, представленный на симпозиум ACM по сортировке (ноябрь 1962), стр. 21]. Позднее Сэкман предложил горизонтальный метод распределения, используемый в алгоритме D; Дональд Шелл [JACM, 14 (1971), 713–719; 15 (1972), 28], независимо развив эту теорию, указал на соотношение (10) и подробно изучил несколько различных алгоритмов распределения. Дальнейшие полезные усовершенствования и упрощения были получены Дерекотом Э. Зэйвом [JACM, будет опубликовано].

Таблица 2

Уровень	Число отрезков, при котором данный уровень оптимален								
	$T = 3$	$T = 4$	$T = 5$	$T = 6$	$T = 7$	$T = 8$	$T = 9$	$T = 10$	
1	2	2	2	2	2	2	2	2	M_1
2	3	4	5	6	7	8	9	10	M_2
3	4	6	8	10	12	14	16	18	M_3
4	6	10	14	14	17	20	23	26	M_4
5	9	18	23	29	20	24	28	32	M_5
6	14	32	35	43	53	27	32	37	M_6
7	22	55	76	61	73	88	35	41	M_7
8	35	96	109	194	98	115	136	44	M_8
9	56	173	244	216	283	148	171	199	M_9
10	90	280	359	269	386	168	213	243	M_{10}
11	145	535	456	779	481	640	240	295	M_{11}
12	234	820	1197	1034	555	792	1002	330	M_{12}
13	378	1635	1563	1249	1996	922	1228	1499	M_{13}
14	611	2401	4034	3910	2486	1017	1432	1818	M_{14}
15	988	4959	5379	4970	2901	4397	1598	2116	M_{15}
16	1598	7029	6456	5841	10578	5251	1713	2374	M_{16}
17	2574	14953	18561	19409	13097	5979	8683	2576	M_{17}
18	3955	20583	22876	23918	15336	6499	10069	2709	M_{18}
19	6528	44899	64189	27557	17029	30164	11259	15787	M_{19}

Таблица 3

Число начальных отрезков, обрабатываемых при оптимальном многофазном слиянии

S	$T = 3$	$T = 4$	$T = 5$	$T = 6$	$T = 7$	$T = 8$	$T = 9$	$T = 10$
10	36	24	19	17	15	14	13	12
20	90	60	49	44	38	36	34	33
50	294	194	158	135	128	121	113	104
100	702	454	362	325	285	271	263	254
1000	10371	6680	5430	4672	4347	3872	3739	3632
5000	63578	41286	32905	28620	26426	23880	23114	22073
S	$\left\{ \begin{array}{l} (1.51 \ 0.951 \ 0.761 \ 0.656 \ 0.589 \ 0.548 \ 0.539 \ 0.488) \cdot S \ln S \\ +(-.11 \ +.14 \ +.16 \ +.19 \ +.21 \ +.20 \ +.02 \ +.18) \cdot S \end{array} \right.$							

Производящая функция (16) была впервые исследована У. Буржем [Proc. IFIP Congress (1971), I, 454–459].

А как обстоит дело с временем перемотки? До сих пор мы использовали ”обрабатываемые начальные отрезки” как единственную меру эффективности для сравнения стратегий ленточного слияния. Но после каждой из фаз 2–6 в примерах в начале этого пункта ЭВМ должна ожидать перемотки двух лент; как предыдущая выводная лента, так и новая текущая выводная лента должны быть перемотаны в начало, прежде чем сможет выполняться следующая фаза. Это может вызвать существенную задержку, так как в общем случае предыдущая выводная лента содержит значительный процент сортируемых записей (см. столбец ”проходы/фазы” в табл. 1). Досадно, когда ЭВМ простаивает во время операций перемотки, тогда как можно было бы, используя иную схему слияния, выполнить полезную работу с остальными лентами.

Таблица 4

Число начальных отрезков, обрабатываемых при стандартном многофазном слиянии

S	$T = 3$	$T = 4$	$T = 5$	$T = 6$	$T = 7$	$T = 8$	$T = 9$	$T = 10$
10	36	24	19	17	15	14	13	12
20	90	62	49	44	41	37	34	33
50	294	194	167	143	134	131	120	114
100	714	459	393	339	319	312	292	277
1000	10730	6920	5774	5370	4913	4716	4597	4552
5000	64740	43210	36497	32781	31442	29533	28817	28080

Эту задачу решает простая модификация многофазной процедуры, хотя она требует не менее пяти лент [см. диссертацию И. Сезари (Univ. of Paris (1968), 25–27), где эта идея приписывается Дж. Кэйрону]. Каждая фаза схемы Кэйрона сливает отрезки с $(T - 3)$ лент на некоторую другую ленту, в то время как остающиеся две ленты перематываются.

Рассмотрим, например, случай шести лент и 49 начальных отрезков. В следующей таблице буквой R помечены ленты, перематывающиеся во время данной фазы; предполагается, что $T5$ содержит первоначальные отрезки:

Фаза	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	Время записи	Время перемотки
1	1^{11}	1^{17}	1^{13}	1^8	—	(R)	49	17
2	(R)	1^9	1^5	—	R	3^8	$8 \times 3 = 24$	$49 - 17 = 32$
3	1^6	1^4	—	R	3^5	R	$5 \times 3 = 15$	$\max(8, 24)$
4	1^2	—	R	5^4	R	3^4	$4 \times 5 = 20$	$\max(13, 15)$
5	—	R	7^2	R	3^3	3^2	$2 \times 7 = 14$	$\max(17, 20)$
6	R	11^2	R	5^2	3^1	—	$2 \times 11 = 22$	$\max(11, 14)$
7	15^1	R	7^1	5^1	—	R	$1 \times 15 = 15$	$\max(22, 24)$
8	R	11^1	7^0	—	R	23^1	$1 \times 23 = 23$	$\max(15, 16)$
9	15^1	11^1	—	R	33^0	R	$0 \times 33 = 0$	$\max(20, 23)$
10	(15^0)	—	R	49^1	(R)	(23^0)	$1 \times 49 = 49$	14

Здесь все перемотки, по существу, совмещены; за исключением фазы 9 (”фиктивная фаза”, которая подготавливает окончательное слияние) и перемотки после начальной фазы распределения (когда перематываются все ленты). Если t есть время, необходимое для слияния такого количества записей, какое содержится в одном начальном отрезке, а r — время перемотки на один начальный отрезок, то этот процесс тратит около $182t + 40r$ плюс время начального распределения и завершающей перемотки.

Соответствующие выражения для стандартного многофазного метода, использующего алгоритм D, есть $140t + 104r$, что несколько хуже, если $r = \frac{3}{4}t$, и несколько лучше, если $r = \frac{1}{2}t$.

Все сказанное о стандартном многофазном методе приложимо к многофазному методу Кэйрона; например, последовательность a_n теперь удовлетворяет рекуррентному соотношению

$$a_n = a_{n-2} + a_{n-3} + a_{n-4} \quad (20)$$

вместо (3). Читателю будет полезно проанализировать этот метод таким же образом, как мы анализировали стандартный многофазный, так как это улучшит понимание обоих методов (см., например, упр. 19 и 20).

В табл. 5 сведены факты о многофазном методе Кэйрона, аналогичные фактам об обычном многофазном методе, приведенным в табл. 1. Заметим, что на самом деле при восьми и более лентах метод Кэйрона становится *лучше* многофазного как по числу обрабатываемых отрезков, так и по времени перемотки, несмотря на то что он выполняет $(T - 3)$ -путевое слияние вместо $(T - 1)$ -путевого!

Это может казаться, парадоксальным, пока мы не поймем, что *высокий порядок слияния не обязательно означает эффективную сортировку*. В качестве крайнего рассмотрим случай, когда на ленту T1 помещается 1 отрезок и n отрезков — на T2, T3, T4, T5; если мы будем поочередно выполнять слияния на T6 и T1, пока T2, T3, T4, T5 не станут пустыми, то время обработки будет равно $(2n^2 + 3n)$ длинам начальных отрезков, т. е., по существу, пропорционально S^2 , а не $S \log S$, хотя все время производится пятипутевое слияние.

Таблица 5

Приблизительное поведение многофазного слияния Кэйрона

Ленты	Фазы	Проходы	Проходы/фазы, в процентах	Отношение роста
5	$3.566 \ln S + 0.158$	$1.463 \ln S + 1.016$	41	1.3247180
6	$2.616 \ln S - 0.166$	$0.951 \ln S + 1.014$	36	1.4655712
7	$2.337 \ln S - 0.472$	$0.781 \ln S + 1.001$	33	1.5341577
8	$2.216 \ln S - 0.762$	$0.699 \ln S + 0.980$	32	1.5701473
9	$2.156 \ln S - 1.034$	$0.654 \ln S + 0.954$	30	1.5900054
10	$2.124 \ln S - 1.290$	$0.626 \ln S + 0.922$	29	1.6013473
20	$2.078 \ln S - 3.093$	$0.575 \ln S + 0.524$	28	1.6179086

Расщепление лент. Эффективное совмещение времени перемотки является проблемой, возникающей во многих приложениях, а не только в сортировке; существует общий подход, который часто может быть использован. Рассмотрим итеративный процесс, который использует ленты следующим образом:

	T1	T2
Фаза 1	Вывод 1	—
	Перемотка	—
Фаза 2	Ввод 1	Вывод 2
	Перемотка	Перемотка
Фаза 3	Вывод 3	Ввод 2
	Перемотка	Перемотка
Фаза 4	Ввод 3	Вывод 4
	Перемотка	Перемотка

и т. д., где "вывод k " означает запись в k -й выводной файл, а "ввод k " означает его чтение. Можно устранить время перемотки, если использовать три ленты, как было предложено К. Вейсертом

[SACM, 5 (1962), 102]:

	T1	T2	T3
Фаза 1	Вывод 1.1	—	—
	Вывод 1.2	—	—
	Перемотка	Вывод 1.3	—
Фаза 2	Ввод 1.1	Вывод 2.1	—
	Ввод 1.2	Перемотка	Вывод 2.2
	Перемотка	Ввод 1.3	Вывод 2.3
Фаза 3	Вывод 3.1	Ввод 2.1	Перемотка
	Вывод 3.2	Перемотка	Ввод 2.2
	Перемотка	Вывод 3.3	Ввод 2.3
Фаза 4	Ввод 3.1	Вывод 4.1	Перемотка
	Ввод 3.2	Перемотка	Вывод 4.2
	Перемотка	Ввод 3.3	Вывод 4.3

и т. д. Здесь "вывод $k.j$ " означает запись j -й трети k -го выводного файла, а "ввод $k.j$ " означает ее чтение. В конце концов будет исключено все время перемотки, если перемотка по крайней мере вдвое быстрее чтения/записи. Подобная процедура, в которой вывод в каждой фазе разделяется между лентами, называется "расщеплением лент".

Л. Мак-Аллестер [SACM, 7 (1964), 158–159] показал, что расщепление лент приводит к эффективному методу совмещения времени перемотки в многофазном слиянии. Его метод можно использовать с четырьмя или большим количеством лент, и он осуществляет $(T - 2)$ -путевое слияние.

Предположим снова, что у нас есть шесть лент и попытаемся построить схему слияния, которая работает следующим образом, расщепляя вывод на каждом уровне (буквы I, O и R обозначают соответственно ввод, вывод и перемотку):

Уровень	T1	T2	T3	T4	T5	T6	Число выводимых
							отрезков
7	I	I	I	I	R	O	u_7
	I	I	I	I	O	R	v_7
6	I	I	I	R	O	I	u_6
	I	I	I	O	R	I	v_6
5	I	I	R	O	I	I	u_5
	I	I	O	R	I	I	v_5
4	I	R	O	I	I	I	u_4
	I	O	R	I	I	I	v_4
3	R	O	I	I	I	I	u_3
	O	R	I	I	I	I	v_3
2	O	I	I	I	I	R	u_2
	R	I	I	I	I	O	v_2
1	I	I	I	I	R	O	u_1
	I	I	I	I	O	R	v_1
0	I	I	I	R	O	I	u_0
	I	I	I	O	R	I	

Чтобы закончить работу с одним отрезком на T4 и пустыми остальными лентами, мы должны иметь

$$\begin{aligned}
 v_0 &= 1, \\
 u_0 + v_1 &= 0, \\
 u_1 + v_2 &= u_0 + v_0, \\
 u_2 + v_3 &= u_1 + v_1 + u_0 + v_0, \\
 u_3 + v_4 &= u_2 + v_2 + u_1 + v_1 + u_0 + v_0, \\
 u_4 + v_5 &= u_3 + v_3 + u_2 + v_2 + u_1 + v_1 + u_0 + v_0, \\
 u_5 + v_6 &= u_4 + v_4 + u_3 + v_3 + u_2 + v_2 + u_1 + v_1,
 \end{aligned}$$

и т. д.; в общем случае требуется, чтобы

$$u_n + v_{n+1} = u_{n-1} + v_{n-1} + u_{n-2} + v_{n-2} + u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4} \quad (22)$$

при всех $n \geq 0$, если считать $u_j = v_j = 0$ при всех $j < 0$.

У этих уравнений нет единственного решения; в самом деле, если положить все u равными нулю, то получим обычное многофазное слияние, причем одна лента будет лишней! Но если выбрать $u_n \approx v_{n+1}$, то время перемотки будет удовлетворительно совмещено.

Мак-Аллестер предложил взять $u_n = v_{n-1} + v_{n-2} + v_{n-3} + v_{n-4}$, $v_{n+1} = u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4}$, так что последовательность

$$\langle x_0, x_1, x_2, x_3, x_4, x_5, \dots \rangle = \langle v_0, u_0, v_1, u_1, v_2, u_2, \dots \rangle$$

удовлетворяет однородному рекуррентному соотношению

$$x_n = x_{n-3} + x_{n-5} + x_{n-7} + x_{n-9}.$$

Оказалось, однако, что лучше положить

$$\begin{aligned} v_{n+1} &= u_{n-1} + v_{n-1} + u_{n-2} + v_{n-2}, \\ u_n &= u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4}. \end{aligned}$$

Эта последовательность не только немного лучше по времени слияния, ее большое преимущество в том, что соответствующее время слияния можно проанализировать математически. [Вариант Мак-Аллестера для анализа крайне труден, потому что в одной фазе могут встречаться отрезки разной длины; мы увидим, что такого не может случиться при (23).]

Можно вывести число отрезков на каждой ленте на каждом уровне, двигаясь назад по схеме (21); мы получаем следующую схему сортировки:

Уровень	T1	T2	T3	T4	T5	T6	Время записи	Время перемотки
	1^{23}	1^{21}	1^{17}	1^{10}	—	1^{11}	82	23
7	1^{19}	1^{17}	1^{13}	1^6	R	$1^{11}4^4$	$4 \times 4 = 16$	$82 - 23$
	1^{13}	1^{11}	1^7	—	4^6	R	$6 \times 4 = 24$	25
6	1^{10}	1^8	1^4	R	4^9	1^84^4	$3 \times 4 = 12$	10
	1^6	1^4	—	4^4	R	1^44^4	$4 \times 4 = 16$	36
5	1^5	1^3	R	4^47^1	4^8	1^34^4	$1 \times 7 = 7$	17
	1^2	—	7^3	R	4^5	4^4	$3 \times 7 = 21$	23
4	1^1	R	7^313^1	4^37^1	4^4	4^3	$1 \times 13 = 13$	21
	—	13^1	R	4^27^1	4^3	4^2	$1 \times 13 = 13$	34
3	R	13^119^1	7^213^1	4^17^1	4^2	4^1	$1 \times 19 = 19$	23
	19^1	R	7^113^1	7^1	4^1	—	$1 \times 19 = 19$	32
2	19^131^0	13^119^1	7^113^1	7^1	4^1	R	$0 \times 31 = 0$	25
	R	19^1	13^1	7^0	—	31^1	$1 \times 31 = 31$	19
1	19^131^0	19^1	13^1	7^0	R	31^152^0	$0 \times 52 = 0$	} $\max(36, 31, 23)$
	19^131^0	19^1	13^1	—	52^0	R	$0 \times 52 = 0$	
0	19^131^0	19^1	13^1	R	52^082^0	31^152^0	$0 \times 82 = 0$	
	(31^0)	(19^0)	—	82^1	(R)	(52^0)	$1 \times 82 = 82$	0

Несовмещенная перемотка встречается только при перемотке вводной ленты T5 (82 единицы), в течение первой половины фазы второго уровня (25 единиц) и в течение окончательных фаз "фиктивного слияния" на уровнях 1 и 0 (36 единиц). Таким образом, время работы можно оценить величиной $273t + 143r$; для алгоритма D соответствующая величина $268t + 208r$ почти всегда хуже.

Нетрудно видеть (см. упр. 23), что длины отрезков, выводимых во время каждой фазы, суть последовательно

$$4, 4, 7, 13, 19, 31, 52, 82, 133, \dots \tag{24}$$

при этом последовательность $\langle t_1, t_2, t_3, \dots \rangle$ удовлетворяет закону

$$t_n = t_{n-2} + 2t_{n-3} + t_{n-4}, \tag{25}$$

если считать $t_n = 1$ при $n \leq 0$. Можно также проанализировать оптимальное размещение фиктивных отрезков, рассмотрев строки чисел слияний, как мы делали для стандартного многофазного метода

[ср. с (8)]:

Уровень	$T1$	$T2$	$T3$	$T4$	$T6$	Окончательный вывод на ленте
1	1	1	1	1	—	$T5$
2	1	1	1	—	1	$T4$
3	21	21	2	2	1	$T3$
4	2221	222	222	22	2	$T2$
5	23222	23222	2322	23	222	$T1$
6	333323222	33332322	333323	3333	2322	$T6$
.....						
n	A_n	B_n	C_n	D_n	E_n	$T(k)$
$n + 1$	$(A''_n E_n + 1)B_n$	$(A''_n E_n + 1)C_n$	$(A''_n E_n + 1)D_n$	$A''_n E_n + 1$	A'_n	$T(k - 1)$
.....						

где $A_n = A'_n A''_n$ и A''_n состоит из последних u_n чисел слияний A_n . Приведенное выше правило перехода с уровня n на уровень $n + 1$ справедливо для *любой* схемы, удовлетворяющей (22). Если мы определяем u и v посредством (23), то строки A_n, \dots, E_n можно выразить в следующем, довольно простом виде [ср. с (9)]:

$$\begin{aligned}
 A_n &= (W_{n-1}W_{n-2}W_{n-3}W_{n-4}) + 1, \\
 B_n &= (W_{n-1}W_{n-2}W_{n-3}) + 1, \\
 C_n &= (W_{n-1}W_{n-2}) + 1, \\
 D_n &= (W_{n-1}) + 1, \\
 E_n &= (W_{n-2}W_{n-3}) + 1,
 \end{aligned}
 \tag{27}$$

где

$$\begin{aligned}
 W_n &= (W_{n-3}W_{n-4}W_{n-2}W_{n-3}) + 1 \quad \text{при } n > 0; \\
 W_0 &= '0' \text{ и } W_n = (\text{пусто}) \quad \text{при } n < 0.
 \end{aligned}
 \tag{28}$$

Исходя из этих соотношений, легко подробно проанализировать случай шести лент.

В общем случае, если имеется $T \geq 5$ лент, то положим $P = T - 2$ и определим последовательности $\langle u_n \rangle, \langle v_n \rangle$ по правилам

$$\begin{aligned}
 v_{n+1} &= u_{n-1} + v_{n-1} + \dots + u_{n-r} + v_{n-r}; \\
 u_n &= u_{n-r-1} + v_{n-r-1} + \dots + u_{n-P} + v_{n-P} \quad \text{при } n \geq 0,
 \end{aligned}
 \tag{29}$$

где $r = \lfloor P/2 \rfloor, v_0 = 1$ и $u_n = v_n = 0$ при $n < 0$. Если $w_n = u_n + v_n$, то имеем

$$\begin{aligned}
 w_n &= w_{n-2} + \dots + w_{n-r} + 2w_{n-r-1} + w_{n-r-2} + \dots + w_{n-P}, \quad n > 0, \\
 w_0 &= 1 \text{ и } w_n = 0 \quad \text{при } n < 0.
 \end{aligned}
 \tag{30}$$

При начальном распределении для уровня $n + 1$ на ленту k помещается $w_n + w_{n-1} + \dots + w_{n-P+k}$ отрезков при $1 \leq k \leq P$ и $w_{n-1} + \dots + w_{n-r}$ — на ленту T ; лента $T - 1$ используется для ввода. Затем u_n отрезков сливаются на ленту T , в то время как лента $T - 1$ перематывается; v_n отрезков сливаются на $T - 1$, пока T перематывается; u_{n-1} отрезков — на $T - 1$, пока $T - 2$ перематывается, и т. д.

Таблица 6

Ленты	Приблизительное поведение фазы	Проходы	Проходы/фазы в процентах	Отношение роста
4	$2.885 \ln S + 0.000$	$1.443 \ln S + 1.000$	50	1.4142136
5	$2.078 \ln S + 0.232$	$0.929 \ln S + 1.022$	45	1.6180340
6	$2.078 \ln S - 0.170$	$0.752 \ln S + 1.024$	34	1.6180340
7	$1.958 \ln S - 0.408$	$0.670 \ln S + 1.007$	34	1.6663019
8	$2.008 \ln S - 0.762$	$0.624 \ln S + 0.994$	31	1.6454116
9	$1.972 \ln S - 0.987$	$0.595 \ln S + 0.967$	30	1.6604077
10	$2.013 \ln S - 1.300$	$0.580 \ln S + 0.94l$	29	1.6433803
20	$2.069 \ln S - 3.164$	$0.566 \ln S + 0.536$	27	1.6214947

Таблица 6 показывает приблизительное поведение этой процедуры, когда S не слишком мало. Столбец "проходы/фазы" примерно указывает, какая часть всего файла перематывается во время каждой

половины фазы и какая часть файла записывается за время каждой полной фазы. *Метод расщепления лент превосходит стандартный многофазный на шести или более лентах* и, вероятно, также на пяти лентах, по крайней мере для больших S .

Если $T = 4$, то указанная процедура стала бы, по существу, эквивалентной сбалансированному двухпутевому слиянию без совмещения времени перемотки, так как w_{2n+1} было бы равно 0 при всех n . Поэтому элементы табл. 6 при $T = 4$ были получены посредством небольшой модификации, состоящей в том, что полагалось

$$v_2 = 0, u_1 = 1, v_1 = 0, u_0 = 0, v_0 = 1 \text{ и } v_{n+1} = u_{n-1} + v_{n-1}, \\ u_n = u_{n-2} + v_{n-2} \quad \text{при } n \geq 2.$$

Это приводит к очень интересной схеме сортировки (см. упр. 25 и 26).

Упражнения

1. [16] На рис. 69 указан порядок, в котором алгоритм D распределяет по пяти лентам отрезки с 34-го по 65-й; в каком порядке распределяются отрезки с 1-го по 33-й?
- >2. [21] Верно ли, что после двух фаз слияния в алгоритме D, т. е. когда мы во второй раз достигнем шага D6, все фиктивные отрезки исчезают?
- >3. [22] Докажите, что по окончании шага D4 всегда выполнено условие $D[1] \geq D[2] \geq \dots \geq D[T]$. Объясните, важность этого условия для правильной работы механизма шагов D2 и D3.
4. [M20] Выведите производящие функции (7).
5. [BM26] (Э. П. Майлс мл., 1960.) Докажите, что при всех $p \geq 2$ многочлен $f_p(z) = z^p - z^{p-1} - \dots - z - 1$ имеет p различных корней, из которых ровно один превосходит 1 по абсолютной величине. [Указание: рассмотрите многочлен $z^{p+1} - 2z^p + 1$.]
6. [BM24] Цель этого упражнения — рассмотреть способ составления табл. 1, 5 и 6. Предположим, что имеется схема слияния, свойства которой следующим образом характеризуются многочленами $p(z)$ и $q(z)$: (1) Число начальных отрезков в "точном распределении", требующем n фаз слияния, равно коэффициенту при z^n в $p(z)/q(z)$. (2) Число начальных отрезков, обрабатываемых в течение этих n фаз слияния, равно коэффициенту при z^n в $p(z)/q(z)^2$. (3) У многочлена $q(z^{-1})$ есть "главный корень" α , такой, что $q(\alpha^{-1}) = 0$, $q'(\alpha^{-1}) \neq 0$, $p(\alpha^{-1}) \neq 0$, и из $q(\beta^{-1}) = 0$ следует, что $\beta = \alpha$ или $|\beta| < |\alpha|$.

Докажите, что существует $\varepsilon > 0$, такое, что если S равно числу отрезков в точном распределении, требующем n фаз слияния, а во время этих фаз обрабатывается ρS отрезков, то $n = a \ln S + b + O(S^\varepsilon)$, $\rho = c \ln S + d + O(S^{-\varepsilon})$, где

$$a = (\ln \alpha)^{-1}, \quad b = -a \ln \left(\frac{p(\alpha^{-1})}{-q'(\alpha^{-1})} \right) - 1, \quad c = a \frac{\alpha}{-q'(\alpha^{-1})}, \\ d = \frac{(b+1)\alpha - p'(\alpha^{-1})/p(\alpha^{-1}) + q''(\alpha^{-1})/q'(\alpha^{-1})}{-q'(\alpha^{-1})}.$$

7. [BM22] Пусть α_p — главный корень многочлена $f_p(z)$ из упр. 5. Каково асимптотическое поведение α_p при $p \rightarrow \infty$?
8. [M20] (Э. Нетто, 1901.) Пусть $N_m^{(p)}$ есть число способов выразить m в виде упорядоченной суммы целых чисел $\{1, 2, \dots, p\}$. Например, если $p = 3$ и $m = 5$, то имеется 13 способов: $1 + 1 + 1 + 1 + 1 = 1 + 1 + 1 + 2 = 1 + 1 + 2 + 1 = 1 + 1 + 3 = 1 + 2 + 1 + 1 = 1 + 2 + 2 = 1 + 3 + 1 = 2 + 1 + 1 + 1 = 2 + 1 + 2 = 2 + 2 + 1 = 2 + 3 = 3 + 1 + 1 = 3 + 2$. Покажите, что $N_m^{(p)}$ являются обобщенными числами Фибоначчи.
9. [M20] Пусть $K_m^{(p)}$ — число последовательностей из нулей и единиц, таких, что в них нет p последовательных единиц. Например, если $p = 3$ и $m = 5$, имеется 24 варианта: 00000, 00001, 00010, 00011, 00100, 00101, 00110, 01000, 01001, \dots , 11011. Покажите, что $K_m^{(p)}$ являются обобщенными числами Фибоначчи.
10. [M27] (Система счисления с обобщенными числами Фибоначчи.) Докажите, что каждое неотрицательное целое n имеет единственное представление в виде суммы различных чисел Фибоначчи p -го порядка $F_j^{(p)}$ при $j \geq p$, удовлетворяющее условию, что не используются никакие p последовательные числа Фибоначчи.
11. [M24] Докажите, что n -й элемент цепочки Q_∞ в (12) равен количеству различных чисел Фибоначчи в представлении элемента $n - 1$ числами Фибоначчи пятого порядка (см. упр. 10).

>12. [M20] Найдите зависимость между степенями матрицы

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

и точным фибоначчиевым распределением в (1).

- >13. [22] Докажите следующее интересное свойство точных фибоначчиевых распределений: если окончательный вывод оказывается на ленте номер T , то число отрезков на всех других лентах *нечетное*, если окончательный вывод оказывается на некоторой ленте, отличной от T , то число отрезков будет *нечетным* на этой ленте и *четным* на остальных [см. (1)].
14. [M35] Пусть $T_n(x) = \sum_{k \geq 0} T_{nk} x^k$, где $T_n(x)$ — многочлены, определенные в (16). (а) Покажите, что для каждого k существует число $n(k)$, такое, что $T_{1k} \leq T_{2k} \leq \dots \leq T_{n(k)k} > T_{n(k)+1,k} \geq \dots$. (б) При условии что $T_{n'k'} < T_{nk'}$ и $n' < n$, докажите, что $T_{n'k} \leq T_{nk}$ для всех $k \geq k'$. (с) Докажите, что существует неубывающая последовательность $\langle M_n \rangle$, такая, что $\sum_n(S) = \min_{j \geq 1} \sum_j(S)$ при $M_n \leq S < M_{n+1}$, но $\sum_n(S) > \min_{j \geq 1} \sum_j(S)$ при $S \geq M_{n+1}$. [См. (19).]
15. [M43] Верно ли, что $\sum_{n-1}(m) < \sum_n(m)$ влечет $\sum_n(m) \leq \sum_{n+1}(m) \leq \sum_{n+2}(m) \leq \dots$? (Такой результат сильно упростил бы вычисление табл. 2.)
16. [M43] Определите асимптотическое поведение многофазного слияния с оптимальным распределением фиктивных отрезков.
17. [32] Верно ли, что отрезки для оптимального многофазного распределения можно разместить таким образом, что распределение $S+1$ начальных отрезков получается путем добавления одного отрезка (на соответствующую ленту) к распределению S начальных отрезков?
18. [30] Верно ли, что оптимальное многофазное распределение дает наилучшую возможную схему слияния в том смысле, что суммарное количество обрабатываемых начальных отрезков минимально, если требуется, чтобы начальные отрезки размещались не более, чем на $T-1$ лентах? (Временем перемотки пренебречь.)
19. [21] Составьте таблицу, аналогичную (1), для многофазного метода сортировки Кэйрона для шести лент.
20. [M24] Какая производящая функция для кэйроновской многофазной сортировки на шести лентах соответствует (7) и (16)? Какие соотношения, аналогичные (9) и (27), определяют строки чисел слияний?
21. [11] Что должно появиться на уровне 7 в (26)?
22. [M21] Каждый член последовательности (24) приблизительно равен сумме двух предыдущих. Наблюдается ли это явление для остальных членов последовательности? Сформулируйте и докажите теорему о $t_n - t_{n-1} - t_{n-2}$.
- >23. [29] Какие изменения надо было бы сделать в (25), (27) и (28), если бы (23) заменилось на $v_{n+1} = u_{n-1} + v_{n-1} + u_{n-2}$, $u_n = v_{n-2} + u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4}$?
23. [BM41] Вычислите асимптотическое поведение многофазной процедуры с расщеплением лент, если элемент v_{n+1} определен как сумма первых q членов $u_{n-1} + v_{n-1} + \dots + u_{n-p} + v_{n-p}$ при различных $P = T-2$ и $0 \leq q \leq 2P$. (В тексте рассматривается только случай $q = 2 \lfloor P/2 \rfloor$; ср. с упр. 23.)
25. [19] Продемонстрируйте, как многофазное слияние с расщеплением лент, упомянутое в конце этого пункта, сортировало бы 32 начальных отрезка. (Дайте анализ фаза за фазой, как это сделано в тексте в примере с 82 отрезками и 6 лентами.)
26. [M21] Проанализируйте поведение многофазного слияния с расщеплением лент на четырех лентах при $S = 2^n$ и при $S = 2^n + 2^{n-1}$ (см. упр. 25).
27. [23] Если начальные отрезки распределены на лентах в соответствии с точным распределением, то многофазная стратегия превращается просто в "сливать до опустошения". Мы сливаем отрезки со всех непустых входных лент, пока одна из них не станет пустой, затем мы используем эту ленту как следующую выводную, а предыдущую выводную ленту используем как вводную.
Верно ли, что эта стратегия "сливать до опустошения" всегда выполняет сортировку независимо от того, как распределены начальные отрезки, при условии что мы распределяем их по крайней мере на две ленты (одна лента, конечно, будет оставлена пустой, чтобы она могла служить первой выводной лентой).
28. [M26] Предыдущее упражнение определяет весьма большое семейства схем слияния. Покажите, что многофазная схема *наилучшая* из них в следующем смысле: если имеется шесть лент и мы рассматриваем класс всех начальных распределений (a, b, c, d, e) , таких, что стратегия "сливать

до опустошения” требует n или меньше фаз для сортировки, то $a + b + c + d + e \leq t_n$, где t_n — соответствующее число для многофазной сортировки (1).

29. [M47] Упр. 28 показывает, что многофазное распределение оптимально среди всех схем ”сливать до опустошения” в смысле минимальности числа фаз. Но является ли оно оптимальным также в смысле минимальности числа проходов?

Пусть числа a и b взаимно простые, и предположим, что $a + b$ есть число Фибоначчи F_n . Верно ли следующее предположение, высказанное Р. М. Карпом: число начальных отрезков, обрабатываемых схемой ”сливать до опустошения”, начинающейся с распределения (a, b) , больше или равно $((n - 5)F_{n+1} + (2n + 2)F_n)/5$? (Указанное значение достигается, когда $a = F_{n+1}$, $b = F_{n-2}$.)

30. [42] Составьте таблицу, аналогичную табл. 2, для многофазного слияния с расщеплением лент.

5.4.3. Каскадное слияние

Другая основная схема, называемая ”каскадным слиянием”, на самом деле была открыта раньше многофазной [Б. К. Бетц и У. К. Картер, ACM Nat’1 Conference, 14 (1959), Paper 14]. Ниже в таблице этот подход иллюстрируется для 6 лент и 190 начальных отрезков с использованием обозначений из п. 5.4.2:

	T1	T2	T3	T4	T5	T6	Количество обработанных начальных отрезков
Проход 1.	1 ⁵⁵	1 ⁵⁰	1 ⁴¹	1 ²⁹	1 ¹⁵	—	190
Проход 2.	—	1 ^{5*}	2 ⁹	3 ¹²	4 ¹⁴	5 ¹⁵	190
Проход 3.	15 ⁵	14 ⁴	12 ³	9 ²	5 ^{1*}	—	190
Проход 4.	—	15 ^{1*}	29 ¹	41 ¹	50 ¹	55 ¹	190
Проход 5.	190 ¹	—	—	—	—	—	190

Каскадное слияние, подобно многофазному, начинается с ”точного распределения” отрезков по лентам, хотя правила точного распределения отличны от правил п. 5.4.2. Каждая строка таблицы представляет полный проход по *всем* данным. Проход 2, например, получается посредством выполнения пятипутевого слияния с T1, T2, T3, T4, T5 на T6, пока T5 не станет пустой (при этом на T6 помещаются 15 отрезков относительной длины 5), затем четырехпутевого слияния с T1, T2, T3, T4 на T5, затем трехпутевого слияния на T4, двухпутевого слияния на T3 и, наконец, однопутевого слияния (операции копирования) с T1 на T2. Проход 3 получается таким же образом путем выполнения сначала пятипутевого слияния, пока одна лента не станет пустой, затем четырехпутевого и т. д. (Похоже, что этому пункту книги следовало бы присвоить номер 5.4.3.2.1, а не 5.4.3!)

Ясно, что операции копирования излишни, и их можно было бы опустить. Фактически, однако, в случае шести лент это копирование занимает только небольшой процент всего времени. Элементы, которые получаются простым копированием, отмечены в приведенной таблице звездочкой. Только 25 из 950 обрабатываемых отрезков принадлежат этому классу. Большая часть времени отводится пятипутевому и четырехпутевому слияниям.

На первый взгляд может показаться, что каскадная схема — довольно плохой вариант в сравнении с многофазной, так как стандартная многофазная схема использует все время $(T - 1)$ -путевое слияние, в то время как каскадная использует $(T - 1)$ -путевое, $(T - 2)$ -путевое, $(T - 3)$ -путевое и т. д., но в действительности она асимптотически *лучше*, чем многофазная, для шести и более лент! Как мы видели в п. 5.4.2, высокий порядок слияния не является гарантией эффективности. В табл. 1 показаны характеристики выполнения каскадного слияния по аналогии с подобной таблицей п. 5.4.2.

Нетрудно вывести ”точные распределения” для каскадного слияния. Для шести лент имеем

Уровень	T1	T2	T3	T4	T5	
0	1	0	0	0	0	
1	1	1	1	1	1	
2	5	4	3	2	1	
3	15	14	12	9	6	
4	55	50	41	29	15	
5	190	175	146	105	55	
.....						
n	a_n	b_n	c_n	d_n	e_n	
$n + 1$	$a_n + b_n + c_n + d_n + e_n$	$a_n + b_n + c_n + d_n$	$a_n + b_n + c_n$	$a_n + b_n$	a_n	

(1)

Таблица 1

Ленты	Характер поведения каскадного слияния		
	Проходы (с копированием)	Проходы (без копирования)	Отношение роста
3	$2.078 \ln S + 0.672$	$1.504 \ln S + 0.992$	1.6180340
4	$1.235 \ln S + 0.754$	$1.102 \ln S + 0.820$	2.2469796
5	$0.946 \ln S + 0.796$	$0.897 \ln S + 0.800$	2.8793852
6	$0.796 \ln S + 0.821$	$0.773 \ln S + 0.808$	3.5133371
7	$0.703 \ln S + 0.839$	$0.691 \ln S + 0.822$	4.1481149
8	$0.639 \ln S + 0.852$	$0.632 \ln S + 0.834$	4.7833861
9	$0.592 \ln S + 0.861$	$0.587 \ln S + 0.845$	5.4189757
10	$0.555 \ln S + 0.869$	$0.552 \ln S + 0.854$	6.0547828
20	$0.397 \ln S + 0.905$	$0.397 \ln S + 0.901$	12.4174426

Отметим интересное свойство этих чисел— их относительные величины являются также и длинами диагоналей правильного $(2T - 1)$ -угольника. Например, пять диагоналей одиннадцатигульника на рис. 73 имеют относительные длины, очень близкие к 190, 175, 146, 105 и 55! Мы докажем этот замечательный факт

Picture: Рис. 73. Геометрическая интерпретация каскадных чисел.

позднее в этом пункте, а также увидим, что относительные времена, затрачиваемые на $(T - 1)$ -путевое слияние, $(T - 2)$ -путевое слияние, ..., однопутевое слияние, приблизительно пропорциональны *квадратам* длин этих диагоналей.

***Начальное распределение отрезков.** Если число начальных отрезков в действительности не есть число Фибоначчи, мы можем, как обычно, вставить фиктивные отрезки. Поверхностный анализ ситуации показывает, что метод приписывания фиктивных отрезков несуществен, так как каскадное слияние всегда осуществляет полные проходы; если имеется 190 начальных отрезков, то каждая запись обрабатывается пять раз, как в приведенном выше примере, но если имеется 191 отрезок, то, очевидно, следует увеличить уровень, и теперь каждая запись будет обрабатываться шесть раз. К счастью, в действительности можно избежать такого резкого скачка. Дэвид Э. Фергюсон нашел способ так распределить начальные отрезки, что многие операции во время первой

Picture: Рис. 74. Эффективность каскадного слияния с распределением по алгоритму D.

фазы слияния сводятся к копированию содержимого ленты. Если обойти такие копирования (просто изменив "логические" номера ленточных устройств по отношению к "физическим" номерам, как в алгоритме 5.4.2D), то получим относительно плавный переход с уровня на уровень, как изображено на рис. 74.

Предположим, что (a, b, c, d, e) , где $a \geq b \geq c \geq d \geq e$ —точное распределение. Переопределив соответствие между логическими и физическими ленточными устройствами, мы можем представить, что реальное распределение—это (e, d, c, b, a) , т. е. a отрезков на T5, b на T4 и т. д. Следующее точное распределение—это $(a + b + c + d + e, a + b + c + d, a + b + c, a + b, a)$; и если ввод исчерпывается прежде, чем мы достигаем этого следующего уровня, то будем считать, что ленты содержат соответственно $(D_1, D_2, D_3, D_4, D_5)$ фиктивных отрезков, где

$$\begin{aligned} D_1 &\leq a + b + c + d, & D_1 &\leq a + b + c, & D_3 &\leq a + b, \\ D_4 &\leq a, & D_5 &= 0; & D_1 &\geq D_2 \geq D_3 \geq D_4 \geq D_5. \end{aligned} \quad (2)$$

Мы вольны представлять себе, что эти фиктивные отрезки появляются на лентах в любом удобном месте. Предполагается, что первый проход слияния даст a отрезков посредством пятипутевого слияния, затем b отрезков посредством четырехпутевого и т. д. Наша цель состоит в расположении фиктивных отрезков таким образом, чтобы заменить слияние копированием. Удобно выполнять первый проход слияния следующим образом:

1. Если $D_4 = a$, то вычесть a из всех D_1, D_2, D_3, D_4 и заявить, что T5—результат слияния. Если $D_4 < a$, то слить a отрезков с лент T1 по T5, используя минимально возможное число фиктивных отрезков на лентах T1–T5 так, чтобы новые значения D_1, D_2, D_3, D_4 удовлетворяли соотношениям

$$\begin{aligned} D_1 &\leq b + c + d, & D_2 &\leq b + c, & D_3 &\leq b, \\ D_4 &= 0; & D_1 &\geq D_2 \geq D_3 \geq D_4. \end{aligned} \quad (3)$$

Таким образом, если D_2 было первоначально $\leq b + c$, то мы не используем ни одного фиктивного отрезка с этой ленты на данном шаге; в то же время, если $b + c < D_2 \leq a + b + c$, мы используем ровно $D_2 - b - c$ фиктивных отрезков.

2. (Этот шаг аналогичен шагу 1, но с некоторым "сдвигом".) Если $D_3 = b$, то вычесть b из всех D_1, D_2, D_3 и заявить, что Т4—результат слияния. Если $D_3 < b$, то слить b отрезков с лент Т1–Т4, уменьшая, если необходимо, число фиктивных отрезков, чтобы достичь

$$D_1 \leq b + c, \quad D_2 \leq b, \quad D_3 = 0; \quad D_1 \geq D_2 \geq D_3.$$

3. И так далее.

Метод Фергюсона распределения отрезков по лентам можно проиллюстрировать, рассмотрев процесс перехода с уровня 3 на уровень 4 в (1). Допустим, что "логические" ленты (Т1, ..., Т5) содержали соответственно (5, 9, 12, 14, 15) отрезков и что мы хотим довести это в конечном итоге до (55, 50, 41, 29, 15). Эта процедура может быть кратко записана так:

Шаг	Добавить к Т1	Добавить к Т2	Добавить к Т3	Добавить к Т4	Добавить к Т5	Сэкономленная величина
(1, 1)	9	0	0	0	0	15 + 14 + 12 + 5
(2, 2)	3	12	0	0	0	15 + 14 + 9 + 5
(2, 1)	9	0	0	0	0	15 + 14 + 5
(3, 3)	2	2	14	0	0	15 + 12 + 5
(3, 2)	3	12	0	0	0	15 + 9 + 5
(3, 1)	9	0	0	0	0	15 + 5
(4, 4)	1	1	1	15	0	14 + 5
(4, 3)	2	2	14	0	0	12 + 5
(4, 2)	3	12	0	0	0	9 + 5
(4, 1)	9	0	0	0	0	5

Сначала помещаем девять отрезков на Т1, затем (3, 12)—на Т1 и Т2 и т. д. Если ввод исчерпается, скажем, во время шага (3, 2), то "сэкономленная величина" составляет 15 + 9 + 5. Это означает, что мы избегаем пятипутевого слияния 15 отрезков, двухпутевого слияния 9 отрезков и однопутевого слияния 5 отрезков посредством присвоения фиктивных отрезков. Другими словами, 15 + 9 + 5 отрезков, присутствующих на уровне 3, не обрабатываются в течение первой фазы слияния.

Следующий алгоритм детально описывает этот процесс.

Алгоритм С. (Сортировка каскадным слиянием со специальным распределением.) Этот алгоритм распределяет начальные отрезки по лентам отрезков за отрезком, пока запас начальных отрезков не исчерпается. Затем он определяет, как следует сливать ленты, предполагая, что имеется $T \geq 3$ лентопротяжных устройств, при этом используется самое большее $(T - 1)$ -путевое слияние и ненужные однопутевые слияния устраняются. Лента T может быть использована для хранения ввода, так как на нее не попадает ни один начальный отрезок. Алгоритм работает со следующими массивами:

- $A[j]$, $1 \leq j \leq T$: Последнее точное каскадное распределение, которое было достигнуто.
- $AA[j]$, $1 \leq j \leq T$: Точное каскадное распределение, к которому мы стремимся.
- $D[j]$, $1 \leq j \leq T$: Число фиктивных отрезков, предполагаемых присутствующими на логическом лентопротяжном устройстве с номером j .
- $M[j]$, $1 \leq j \leq T$: Максимальное число фиктивных отрезков, которое желательно иметь на логическом лентопротяжном устройстве с номером j .
- $TARE[j]$, $1 \leq j \leq T$: Номер физического лентопротяжного устройства, соответствующего логическому лентопротяжному устройству с номером j .

С1 [Начальная установка.] Установить $A[k] \leftarrow AA[k] \leftarrow D[k] \leftarrow 0$ при $2 \leq k \leq T$; установить $A[1] \leftarrow 0$, $AA[1] \leftarrow 1$, $D[1] \leftarrow 1$; установить $TARE[k] \leftarrow k$ при $1 \leq k \leq T$. Наконец, установить $i \leftarrow T - 2$, $j \leftarrow 1$, $k \leftarrow 1$, $l \leftarrow 0$, $m \leftarrow 1$ и перейти к шагу **С5** (эти маневры являются одним из способов начать работу непосредственно во внутреннем цикле с соответствующей установкой управляющих переменных).

С2 [Начать новый уровень.] (Мы только что получили точное распределение. Но так как еще имеются исходные данные, то необходимо подготовиться к следующему уровню.) Увеличить l на 1. Установить $A[k] \leftarrow AA[k]$ при $1 \leq k \leq T$, затем установить $AA[T - k] \leftarrow AA[T - k + 1] + A[k]$ при $k = 1, 2, \dots, T - 1$ (в таком порядке). Установить $(TARE[1], TARE[2], \dots, TARE[T - 1]) \leftarrow (TARE[T - 1], \dots, TARE[2], TARE[1])$ и установить $D[k] \leftarrow AA[k + 1]$ при $1 \leq k < T$. Наконец, установить $i \leftarrow 1$.

- С3** [Начать подуровень i .] Установить $j \leftarrow i$. (Переменные i и j представляют "шаг (i, j) " в таблице, иллюстрирующей метод Фергюсона.)
- С4** [Начать шаг (i, j) .] Установить $k \leftarrow j$ и $m \leftarrow A[T - j - 1]$. Если $m = 0$ и $i = j$, то установить $i \leftarrow T - 2$ и вернуться к **С3**; если $m = 0$ и $i \neq j$, вернуться к **С2**. (Переменная m представляет собой число отрезков, которое должно быть записано на ленту $\text{TARE}[k]$; m бывает равно 0, только если $l = 1$.)
- С5** [Ввести на ленту $\text{TARE}[k]$.] Записать один отрезок на ленту номер $\text{TARE}[k]$ и уменьшить $D[k]$ на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу **С7**.
- С6** [Продвижение.] Уменьшить m на 1. Если $m > 0$, вернуться к шагу **С5**. В противном случае уменьшить k на 1; если $k > 0$, установить $m \leftarrow A[T - j - 1] - A[T - j]$ и вернуться к **С5**, если $m > 0$. В противном случае уменьшить j на 1; если $j > 0$, перейти к шагу **С4**. В противном случае увеличить i на 1; если $i < T < 1$, вернуться к шагу **С3**. В противном случае перейти к **С2**.
- С7** [Подготовка к слиянию.] (К этому моменту начальное распределение завершено, и таблицы A , AA , D и TARE описывают состояние всех лент в данный момент.) Установить $M[k] \leftarrow AA[k + 1]$ при $1 \leq k < T$ и установить $\text{FIRST} \leftarrow 1$. (Переменная FIRST принимает ненулевое значение только во время первого прохода слияния.)
- С8** [Каскад.] Если $l = 0$, остановиться; сортировка завершена, вывод находится на $\text{TARE}[1]$. В противном случае при $p = T - 1, T - 2, \dots, 1$ (в таком порядке) выполнять p -путевое слияние с лент $\text{TARE}[1], \dots, \text{TARE}[p]$ на $\text{TARE}[p + 1]$ следующим образом:
Если $p = 1$, то моделировать однопутевое слияние обычной перемоткой $\text{TARE}[2]$ и заменой $\text{TARE}[1] \leftrightarrow \text{TARE}[2]$. В противном случае, если $\text{FIRST} = 1$ и $D[p - 1] = M[p - 1]$, то моделировать p -путевое слияние, просто поменяв $\text{TARE}[p] \leftrightarrow \text{TARE}[p + 1]$, перемотав $\text{TARE}[p]$ и вычтя $M[p - 1]$ из каждого $D[1], \dots, D[p - 1]$, $M[1], \dots, M[p - 1]$. В противном случае вычтеть $M[p - 1]$ из всех $M[1], \dots, M[p - 1]$. Затем слить по одному отрезку с каждой $\text{TARE}[j]$, такой, что $1 \leq j \leq p$ и $D[j] \leq M[j]$; вычтеть единицу из каждого $D[j]$, такого, что $1 \leq j \leq p$ и $D[j] > M[j]$, и поместить выводной отрезок на $\text{TARE}[p + 1]$. Продолжать работу, пока $\text{TARE}[p]$ не станет пустой. Затем перемотать $\text{TARE}[p]$ и $\text{TARE}[p + 1]$.
- С9** [Опуститься на один уровень.] Уменьшить l на 1, установить $\text{FIRST} \leftarrow 0$, установить $(\text{TARE}[1], \dots, \text{TARE}[T]) \leftarrow (\text{TARE}[T], \dots, \text{TARE}[1])$. (К этому моменту все D и M —нули и таковыми останутся.) Вернуться к **С8**. ■

Шаги **С1–С6** этого алгоритма выполняют распределение, шаги **С7–С9** выполняют слияние; эти две части совершенно независимы одна от другой, и можно было бы хранить $M[k]$ и $AA[k + 1]$ в одних и тех же ячейках памяти.

Picture: Рис. 75. Каскадное слияние со специальным распределением.

***Анализ каскадного слияния.** Каскадное слияние поддается анализу с большим трудом, чем многофазное. Но этот анализ особенно интересен, поскольку содержит много замечательных формул. Настоятельно рекомендуем читателям, интересующимся дискретной математикой, самостоятельно проанализировать каскадное распределение, прежде чем читать дальше, ведь числа имеют так много необычных свойств, открывать которые — одно удовольствие! Мы обсудим здесь лишь один из многих подходов, обращая особое внимание на методы получения результатов.

Для удобства рассмотрим случай шести лент. При этом будем стараться получить формулы, которые обобщаются на случай любого T . Соотношения (1) приводят к первой основной системе:

$$\begin{aligned}
 a_n &= a_n &= \binom{0}{0} a_n, \\
 b_n &= a_n - e_{n-1} = a_n - a_{n-2} &= \binom{1}{0} a_n - \binom{2}{2} a_{n-2}, \\
 c_n &= b_n - d_{n-1} = b_n - a_{n-2} - b_{n-2} &= \binom{2}{0} a_n - \binom{3}{2} a_{n-2} + \binom{4}{4} a_{n-4}, \\
 d_n &= c_n - c_{n-1} = c_n - a_{n-2} - b_{n-2} - c_{n-2} &= \binom{3}{0} a_n - \binom{4}{2} a_{n-2} + \binom{5}{4} a_{n-4} - \binom{6}{6} a_{n-6}, \\
 e_n &= d_n - b_{n-1} = d_n - a_{n-2} - b_{n-2} - c_{n-2} - d_{n-2} &= \binom{4}{0} a_n - \binom{5}{2} a_{n-2} + \binom{6}{4} a_{n-4} - \binom{7}{6} a_{n-6} + \binom{8}{8} a_{n-8}.
 \end{aligned} \tag{4}$$

Обозначим $A(z) = \sum_{n \geq 0} a_n z^n, \dots, E(z) = \sum_{n \geq 0} e_n z^n$ и определим многочлены

$$\begin{aligned}
 q_m(z) &= \binom{m}{0} - \binom{m+1}{2} z^2 + \binom{m+2}{4} z^4 - \dots = \\
 &= \sum_{k \geq 0} \binom{m+k}{2k} (-1)^k z^{2k} = \sum_{0 \leq k \leq m} \binom{2m-k}{k} (-1)^{m-k} z^{2m-2k}.
 \end{aligned} \tag{5}$$

Результат (4) кратко можно истолковать так, что $B(z) - q_1(z) \times A(z), \dots, E(z) - q_4(z)A(z)$ сводятся к конечным суммам, соответствующим граничным условиям, а именно значениям $a_{-1}, a_{-2}, a_{-3}, \dots$, которые появляются в (4) (при небольших n), но не в $A(z)$. Чтобы получить подходящие граничные условия, применим рекуррентное соотношение в обратную сторону для отрицательных уровней до уровня -8 :

n	a_n	b_n	c_n	d_n	e_n
0	1	0	0	0	0
-1	0	0	0	0	1
-2	1	-1	0	0	0
-3	0	0	0	-1	2
-4	2	-3	1	0	0
-5	0	0	1	-4	5
-6	5	-9	5	-1	0
-7	0	-1	6	-14	14
-8	14	-28	20	-7	1

(Для семи лент таблица была бы аналогичной, однако строки с нечетными n были бы сдвинуты вправо на один элемент.) Тайна последовательности $a_0, a_{-2}, a_{-4}, \dots = 1, 1, 2, 5, 14, \dots$ мгновенно раскрывается специалистом по информатике, так как эта последовательность встречается в связи с очень многими рекуррентными алгоритмами (например, в упр. 2.2.1-4 и формуле 2.3.4.4-13). Итак, мы предполагаем, что в случае T лент

$$\begin{aligned} a_{-2n} &= \binom{2n}{n} \frac{1}{n+1} && \text{при } 0 \leq n \leq T-2; \\ a_{-2n-1} &= 0 && \text{при } 0 \leq n \leq T-3. \end{aligned} \quad (6)$$

Чтобы проверить правильность этого предположения, достаточно показать, что (6) и (4) приводят к правильным результатам для уровней 0 и 1. Для уровня 1 это очевидно, а для уровня 0 нам надо проверить, что

$$\binom{m}{0} a_0 - \binom{m+1}{2} a_{-2} + \binom{m+2}{4} a_{-4} - \binom{m+3}{6} a_{-6} + \dots = \sum_{k \geq 0} \binom{m+k}{2k} \binom{2k}{k} \frac{(-1)^k}{k+1} = \delta_{m0} \quad (7)$$

для $0 \leq m \leq T-2$. К счастью, эту сумму можно вычислить стандартными методами (это "задача 2", один из основных примеров в тексте п. 4.2.6).

Теперь можно вычислить коэффициенты $B(z) - q_1(z)A(z)$ и т. д. Рассмотрим, например, коэффициент при z^{2m} в $D(z) - q_3(z)A(z)$. Он равен

$$\begin{aligned} \sum_{k \geq 0} \binom{3+m+k}{2m+2k} (-1)^{m+k} a_{-2k} &= \sum_{k \geq 0} \binom{3+m+k}{2m+2k} \binom{2k}{k} \frac{(-1)^{m+k}}{k+1} = \\ &= (-1)^m \left(\binom{2+m}{2m-1} - \binom{3+m}{2m} \right) = \\ &= (-1)^{m+1} \binom{2+m}{2m} \end{aligned}$$

из результата "задачи 3" в п. 1.2.6. Таким образом, мы вывели формулы

$$\begin{aligned} A(z) &= q_0(z)A(z); \\ B(z) &= q_1(z)A(z) - q_0(z); \\ C(z) &= q_2(z)A(z) - q_1(z); \\ D(z) &= q_3(z)A(z) - q_2(z); \\ E(z) &= q_4(z)A(z) - q_3(z). \end{aligned} \quad (8)$$

Кроме того, имеем $e_{n+1} = a_n$; следовательно, $zA(z) = E(z)$ и

$$A(z) = q_3(z)/(q_4(z) - z). \quad (9)$$

Производящие функции были выражены при помощи q -многочленов, поэтому мы хотим лучше изучить q . В этом отношении полезно упр. 1.2.9-15, так как оно дает выражение в замкнутом виде, которое может быть записано как

$$q_m(z) = \frac{((\sqrt{4-z^2} + iz)/2)^{2m+1} + ((\sqrt{4-z^2} - iz)/2)^{2m+1}}{\sqrt{4-z^2}} \quad (10)$$

Все упрощается, если теперь положить $z = 2 \sin \theta$:

$$q_m(2 \sin \theta) = \frac{(\cos \theta + i \sin \theta)^{2m+1} + (\cos \theta - i \sin \theta)^{2m+1}}{2 \cos \theta} = \frac{\cos(2m+1)\theta}{\cos \theta}. \tag{11}$$

(Это совпадение заставляет думать, что многочлены $q_m(z)$ хорошо известны в математике; и действительно, взглянув в соответствующие таблицы, видим, что $q_m(z)$, по существу, многочлен Чебышева второго рода, а именно $(-1)^m U_{2m}(z/2)$ в обычных обозначениях.)

Теперь можно определить корни знаменателя в (9): $q_4(2 \sin \theta) = 2 \sin \theta$ сводится к

$$\cos 9\theta = 2 \sin \theta \cos \theta = \sin 2\theta.$$

Решения этого соотношения получаем, если только $\pm 9\theta = 2\theta + (2n - \frac{1}{2})\pi$; все такие θ дают корни знаменателя в (9) при условии, что $\cos \theta \neq 0$. (Если $\cos \theta = 0$, то $q_m(\pm 2) = \pm(2m + 1)$, никогда не равно ± 2 .) Следовательно, получаем 8 различных корней:

$$q_4(z) - z = 0 \quad \text{при } z = 2 \sin \frac{-5}{14}\pi, \quad 2 \sin \frac{-1}{14}\pi, \quad 2 \sin \frac{3}{14}\pi, \\ 2 \sin \frac{-7}{22}\pi, \quad 2 \sin \frac{-3}{22}\pi, \quad 2 \sin \frac{1}{22}\pi, \quad 2 \sin \frac{5}{22}\pi, \quad 2 \sin \frac{9}{22}\pi.$$

Так как $q_4(z)$ —многочлен степени 8, мы учли все корни. Первые три из этих значений дают $q_3(z) = 0$, так что $q_3(z)$ и $q_4(z) - z$ имеют общим делителем многочлен третьей степени. Остальные пять корней управляют асимптотическим поведением коэффициентов $A(z)$, если разложить (9) в элементарные дроби.

Перейдя к рассмотрению общего случая T лент, положим $\theta_k = (4k + 1)\pi/(4T - 2)$. Производящая функция $A(z)$ для T -ленточных каскадных чисел принимает вид

$$\frac{4}{2T-1} \sum_{-T/2 < k < \lfloor T/2 \rfloor} \frac{\cos^2 \theta_k}{1 - z/(2 \sin \theta_k)} \tag{12}$$

(см. упр. 8); следовательно,

$$a_n = \frac{4}{2T-1} \sum_{-T/2 < k < \lfloor T/2 \rfloor} \cos^2 \theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n. \tag{13}$$

Соотношения (8) приводят теперь к аналогичным формулам:

$$b_n = \frac{4}{2T-1} \sum_{-T/2 < k < \lfloor T/2 \rfloor} \cos \theta_k \cos 3\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n; \\ c_n = \frac{4}{2T-1} \sum_{-T/2 < k < \lfloor T/2 \rfloor} \cos \theta_k \cos 5\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n; \\ d_n = \frac{4}{2T-1} \sum_{-T/2 < k < \lfloor T/2 \rfloor} \cos \theta_k \cos 7\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n \tag{14}$$

и т. д. В упр. 9 показано, что эти уравнения справедливы для всех $n \geq 0$, а не только для больших n . В каждой сумме член с $k = 0$ значительно превосходит все остальные, особенно если n достаточно велико; следовательно, "отношение роста" есть

$$\frac{1}{2 \sin \theta_0} = \frac{2}{\pi} T - \frac{1}{\pi} + \frac{\pi}{48T} + O(T^{-2}). \tag{15}$$

Каскадная сортировка впервые была исследована У. К. Картером [Proc. IFIP Congress (1962), 62–66], который получил численные результаты для небольших значений T , и Дэвидом Фергюсоном [SASM, 7 (1964), 297], который открыл первые два члена в асимптотическом поведении (15) отношения роста. Летом 1964 г. Р. У. Флloyd получил явный вид $1/(2 \sin \theta_0)$ для отношения роста, так что точные формулы могли быть использованы для всех T . Глубокий анализ каскадных чисел был

независимо выполнен Дж. Н. Рэйни [*Canadian J. Math.*, **18** (1966), 332–349], который наткнулся на них совершенно другим путем, не имея дела с сортировкой. Рэйни подметил аналогию с диагоналями (рис. 73) и вывел много других интересных свойств этих чисел. Флloyd и Рэйни в своих доказательствах оперировали с матрицами (см. упр. 6).

Модификация каскадной сортировки. Если добавлена еще одна лента, то почти все время перемотки в течение каскадной сортировки можно совместить. Например, мы можем сливать T1–T5 на T7, затем T1–T4 на T6, затем T1–T3 на T5 (которая к этому моменту уже перемотана), затем T1–T2 на T4 и начать следующий проход, когда на T4 будет перемотано сравнительно немного данных. Эффективность этого процесса можно предсказать на основании изложенного выше анализа каскадного метода (дальнейшие подробности см. в п. 5.4.6).

Схема "компромиссного слияния", которая включает многофазную и каскадную схемы как частные случаи, была предложена Д. Э. Кнутом в [*CACM*, **6** (1963), 585–587]. Каждая фаза состоит из $(T - 1)$ -путевого, $(T - 2)$ -путевого, ..., P -путевого слияний, где P —любое фиксированное число между 1 и $T - 1$. Если $P = T - 1$, то это—многофазный метод; если $P = 1$, это—чистый каскадный метод; если $P = 2$, это—каскадный метод без фаз копирования. Анализ такой схемы был проделан Ч. Радке [*IBM Systems J.*, **5** (1966), 226–247] и У. Х. Буржем [*Proc. IFIP Congress*, **1** (1971), 454–459]. Бурж нашел производящую функцию $\sum T_n(x)z^n$ для каждого (P, T) -компромиссного слияния, обобщающую соотношение (5.4.2-16); он показал, что наилучшее значение P (с точки зрения наименьшего числа обрабатываемых начальных отрезков), как функции от S при $S \rightarrow \infty$ (если непосредственно использовать схему распределения и пренебречь временем перемотки), есть соответственно (2, 3, 3, 4, 4, 4, 3, 3, 4) при $T = (3, 4, 5, 6, 7, 8, 9, 10, 11)$. Эти значения P с ростом T сильнее отклоняются в сторону каскадного, а не многофазного метода, и оказывается, что компромиссное слияние никогда не станет существенно лучше каскадного. С другой стороны, при оптимальном выборе уровней и распределении фиктивных отрезков, как описано в п. 5.4.2, чистый многофазный метод кажется наилучшим среди всех компромиссных слияний. К сожалению, оптимальное распределение сравнительно трудно реализовать.

Т. Л. Йонсен [*BIT*, **6** (1966), 129–143] исследовал сочетания сбалансированного и многофазного слияний; модификация сбалансированного слияния с совмещением перемоток была предложена М. Готцем [*Digital Computer User's Handbook*, ed. by M. Klerer and G. A. Korn (New York: McGraw-Hill, 1967), 1.311–1.312]; можно представить себе и многие другие гибридные схемы.

Упражнения

1. [10] Используя табл. 1, сравните каскадное слияние с описанной в п. 5.4.2 версией многофазного слияния с "расщеплением лент". Какой метод лучше? (Временем перемотки пренебречь.)
- >2. [22] Сравните каскадную сортировку с тремя лентами, использующую алгоритм С, и многофазную сортировку с тремя лентами, использующую алгоритм 5.4.2D. Какие сходства и различия вы заметите?
3. [20] Составьте таблицу, показывающую, что происходит при сортировке на шести лентах 100 начальных отрезков при помощи алгоритма С.
4. [M20] (Дж. Н. Рэйни.) "Каскадное распределение n -го уровня" есть мультимножество, определенное следующим образом (в случае шести лент): $\{1, 0, 0, 0, 0\}$ есть каскадное распределение нулевого уровня; если $\{a, b, c, d, e\}$ —каскадное распределение n -го уровня, то $\{a + b + c + d + e, a + b + c + d, a + b + c, a + b, a\}$ будет каскадным распределением $(n + 1)$ -го уровня (Так как мультимножество не упорядочено, то из единственного распределения n -го уровня можно образовать до 5 различных распределений $(n + 1)$ -го уровня.) (а) Докажите, что любое мультимножество $\{a, b, c, d, e\}$ из взаимно простых чисел является каскадным распределением n -го уровня при некотором n . (б) Докажите, что распределение, используемое в каскадной сортировке, оптимально в том смысле, что если $\{a, b, c, d, e\}$ —любое распределение n -го уровня, причем $a \geq b \geq c \geq d \geq e$, то будем иметь $a \leq a_n, b \leq b_n, c \leq c_n, d \leq d_n, e \leq e_n$, где $\{a_n, b_n, c_n, d_n, e_n\}$ —распределение, определенное в (1).
- >5. [20] Докажите, что каскадные числа, определенные в (1), удовлетворяют закону

$$a_k a_{n-k} + b_k b_{n-k} + c_k c_{n-k} + d_k d_{n-k} + e_k e_{n-k} = a_n \quad \text{при } 0 \leq k \leq n.$$

[Указание: для лучшего понимания этого соотношения рассмотрите, сколько отрезков различной длины выводится в течение k -го прохода полной каскадной сортировки.]

6. [M20] Найдите 5×5 -матрицу Q , такую, что первая строка Q^n содержит каскадные числа для шести лент $a_n b_n c_n d_n e_n$ при всех $n \geq 0$.

7. [M20] При условии что каскадное слияние применяется к точному распределению a_n начальных отрезков, найдите формулу для величины сэкономленной работы, когда исключается однопутевое слияние.
8. [BM23] Выведите формулу (12).
9. [BM26] Выведите формулу (14).
- >10. [M28] Вместо системы (4) для изучения каскадных чисел воспользуйтесь в качестве исходных тождествами

$$\begin{aligned}
 e_n &= a_{n-1} &&= \binom{1}{1} a_{n-1}; \\
 d_n &= 2a_{n-1} - e_{n-2} &&= \binom{2}{1} a_{n-1} - \binom{3}{3} a_{n-3}; \\
 c_n &= 3a_{n-1} - d_{n-2} - 2e_{n-2} &&= \binom{3}{1} a_{n-1} - \binom{4}{3} a_{n-3} + \binom{5}{5} a_{n-5}
 \end{aligned}$$

и т. д. Полагая

$$r_m(z) = \binom{m}{1} z - \binom{m+1}{3} z^3 + \binom{m+2}{5} z^5 - \dots,$$

выразите $A(z)$, $B(z)$ и т. д. через эти r -многочлены.

11. [M38] Пусть

$$f_m(z) = \sum_{0 \leq k \leq m} \binom{\lfloor (m+k)/2 \rfloor}{k} (-1)^{\lfloor k/2 \rfloor} z^k.$$

Докажите, что производящая функция $A(z)$ для T -ленточных каскадных чисел равна $f_{T-3}(z)/f_{T-1}(z)$, причем числитель и знаменатель этого выражения не имеют общих делителей.

12. [M40] Докажите, что схема распределения Фергюсона оптимальна в том смысле, что при любом другом методе размещения фиктивных отрезков, удовлетворяющем (2), во время первого прохода будет обрабатываться не меньше начальных отрезков, *при условии* что во время этого прохода используется стратегия шагов C7–C9.
13. [40] В тексте предлагается большую часть времени перемотки совмещать путем добавления дополнительной ленты. Разработайте эту идею. (Например, схема, изложенная в тексте, включает ожидание перемотки ленты T4; не будет ли лучше исключить T4 из первой фазы слияния следующего прохода?)

5.4.4. Чтение ленты в обратном направлении

Многие лентопротяжные устройства позволяют читать ленту в направлении, противоположном тому, в котором шла запись на нее. Схемы слияния, с которыми мы встречались до сих пор, всегда записывают информацию на ленту в прямом направлении, затем перематывают ленту, читают ее в прямом направлении и вновь перематывают. (Файлы на ленте, следовательно, ведут себя как очереди "первым включается—первым исключается".) Обратное чтение позволяет обойтись без обеих операций перемотки: мы записываем на ленту в прямом направлении и читаем ее в обратном. (В этом случае файлы ведут себя как стеки, поскольку здесь действует правило "последним включается—первым исключается".)

Схемы сбалансированного, многофазного и каскадного слияний можно приспособить для обратного чтения. Основное отличие состоит в том, что *слияние изменяет порядок отрезков*, если мы читаем в прямом направлении и записываем в обратном. Если два отрезка находятся на ленте в возрастающем порядке, то их можно слить, читая в обратном направлении, но при этом порядок станет убывающим. Полученные таким путем убывающие отрезки станут возрастающими на следующем проходе; таким образом, алгоритм слияния должен уметь работать с отрезками обоих направлений. Программисту, впервые столкнувшемуся с обратным чтением, может показаться, что он стоит на голове!

В качестве примера обратного чтения рассмотрим процесс слияния 8 начальных отрезков с использованием *сбалансированного* слияния на 4 лентах. Можно следующим образом записать наши действия:

	T1	T2	T3	T4	
Проход 1.	$A_1 A_1 A_1 A_1$	$A_1 A_1 A_1 A_1$	—	—	Начальное распределение
Проход 2.	—	—	$D_2 D_2$	$D_2 D_2$	Слияние на T3 и T4
Проход 3.	A_4	A_4	—	—	Слияние на T1 и T2
Проход 4.	—	—	D_8	—	Окончательное слияние на T3

Здесь A_r обозначает отрезок, имеющий относительную длину r и расположенный в возрастающем порядке, если лента читается в прямом направлении, как в предыдущих наших примерах; D_r — аналогичное обозначение для убывающего отрезка длины r . Во время 2-го прохода возрастающие отрезки становятся убывающими: они оказываются убывающими при вводе, так как мы читаем в обратном направлении. Они вновь изменяют ориентацию на 3-м проходе.

Заметим, что описанный процесс завершается результатом на ленте $T3$ в *убывающем* порядке. Если это плохо (что зависит от того, должен ли результат читаться в обратном направлении, или же лента, содержащая его, должна быть снята и отложена для будущего использования), мы можем скопировать его на другую ленту, обратив направление. Более быстрым способом была бы перемотка $T1$ и $T2$ после 3-го прохода, при этом во время 4-го прохода получается A_8 . Еще быстрее было бы начать с восьми *убывающих* отрезков на 1-м проходе, так как это поменяет местами все A и D . Однако для сбалансированного слияния 16 начальных отрезков потребовалось бы, чтобы начальные отрезки были возрастающими, а так как мы обычно не знаем заранее, сколько начальных отрезков будет образовано, то необходимо выбрать одно постоянное направление. Следовательно, идея перемотки после 3-го прохода, вероятно, наилучшая.

Каскадное слияние преобразуется таким же способом. Рассмотрим, например, сортировку 14 начальных отрезков на четырех лентах:

	$T1$	$T2$	$T3$	$T4$
Проход 1.	$A_1 A_1 A_1 A_1 A_1 A_1$	$A_1 A_1 A_1 A_1 A_1$	$A_1 A_1 A_1$	—
Проход 2.	—	D_1	$D_2 D_2$	$D_3 D_3 D_3$
Проход 3.	A_6	A_5	A_3	—
Проход 4.	—	—	—	D_{14}

Снова вместо D_{14} можно получить A_{14} , если перемотать $T1, T2, T3$ непосредственно перед последним проходом. Заметим, что это "чистое" каскадное слияние в том смысле, что все однопутевые слияния выполнены явным образом. Если бы мы запретили операции копирования, как в алгоритме 5.4.3D, то после 2-го прохода столкнулись бы с ситуацией

A_1	—	$D_2 D_2$	$D_3 D_3 D_3$
-------	---	-----------	---------------

и было бы невозможно продолжать работу, используя трехпутевое слияние, так как мы не можем сливать отрезки противоположных направлений! Можно было бы избежать операции копирования $T1$ на $T2$, если перемотать $T1$ и начать читать ее в прямом направлении во время следующей фазы слияния (в то время как $T3$ и $T4$ читаются в обратном направлении). Но тогда пришлось бы вновь перемотать $T1$ после слияния, так что этот прием заменяет одно копирование на две перемотки.

Таким образом, метод распределения алгоритма 5.4.3С работает для обратного чтения не столь эффективно, как для прямого чтения; временные затраты резко возрастают каждый раз, когда число начальных отрезков проходит через "точное" каскадное распределение. Чтобы получить более гладкий проход между точными каскадными распределениями, можно использовать иной метод распределения (см. упр. 17).

Обратное чтение в многофазном слиянии. На первый взгляд (и даже на второй и третий!) схема многофазного слияния кажется совершенно неподходящей для обратного чтения. Предположим, например, что имеются 13 начальных отрезков и три ленты.

	$T1$	$T2$	$T3$
Фаза 1.	$A_1 A_1 A_1 A_1 A_1$	$A_1 A_1 A_1 A_1 A_1 A_1 A_1$	—
Фаза 2.	—	$A_1 A_1 A_1$	$D_2 D_2 D_2 D_2 D_2$

Здесь мы встаем в тупик; можно было бы перемотать $T2$ или $T3$ и затем читать их в прямом направлении, в то время как остальные ленты — в обратном. Но это сильно запутало бы дело, и выгода от обратного чтения была бы относительно мала.

Остроумная идея, спасающая положение, состоит в том, чтобы *чередовать направления отрезков на каждой ленте*. Тогда слияние может происходить вполне согласованно:

Фаза 1.	$A_1 D_1 A_1 D_1 A_1$	$D_1 A_1 D_1 A_1 D_1 A_1 D_1 A_1$	—
Фаза 2.	—	$D_1 A_1 D_1$	$D_2 A_2 D_2 A_2 D_2$
Фаза 3.	$A_3 D_3 A_3$	—	$D_2 A_2$
Фаза 4.	A_3	$D_5 A_5$	—
Фаза 5.	—	D_5	D_8
Фаза 6.	A_{13}	—	—

Этот принцип был кратко упомянут Р. Л. Гилстэдом в его ранней статье о многофазном слиянии, более полно он описал его в *САСМ*, 6 (1963), 220–223.

Этот ADA-метод четко работает для многофазного слияния с *любым* числом лент; можно показать, что A и D согласуются соответствующим образом на каждой фазе, при том только условии, что проход начального распределения порождает чередующиеся отрезки A и D на каждой ленте и что каждая лента кончается отрезком A (или каждая лента кончается отрезком D). Так как последний отрезок, записываемый в файл вывода во время одной фазы, имеет направление, противоположное направлению последнего использованного отрезка из файла ввода, то следующая фаза всегда находит свои отрезки с надлежащей ориентацией. Далее, мы видели в упр. 5.4.2-13, что большинство точных фибоначчиевых распределений требует *нечетного* числа отрезков на одной ленте (окончательной выводной ленте) и *четного* числа отрезков на всех остальных лентах. Если T_1 предназначена для конечного вывода, то мы можем, следовательно, гарантировать, что все ленты будут кончаться отрезком A , если ленту T_1 начнем с A , а все остальные ленты — с D . Можно использовать метод распределения, аналогичный алгоритму 5.4.2D, изменив его таким образом, чтобы распределение на каждом уровне имело в качестве выводной ленты T_1 . (Мы пропускаем уровни $1, T+1, 2T+1, \dots$, так как это те уровни, на которых конечной выводной лентой является первоначально пустая лента.) Например, в случае шести лент можно использовать вместо (5.4.2-1) следующее распределение отрезков:

Уровень	T_1	T_2	T_3	T_4	T_5	Сумма	Окончательный вывод на ленте
0	1	0	0	0	0	1	T_1
2	1	2	2	2	2	9	T_1
3	3	4	4	4	2	17	T_1
4	7	8	8	6	4	33	T_1
5	15	16	14	12	8	65	T_1
6	31	30	28	24	16	129	T_1
8	61	120	116	108	92	497	T_1

Таким образом, на T_1 всегда помещается нечетное число отрезков, тогда как на ленты с T_2 по T_5 — четные числа (в убывающем порядке для упрощения присвоения фиктивных отрезков). Такое распределение имеет то преимущество, что конечная выводная лента известна заранее независимо от числа начальных отрезков, которые придется обрабатывать. Оказывается (см. упр. 3), что если используется эта схема, то результат всегда будет на T_1 в возрастающем порядке.

Другой способ осуществить распределение для многофазной схемы с обратным чтением был предложен Д. Т. Гудвином и Дж. Л. Венном [SACM, 7 (1964), 315]. Мы можем распределять отрезки, почти как в алгоритме 5.4.2D, начиная с D -отрезка на каждой ленте. Когда ввод исчерпан, мы представляем себе фиктивный A -отрезок расположенным в начале единственной "нечетной" ленты, если только не достигнуто распределение со всеми нечетными числами. Остальные фиктивные отрезки мы представляем себе расположенными в конце лент или сгруппированными в пары в середине. Вопрос об оптимальном размещении фиктивных отрезков анализируется ниже в упр. 5.

Оптимальные схемы слияния. До сих пор мы обсуждали различные схемы слияния с лентами, не пытаясь найти метод, "наилучший из возможных". Определение оптимальной схемы кажется особенно сложным в случае прямого чтения, где взаимодействие времени перемотки и времени слияния с трудом поддается анализу. С другой стороны, если слияние осуществляется посредством обратного чтения и прямой записи, то, по существу, все перемотки устраняются и оказывается возможным дать довольно хорошую характеристику оптимальным схемам слияния. Ричард М. Карп предложил несколько интересных подходов к этой задаче, и мы завершаем этот пункт обсуждением развитой им теории.

Во-первых, нам необходим более удобный способ описания схем слияния вместо довольно таинственных таблиц "содержимого лент", которые использовались выше. Карп предложил два таких способа — *векторное представление схемы слияния* и *представление в виде дерева*. Оба представления полезны на практике, и мы опишем их по очереди.

Векторное представление схемы слияния состоит из последовательности "векторов слияния" $y^m \dots y^1 y^0$, каждый из которых имеет T компонент; $y^{(i)}$ следующим образом изображает i -й с конца шаг слияния:

$$y_j^{(i)} = \begin{cases} +1, & \text{если лента с номером } j \text{ является вводной для данного слияния;} \\ 0, & \text{если лента с номером } j \text{ не используется в данном слиянии;} \\ -1, & \text{если на ленту с номером } j \text{ выводится результат данного слияния.} \end{cases} \quad (2)$$

Таким образом, ровно одна компонента $y^{(i)}$ равна -1 , остальные компоненты равны 0 и 1 . Итоговый вектор $y^{(0)}$ особый; это единичный вектор, имеющий 1 в позиции j , если окончательный отсортированный результат оказывается на устройстве j , и 0 в остальных местах. Из этого определения следует,

что векторная сумма

$$v^{(i)} = y^{(i)} + y^{(i-1)} + \dots + y^{(0)} \quad (3)$$

представляет собой распределение отрезков на лентах непосредственно перед i -м с конца шагом слияния, причем на ленте j находится $v_j^{(i)}$ отрезков. В частности, по $v^{(m)}$ можно судить, сколько отрезков помещает на каждую ленту начальный проход распределения.

Три схемы слияния, описанные ранее в этом пункте в табличной форме, имеют следующие векторные представления:

Сбалансированная ($T = 4, S = 8$)	Каскадная ($T = 4, S = 14$)	Многофазная ($T = 3, S = 13$)
$v^{(7)} = (4, 4, 0, 0)$	$v^{(10)} = (6, 5, 3, 0)$	$v^{(12)} = (5, 8, 0)$
$y^{(7)} = (+1, +1, -1, 0)$	$y^{(10)} = (+1, +1, +1, -1)$	$y^{(12)} = (+1, +1, -1)$
$y^{(6)} = (+1, +1, 0, -1)$	$y^{(9)} = (+1, +1, +1, -1)$	$y^{(11)} = (+1, +1, -1)$
$y^{(5)} = (+1, +1, -1, 0)$	$y^{(8)} = (+1, +1, +1, -1)$	$y^{(10)} = (+1, +1, -1)$
$y^{(4)} = (+1, +1, 0, -1)$	$y^{(7)} = (+1, +1, -1, 0)$	$y^{(9)} = (+1, +1, -1)$
$y^{(3)} = (-1, 0, +1, +1)$	$y^{(6)} = (+1, +1, -1, 0)$	$y^{(8)} = (+1, +1, -1)$
$y^{(2)} = (0, -1, +1, +1)$	$y^{(5)} = (+1, -1, 0, 0)$	$y^{(7)} = (-1, +1, +1)$
$y^{(1)} = (+1, +1, -1, 0)$	$y^{(4)} = (-1, +1, +1, +1)$	$y^{(6)} = (-1, +1, +1)$
$y^{(0)} = (0, 0, 1, 0)$	$y^{(3)} = (0, -1, +1, +1)$	$y^{(5)} = (-1, +1, +1)$
	$y^{(2)} = (0, 0, -1, +1)$	$y^{(4)} = (+1, -1, +1)$
	$y^{(1)} = (+1, +1, +1, -1)$	$y^{(3)} = (+1, -1, +1)$
	$y^{(0)} = (0, 0, 0, 1)$	$y^{(2)} = (+1, +1, -1)$
		$y^{(1)} = (-1, +1, +1)$
		$y^{(0)} = (1, 0, 0)$

Может показаться неудобным нумеровать эти векторы с конца так, чтобы $y^{(m)}$ появлялся первым, а $y^{(0)}$ — последним, но эта особая точка зрения оказывается предпочтительной при разработке теории. Для поиска оптимального метода неплохо начать с отсортированного вывода и представить себе, как его можно "разлить" на различные ленты, затем "разлить" их и т. д., рассматривая последовательные распределения $v^{(0)}, v^{(1)}, v^{(2)}, \dots$ в порядке, обратном тому, в котором они в действительности появляются в процессе сортировки. Фактически именно этот подход был уже использован нами при анализе многофазного и каскадного слияния.

Каждая схема слияния, очевидно, имеет векторное представление. И обратно, как легко видеть, последовательность векторов $y^{(m)} \dots y^{(1)} y^{(0)}$ соответствует реальной схеме слияния тогда и только тогда, когда выполняются следующие три условия:

- i) вектор $y^{(0)}$ является единичным;
- ii) ровно одна компонента вектора $y^{(i)}$ равна -1 , все остальные компоненты равны 0 или $+1$,
 $m \geq i \geq 1$;
- iii) все компоненты вектора $y^{(i)} + \dots + y^{(1)} + y^{(0)}$ неотрицательны, $m \geq i \geq 1$.

Представление схемы слияния в виде дерева дает другое изображение той же информации. Мы строим дерево с одним внешним "лиственным" узлом для каждого начального отрезка и с одним внутренним узлом для каждого отрезка, полученного в результате слияния, таким образом, что потомками любого внутреннего узла являются отрезки, из которых он был сформирован. Каждый внутренний узел помечается номером шага, на котором был образован соответствующий отрезок, при этом шаги нумеруются в обратном порядке, как в векторном представлении; кроме того, ребра непосредственно над каждым узлом помечаются именем ленты, на которой оказывается этот отрезок. Например, три приведенные выше схемы имеют представления в виде дерева, изображенные на рис. 76, если мы назовем ленты A, B, C, D , а не $T1, T2, T3, T4$.

Это удобное и наглядное представление многих существенных свойств схемы слияния; например, если отрезок на уровне 0 дерева (корень) должен быть возрастающим, то отрезки на уровне 1 должны быть убывающими, отрезки на уровне 2 — возрастающими и т. д.; некоторый начальный отрезок является возрастающим тогда и только тогда, когда соответствующий внешний узел находится на уровне с четным номером. Далее, суммарное количество начальных отрезков, обрабатываемых при слиянии (не включая начальное распределение), в точности равно длине внешнего пути дерева, так как каждый начальный отрезок на уровне k обрабатывается ровно k раз.

Picture: Рис. 76. Представления трех схем слияния в виде дерева.

Любая схема слияния имеет представление в виде дерева, но не каждое дерево определяет схему слияния. Дерево, внутренние узлы которого помечены числами от 1 до m и ребра которого помечены

именами лент, изображает правильную схему слияния с обратным чтением тогда и только тогда, когда

- а) никакие два ребра, смежные с одним внутренним узлом, не имеют одинакового имени ленты;
- б) если $i > j$ и если A есть имя ленты, то дерево не содержит конфигурации

Picture: p.362

- в) если $i < j < k < l$ и если A —имя ленты, то дерево не содержит таких пар:

Picture: p.363.1

Условие (а) очевидно, так как вводные и выводная ленты слияния должны быть различны; условие (б) также очевидно. Условие "непересечения" (в) отражает характерное для операций обратного чтения ленты ограничение "последним включается — первым исключается". Отрезок, образованный на шаге k , должен быть удален прежде отрезка, сформированного ранее на той же ленте; следовательно, конфигурации (4) невозможны. Нетрудно проверить, что любое помеченное дерево, удовлетворяющее условиям (а), (б), (в), действительно соответствует некоторой схеме слияния с обратным чтением.

Если имеется T ленточных устройств, то из условия (а) следует, что степень каждого внутреннего узла равна $T - 1$ или меньше. Приписывание подходящих меток всем таким деревьям не всегда возможно; например, если $T = 3$, то не существует схемы слияния с деревом вида

Picture: p.363.2

Такая форма дерева привела бы к оптимальной схеме слияния, если бы мы смогли соответствующим образом приписать номера шагов и номера лент, поскольку это единственное дерево с минимальной длиной внешнего пути среди деревьев, имеющих четыре внешних узла. Но в силу симметрии этой диаграммы имеется, по существу, только один способ пометить ее, соблюдая условия (а) и (б):

Picture: p.363.3

Однако при этом нарушается условие (в). Дерево, которое может быть помечено в соответствии с упомянутыми условиями с использованием T или меньше имен лент, называется T -lifo¹² деревом.

Другой способ характеризовать все помеченные деревья, которые могут возникнуть из схемы слияния, состоит в том, чтобы рассмотреть, как все подобные деревья могут быть "выращены". Начнем с некоторого имени ленты, скажем A , и с ростка дерева

Picture: p.364.1

Шаг i роста дерева состоит в выборе различных имен лент B, B_1, B_2, \dots, B_k и замене *позже всего образованного* узла, соответствующего B ,

Picture: p.364.2

Это правило "последним образован—первым растет" в точности описывает способ построения представления в виде дерева непосредственно из векторного представления.

Определение строго оптимальной T -ленточной схемы слияния, т. е. дерева, имеющего минимальную длину пути среди всех T -lifo деревьев с данным числом внешних узлов, кажется весьма трудной задачей. Следующая неочевидная схема, например, оказывается наилучшим способом слить семь начальных отрезков, имея четыре ленты и читая в обратном направлении: Однопутевое слияние необходимо по существу для достижения оптимума! (См. упр. 8.) С другой стороны, не так трудно дать конструкции, *асимптотически* оптимальные для любого фиксированного T .

Пусть $K_T(n)$ —минимальная длина внешнего пути, достижимая в T -lifo дереве с n внешними узлами. Используя теорию, развитую в п. 2.3.4.5, нетрудно доказать, что

$$K_T(n) \geq nq - \lfloor ((T-1)^q - n)/(T-2) \rfloor, \quad q = \lceil \log_{T-1} n \rceil, \quad (9)$$

¹² "Lifo"—аббревиатура для "last in—first out" (последним включается— первым исключается).—
Прим. перев.

так как это минимальная длина внешнего пути любого дерева с n внешними узлами и степенью любого узла $< T$. К настоящему моменту известны относительно немногие точные значения $K_T(n)$. Здесь приведены некоторые верхние оценки, которые, вероятно, точны:

$$\begin{array}{cccccccccccccccc} n = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ K_3(n) \leq & 0 & 2 & 5 & 9 & 12 & 16 & 21 & 25 & 30 & 34 & 39 & 45 & 50 & 56 & 61 \\ K_4(n) \leq & 0 & 2 & 3 & 6 & 8 & 11 & 14 & 17 & 20 & 24 & 27 & 31 & 33 & 37 & 40 \end{array} \quad (10)$$

Карп обнаружил, что любое дерево с внутренними узлами степени $< T$ является *почти T-lifo* деревом в том смысле, что оно может быть сделано *T-lifo* с помощью замены некоторых внешних узлов на однопутевые слияния. Фактически построение подходящей расстановки меток выполняется, довольно просто. Пусть A —конкретное имя ленты. Будем действовать следующим образом:

Шаг 1. Приписать имена лент ребрам диаграммы дерева любым способом, совместимым с условием (а), указанным выше, однако так, чтобы специальное имя A использовалось только в самом левом ребре ветви.

Шаг 2. Заменить каждый внешний узел вида

Picture: p.365

если только $B \neq A$.

Шаг 3. Занумеровать внутренние узлы в прямом порядке¹³. Результатом будет расстановка меток, удовлетворяющая условиям (а), (б), (с).

Например, если мы начнем с дерева

Picture: p.366.1

и трех лент, то эта процедура могла бы приписать метки таким образом:

Picture: p.366.2

Нетрудно проверить, что конструкция Карпа удовлетворяет дисциплине "последним образован—первым растет" в силу свойств прямого порядка (см. упр. 12).

Заметим, что результатом этого построения является схема слияния, в которой все начальные отрезки появляются на ленте A . Это предполагает следующую схему распределения и сортировки, которую можно назвать слиянием в *прямом порядке*.

- P1.** Распределять начальные отрезки на ленту A , пока ввод не будет исчерпан. Пусть S —число всех начальных отрезков.
- P2.** Выполнять приведенное выше построение, используя $(T - 1)$ -арное дерево с S внешними узлами с минимальной длиной пути, получая *T-lifo* дерево, длина внешнего пути которого превышает нижнюю границу (9) не более, чем на S .
- P3.** Сливать отрезки в соответствии с этой схемой. ■

Результат в указанной схеме получается на какой угодно ленте. Но *эта схема имеет один серьезный изъян*. (Видит ли читатель, что именно здесь будет неправильно работать?) Дело в том, что схема требует, чтобы первоначально некоторые из отрезков на A были возрастающими, а некоторые—убывающими в зависимости от того, появляется ли соответствующий внешний узел на нечетном или четном уровне. Эту проблему можно разрешить, не зная S заранее, путем копирования отрезков, которые должны быть убывающими, на вспомогательную ленту (или ленты) непосредственно перед тем, как они требуются. Тогда суммарное количество операций, измеряемое в длинах начальных отрезков, окажется равным

$$S \log_{T-1} S + O(S). \quad (13)$$

Таким образом, слияние в прямом порядке определенно лучше многофазного или каскадного при $S \rightarrow \infty$; в действительности оно асимптотически *оптимально*, так как (9) показывает, что $S \log_{T-1} S + O(S)$ —это наилучшее, что мы вообще можем надеяться получить с T лентами. С другой стороны, для сравнительно небольших значений S , обычно встречающихся на практике, слияние в прямом порядке весьма неэффективно; многофазный или каскадный методы проще и быстрее, если S относительно

¹³ См. п. 2.3.1.—Прим. перев.

мало. Возможно, удастся изобрести простую схему распределения и слияния, которая сравнима с многофазной и каскадной для небольших S и асимптотически оптимальна при больших S .

Ниже в упражнениях второй части демонстрируется, как Карп аналогичным образом поставил вопрос для слияния с *прямым чтением*. Теория оказывается в этом случае значительно более сложной, хотя были получены некоторые весьма интересные результаты.

Упражнения

(ПЕРВАЯ ЧАСТЬ)

1. [17] При слиянии с прямым чтением часто удобно отмечать конец каждого отрезка на ленте путем добавления искусственной "концевой" записи с ключом $+\infty$. Как следует видоизменить этот метод при обратном чтении?
2. [20] Будут ли столбцы таблицы, аналогичной (1), всегда неубывающими, или бывают случаи, когда нам придется "вычитать" отрезки с некоторой ленты при переходе от одного уровня к следующему?
- >3. [20] Докажите, что метод многофазного распределения, описанный в связи с (1), дает после завершения сортировки на ленте T_1 , всегда отрезок A , если первоначально на T_1 было $ADA\dots$, а на T_2-T_5 было $DAD\dots$.
4. [22] Как вы оцениваете идею выполнять многофазное слияние с обратным чтением, распределив все отрезки в *возрастающем* порядке и считая, что все позиции "D" первоначально заполнены фиктивными отрезками?
- >5. [23] Какие формулы для строк чисел слияния вместо (8), (9), (10) и (11) из 5.4.2 будут справедливы для многофазного слияния с обратным чтением? Изобразите числа слияния для распределения пятого уровня на шести лентах, нарисовав диаграмму, аналогичную рис. 71(a).
6. [07] Каково векторное представление схемы слияния, представлением которой в виде дерева является (8)?
7. [16] Нарисуйте представление в виде дерева схемы слияния с обратным чтением, определенной следующей последовательностью векторов:

$$\begin{aligned}
 v^{(33)} &= (20, 9, 5) & y^{(22)} &= (+1, -1, +1) & y^{(10)} &= (+1, +1, -1) \\
 y^{(33)} &= (+1, -1, +1) & y^{(21)} &= (-1, +1, +1) & y^{(9)} &= (+1, -1, +1) \\
 y^{(32)} &= (+1, +1, -1) & y^{(20)} &= (+1, +1, -1) & y^{(8)} &= (+1, +1, -1) \\
 y^{(31)} &= (+1, +1, -1) & y^{(19)} &= (-1, +1, +1) & y^{(7)} &= (+1, +1, -1) \\
 y^{(30)} &= (+1, +1, -1) & y^{(18)} &= (+1, +1, -1) & y^{(6)} &= (+1, +1, -1) \\
 y^{(29)} &= (+1, -1, +1) & y^{(17)} &= (+1, +1, -1) & y^{(5)} &= (-1, +1, +1) \\
 y^{(28)} &= (-1, +1, +1) & y^{(16)} &= (+1, +1, -1) & y^{(4)} &= (+1, -1, +1) \\
 y^{(27)} &= (+1, -1, +1) & y^{(15)} &= (+1, +1, -1) & y^{(3)} &= (-1, +1, +1) \\
 y^{(26)} &= (+1, -1, +1) & y^{(14)} &= (+1, -1, +1) & y^{(2)} &= (+1, -1, +1) \\
 y^{(25)} &= (+1, +1, -1) & y^{(13)} &= (+1, -1, +1) & y^{(1)} &= (-1, +1, +1) \\
 y^{(24)} &= (+1, -1, +1) & y^{(12)} &= (-1, +1, +1) & y^{(0)} &= (1, 0, 0) \\
 y^{(23)} &= (+1, -1, +1) & y^{(11)} &= (+1, +1, -1) & &
 \end{aligned}$$

8. [23] Докажите, что (8)—оптимальный способ слияния с обратным чтением при $S = 7$ и $T = 4$ и что все методы, избегающие однопутевого слияния, хуже.
9. [M22] Докажите нижнюю оценку (9).
10. [41] При помощи ЭВМ составьте таблицу точных значений $K_T(n)$.
- >11. [20] Верно ли утверждение, что для любой схемы слияния с обратным чтением, не использующей ничего, кроме $(T - 1)$ -путевого слияния, необходимо, чтобы отрезки на каждой ленте чередовались: $ADAD\dots$, т. е. она не будет работать, если два соседних отрезка окажутся одинаково упорядоченными?
12. [22] Докажите, что конструкция с прямым порядком Карпа всегда порождает помеченное дерево, удовлетворяющее условиям (a), (b), (c).
13. [16] Сделайте (12) более эффективным, удалив как можно больше однопутевых слияний, однако так, чтобы прямой порядок все еще давал правильную расстановку меток у внутренних узлов.
14. [40] Придумайте алгоритм, который выполняет слияние в прямом порядке без явного построения дерева в шагах P2 и P3, используя только $O(\log S)$ слов памяти для управления слиянием.
15. [M39] Конструкция Карпа с прямым порядком порождает деревья с однопутевым слиянием в некоторых терминальных узлах. Докажите, что если $T = 3$, то можно построить асимптотически оптимальные 3-lifo деревья, в которых используется только двухпутевое слияние.

Другими словами, пусть $\hat{K}_T(n)$ будет минимальной длиной внешнего пути среди всех T -lifo деревьев с n внешними узлами, таких, что каждый внутренний узел имеет степень $T - 1$. Докажите, что $\hat{K}_3(n) = n \log_2 n + O(n)$.

16. [M46] (Сохраняются обозначения упр. 15) Верно ли, что $\hat{K}_T(n) = n \log_{T-1}(n) + O(n)$ для всех $T \geq 3$, если $n \equiv 1 \pmod{T-1}$?
- >17. [28] (Ричард Д. Пратт.) Чтобы получить возрастающий порядок в каскадном слиянии с обратным чтением, мы могли бы потребовать *четного* числа проходов слияния; это предполагает, что метод начального распределения несколько отличен от алгоритма 5.4.3С. (а) Измените (5.4.3-1) так, чтобы были представлены только точные распределения, которые требуют четного числа проходов слияния. (б) Сконструируйте схему начального распределения, осуществляющую интерполяцию между этими точными распределениями. [Иначе говоря, если число начальных отрезков попадает между точными распределениями, то желательно слить некоторые (но не все) отрезки дважды, чтобы достигнуть точного распределения.]
18. [46] Предположим, что имеется T ленточных устройств для некоторого $T \geq 3$ и что T_1 содержит N записей, в то время как остальные ленты пусты. Возможно ли обратить порядок записей на T_1 за число шагов, меньшее $O(N \log N)$, без обратного чтения? (Эта операция является, конечно, тривиальной, если допускается обратное чтение.) См. в упр. 5.2.5-14 класс таких алгоритмов, которые, однако, тратят порядка $N \log N$ шагов.

Упражнения

(ВТОРАЯ ЧАСТЬ)

Следующие упражнения развивают теорию ленточного слияния с прямым чтением. В этом случае каждая лента действует как очередь, а не как стек. Схему слияния можно представить последовательностью векторов $y^{(n)} \dots y^{(1)}y^{(0)}$ точно так же, как в тексте, но когда мы преобразуем векторное представление в представление в виде дерева, то мы заменяем правило "последним образован—первым растет" на "первым образован—первым растет". Таким образом, недопустимая конфигурация (4) должна быть заменена на

Picture: p.369

Дерево, которое может быть помечено так, чтобы изображать слияние с прямым чтением на T лентах, называется T -fifo¹⁴ по аналогии с термином T -lifo в случае обратного чтения.

Если ленты можно прочитать в обратном направлении, они образуют очень хорошие стеки. Но, к сожалению, они не могут образовать очень хорошие универсальные очереди. Если мы в произвольном порядке записываем и читаем по правилу "первым включается—первым исключается", то приходится тратить много времени на перемотку от одной части ленты к другой. Хуже того—мы вскоре проскочим конец ленты! Мы сталкиваемся с той же проблемой, что и проблема выхода очереди за границу памяти [см. соотношения (2.2.2-4) и (2.2.2-5)], но решение в виде (2.2.2-6) и (2.2.2-7) не применимо к лентам, так как они не являются кольцевыми. Поэтому дерево будем называть *сильным T -fifo* деревом, если оно может быть помечено так, чтобы соответствующая схема слияния использовала ленты, подчиняясь особой дисциплине: "записать, перемотать, прочитать все, перемотать; записать, перемотать, прочитать все, перемотать; и т. д."

1. [22] (Р. М. Карп.) Найдите бинарное дерево, которое не является 3-fifo.
- >2. [22] Сформулируйте условие "сильного T -fifo" дерева в терминах достаточно простого правила относительно недопустимых конфигураций меток лент, аналогичного (4').
3. [18] Нарисуйте представление в виде дерева схемы слияния с прямым чтением; определенной посредством векторов в упр. 7. Является ли это дерево сильным 3-fifo?
4. [28] (Р. М. Карп.) Докажите, что представления в виде дерева многофазного и каскадного слияния с точным распределением полностью одинаковы как в случае обратного чтения, так и в случае прямого чтения, за исключением номеров, которыми помечены внутренние узлы. Найдите более широкий класс векторных, представлений схем слияния, для которых это верно.
5. [24] (Р. М. Карп.) Будем говорить, что отрезок $y^{(a)} \dots y^{(r)}$ схемы слияния является *стадией*, если никакая выводная лента не используется в дальнейшем как вводная, т. е. если не существует i, j, k , таких, что $q \geq i > k \geq r$ и $y_j^{(i)} = -1, y_j^{(k)} = +1$. Цель этого упражнения—доказать, что *каскадное слияние минимизирует число стадий* среди всех схем слияния с тем же числом лент и начальных отрезков.

¹⁴ "Fifo"—аббревиатура от "first in first out" (первым включается—первым исключается).—Прим. перев.

Удобно ввести некоторые обозначения. Будем писать $v \rightarrow w$, если v и w —такие T -векторы, что существует схема слияния, которая в своей первой стадии переводит w в v (т. е. существует схема слияния $y^{(m)} \dots y^{(0)}$, такая, что $y^{(m)} \dots y^{(l+1)}$ является стадией, $w = y^{(m)} + \dots + y^{(0)}$ и $v = y^{(l)} + \dots + y^{(0)}$). Будем писать $v \preceq w$, если v и w — T -векторы, такие, что сумма наибольших k элементов вектора v не превышает суммы наибольших k элементов вектора w при $1 \leq k \leq T$. Так, например, $(2, 1, 2, 2, 2, 1) \preceq (1, 2, 3, 0, 3, 1)$, так как $2 \leq 3, 2 + 2 \leq 3 + 3, \dots, 2 + 2 + 2 + 2 + 1 + 1 \leq 3 + 3 + 2 + 1 + 1 + 0$. Наконец, если $v = (v_1, \dots, v_T)$, то пусть $C(v) = (s_T, s_{T-2}, s_{T-3}, \dots, s_1, 0)$, где s_k есть сумма наибольших k элементов вектора v .

(а) Докажите, что $v \rightarrow C(v)$. (б) Докажите, что $v \preceq w$ влечет $C(v) \preceq C(w)$. (с) Считая известным результат упр. 24, докажите, что каскадное слияние минимизирует число стадий.

6. [M35] Используя обозначения упр. 23, докажите, что $v \rightarrow w$ влечет $w \preceq C(v)$.

7. [M36] (Р. М. Карп.) Будем говорить, что сегмент $y^{(a)} \dots y^{(r)}$ схемы слияния является *фазой*, если ни одна из лент не используется и для ввода, и для вывода, т. е. если не существует i, j, k , таких, что $q \geq i, k \geq r$ и $y_j^{(i)} = +1, y_j^{(k)} = -1$. Цель этого упражнения—исследовать схему слияния, которая минимизирует число фаз. Мы будем писать $v \Rightarrow w$, если w может быть преобразовано в v за одну фазу (ср. с подобным обозначением, введенным в упр. 23), и пусть $D_k(v) = (s_k + t_{k+1}, s_k + t_{k+2}, \dots, s_k + t_T, 0, \dots, 0)$, где t_j обозначает j -й в порядке убывания элемент v и $s_k = t_1 + \dots + t_k$. (а) Докажите, что $v \Rightarrow D_k(v)$ при $1 \leq k < T$. (б) Докажите, что из $v \preceq w$ следует $D_k(v) \preceq D_k(w)$ при $1 \leq k < T$. (с) Докажите, что из $v \Rightarrow w$ следует $w \preceq D_k(v)$ для некоторого $k, 1 \leq k < T$. (д) Следовательно, схема слияния, сортирующая максимальное число начальных отрезков на T лентах за q фаз, может быть изображена последовательностью целых чисел $k_1 k_2 \dots k_q$, такой, что начальное распределение есть $D_{k_q}(\dots D_{k_2}(D_{k_1}(u)) \dots)$, где $u = (1, 0, \dots, 0)$. Эта стратегия минимума фаз, имеет сильное T -fifo представление, и она также входит в класс схем упр. 22. Когда $T = 3$, это *многофазная* схема, а при $T = 4, 5, 6, 7$ это вариация *сбалансированной* схемы.

8. [M46] (Р. М. Карп). Верно ли, что оптимальная последовательность $k_1 k_2 \dots k_q$, упомянутая в упр. 25, всегда равна $1 \lceil T/2 \rceil \lfloor T/2 \rfloor \lceil T/2 \rceil \lfloor T/2 \rfloor \dots$ для всех $T \geq 4$ и всех достаточно больших q ?

5.4.5. Осциллирующая сортировка

Еще один подход к сортировке слиянием был предложен Шелдоном Собедем в [JACM, 9 (1962), 372–375]. Вместо того чтобы начинать с прохода распределения, когда все начальные отрезки распределяются по лентам, он предложил алгоритм, который переключается то на распределение, то на слияние, так что большая часть сортировки происходит еще до того, как вся исходная информация будет полностью просмотрена. Предположим, например, что для слияния используется пять лент. По методу Собеда 16 начальных отрезков будут сортироваться следующим образом:

Операция	$T1$	$T2$	$T3$	$T4$	$T5$	Стоимость
Фаза 1. Распределение	A_1	A_1	A_1	A_1	—	4
Фаза 2. Слияние	—	—	—	—	D_4	4
Фаза 3. Распределение	—	A_1	A_1	A_1	$D_4 A_1$	4
Фаза 4. Слияние	D_4	—	—	—	D_4	4
Фаза 5. Распределение	$D_4 A_1$	—	A_1	A_1	$D_4 A_1$	4
Фаза 6. Слияние	D_4	D_4	—	—	D_4	4
Фаза 7. Распределение	$D_4 A_1$	$D_4 A_1$	—	A_1	$D_4 A_1$	4
Фаза 8. Слияние	D_4	D_4	D_4	—	D_4	4
Фаза 9. Слияние	—	—	—	A_{16}	—	16

Здесь, как и в п. 5.4.4, мы используем A_r и D_r для обозначения соответственно возрастающих и убывающих отрезков относительной длины r . Рассматриваемый метод начинается с записи по одному начальному отрезку на каждую из четырех лент и сливает их (читая в обратном направлении) на пятую ленту. Опять возобновляется распределение, на этот раз циклически сдвинутое на 1 вправо по отношению к лентам, и второе слияние дает еще один отрезок D_4 . Когда этим способом сформированы четыре отрезка D_4 , дополнительное слияние создает A_{16} . Процесс можно продолжать, создавая еще три A_{16} , сливая их в D_{64} и т. д. до тех пор, пока не исчерпаются исходные данные. Не нужно знать заранее длину исходных данных.

Если число начальных отрезков S есть 4^m , то нетрудно видеть, что этот метод обрабатывает каждую запись ровно $m + 1$ раз (один раз во время распределения и m раз во время слияния). Если S лежит между 4^{m-1} и 4^m , то можно с помощью фиктивных отрезков увеличить S до 4^m ; следовательно, общее время сортировки будет определяться $\lceil \log_4 S \rceil + 1$ проходами по всем данным. Это как раз то, что достигается при сбалансированной сортировке на *восьми* лентах; в общем случае осциллирующая сортировка с T рабочими лентами эквивалентна сбалансированному слиянию с $2(T - 1)$ лентами, так

как она делает $\lceil \log_{T-1} S \rceil + 1$ проходов по данным. Если S оказывается степенью $T - 1$, то это самое лучшее, что можно получить при *любом* методе с T лентами, так как здесь достигается нижняя оценка из соотношения (5.4.4-9). С другой стороны, если S равно $(T - 1)^{m-1} + 1$, т. е. ровно на единицу больше степени $T - 1$, то этот метод теряет почти целый проход.

В упр. 2 показано, как устранить часть этой лишней работы, используя специальную программу окончания. Еще одно усовершенствование было предложено в 1966 г. Денисом Л. Бэнчером, который назвал свою процедуру перекрестным слиянием. [См. Н. Wedekind, Datenorganisation (Berlin W. de Gruyter, 1970). 164–166, и U. S. Patent 3540000 (10 ноября 1970).] Основная идея состоит в том, чтобы отложить слияние до тех пор, пока не будет накоплено больше сведений об S . Мы обсудим несколько измененную форму первоначальной схемы Бэнчера.

Эта улучшенная осциллирующая сортировка действует следующим образом:

	Операция	$T1$	$T2$	$T3$	$T4$	$T5$	Стоимость
Фаза 1.	Распределение	–	A_1	A_1	A_1	A_1	4
Фаза 2.	Распределение	–	A_1	A_1A_1	A_1A_1	A_1A_1	3
Фаза 3.	Слияние	D_4	–	A_1	A_1	A_1	4
Фаза 4.	Распределение	D_4A_1	–	A_1	A_1A_1	A_1A_1	3
Фаза 5.	Слияние	D_4	D_4	–	A_1	A_1	4
Фаза 6.	Распределение	D_4A_1	D_4A_1	–	A_1	A_1A_1	3
Фаза 7.	Слияние	D_4	D_4	D_4	–	A_1	4
Фаза 8.	Распределение	D_4A_1	D_4A_1	D_4A_1	–	A_1	3
Фаза 9.	Слияние	D_4	D_4	D_4	D_4	–	4

В этот момент мы не сливаем все D_4 в A_{16} , (если только не окажется, что исходные данные исчерпаны); лишь после того, как закончится

Фаза 15. Слияние D_4D_4 D_4D_4 D_4D_4 D_4 – 4

будет получен первый отрезок A_{16} :

Фаза 16. Слияние D_4 D_4 D_4 – A_{16} 16

Второй отрезок A_{16} появится после создания еще трех D_4 :

Фаза 22. Слияние D_4D_4 D_4D_4 D_4 – $A_{16}D_4$ 4

Фаза 23. Слияние D_4 D_4 – A_{16} A_{16} 16

и т. д. (ср. с фазами 1–5). Преимущества схемы Бэнчера можно видеть, если имеется, например, только пять начальных отрезков: осциллирующая сортировка (ее модификация из упр. 2) выполняла бы четырехпутевое слияние (на фазе 2), за которым следовало бы двухпутевое слияние с общей стоимостью $4 + 4 + 4 + 1 + 5 = 14$, тогда как схема Бэнчера выполняла бы двухпутевое слияние (на фазе 3), за которым следовало бы четырехпутевое слияние с общей стоимостью $4 + 1 + 2 + 5 = 12$. (Оба метода также требуют небольших дополнительных затрат, именно однократной перемотки перед окончательным слиянием.)

Точное описание метода Бэнчера содержится ниже в алгоритме В. К сожалению, соответствующую процедуру, по-видимому, трудней понять, чем запрограммировать; легче объяснить этот метод ЭВМ, чем программисту! Частично это происходит по той причине, что рекурсивный метод выражен в итеративном виде и затем подвергнут некоторой оптимизации; читатель, возможно, обнаружит, что необходимо несколько раз проследить за работой алгоритма, чтобы действительно осознать, что же происходит.

Алгоритм В. (Осциллирующая сортировка с перекрестным распределением.) Этот алгоритм берет первоначальные отрезки и распределяет их по лентам, время от времени прерывая процесс распределения, чтобы слить содержимое некоторых лент.

Picture: Рис. 77. Осциллирующая сортировка с перекрестным распределением.

В алгоритме используется P -путевое слияние и предполагается, что есть $T = P + 1 \geq 3$ ленточных устройств (не считая устройства, которое может быть необходимо для хранения исходных данных). Ленточные устройства должны допускать чтение как в прямом, так и в обратном направлении; они обозначены числами $0, 1, \dots, P$. Используются следующие массивы:

$D[j]$, $0 \leq j \leq P$ —число фиктивных отрезков, наличие которых предполагается в конце ленты j .

$A[l, j]$, $0 \leq l \leq L$, $0 \leq j \leq P$. Здесь L —достаточно большое число, такое, что будет введено не более P^{L+1} начальных отрезков. Если $A[l, j] = k \geq 0$, то на ленте j имеется отрезок

номинальной длины P^k , соответствующий "уровню l " работы алгоритма. Этот отрезок возрастающий, если k четно, и убывающий, если k нечетно. $A[l, j] = -1$ означает, что на уровне l лента j не используется.

Инструкция "записать начальный отрезок на ленту j " является сокращенным обозначением следующих действий:

Установить $A[l, j] \leftarrow 0$. Если исходные данные исчерпаны, то увеличить $D[j]$ на 1; в противном случае записать отрезок на ленту j (в возрастающем порядке).

Инструкция "слить на ленту j " используется как краткое обозначение следующих действий:

Если $D[i] > 0$ для всех $i \neq j$, то уменьшить $D[i]$ на 1 при всех $i \neq j$ и увеличить $D[j]$ на 1. В противном случае слить один отрезок на ленту j со всех лент $i \neq j$, таких, что $D[i] = 0$, и уменьшить $D[i]$ на 1 для всех остальных $i \neq j$.

В1 [Начальная установка.] Установить $D[j] \leftarrow 0$ при $0 \leq j \leq P$. Затем записать начальный отрезок на ленту j при $1 \leq j \leq P$. Установить $A[0, 0] \leftarrow -1, l \leftarrow 0, q \leftarrow 0$.

В2 [Ввод завершен?] (В этот момент лента q пуста и всякая другая лента содержит самое большое один отрезок.) Если еще есть исходные данные, перейти к шагу **В3**. Однако если ввод исчерпан, то перемотать все ленты $j \neq q$, такие, что $A[0, j]$ четно; затем слить на ленту q , читая все только что перемотанные ленты в прямом направлении, а остальные ленты — в обратном. Этим завершается сортировка; результат находится на ленте q в возрастающем порядке.

В3 [Начать новый уровень.] Установить $l \leftarrow l + 1, r \leftarrow q, s \leftarrow 0$ и $q \leftarrow (q + 1) \bmod T$. Записать начальный отрезок на ленту $(q + j) \bmod T$ при $1 \leq j \leq T - 2$. (Таким образом, начальные отрезки записываются на все ленты, кроме лент q и r .) Установить $A[l, q] \leftarrow -1$ и $A[l, r] \leftarrow -1$.

В4 [Можно ли сливать?] Если $A[l - 1, q] \neq s$, вернуться к шагу **В3**.

В5 [Слияние.] (В этот момент $A[l - 1, q] = A[l, j] = s$ при всех $j \neq q, j \neq r$.) Слить на ленту r . (См. выше определение этой операции.) Затем установить $s \leftarrow s + 1, l \leftarrow l - 1, A[l, r] \leftarrow s$ и $A[l, q] \leftarrow -1$. Установить $r \leftarrow (2q - r) \bmod T$. (В общем случае мы имеем $r = (q - 1) \bmod T$, если s четно, и $r = (q + 1) \bmod T$, если s нечетно.)

В6 [Закончен ли уровень?] Если $l = 0$, перейти к **В2**. В противном случае, если $A[l, j] = s$ для всех $j \neq q$ и $j \neq r$, то перейти к **В4**. В противном случае вернуться к **В3**. ■

Чтобы показать правильность этого алгоритма, мы можем использовать доказательство типа "рекурсивной индукции", так же как мы делали для алгоритма 2.3.1Т. Предположим, что мы начинаем с шага **В3** с $l = l_0, q = q_0, s_+ = A[l_0, (q_0 + 1) \bmod T]$ и $s_- = A[l_0, (q_0 - 1) \bmod T]$, и допустим, кроме того, что либо $s_+ = 0$, либо $s_- = 1$, либо $s_+ = 2$, либо $s_- = 3$, либо ... Можно проверить по индукции, что алгоритм в конце концов придет к шагу **В5**, не изменив с нулевой по l -ю строки A и со значениями переменных $l = l_0 + 1, q = q_0 \pm 1, r = q_0$ и $s = s_+$ или s_- , причем мы выбираем знак $+$, если $s_+ = 0$ или ($s_+ = 2$ и $s_- \neq 1$) или ($s_+ = 4$ и $s_- \neq 1, 3$) или ..., и мы выбираем знак $-$, если ($s_- = 1$ и $s_+ = 0$) или ($s_- = 3$ и $s_+ \neq 0, 2$) или ... Приведенный здесь набросок доказательства не очень элегантен, но и сам алгоритм сформулирован в виде, который больше годится для реализации, чем для проверки правильности.

Picture: Рис. 78 Эффективность осциллирующей сортировки, использующей метод алгоритма В и упр. 3.

На рис. 78 показана эффективность алгоритма В, выраженная средним числом слияний каждой записи в зависимости от S —числа начальных отрезков, причем предполагается, что начальные отрезки приблизительно равны по длине. (Соответствующие графики для многофазной и каскадной сортировки приведены на рис. 70 и 74.) При подготовке рис. 78 учтено небольшое усовершенствование, упомянутое в упр. 3.

Прямое чтение. Схема осциллирующей сортировки, по-видимому, требует возможности обратного чтения, поскольку приходится где-то накапливать длинные отрезки по мере того, как мы сливаем вновь введенные короткие отрезки. Тем не менее М. А. Готц [Proc. AFIPS Spring Jt. Contr. Conf.; 25 (1964), 599–607] нашел способ выполнить осциллирующую сортировку, используя только прямое чтение и простую перемотку. Его метод в корне отличается от остальных схем, которые мы видели в этой главе, поскольку

а) данные иногда записываются в начало ленты, причем предполагается, что данные, находящиеся в *середине* этой ленты, не разрушаются;

б) все начальные строки имеют фиксированную максимальную длину. Условие (а) нарушает свойство "первым включается—первым исключается", которое, как мы предположили, является характеристикой прямого чтения, однако оно может быть надежно реализовано, если между отрезками

оставлять достаточное количество чистой ленты и если в нужные моменты пренебречь "ошибками четности". Условие (b) оказывается до некоторой степени противоречащим эффективному использованию выбора с замещением.

Осциллирующая сортировка Готца с прямым чтением имеет одно темное пятно—это один из первых алгоритмов, который был запатентован как алгоритм, а не как физическое устройство [U. S. Patent 3380029 (23 апреля 1968)]. Если положение не изменится, то это означает, что алгоритм нельзя использовать в программе без разрешения владельца патента. Метод Бэнчера (осциллирующая сортировка с обратным чтением) был запатентован IBM несколькими годами позже. [Таким образом, наступил конец эры, когда удовольствие от открытия нового алгоритма считалось достаточным вознаграждением! Так как программирование неотделимо от создания машины, а программы для ЭВМ теперь стоят денег, то патентование алгоритмов является неизбежным. Конечно, действия недалековидных людей, сохраняющих новые алгоритмы в строгом секрете, значительно хуже, чем широкая доступность алгоритмов, которые являются собственностью в течение лишь ограниченного времени.]

Центральная идея в методе Готца состоит в таком использовании лент, чтобы каждая лента начиналась с отрезка относительной длины 1, за которым следовал бы отрезок относительной длины P , затем P^2 и т. д. Например, если $T = 5$, то сортировка начинается следующим образом ("." указывает текущее положение головки чтения-записи на каждой ленте):

	Операция	T_1	T_2	T_3	T_4	T_5	Стоимость	Примечания
Фаза 1.	Распределение	A_1	$.A_1$	$.A_1$	$.A_1$	$A_1.$	5	[T_5 не перематывается]
Фаза 2.	Слияние	$A_1.$	$A_1.$	$A_1.$	$A_1.$	$A_1A_4.$	4	[Перемотка всех лет]
Фаза 3.	Распределение	A_1	$.A_1$	$.A_1$	$A_1.$	$.A_1A_4$	4	[T_4 не перематывается]
Фаза 4.	Слияние	$A_1.$	$A_1.$	$A_1.$	$A_1A_4.$	$A_1.A_4$	4	[Перемотка всех лент]
Фаза 5.	Распределение	A_1	$.A_1$	$A_1.$	$.A_1A_4$	$.A_1A_4$	4	[T_3 не перематывается]
Фаза 6.	Слияние	$A_1.$	$A_1.$	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	4	[Перемотка всех лент]
Фаза 7.	Распределение	A_1	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T_2 не перематывается]
Фаза 8.	Слияние	$A_1.$	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	4	[Перемотка всех лент]
фаза 9.	Распределение	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T_1 не перематывается]
Фаза 10.	Слияние	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	4	[Нет перемотки]
Фаза 11.	Слияние	$A_1A_4A_{16}.$	$A_1A_4.$	$A_1A_4.$	$A_1A_4.$	$A_1A_4.$	16	[Перемотка всех лент]

И так далее. Во время фазы 1 лента T_1 перематывается и одновременно на T_2 записываются исходные данные, затем перематывается T_2 и одновременно на T_3 записываются исходные данные и т. д. В конце концов, когда исходные данные исчерпаны, начинают появляться фиктивные отрезки, и иногда необходимо вообразить, что они записаны явно на ленте полной длины. Например, если $S = 18$, то отрезки A_1 на T_4 и T_5 будут фиктивными во время фазы 9; нам придется продвинуться вперед по T_4 и T_5 при слиянии с T_2 и T_3 на T_1 во время фазы 10, так как нам надо добраться до отрезков A_4 на T_4 и T_5 для подготовки к фазе 11. С другой стороны, фиктивный отрезок A_1 на T_1 не обязательно должен существовать явно. Таким образом, "конец игры" несколько замысловат. Еще с одним примером применения этого метода мы встретимся в следующем пункте.

Упражнения

- [22] В тексте имеется иллюстрация осциллирующей сортировки Собея в ее первоначальном виде для $T = 5$ и $S = 16$. Дайте точное определение алгоритма, в котором эта процедура обобщается и сортируются $S = P^L$ начальных отрезков на $T = P + 1 \geq 3$ лентах. Постарайтесь найти алгоритм, который может быть очень просто описан.
- [24] Если в изначальном методе Собея $S = 6$, то мы могли бы заявить, что $S = 16$ и что имеется 10 фиктивных отрезков. Тогда фаза 3 в примере в тексте поместила бы фиктивные отрезки A_0 на T_4 и T_5 ; фаза 4 слила бы отрезки A_1 на T_2 и T_3 в D_2 на T_1 ; фазы 5–8 не делали бы ничего; фаза 9 породила бы A_6 на T_4 . Лучше бы перемотать T_2 и T_3 сразу после фазы 3 и затем немедленно получать A_6 на T_4 с помощью трехпутевого слияния. Покажите, как, основываясь на этой идее, улучшить окончание алгоритма из упр. 1, если S не является точной степенью P .
- [24] Составьте таблицу, показывающую поведение алгоритма В, если $T = 3$, предполагая, что имеется 9 начальных отрезков. Покажите, что эта процедура очевидно неэффективна в одном месте, и предложите изменения в алгоритме В, которые исправляют положение.
- [21] На шаге В3 имеется установка как $A[l, q]$, так и $A[l, r]$ в -1 . Покажите, что одна из этих операций всегда лишняя, так как соответствующий элемент массива A никогда не рассматривается.
- [M25] Пусть S —число начальных отрезков в имеющихся исходных данных для алгоритма В. При каких значениях S не требуется ни одной перемотки на шаге В2?

5.4.6. Практические аспекты слияния на лентах

Здесь всплывают разные мелочи. После того как мы обсудили различные семейства схем слияния, самое время посмотреть, как они отображаются на реальные конфигурации ЭВМ и магнитных лент, и сравнить разумным образом эти схемы. Изучение внутренней сортировки показало, что невозможно адекватно оценить эффективность метода сортировки, просто подсчитывая число выполняемых им сравнений; подобно этому, мы не можем правильно оценить метод внешней сортировки, просто выясняя число выполняемых им проходов по данным.

В этом пункте мы рассмотрим характеристики типичных ленточных устройств и их влияние на начальное распределение и слияние. В частности, изучим несколько схем распределения буферов и их воздействие на время счета. Будет также вкратце рассмотрена конструкция программ-генераторов сортировки.

Как работает лента. В характеристиках ленточных устройств, производимых разными изготовителями, имеются значительные различия. Определим для удобства гипотетическое ленточное устройство MIXT, достаточно типичное для оборудования того времени, когда была написана эта книга. Изучив, как построить алгоритм сортировки для лент MIXT, вы лучше поймете, как обращаться с другими конкретными ленточными устройствами.

MIXT читает или записывает 800 литер на дюйм ленты со скоростью 75 дюймов в секунду. Это означает, что одна литера читается или записывается в течение $1/60$ мс, т. е. $16\frac{2}{3}$ мкс, если лента находится в активном состоянии. [Реальные лентопротяжные устройства, широко распространенные в настоящее время, имеют плотность в диапазоне от 200 до 1600 литер на дюйм и скорость в диапазоне от $37\frac{1}{2}$ до 150 дюймов в секунду, так что их эффективная скорость в сравнении с MIXT изменяется в диапазоне от $\frac{1}{8}$ до 4. С практической точки зрения большой объем сортировки выполняется в коммерческих системах на относительно небольшом и недорогом оборудовании, которое медленнее, чем рассматриваемое здесь. С другой стороны, лентопротяжные устройства вскоре могут резко измениться, что сделает настоящие предположения устаревшими. Наша основная цель состоит не в получении конкретных ответов, а в том, чтобы научиться разумно сочетать теорию с практикой.]

Одно из важных соображений, которое надо иметь в виду, состоит в том, что ленты имеют конечную длину. Каждая бобина содержит 2400 футов ленты или меньше; следовательно, на одной бобине ленты MIXT есть место для самое большее примерно 23 000 000 литер, и, чтобы прочесть их все, требуется около $23\,000\,000/3\,600\,000 \approx 6.4$ мин. Если требуется сортировать большой файл, то обычно лучше всего сортировать за один раз одну полную бобину и затем слить отдельные отсортированные бобины с целью избежать избыточной работы по установке лент. Это означает, что число начальных отрезков S , реально присутствующих в схемах слияния, которые мы изучали, никогда не будет очень большим. Мы никогда не столкнемся со

Picture: Рис. 79. Магнитная лента с блоками переменной длины.

случаем $S > 5000$ даже при очень маленькой внутренней памяти, производящей начальные отрезки длиной только в 5000 литер. Следовательно, формулы, дающие асимптотическую эффективность алгоритмов при $S \rightarrow \infty$, имеют главным образом академический интерес.

Данные хранятся на ленте в виде *блоков* (рис. 79), и каждая инструкция чтения/записи передает один блок. Блоки часто называются "записями", но мы будем избегать этого термина, чтобы не путать его с "записями" файла, которые участвуют в сортировке. Для многих ранних программ сортировки, написанных в 50-х годах, это различие было необязательным, так как в одном блоке хранилась одна запись; но мы увидим, что объединение нескольких записей в одном блоке на ленте обычно дает определенные преимущества.

Соседние блоки разделяются *межблочными промежутками* длиной по 480 литер, что позволяет ленте остановиться или разогнаться между отдельными командами чтения или записи. Межблочные промежутки приводят к уменьшению числа литер на одной бобине ленты, зависящему от числа литер в одном блоке (рис. 80); в той же степени уменьшается количество литер, передаваемых за секунду, так как лента движется с постоянной скоростью.

Многие "устаревшие" модели ЭВМ имеют фиксированные и весьма малые размеры блока. Например, MIX, как она описана в гл. 1, всегда читает и пишет блоки по 100 слов, таким образом, это составляет примерно 500 литер на блок и 480 литер на промежуток, т. е. почти половина ленты пропадает! Сейчас большая часть машин допускают переменный размер блока, и поэтому ниже мы обсудим проблему выбора подходящего размера блока.

Picture: Рис. 80. Число литер на одной бобине ленты MIXT как функция от размера блока.

В конце операции чтения или записи лента проходит с полной скоростью около 66 первых литер межблочного промежутка. Если в это время будет инициирована следующая операция для этой же ленты, то движение ленты продолжится без перерыва. Но если следующая операция не начнется достаточно быстро,

Picture: Рис. 81. Как вычислить время стартстопной задержки. (Оно добавляется ко времени, используемому при чтении или записи блоков и промежутков.)

то лента остановится и к тому же потребуется некоторое время, чтобы разогнать ее до полной скорости в следующей операции. Суммарная стартстопная задержка составляет 5 мс, 2 для остановки и 3 для разгона (рис. 81). Таким образом, если мы не поддерживаем непрерывного, безостановочного движения ленты с полной скоростью, то эффект во времени счета будет, в сущности, такой же, как если бы в межблочном промежутке было 780 литер вместо 480.

Рассмотрим теперь операцию *перемотки*. К сожалению, обычно трудно точно охарактеризовать время, требуемое для перемотки ленты на заданное число литер n . На некоторых машинах имеется "быстрая перемотка", которая применяется, только если n превышает число порядка 5 миллионов; для меньших значений n перемотка происходит с нормальной скоростью чтения/записи.

Picture: Рис. 82. Приблизительное время счета при двух обычно используемых методах перемотки.

На других машинах имеется специальный мотор, используемый во всех операциях перемотки; он постепенно ускоряет бобину ленты до определенного числа оборотов в минуту, а затем тормозит ее, когда подходит время остановки; действительная скорость ленты в этом случае зависит от заполненности бобины. Мы примем для простоты, что MIXT требует $\max(30, n/150)$ мс для перемотки на n литерных позиций (включая промежутки), т. е. примерно две пятых того, что требуется для их чтения/записи. Это достаточно хорошее приближение к поведению многих реальных устройств, где отношение времени чтения/записи ко времени перемотки обычно заключено между 2 и 3, но оно не дает адекватной модели эффекта комбинированной нормальной и быстрой перемотки, имеющейся на многих других машинах (рис. 82).

При первоначальной установке и/или перемотке к началу лента помещается в точку загрузки, и для любой операции чтения или записи, начинающейся из этого положения, требуется дополнительно 110 мс. Если лента не находится в точке загрузки, она может быть прочитана в обратном направлении; 32 мс добавляется ко времени любой обратной операции, следующей за прямой, или любой прямой операции, следующей за обратной.

Наконец, следует рассмотреть возможность одновременного ввода и вывода. Часто по экономическим причинам несколько ленточных устройств присоединяются к одному *ленточному контроллеру* (устройству управления лентами), который может одновременно работать только с одной или двумя лентами, поскольку число линий передачи данных между ним и ЭВМ ограничено. Иногда контроллеры не способны работать более чем с одной лентой одновременно. Однако часто они могут читать одну ленту во время записи другой. Несколько реже встречаются контроллеры, которые могут читать одновременно с двух устройств, и автор никогда не видел контроллера, который мог бы писать на два устройства одновременно. Перемотка—это особый случай: через 30 мс после начала перемотки ленточное устройство MIXT "отключается" от своего контроллера, который после этого способен выполнять операции с другими устройствами. Таким образом, очень большое число ленточных устройств могут одновременно осуществлять перемотку.

Почти все машины допускают выполнение ввода/вывода параллельно с вычислениями, хотя многие ЭВМ, когда происходит ввод/вывод, работают на 20–40% медленнее из-за "разделения циклов памяти".

Снова о слиянии. Обратимся вновь к процессу P -путевого слияния с упором на функционирование устройств ввода и вывода. Будем считать, что для вводных и выводных файлов используется $P + 1$ ленточных устройств. Наша цель—максимально совместить операции ввода/вывода друг с другом и со счетом по программе так, чтобы минимизировать общее время слияния.

Поучительно рассмотреть следующий частный случай, в котором на возможную степень одновременности наложены серьезные ограничения. Предположим, что

- a) можно записывать не более чем на одну ленту одновременно;
- b) можно читать не более чем с одной ленты одновременно;
- c) чтение, запись и вычисления могут происходить одновременно, только если операции чтения и записи были начаты одновременно.

Оказывается, что даже при таких ограничениях достаточно иметь $2P$ буферов ввода и 2 буфера вывода, чтобы поддерживать, в сущности, максимальную скорость движения лент, если только мы имеем дело не с очень медленной ЭВМ. Заметим, что (а) не является на самом деле ограничением, так как имеется только одна выводная лента. Кроме того, объем ввода равен объему вывода, так что читается в среднем только одна лента в любой данный момент времени; если условие (b) не выполняется, то обязательно должны быть периоды, когда вообще не происходит ввода. Следовательно, для того чтобы минимизировать время слияния, достаточно поддерживать выводную ленту в состоянии работы.

Важный метод, называемый предсказанием или *прогнозированием* (forecasting), дает желаемый эффект. Во время выполнения P -путевого слияния обычно имеется P "текущих буферов ввода", которые используются как источники данных; некоторые из них заполнены больше других в зависимости от того, какая часть данных в них уже просмотрена. Если все они опустошатся примерно в одно и то же время, то, прежде чем продолжить работу, нам придется выполнить много чтений, если только мы не предпримем нужных мер заранее. К счастью, всегда можно сказать, какой буфер первым станет пустым, просто посмотрев на *последнюю* запись в каждом буфере. Буфер, последняя запись которого имеет наименьший ключ, обязательно будет первым пустым буфером независимо от значений каких-либо других ключей, так что мы всегда знаем, какой файл послужит причиной нашей следующей команды ввода. Алгоритм F раскрывает этот принцип в деталях.

Алгоритм F. (Прогнозирование с плавающими буферами.) Этот алгоритм управляет буферизацией во время P -путевого слияния длинных вводных файлов при $P \geq 2$. Допустим, что вводные ленты и файлы занумерованы $1, 2, \dots, P$. В алгоритме используются $2P$ буферов ввода $I[1], \dots, I[2P]$, два буфера вывода $O[0]$ и $O[1]$ и следующие вспомогательные массивы:

- $A[j], 1 \leq j \leq 2P$: 0 если в буфер $I[j]$ можно вводить данные;
 1 в противном случае.
 $V[i], 1 \leq i \leq P$: Буфер, содержащий последний прочитанный блок из файла i .
 $C[i], 1 \leq i \leq P$: Буфер, используемый в настоящий момент для ввода из файла i .
 $L[i], 1 \leq i \leq P$: Последний ключ, прочитанный из файла i .
 $S[j], 1 \leq j \leq 2P$: Буфер, который следует использовать, когда $I[j]$ опустошится.

В описываемом виде алгоритм никогда не остановится; подходящий способ его "выключения" обсуждается ниже.

- F1 [Начальная установка.]** Прочитать первый блок с ленты i в буфер $I[i]$, установить $A[i] \leftarrow 1, A[P+i] \leftarrow 0, V[i] \leftarrow i, C[i] \leftarrow i$ и установить $L[i]$ равным ключу последней записи в $I[i]$ при $1 \leq i \leq P$. Затем найти m , такое, что $L(m) = \min\{L[1], \dots, L[P]\}$, и установить $t \leftarrow 0, k \leftarrow P+1$. Начать читать с ленты от в буфер $I[k]$.
- F2 [Слияние.]** Слить записи из буферов $I[C[1]], \dots, I[C[P]]$ в $O[t]$, пока буфер $O[t]$ не заполнится. Если во время этого процесса какой-нибудь буфер ввода, скажем $I[C[i]]$, станет пустым, а $O[t]$ еще не заполнен, то установить $A[C[i]] \leftarrow 0, C[i] \leftarrow S[C[i]]$ и продолжать слияние.
- F3 [Ввод/вывод завершен.]** Ждать, пока не завершится предыдущая операция чтения (или чтения/записи). Затем установить $A[k] \leftarrow 1, S[B[m]] \leftarrow k, V[m] \leftarrow k$ и установить $L[m]$ равным ключу последней записи в $I[k]$.
- F4 [Прогнозирование.]** Найти m , такое, что $L[m] = \min\{L[1], \dots, L[P]\}$, и найти k , такое, что $A[k] = 0$.
- F5 [Чтение/запись.]** Начать читать с ленты m в буфер $I[k]$ и писать из буфера $O[t]$ на выводную ленту, затем положить $t \leftarrow 1 - t$ и вернуться к **F2**. ■

Picture: Рис. 83. Прогнозирование с плавающими буферами.

Пример на рис. 84 показывает, как работает метод прогнозирования при $P = 2$ в предположении, что каждый блок на ленте содержит только две записи. Здесь представлено содержимое буферов ввода во все моменты, когда мы достигаем начала шага F2. Алгоритм F, в сущности, образует P *очередей буферов*, где $C[i]$ указывает на начало i -й очереди, $V[i]$ — на ее конец, $S[j]$ указывает на преемника буфера $I[j]$; этим указаниям на рис. 84 соответствуют стрелки. Строка 1 отражает состояние дел после начальной установки. Для каждого вводного файла есть один буфер, и еще один блок читается из файла 1 (так как $03 < 05$). Строка 2 показывает положение вещей после того, как слит первый блок: мы выводим блок, содержащий "01 02", и вводим следующий блок из файла 2 (так как $05 < 09$). Заметим, что в строке 3 три из четырех буферов ввода, по сути дела, предоставлены файлу 2, так как мы читаем из этого файла и в его очереди уже есть полный буфер и частично заполненный буфер.

Этот механизм ”плавающих буферов” является важной чертой алгоритма F, так как мы не смогли бы продолжить работу в строке 4, если бы в строке 3 выбрали для ввода файл 1 вместо файла 2.

Picture: Рис. 84. Очереди буферов в соответствии с алгоритмом F.

Чтобы доказать правильность алгоритма F, мы должны установить два факта:

- i) всегда имеется свободный буфер (т. е. мы всегда можем найти k на шаге F4);
- ii) если буфер ввода исчерпывается во время слияния, то его преемник уже присутствует в памяти (т. е. $S[C[i]$ в шаге F2 имеет осмысленное значение).

Допустим, что (i) не имеет места, т. е. все буферы заняты в некоторый момент, когда мы достигаем шага F4. Каждый раз, когда мы приходим к этому шагу, суммарный объем необработанных данных во всех буферах составляет ровно P емкостей буфера, т. е. данных ровно столько, чтобы, переместив их, заполнить P буферов, ибо данные вводятся и выводятся с одинаковой скоростью. Некоторые буферы заполнены лишь частично, однако для каждого файла частично заполнен самое большее один буфер, так что всего таких буферов не более P . По предположению все $2P$ буферов заняты, так что по меньшей мере P из них должны быть заполнены целиком. Это может случиться, только если P буферов полны и P пусты, иначе мы бы имели слишком много данных. Но самое большее один буфер может быть одновременно пуст и занят; следовательно, (i) не может не выполняться.

Допустим, что (ii) не имеет места, т. е. для некоторого файла в памяти нет необработанных записей, но текущий буфер вывода еще не полон. Согласно принципу прогнозирования, нужно иметь не более одного блока данных для всех остальных файлов, так как мы не читаем блок из файла, если этот блок не потребуется прежде, чем будут исчерпаны буферы какого-нибудь другого файла. Таким образом, общее число необработанных записей составляет самое большее $P - 1$ блоков; добавление неполного буфера вывода дает менее P буферных емкостей данных в памяти; получили противоречие.

Эти рассуждения устанавливают справедливость алгоритма F; они также показывают, что возможны патологические обстоятельства, при которых алгоритм едва-едва избегает крушения. Нами здесь не упомянута некая важная тонкость, касающаяся возможного равенства ключей; это обсуждается в упр. 5.

Один из способов изящно завершить алгоритм F состоит в том, чтобы присвоить $L[m]$ значение ∞ на шаге F3, если только что прочитанный блок был последним в отрезке. (Конец отрезка всегда указывается некоторым особым образом.) После того как будут прочитаны все данные во всех файлах, мы в конце концов обнаружим на шаге F4, что все L равны ∞ ; тогда обычно можно начать чтение первых блоков следующих отрезков в каждом файле, выполняя начальную установку для следующей фазы слияния по мере вывода последних $P + 1$ блоков.

Таким образом, можно поддерживать полную скорость работы выводной ленты, читая в любое время не более одной ленты. Исключение из этого правила встречается на шаге F1, где было бы полезно читать сразу несколько лент, чтобы обеспечить возможность начать работу; но обычно шаг F1 можно устроить так, чтобы он совмещался с предыдущей частью вычислений.

Идея просмотра последней записи каждого блока с целью предсказания, какой буфер первым станет пустым, была высказана в 1953 г. Ф. Э. Гольбертон. Сам метод впервые был опубликован Э. Х. Фрэндом [*JACM*, 3 (1956), 144–145, 165]. Его довольно сложный алгоритм использовал $3P$ буферов ввода, и каждому файлу ввода предназначалось по три буфера; алгоритм F улучшает положение, используя ”плавающие буферы” и позволяя любому одному файлу потребовать сразу даже $P + 1$ буферов ввода, хотя всего требуется не более $2P$ буферов.

Слияние с менее чем $2P$ буферами обсуждается в конце этого пункта. Некоторые ЭВМ имеют возможность ”чтения вразброс—записи со сборкой”, что позволяет осуществлять ввод/вывод из непоследовательных ячеек памяти; использование такой возможности выходит за рамки этой книги.

Сравнительное поведение схем слияния. Используем теперь наши знания о лентах и слиянии, чтобы сравнить эффективность различных схем слияния, изученных нами в п. 5.4.2–5.4.5. Будет весьма поучительно разработать детали каждого метода в применении к конкретному ”беспристрастному” примеру. Рассмотрим поэтому задачу сортировки файла, каждая запись которого содержит 100 литер, причем в памяти для записи данных доступно 100000 литерных позиций (не считая места для программы, ее вспомогательных переменных и сравнительно небольшого пространства, необходимого для ссылок в дереве выбора). Исходные данные расположены на ленте в случайном порядке блоками по 5000 литер каждый, и результат должен получиться в том же формате. Для работы имеется пять рабочих лент в добавление к устройству, на котором находится вводная лента.

Общее число сортируемых записей 100000, но эта информация алгоритму сортировки заранее не известна.

В схему А (здесь и далее см. вкладку) сведены те действия, которые происходят, когда к нашим данным применяется десять различных схем слияния. Обратившись к этой важной иллюстрации, очень полезно вообразить, что вы наблюдаете за тем, как происходит реальная сортировка: медленно просматривайте каждую строку слева направо, мысленно представляя себе шесть лент, осуществляющих чтение, запись, перемотку и/или обратное чтение, как указано на диаграмме. В течение P -путевого слияния вводные ленты будут находиться в движении в P раз реже, чем выводная лента. Заметим, что в схеме А предполагается, что, когда первоначальная вводная лента полностью прочитана (и перемотана, чтобы ее убрать), умелый оператор снимает ее и заменяет рабочей лентой за 30 с. В примерах 2, 3 и 4 это и есть "время критического пути", когда ЭВМ в бездействии ожидает оператора. Но в остальных примерах операция снятия и установки лент совмещена с другой работой. (На схеме А по горизонтали указано время в мин.)

Пример 1. Сбалансированное слияние с прямым чтением. Напомним описание задачи: записи имеют длину в 100 литер, внутренней памяти достаточно для одновременного хранения 1000 записей и каждый блок вводной ленты содержит 5000 литер (50 записей). Всего имеется 100 000 записей (т. е. 10 000 000 литер, или 2000 блоков).

Мы ничем не связаны в выборе размера блоков для промежуточных файлов. Шестиленточное сбалансированное слияние использует трехпутевое слияние, так что техника алгоритма F требует 8 буферов; можно, следовательно, использовать блоки, содержащие каждый по $1000/8 = 125$ записей (или 12500 литер).

Проход начального распределения может использовать выбор с замещением (п. 5.4.1), и, чтобы поддерживать непрерывную работу лент, будем использовать два буфера ввода по 50 записей каждый, плюс два буфера вывода по 125 записей каждый. Это оставляет для дерева выбора место в 650 записей. Большая часть начальных отрезков будет, следовательно, иметь длину около 1300 записей (10 или 11 блоков); на схеме А получилось 78 начальных отрезков, причем последний отрезок короткий.

Первый проход слияния, как показано, сливает девять отрезков на ленту 4, а не чередует ленты 4, 5 и 6. Это дает возможность выполнять полезную работу в то время, когда оператор вычислительной машины устанавливает рабочую ленту на устройство 6; так как общее число отрезков S известно сразу после завершения начального распределения, то алгоритм знает, что на ленту 4 должно быть слито $\lceil S/9 \rceil$ отрезков, затем $\lceil (S - 3)/9 \rceil$ — на ленту 5, затем $\lceil (S - 6)/9 \rceil$ — на ленту 6.

Вся процедура сортировки в этом примере может быть следующим образом изображена с использованием обозначений, введенных в п. 5.4.2:

1^{26}	1^{26}	1^{26}	—	—	—
—	—	—	3^9	3^9	3^8
9^3	9^3	$9^2 6^1$	—	—	—
—	—	—	27^1	27^1	24^1
78^1	—	—	—	—	—

Пример 2. Многофазное слияние с прямым чтением. Второй пример на схеме А иллюстрирует многофазное слияние в соответствии с алгоритмом 5.4.2D. В этом случае мы выполняем пятипутевое слияние, поэтому память разбита на 12 буферов по 83 записи каждый. В течение первоначального выбора с замещением мы имеем два буфера ввода в 50 записей и два буфера вывода в 83 записи, что оставляет 734 записи в дереве; таким образом, начальные отрезки в этот раз будут иметь длину около 1468 записей (17 или 18 блоков). В данной ситуации получено $S = 70$ начальных отрезков, причем длины двух последних в действительности равны только четырем блокам и одному блоку соответственно. Схему слияния можно изобразить так:

$0^{13}1^{18}$	$0^{13}1^{17}$	$0^{13}1^{15}$	$0^{12}1^{12}$	$0^8 1^1$	—
1^{15}	1^{14}	1^{12}	1^8	—	$0^8 1^4 2^1 5^3$
1^7	1^6	1^4	—	4^8	$1^4 2^1 5^3$
1^3	1^2	—	8^4	4^4	$2^1 5^3$
1^1	—	$16^1 19^1$	8^2	4^2	5^2
—	34^1	19^1	8^1	4^1	5^1
70^1	—	—	—	—	—

Удивительно, что многофазное слияние занимает на 25 с *больше* времени, чем значительно более простое сбалансированное слияние! Это объясняется двумя основными причинами:

- 1) Этот случай особенно удачен для сбалансированного слияния, так как $S = 78$ очень близко к точной степени 3. Если бы было получено 82 начальных отрезка, то сбалансированное слияние заняло бы еще один проход.

- 2) При многофазном слиянии теряется 30 с во время замены вводной ленты и в целом свыше 5 мин проходит в ожидании завершения операций перемотки. В противоположность этому сбалансированное слияние требовало сравнительно небольшого времени перемотки. Во второй фазе многофазного слияния сэкономлено 13 с, так как 8 фиктивных отрезков на ленте 6 можно считать присутствующими даже во время перемотки этой ленты, но дальше не происходит никакого совмещения перемотки. Таким образом, многофазный метод проигрывает, несмотря на то, что он требует значительно меньшего времени чтения/записи.

Пример 3. Каскадное слияние с прямым чтением. Этот случай аналогичен предыдущему, только использует алгоритм 5.4.3С. Слияние изображается так:

1^{14}	1^{15}	1^{12}	1^{14}	1^{15}	—
1^5	1^9	—	1^{14}	1^{15}	$1^3 2^3 3^6$
$5^1 6^3$	5^3	$5^3 6^2$	—	1^1	2^2
—	12^1	6^1	18^1	18^1	16^1
70^1	—	—	—	—	—

(Просматривая схему А, не забывайте представлять каждый пример в действии.)

Пример 4. Многофазное слияние с расщеплением лент. Эта процедура, описанная в конце п. 5.4.2, позволяет совместить большую часть времени перемотки. Она использует четырехпутевое слияние, так что мы делим память на десять буферов по 100 записей; в дереве выбора имеется 700 записей, и в результате оказывается, что образовано 72 начальных отрезка. Последний отрезок вновь очень короткий. Использована схема распределения, аналогичная алгоритму 5.4.2D, за ней следует простой, но до некоторой степени специальный метод размещения фиктивных отрезков:

1^{21}	1^{19}	1^{15}	1^8	—	$0^{21} 9$
$0^{21} 17$	$0^{21} 15$	$0^{21} 11$	$0^{21} 4$	—	$0^{21} 9 4^4$
1^{13}	1^{11}	1^7	—	$0^{24} 4$	$0^{21} 9 4^4$
1^{10}	1^8	1^4	—	$0^{24} 4^3 2^4 1$	$1^8 4^4$
1^6	1^4	—	4^4	$0^{24} 4^3 2^4 1$	$1^4 4^4$
1^5	1^3	—	$4^4 3^1$	$0^{14} 4^3 2^4 1$	$1^3 4^4$
1^2	—	$3^1 7^2$	$4^4 3^1$	$4^2 3^2 4^1$	4^4
1^1	—	$3^1 7^2 13^1$	$4^3 3^1$	$4^1 3^2 4^1$	4^3
—	13^1	$3^1 7^2 13^1$	$4^2 3^1$	$3^2 4^1$	4^2
—	$13^1 14^1$	$7^2 13^1$	$4^1 3^1$	$3^1 4^1$	4^1
18^1	$13^1 14^1$	$7^1 13^1$	3^1	4^1	—
18^1	14^1	13^1	—	—	27^1
—	—	—	72^1	—	—

Среди всех примеров на схеме А, которые не читают в обратном направлении, в этом, как оказывается, наилучшее время выполнения. Так как S никогда не бывает очень большим, можно разработать более сложный алгоритм, который размещает фиктивные отрезки еще лучше (см. упр. 5.4.2-26).

Пример 5. Каскадное слияние с совмещением перемоток. Эта процедура работает почти так же быстро, как предыдущая, хотя управляющий ею алгоритм более прост. Мы используем для начального распределения метод каскадной сортировки, как в алгоритме 5.4.3С, но с $T = 5$, а не $T = 6$. Затем использование лент в каждой фазе каждого "каскада" чередуется таким образом, что мы обычно не пишем на ленту, пока она почти наверняка не окажется перемотанной. Короче говоря, схема такова:

1^{21}	1^{22}	1^{19}	1^{10}	—	—
1^4	1^7	—	—	$1^2 2^2 3^5$	4^{10}
7^2	—	8^3	$7^2 8^2$	—	4^1
—	26^1	—	8^1	22^1	16^1
72^1	—	—	—	—	—

Пример 6. Сбалансированное слияние с обратным чтением. Этот пример похож на пример 1, но все перемотки устранены:

A_1^{26}	A_1^{26}	A_1^{26}	—	—	—
—	—	—	D_3^9	D_3^9	D_3^8
A_9^3	A_9^3	$A_9^2 A_6^1$	—	—	—
—	—	—	D_{24}^1	D_{27}^1	D_{27}^1
A_{78}^1	—	—	—	—	—

Так как в примере 1 было сравнительно мало перемоток, то эта схема не намного лучше, чем в случае прямого чтения. Фактически она оказывается несколько медленней многофазной схемы с расщеплением лент, несмотря на удачное значение $S = 78$.

Пример 7. Многофазное слияние с обратным чтением. В этом примере используется только пять лент из шести, чтобы устранить время перемотки и смены вводной ленты. Таким образом, используется только четырехпутевое слияние и такая же структура буферов, как в примерах 4 и 5. Используется распределение, аналогичное алгоритму 5.4.2D, но направление отрезков чередуется, и лента 1 зафиксирована, как конечная выводная лента. Первым записывается возрастающий отрезок на ленту 1; затем убывающие отрезки на ленты 2, 3, 4; затем возрастающие отрезки на 2, 3, 4; затем убывающие на 1, 2, 3 и т. д. Всякий раз, как мы переключаем направление, выбор с замещением обычно дает более короткий отрезок, поэтому оказалось образовано 77 начальных отрезков вместо 72 в примерах 4 и 5.

Эта процедура в результате дает распределение (22, 21, 19, 15) отрезков, а ближайшее точное распределение — (29, 56, 52, 44). Упражнение 5.4.4-5 показывает, как построить строки чисел слияния, которые могут быть использованы для размещения фиктивных отрезков "оптимальным" образом; такая процедура возможна на практике, поскольку конечность бобины гарантирует, что S никогда не будет слишком большим. Поэтому пример на схеме А был построен с использованием такого метода размещения фиктивных отрезков (см. упр. 7). Он оказался самым быстрым из всех представленных примеров.

Пример 8. Каскадное слияние с обратным чтением. Как и в примере 7, здесь участвует только пять лент. Эта процедура следует алгоритму 5.4.3С, используя перемотку и прямое чтение, чтобы избежать однопутевого слияния (так как перемотка более чем в два раза быстрее чтения на устройствах МЛХТ). Распределение, следовательно, то же, что и в примере 6. Используя символ \downarrow для обозначения перемотки, изобразим эту схему так:

$$\begin{array}{ccccc}
 A_1^{21} & A_1^{22} & A_1^{19} & A_1^{10} & - \\
 A_1^4 \downarrow & A_1^7 \downarrow & - & {}_1^2 D_2^2 D_3^5 & D_4^{10} \\
 A_8 A_7^2 & A_5^2 & A_9^4 & - & D_4^1 \downarrow \\
 - & D_{17} & A_9 \downarrow & D_{25} & D_{21}
 \end{array}$$

Пример 9. Осциллирующая сортировка с обратным чтением. Осциллирующая сортировка с $T = 5$ (алгоритм 5.4.5В) может использовать распределение буферов, как в примерах 4, 5, 7 и 8, так как она выполняет четырехпутевое слияние. Однако выбор с замещением действует здесь иначе, поскольку непосредственно перед входом в каждую фазу слияния выводится отрезок длины 700 (а не примерно 1400), чтобы очистить внутреннюю память. Следовательно, здесь порождается 85 отрезков вместо 72. Некоторые ключевые шаги этого процесса таковы:

$$\begin{array}{ccccc}
 - & A_1 & A_1 A_1 & A_1 A_1 & A_1 A_1 \\
 D_4 & - & A_1 & A_1 & A_1 \\
 \dots\dots\dots & & & & \\
 D_4 D_4 & D_4 D_4 & D_4 D_4 & D_4 & - \\
 D_4 & D_4 & D_4 & - & A_{16} \\
 \dots\dots\dots & & & & \\
 D_4 & A_{16} D_4 D_4 & A_{16} D_4 & A_{16} D_4 A_1 & A_{16} \\
 D_4 & A_{16} D_4 D_4 & A_{16} D_4 D_1 & A_{16} D_4 & A_{16} \\
 - & A_{16} D_4 & A_{16} D_4 & A_{16} & A_{16} A_{13} \\
 - & A_{16} D_4 & A_{16} & A_{16} A_4 & A_{16} A_{13} \\
 - & A_{16} & A_{16} A_4 & A_{16} A_4 & A_{16} A_{13} \\
 D_{37} & - & A_{16} \downarrow & A_{16} \downarrow & A_{16} \downarrow \\
 - & A_{85} & - & - & -
 \end{array}$$

Пример 10. Осциллирующая сортировка с прямым чтением. В последнем примере выбор с замещением не используется, так как все начальные отрезки должны быть одной длины. Следовательно, будет происходить внутренняя сортировка 1000 записей (полной емкости памяти) каждый раз, когда

требуется начальный отрезок; это дает $S = 100$. Вот некоторые ключевые шаги процесса:

A_1	A_1	A_1	A_1	A_1
—	—	—	—	$A_1 A_4$
.....				
A_1	A_1	A_1	A_1	$A_1 A_4$
—	—	—	$A_1 A_4$	$A_1 A_4$
—	—	—	$A_1 A_4$	$A_1 A_4$
.....				
A_1	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$
$A_1 A_4$	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$
$A_1 A_4 A_{16}$	—	—	—	—
.....				
—	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$	$A_1 A_4 A_{16} A_{64}$
A_4	$A_1 A_4$	$A_1 A_4$	$A_1 A_4$	$A_1 A_4 A_{16} A_{64}$
$A_4 A_{16}$	—	—	—	$A_1 A_4 A_{16} A_{64}$
$A_4 A_{16}$	A_4	—	—	$A_1 A_4 A_{16} A_{64}$
—	—	—	A_{36}	$A_1 A_4 A_{16} A_{64}$
A_{100}	—	—	—	—

Эта программа оказывается самой медленной из всех частично из-за того, что она не использует выбор с замещением, но большей частью вследствие ее весьма нескладного конца (двухпутевое слияние).

Оценка времени выполнения. Посмотрим теперь, как вычислить приблизительное время выполнения метода сортировки, использующего ленты MIXT. Можно ли предсказать результаты, изображенные на схеме А, не выполняя детального моделирования?

Один способ, который традиционно использовался для сравнения различных схем слияния, состоит в том, чтобы наложить друг на друга графики, подобные представленным на рис. 70, 74 и 78. Эти графики изображают эффективное число проходов по данным как функцию от числа начальных отрезков в предположении, что все начальные отрезки имеют примерно равную длину (рис. 85). Но это *не* дает очень реалистичного сравнения, потому что, как мы видели, разные методы приводят к различному числу начальных отрезков; кроме того, имеются различные "накладные расходы", зависящие от относительной частоты межблочных промежутков; значительное воздействие оказывает также время перемотки. Все эти зависящие от машины особенности делают невозможным подготовить на машинно-независимой основе схемы, осуществляющие истинное сравнение методов. С другой стороны, из рис. 85 все же явствует, что, за исключением сбалансированного слияния, эффективное число проходов может быть достаточно хорошо аппроксимировано плавными кривыми вида $\alpha \ln S + \beta$. Следовательно, мы можем неплохо сравнивать методы в любой практической ситуации, изучив формулы, аппроксимирующие время выполнения. Конечно, наша цель — найти формулы простые, но достаточно реалистичные.

Picture: Рис. 85. Несколько обманчивый способ сравнения схем слияния.

Попытаемся теперь вывести такие формулы в терминах следующих параметров:

N = число сортируемых записей;

C = число литер в записи;

M = число доступных литерных позиций внутренней памяти (предполагаемое кратным C);

τ = время в секундах, нужное для того, чтобы прочитать или записать одну литеру;

$\rho\tau$ = время в секундах для перемотки одной литеры;

$\sigma\tau$ = время в секундах стартстопной задержки;

γ = число литер в межблочном промежутке;

δ = время в секундах, нужное оператору для снятия и замены вводной ленты;

B_i = число литер в блоке неотсортированного ввода;

B_o = число литер в блоке отсортированного вывода.

Для MIXT имеем $\tau = 1/60\,000$, $\rho = 2/5$, $\sigma = 300$, $\gamma = 480$. В примере, рассмотренном выше, $N = 100\,000$, $C = 100$, $M = 100\,000$, $B_i = B_o = 5000$, $\delta = 30$. Обычно именно эти характеристики машины и данных решающим образом влияют на время сортировки (хотя время перемотки часто задается более сложным выражением, чем просто коэффициентом ρ). Имея указанные параметры и схему слияния,

вычислим еще некоторые величины:

$$\begin{aligned}
 P &= \text{максимальный порядок слияния в схеме;} \\
 P' &= \text{число записей в дереве выбора с замещением;} \\
 S &= \text{число начальных отрезков;} \\
 \pi = \alpha \ln S + \beta &= \text{приблизительное среднее число чтений и записей каждой литеры, не считая начального распределения и окончательного слияния;} \\
 \pi' = \alpha' \ln S + \beta' &= \text{приблизительное среднее число перемоток каждой литеры во время промежуточных фаз слияния;} \\
 B &= \text{число литер в блоке в промежуточных фазах слияния;} \\
 \omega_i, \omega, \omega_o &= \text{"коэффициенты накладных расходов"—эффективные времена, требуемые для чтения или записи одной литеры (с учетом промежутков и стартстопного времени), деленные на время } \tau.
 \end{aligned}$$

В примерах схемы А размеры блоков и буферов выбраны в соответствии с формулой

$$B = \left\lfloor \frac{M}{C(2P + 2)} \right\rfloor C, \quad (1)$$

так чтобы блоки могли быть самыми большими, какие возможны при условии совместимости со схемой буферизации алгоритма F. (Чтобы избежать забот во время последнего прохода, величина P должна быть достаточно малой, чтобы (1) обеспечило $B \geq B_o$.) Размер дерева во время выбора с замещением будет, следовательно,

$$P' = (M - 2B_i - 2B)/C. \quad (2)$$

Для случайных данных число начальных отрезков можно оценить, используя результаты п. 5.4.1, формулой

$$S \approx \left\lceil \frac{N}{2P'} + \frac{7}{6} \right\rceil. \quad (3)$$

Предполагая, что $B_i < B$ и что вводная лента во время распределения может работать с полной скоростью (см. ниже), распределение начальных отрезков займет примерно $NC\omega_i\tau$ с, где

$$\omega_i = (B_i + \gamma)/B_i. \quad (4)$$

Во время слияния схема буферизации допускает совмещение чтения, записи и вычислений, но частое переключение вводных лент означает, что мы должны учесть стартстопное время; поэтому положим

$$\omega = (B + \gamma + \sigma)/B \quad (5)$$

и время слияния оценим формулой

$$(\pi + \rho\pi')NC\omega\tau. \quad (6)$$

Эта формула слегка преувеличивает время перемотки, так как ω включает стартстопное время, но другие соображения (такие, как взаимная блокировка перемоток и потери на чтение с точки загрузки) обычно компенсирует это. Окончательный проход слияния в предположении $B_o \leq B$ ограничивается коэффициентом накладных расходов

$$\omega_o = (B_o + \gamma)/B_o. \quad (7)$$

Мы можем оценить время выполнения последнего слияния и перемотки как

$$NC(1 + \rho)\omega_o\tau;$$

на практике оно могло бы быть несколько больше из-за наличия неравных блоков (ввод и вывод не синхронизованы, как в алгоритме F), но это время будет в основном одинаково для всех схем слияния.

Прежде чем переходить к более специфическим формулам для отдельных схем, попробуем обосновать два из сделанных выше предположения.

- а) *Может ли выбор с замещением успеть за вводной лентой?* В примере на схеме А, вероятно, может, так как для выбора новой записи требуется около десяти итераций внутреннего цикла алгоритма 5.4.1R, и мы имеем время $C\omega_i\tau > 1667$ мкс, за которое

следует выполнить эти итерации. Тщательно запрограммировав цикл выбора с замещением, мы можем достигнуть этого на многих (но не на всех) машинах. Заметим, что при слиянии положение несколько менее критическое: время вычисления для одной записи почти всегда меньше времени работы ленты при P -путевом слиянии, так как P не очень велико.

- б) Должны, ли мы на самом деле выбирать в качестве B максимально возможный размер буфера, как в (1)? Большой размер буфера сокращает отношение издержек ω в (5), но он также увеличивает число начальных отрезков S , так как P' уменьшается. Непосредственно не ясно, какой фактор более важен. Рассматривая время слияния как функцию от $x = CP'$, можно выразить его в виде

$$\left(\theta_1 \ln \left(\frac{N}{x} + \frac{7}{6}\right) + \theta_2\right) \left(\frac{\theta_3 - x}{\theta_4 - x}\right) \quad (8)$$

для некоторых подходящих констант $\theta_1, \theta_2, \theta_3, \theta_4$, причем $\theta_3 > \theta_4$. Дифференцирование по x показывает, что есть некоторое N_0 , такое, что для всех $N \geq N_0$ невыгодно увеличивать x за счет размера буфера. В примерах, приведенных на схеме А, N_0 оказалось, грубо говоря, равным 10 000; при сортировке более 10 000 записей большой размер буфера предпочтительнее.

Заметим, однако, что при сбалансированном слиянии число проходов резко изменяется, когда S проходит через степень P . Если заранее известно приближенное значение N , то размер буфера следует выбрать так, чтобы S с большой вероятностью оказалось немного меньше степени P . Например, размер буфера для первой строки схемы А был равен 12 500, так как $S = 78$. Это было вполне удовлетворительно, но если бы S оказалось равным 82, то было бы значительно лучше немного уменьшить размер буфера.

Формулы для десяти примеров. Возвращаясь к схеме А, попытаемся дать формулы, аппроксимирующие время работы для каждого из десяти методов. В большинстве случаев основная формула

$$NC\omega_i\tau + (\pi + \rho\pi')NC\omega\tau + (1 + \rho)NC\omega_o\tau \quad (9)$$

будет достаточно хорошим приближением к суммарному времени сортировки, если мы определим число промежуточных проходов слияния $\pi = \alpha \ln S + \beta$ и число промежуточных проходов перемотки $\pi' = \alpha' \ln S + \beta$. Иногда необходимо внести в (9) некоторую поправку; специфика каждого метода учитывается следующим образом:

Пример 1. Сбалансированное слияние с прямым чтением. Формулы

$$\pi = \lceil \ln S / \ln P \rceil - 1, \quad \pi' = \lceil \ln S / \ln P \rceil / P$$

могут быть использованы для P -путевого слияния с $2P$ лентами.

Пример 2. Многофазное слияние с прямым чтением. Можно положить $\pi' \approx \pi$, так как за каждой фазой обычно следует перемотка приблизительно такой же длины, как предшествующее слияние. Из табл. 5.4.2-1 получаем в случае шести лент значения $\alpha = 0.795$, $\beta = 0.846 - 2$. (Величина "–2" возникает из-за того, что элементы таблицы включают наряду с промежуточными проходами также начальный и конечный.) К (9) нужно добавить время перемотки вводной ленты после начального распределения, а именно $\rho NC\omega_i\tau + \delta$.

Пример 3. Каскадное слияние с прямым чтением. Таблица 5.4.3-1 приводит к значениям $\alpha = 0.773$, $\beta = 0.808 - 2$. Время перемотки сравнительно трудно оценить; возможно, предположение $\pi' = \pi$ достаточно точно. Как и в примере 2, мы должны добавить к (9) время начальной перемотки.

Пример 4. Многофазное слияние с расщеплением лент. Из табл. 5.4.2-5 получаем $\alpha = 0.752$, $\beta = 1.024 - 2$. Время перемотки почти все совмещается, за исключением перемотки после начальной установки ($\rho NC\omega_i\tau + \delta$) и двух фаз вблизи конца (36% от $2\rho NC\omega\tau$). Мы можем также вычесть 0.18 из β , так как первая половина фазы совмещается с начальной перемоткой.

Пример 5. Каскадное слияние с совмещением перемотки. Здесь, используя табл. 5.4.3-1 для $T = 5$, получаем $\alpha = 0.897$, $\beta = 0.800 - 2$. Почти вся несовмещенная перемотка встречается непосредственно после начального распределения и после каждого двухпутевого слияния. После точного начального распределения самая длинная лента содержит примерно $1/g$ всех данных, где g есть "отношение роста". После каждого двухпутевого слияния объем перемотки в случае шести лент равен $d_k d_{n-k}$ (см. упр. 5.4.3-5), и можно показать, что в случае T лент объем перемотки после двухпутевых слияний приблизительно равен

$$(2/(2T - 1))(1 - \cos(4\pi/(2T - 1)))$$

от всего файла. В нашем случае ($T = 5$) это составляет $\frac{2}{9}(1 - \cos 80^\circ) \approx 0.183$ файла, и это происходит в $0.946 \ln S + 0.796 - 2$ случаях.

Пример 6. Сбалансированное слияние с обратным чтением. Оно напоминает пример 1, за исключением того, что значительная часть перемотки устраняется. Изменение направления от прямого к обратному вызывает некоторые задержки, но они не существенны. С вероятностью $1/2$ перед последним проходом нужна будет перемотка, поэтому можно взять $\pi' = 1/(2P)$.

Пример 7. Многофазное слияние с обратным чтением. Так как в этом случае выбор с замещением порождает отрезки, меняющие направление примерно каждые P раз, то следует заменить (3) другой формулой для S . Достаточно хорошим приближением (см. упр. 5.4.1-24) будет $S = \lceil N(3 + 1/P)6P' \rceil + 1$. Все время перемотки устраняется, и табл. 5.4.2-1 дает $\alpha = 0.863, \beta = 0.921 - 2$.

Пример 8. Каскадное слияние с обратным чтением. Из табл. 5.4.3-1 имеем $\alpha = 0.897, \beta = 0.800 - 2$. Время перемотки по этой таблице можно оценить как удвоенную разность [”проходы с копированием” минус ”проходы без копирования”] плюс $1/(2P)$ в том случае, если перед окончательным слиянием необходима перемотка для получения возрастающего порядка.

Пример 9. Осциллирующая сортировка с обратным чтением. В этом случае выбор с замещением должен много раз начинаться и останавливаться; за один раз распределяется серия от $P - 1$ до $2P - 1$ отрезков (в среднем P); средняя длина отрезков, следовательно, оказывается приблизительно равной $P'(2P - 4/3)/P$, и можно оценить $S = \lceil N/((2 - 4/(3P))P') \rceil + 1$. Некоторое время расходуется на переключение от слияния к распределению и обратно; это приблизительно время, требуемое, чтобы прочитать с вводной ленты P' записей, а именно $P' C \omega_i \tau$, и это происходит примерно S/P раз. Время перемотки и время слияния можно оценить, как в примере 6.

Пример 10. Осциллирующая сортировка с прямым чтением. Этот метод нелегко проанализировать, поскольку окончательная фаза ”чистки”, выполняемая после исчерпания ввода, не так эффективна, как предыдущие. Пренебрегая этим трудным аспектом и просто считая, что есть один дополнительный проход, можно оценить время слияния, полагая $\alpha = 1/\ln P, \beta = 0$ и $\pi' = \pi/P$. Распределение отрезков в этом случае несколько иное, так как не используется выбор с замещением; мы устанавливаем $P' = M/C$ и $S = \lceil N/P' \rceil$. Приложив усилия, можно совместить вычисление, чтение и запись во время распределения, вводя дополнительный коэффициент накладных расходов около $(M + 2B)/M$. Время переключения режимов, упомянутое в примере 9, в настоящем случае не нужно, так как оно совмещается с перемоткой. Итак, оценкой времени сортировки будет (9) плюс $2BNC\omega_i\tau/M$.

Таблица 1

Пример	P	B	P'	S	Сводная таблица оценок времени сортировки					(9)	Добавка к (9)	Оценка итога	Реальный итог
					ω	α	β	α'	β'				
1	3	12500	650	79	1.062	0.910	-1.000	0.303	0.000	1064		1064	1076
2	5	8300	734	70	1.094	0.795	-1.136	0.795	-1.136	1010	$\rho NC\omega_i\tau + \delta$	1113	1103
3	5	8300	734	70	1.094	0.773	-1.192	0.773	-1.192	972	$\rho NC\omega_i\tau + \delta$	1075	1127
4	4	10000	700	73	1.078	0.752	-0.994	0.000	0.720	844	$\rho NC\omega_i\tau + \delta$	947	966
5	4	10000	700	73	1.078	0.897	-1.200	0.173	0.129	972		972	992
6	3	12500	650	79	1.062	0.910	-1.000	0.000	0.167	981		981	980
7	4	10000	700	79	1.078	0.863	-1.079	0.000	0.000	922		922	907
8	4	10000	700	73	1.078	0.897	-1.200	0.098	0.117	952		952	949
9	4	10000	700	87	1.078	0.721	-1.000	0.000	0.125	846	$P' SC\omega_i\tau/P$	874	928
10	4	10000	-	100	1.078	0.721	0.000	0.180	0.000	1095	$2BNC\omega_i\tau/M$	1131	1158

Табл. 1 показывает, что оценки в этих примерах не слишком плохи, хотя в нескольких случаях расхождение составляет порядка 50 с. Формулы в примерах 2 и 3 показывают, что каскадное слияние должно быть предпочтительней многофазного на шести лентах, тем не менее на практике многофазное слияние лучше! Причина этого кроется в том, что графики, подобные изображенным на рис. 85 (где показан случай пяти лент), ближе к прямым линиям для многофазного алгоритма; каскадный метод превосходит многофазный на шести лентах для $14 \leq S \leq 15$ и $43 \leq S \leq 55$ вблизи ”точных” каскадных чисел 15 и 55, но многофазное распределение по алгоритму 5.4.2D лучше или эквивалентно для всех остальных $S \leq 100$. Каскадный метод предпочтительнее многофазного при $S \rightarrow \infty$, но фактически S не приближается к ∞ . Заниженная оценка в примере 9 обусловлена аналогичными обстоятельствами; многофазная сортировка лучше осциллирующей, несмотря на то что асимптотическая теория говорит нам, что для больших S осциллирующая сортировка будет наилучшей.

Несколько дополнительных замечаний. Сейчас самое время сделать несколько более или менее случайных наблюдений относительно ленточного слияния:

1) Приведенные формулы показывают, что сортировка является, в сущности, функцией от произведения N и C , а не от N и C порознь. За исключением нескольких относительно незначительных соображений (например, что B выбирается кратным C), из наших формул следует, что сортировка одного миллиона записей по 10 литер каждая займет примерно столько же времени, что и сортировка 100000 записей по 100 литер каждая. На самом деле здесь может появиться различие, не обнаружимое в наших формулах, так как во время выбора с замещением некоторое пространство используется для полей связи. В любом случае размер *ключа* едва ли окажет какое-либо влияние, если только ключи не будут столь длинными и сложными, что внутренние вычисления не смогут угнаться за лентами.

При длинных записях и коротких ключах соблазнительно выделить ключи, отсортировать их, а затем как-нибудь переставить записи целиком. Но эта идея, кажется, не работает: она может только отсрочить агонию, поскольку процедура окончательной перестановки требует почти столько же времени, сколько потребовала бы общепринятая сортировка слиянием.

2) Мы видели, что требуется от 15 до 19 мин, чтобы отсортировать 100000 записей по 100 литер при соблюдении наших предположений. Сколько времени заняла бы их сортировка на карточном сортировщике? Этот вопрос имеет практический интерес потому, что карточные сортировщики дешевле ЭВМ. Допуская, что каждая запись может быть втиснута в 80-колонную карту и что алфавитный ключ занимает шесть колонок, причем для сортировки по каждой колонке требуется в среднем $1\frac{2}{3}$ прохода, получаем, что мы должны пропустить каждую карту через машину примерно 10 раз. При скорости 1000 карт в минуту это заняло бы 1000 мин, т. е. почти 17 ч. (Имеется весьма большая вероятность, что за это время некоторые карты окажутся нечаянно "перепутанными" или будут "замяты" в машине.)

3) При написании программы сортировки, которая должна использоваться многократно, разумно очень тщательно оценить время работы и сравнить теорию с действительными наблюдаемыми характеристиками выполнения. Так как теория сортировки развита довольно хорошо, то эта процедура, как известно, способна внезапно выявить дефекты в оборудовании или программном обеспечении ввода/вывода в существующих системах. Оказывается обслуживание работало медленнее, чем следовало, и никто этого не замечал, пока это не проявилось на программе сортировки!

4) Некоторые вычислительные системы имеют два "банка" лентопротяжных устройств, присоединенных к отдельным "каналам" таким способом, что одновременное чтение и запись допускается только для лент из разных банков. Для такой конфигурации больше всего подходит сбалансированное слияние. Рассмотрим, например, случай шести лент по три в каждом банке и предположим, что мы хотим выполнить многофазное слияние с $T = 6$. Во время пятипутевого слияния две из вводных лент будут не в том банке, так что, грубо говоря, две пятых времени ввода не будут совмещены с выводом. Это добавляет ко времени сортировки приблизительно 40%, так что сбалансированное слияние окажется лучше многофазного даже в случае обратного чтения.

5) Наш анализ выбора с замещением был выполнен для "случайных" файлов, но файлы, встречающиеся на практике, очень часто уже упорядочены в той или иной степени. (Фактически иногда люди будут сортировать файлы, уже упорядоченные, только чтобы убедиться в этом.) Таким образом, опыт даже в большей мере, чем указывают наши формулы, показал, что выбор с замещением предпочтительнее других видов внутренней сортировки. Это преимущество несколько ослабляется в случае многофазной сортировки с обратным чтением, так как должен быть порожден ряд убывающих отрезков; на самом деле Р. Л. Гилстэд (он первый опубликовал многофазное слияние) первоначально по этой причине отверг метод обратного чтения. Но позднее он заметил, что чередование направлений будет все же давать длинные возрастающие отрезки. Кроме того, многофазный метод с обратным чтением—это единственный стандартный метод, который благосклонен к убывающим входным файлам, равно как и к возрастающим.

6) Другое преимущество выбора с замещением состоит в том, что этот метод допускает совмещение процессов чтения, записи и вычислений. Если бы мы просто выполняли внутреннюю сортировку очевидным способом—заполняя память, сортируя ее и затем записывая ее по мере того, как она заполняется новым содержимым,—то проход распределения занял бы примерно вдвое больше времени!

Из рассмотренных нами методов внутренней сортировки еще только один можно приспособить к одновременному чтению, записи и вычислениям—пирамидальную сортировку. (Эта идея была использована при подготовке примера 10 схемы А.) Предположим для удобства, что внутренняя память содержит 1000 записей, а каждый блок на ленте—по 100. Действовать можно следующим образом (через B_1, B_2, \dots, B_{10} обозначено содержимое памяти, разделенной на 10 блоков по 100 записей).

Шаг 0. Заполнить память и сделать так, чтобы элементы $B_2 \dots B_{10}$ удовлетворяли неравенствам пирамиды (с наименьшим элементом в вершине).

Шаг 1. Свести $B_1 \dots B_{10}$ в пирамиду, затем выбрать наименьшие 100 записей и переписать их в B_{10} .

Шаг 2. Записать из B_{10} ; в то же время выбрать наименьшие 100 записей из $B_1 \dots B_9$ и поместить их в B_9 .

Шаг 3. Прочитать в B_{10} и записать из B_9 ; в то же время выбрать наименьшие 100 записей из $B_1 \dots B_8$ и поместить их в B_8 .

...

Шаг 9. Прочитать в B_4 и записать из B_3 ; в то же время выбрать наименьшие 100 записей из $B_1 B_2$, поместить их в B_2 и сделать так, чтобы неравенства пирамиды были справедливы для $B_5 \dots B_{10}$.

Шаг 10. Прочитать в B_3 и записать из B_2 , сортируя B_1 , в то же время сделать так, чтобы неравенства пирамиды были справедливы для $B_4 \dots B_{10}$.

Шаг 11. Прочитать в B_2 и записать из B_1 ; в то же время сделать так, чтобы $B_3 \dots B_{10}$ удовлетворяли неравенствам пирамиды.

Шаг 12. Прочитать в B_1 , делая так, чтобы $B_2 \dots B_{10}$ удовлетворяли неравенствам пирамиды. Вернуться к шагу 1. ■

7) Мы предполагаем, что число сортируемых записей N не известно заранее. На самом же деле в большинстве вычислительных машин есть возможность все время следить за числом записей во всех файлах, и мы могли бы считать, что наша вычислительная система способна сообщить значение N . Насколько бы нам это помогло? К сожалению, не очень! Мы видели, что выбор с замещением весьма выгоден, но он ведет к непредсказуемому числу начальных отрезков. В сбалансированном слиянии мы могли бы использовать информацию об N для установления такого размера буфера B , чтобы S оказалось, скорее всего, чуть меньше степени P ; в многофазном распределении с оптимальным размещением фиктивных отрезков мы могли бы использовать информацию об N , чтобы решить, какой уровень выбрать (см. табл. 5.4.2-2).

Идя по другому пути и не используя выбора с замещением, мы могли бы применить слияние в прямом порядке, описанное в конце п. 5.4.4, встроив его в осциллирующую сортировку, распределяющую начальные отрезки соответствующего направления в соответствующий момент (вместо того чтобы брать их с "ленты А", как описано). Этот метод, в сущности, *оптимален* среди всех методов, выполняющих внутреннюю сортировку без выбора с замещением, так как он сливает в соответствии с наилучшим возможным P -арным деревом, но он все же работает медленнее методов, основанных на выборе с замещением.

8) Лентопротяжные устройства часто оказываются наименее надежными частями ЭВМ. Обслуживающий персонал обычно получает вызовы для исправления лентопротяжных устройств чаще, чем для любого другого компонента машины, и операторы вычислительной машины должны уметь возобновлять работу после сбоя ленты. Автор никогда даже не видел установок с 10 или более лентопротяжными устройствами, которые бы все одновременно находились в хорошем рабочем состоянии. Следовательно, можно принять за аксиому, что *исходная вводная лента ни в коем случае не должна изменяться, пока не станет известно, что вся сортировка удовлетворительно завершена*. В некоторых примерах схемы А существует досадное "время, пока оператор не сменит ленту", но было бы слишком рискованно затирать исходные данные ввиду возможности какой-либо неисправности во время длинной сортировки.

9) При переходе от прямой записи к обратному чтению мы можем сэкономить некоторое время, вовсе не записывая последний буфер на ленту; он в любом случае будет вновь прочитан! Схема А показывает, что этот прием в действительности экономит сравнительно немного времени, за исключением случая осциллирующей сортировки, когда направления меняются часто.

10) Если в нашем распоряжении много ленточных устройств, то не всегда стоит использовать их все в целях получения "высокого порядка слияния". Так, например, более высокий порядок слияния обычно означает меньший размер блока, а процентная разность между $\log_P S$ и $\log_{P+1} S$ не очень велика при больших P . Подумайте также о бедном операторе ЭВМ, который должен установить все эти рабочие ленты. С другой стороны, в упр. 12 описан интересный способ использования дополнительных лентопротяжных устройств, группируемых так, чтобы совместить время ввода и вывода без увеличения порядка слияния.

11) На машинах, подобных MIX, которые имеют фиксированный и довольно маленький размер блоков, для слияния едва ли требуется много внутренней памяти. Здесь осциллирующая сортировка более предпочтительна, потому что становится возможным сохранять дерево выбора с замещением в памяти во время слияния. На самом деле в этом случае можно усовершенствовать осциллирующую сортировку (как предложил К. Дж. Белл в 1962 г.), сливая новый начальный отрезок в вывод каждый раз, когда мы сливаем с рабочих лент!

12) Мы видели, что файлы на нескольких бобинах должны сортироваться последовательно бобина за бобиной, чтобы избежать чрезмерной работы по перестановке лент. Фактически сбалансированное

слияние с шестью лентами, если оно тщательно запрограммировано, может сортировать *три* бобины до момента окончательного слияния.

Для слияния относительно большого числа отдельно отсортированных бобин быстрее будет дерево слияния с минимальной длиной пути (ср. с п. 5.4.4). Это построение было впервые осуществлено Э. Х. Фрэндом [JACM, 3(1956), 166–167] и затем У. Х. Буржем [Information and Control, 1 (1958), 181–197], которые отметили, что оптимальный способ слияния отрезков данных (возможно, неравных) длин получается с помощью построения дерева с минимальной *взвешенной* длиной пути, используя длины отрезков в качестве весов (см. п. 2.3.4.5 и 5.4.9), если пренебречь временем установки лент. Но файлы, занимающие несколько бобин, вероятно, следует хранить на дисках или другом запоминающем устройстве большой емкости, а не на лентах.

13) В нашем обсуждении мы, не задумываясь, предполагали, что имеется возможность использовать непосредственно инструкции ввода/вывода и что никакой сложный системный интерфейс не мешает нам использовать ленты с такой эффективностью, на какую рассчитывали конструкторы аппаратуры. Эти идеальные предположения позволили нам проникнуть в суть проблем слияния, и они могут дать некоторый подход к конструированию соответствующих операционных систем. Однако следует понимать, что мультипрограммирование и мультипроцессирование могут значительно усложнить ситуацию.

14) Обсуждаемые нами вопросы были впервые рассмотрены в печати Э. Х. Фрэндом [JACM, 3 (1956), 134–165], У. Зобербергом [Electron. Datenverarb., 5 (1960), 28–44] и М. А. Готцем [Digital Computer User's Handbook (New York, McGraw-Hill, 1967) 1.292–1.320].

Резюме. Мы можем следующим образом вкратце выразить все, что узнали о сравнении различных схем слияния:

Теорема А. *Трудно решить, какая схема слияния является наилучшей в конкретной ситуации.* ■

Примеры, которые мы видели на схеме А, показывают, как 100000 записей по 100 литер (или 1 миллион записей по 10 литер), расположенных в случайном порядке, могли бы быть отсортированы с использованием шести лент при достаточно реалистических предположениях. Эти данные занимают около половины ленты и могут быть отсортированы приблизительно за 15–19 мин; однако существующее ленточное оборудование сильно различается по возможностям, и время выполнения такой работы на разных машинах изменяется в диапазоне приблизительно от четырех минут до двух часов. В наших примерах около 3 мин расходуется на начальное распределение отрезков и внутреннюю сортировку, около 4.5 мин — на окончательное слияние и перемотку выводной ленты и около 7.5–11.5 мин — на промежуточные стадии слияния.

Если имеется шесть лент, которые нельзя читать в обратном направлении, то наилучшим методом сортировки при наших предположениях было "многофазное слияние с расщеплением лент" (пример 4), а для лент, допускающих обратное чтение, наилучшим методом оказался многофазный метод с обратным чтением со сложным размещением фиктивных отрезков (пример 7). Осциллирующая сортировка (пример 9) занимает второе место. В обоих случаях каскадное слияние более просто и лишь незначительно медленнее (примеры 5 и 8). В случае прямого чтения обычное сбалансированное слияние (пример 1) оказалось удивительно эффективным, частично из-за удачи в этом конкретном примере, а частично из-за того, что оно тратит сравнительно мало времени на перемотку.

Положение несколько изменилось бы, если бы в нашем распоряжении было другое число лент.

Генераторы сортировки. В условиях большого разнообразия характеристик данных и оборудования почти невозможно написать единственную программу внешней сортировки, которая была бы удовлетворительной в подавляющем большинстве случаев. Также весьма трудно создать программу, которая в реальных условиях эффективно работает с лентами. Следовательно, изготовление программного обеспечения сортировки — самостоятельная задача, требующая большой работы. *Генератор сортировки* — это программа, которая, основываясь на параметрах, описывающих формат данных и конфигурацию оборудования, порождает машинную программу, специально приспособленную к конкретному применению сортировки. Подобная программа часто связана с языками высокого уровня, такими, как Кобол или PL/1, или она может быть написана как набор макроопределений для использования совместно с макроассемблером.

Одной из особенностей, обычно обеспечиваемых генератором сортировки, является возможность вставлять "собственные команды" — особые инструкции, которые должны включаться в первый и последний проходы программы сортировки. Собственные команды первого прохода обычно используются, чтобы отредактировать исходные записи, часто сокращая их или незначительно удлиняя, чтобы привести их к форме, более простой для сортировки. Пусть, например, исходные записи должны быть отсортированы по девятилитерному ключу, изображающему дату в формате месяц-день-год:

Трехбуквенные коды месяцев можно найти в таблице и заменить числами, причем наиболее значащие поля могут быть помещены слева:

17760704 15171031 16051105 17890714 19171120

Это уменьшает длину записей и делает более простым последующее сравнение. (Код ключей мог бы быть сделан даже более компактным.) Собственные команды последнего прохода могут использоваться для восстановления исходного формата и/или для внесения других желательных изменений в файл и/или для вычисления какой-либо функции от выводных записей. Алгоритмы слияния, которые мы изучили, организованы таким образом, что последний проход легко отличить от остальных фаз слияния. Заметим, что если имеются собственные команды, то должно быть по крайней мере два прохода по файлу, даже если он первоначально находился в порядке. Собственные команды, изменяющие размер записей, могут затруднить совмещение некоторых операций ввода/вывода в осциллирующей сортировке.

Генераторы сортировки также заботятся о системных деталях, таких, как соглашения о метках лент; они также часто обеспечивают подсчет контрольной суммы или иные проверки того, что никакая часть файла не пропала и не изменилась. Иногда имеются средства для остановки сортировки в удобных местах и возобновления ее позднее. Самые высококачественные генераторы позволяют записям иметь динамически меняющиеся длины [см. D. J. Waks, *CACM*, 6 (1963), 267–272].

***Слияние с меньшим числом буферов.** Мы видели, что $2P + 2$ буферов достаточно для поддержания быстрого движения лент в течение P -путевого слияния. В завершение этого пункта проведем математический анализ времени слияния в том случае, когда имеется меньше $2P + 2$ буферов.

Очевидно, что желательны два буфера вывода, так как мы сможем записывать из одного буфера, образуя в это же время следующий блок вывода в другом. Поэтому мы можем вообще не рассматривать вопрос вывода и заняться только вводом.

Допустим, имеется $P + Q$ буферов ввода, где $1 \leq Q \leq P$. Воспользуемся для описания нашей ситуации моделью, предложенной Л. Дж. Вудрамом [*IBM Systems J.*, 9 (1970), 118–144]. Чтение одного блока ленты занимает одну единицу времени. Имеется вероятность p_0 того, что в течение этого времени ни один из буферов ввода не станет пустым, p_1 — что один буфер станет пустым, $p_{\geq 2}$ — что два или больше буферов станут пустыми и т. д. По завершении чтения ленты мы оказываемся в одном из $Q + 1$ состояний:

Состояние 0: Q буферов пусты. Мы начинаем читать блок подходящего файла в один из них, используя метод прогнозирования, описанный ранее в этом пункте. Через одну единицу времени мы переходим в состояние 1 с вероятностью p_0 , в противном случае мы остаемся в состоянии 0.

Состояние 1: $Q - 1$ буферов пусты. Мы начинаем читать в один из них, предсказывая подходящий файл. Через одну единицу времени мы переходим в состояние 2 с вероятностью p_0 , в состояние 1 с вероятностью p_1 и в состояние 0 с вероятностью $p_{\geq 2}$.

⋮

Состояние $Q - 1$: один буфер пуст. Мы начинаем читать в него, предсказывая подходящий файл. Через одну единицу времени мы переходим в состояние Q с вероятностью p_0 , в состояние $Q - 1$ с вероятностью p_1, \dots , в состояние 1 с вероятностью p_{Q-1} , и в состояние 0 с вероятностью $p_{\geq Q}$.

Состояние Q : все буферы заполнены. Чтение лент останавливается в среднем на μ единиц времени, и затем мы переходим в состояние 1.

Мы начинаем с состояния 0. Эта модель ситуации соответствует марковскому процессу (см. упр. 2.3.4.2-26), который можно проанализировать с помощью производящих функций следующим интересным способом. Пусть z — произвольный параметр, и предположим, что каждый раз, когда мы решили читать с ленты, делаем это с вероятностью z , а с вероятностью $1 - z$ завершаем алгоритм. Пусть $g_Q(z) = \sum_{n \geq 0} a_n^{(Q)} z^n (1 - z)$ будет средним числом появлений в этом процессе состояния Q ; отсюда следует, что $a_n^{(Q)}$ — это среднее число появлений состояния Q , если было прочитано ровно n блоков. Тогда $n + a_n \mu$ — среднее суммарное время, затраченное на ввод и вычисления. Если бы имелось полное совмещение, как в алгоритме с $(2P + 2)$ буферами, то суммарное время включало бы только n единиц, так что $a_n \mu$ представляет время задержки чтения.

Пусть A_{ij} — вероятность того, что мы переходим из состояния i в состояние j в этом процессе при $0 \leq i, j \leq Q + 1$, где $Q + 1$ — новое состояние "остановки". Например, для $Q = 1, 2, 3$ матрицы A

будут следующими:

$$Q = 1 : \begin{pmatrix} p_{\geq 1}z & p_0z & 1-z \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$Q = 2 : \begin{pmatrix} p_{\geq 1}z & p_0z & 0 & 1-z \\ p_{\geq 2}z & p_1z & p_0z & 1-z \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$Q = 3 : \begin{pmatrix} p_{\geq 1}z & p_0z & 0 & 0 & 1-z \\ p_{\geq 2}z & p_1z & p_0z & 0 & 1-z \\ p_{\geq 3}z & p_2z & p_1z & p_0z & 1-z \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Из упр. 2.3.4.2-26b имеем, что $g_Q(z) =$ алгебраическое дополнение $Q_0(I - A)/\det(I - A)$. Так, например, если $Q = 1$, имеем

$$g_1(z) = \det \begin{pmatrix} 0 & -p_0z & z-1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} / \det \begin{pmatrix} 1-p_{\geq 1}z & -p_0z & z-1 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} =$$

$$= \frac{p_0z}{1-p_1z-p_0z} = \frac{p_0z}{1-z} = \sum_{n \geq 0} np_0z^n(1-z),$$

так что $a_n^{(1)} = np_0$. Это, конечно, очевидно заранее, так как при $Q = 1$ задача очень проста. Аналогичное вычисление для $Q = 2$ (см. упр. 14) дает менее очевидную формулу:

$$a_n^{(2)} = \frac{p_0^2 n}{1-p_1} - \frac{p_0^2(1-p_1^n)}{(1-p_1)^2}. \quad (10)$$

В общем случае можно показать, что $a_n^{(Q)}$ имеет вид $\alpha^{(Q)}n + O(1)$ при $n \rightarrow \infty$, где константу $\alpha^{(Q)}$ не слишком трудно вычислить. (См. упр. 15.) Как оказывается, $\alpha^{(3)} = p_0^3/((1-p_1)^2 - p_0p_2)$.

Исходя из природы слияния, довольно разумно предположить, что $\mu = 1/P$ и что вероятности p_k соответствуют биномиальному распределению

$$p_k = \binom{P}{k} \left(\frac{1}{P}\right)^k \left(\frac{P-1}{P}\right)^{P-k}$$

Например, если $P = 5$, то $p_0 = .32768$, $p_1 = .4096$, $p_2 = .2048$, $p_3 = .0512$, $p_4 = .0064$ и $p_5 = .00032$; следовательно, $\alpha^{(1)} = 0.328$, $\alpha^{(2)} = 0.182$ и $\alpha^{(3)} = 0.127$. Другими словами, если мы используем $5 + 3$ вводных буферов вместо $5 + 5$, то можно ожидать дополнительного времени задержки чтения порядка $0.127/5 \approx 2.5\%$.

Конечно, эта модель—только очень грубое приближение. Мы знаем, что при $Q = P$ вообще нет времени задержки, но если судить по модели, то есть. Дополнительное время задержки чтения для меньших Q почти точно уравнивает выигрыш в накладных расходах, получаемый от использования более крупных блоков, так что простая схема с $Q = P$ кажется оправданной.

Упражнения

1. [13] Выведите формулу для точного числа литер на ленте, если каждый блок содержит n литер. Считайте, что лента могла бы вместить ровно 23000000 литер, если бы не было межблочных промежутков.
2. [15] Объясните, почему первый буфер файла 2 в строке 6 рис. 84 совсем пуст.
3. [20] Будет ли алгоритм F работать должным образом, если вместо $2P$ буферов ввода имеется только $2P - 1$? Если да, докажите это, если нет—приведите пример, когда алгоритм терпит неудачу.
4. [20] Как изменить алгоритм F, чтобы он работал также и при $P = 1$?
5. [21] Если в различных файлах имеются равные ключи, необходимо в процессе прогнозирования действовать очень аккуратно. Объясните, почему, и покажите, как избежать трудностей, если более строго определить операции слияния и прогнозирования в алгоритме F.

6. [22] Какие изменения следует сделать в алгоритме 5.4.3С, чтобы преобразовать его в алгоритм каскадного слияния с совмещением перемотки на $T + 1$ лентах?
 >7. [26] Начальное распределение в примере 7 схемы А порождает

$$(A_1 D_1)^{11} D_1 (A_1 D_1)^{10} D_1 (A_1 D_1)^9 D_1 (A_1 D_1)^7$$

на лентах 1–4, где $(A_1 D_1)^7$ означает $A_1 D_1 A_1 D_1 A_1 D_1 A_1 D_1 A_1 D_1 A_1 D_1 A_1 D_1$. Покажите, как вставить дополнительные отрезки A_0 и D_0 наилучшим из возможных способов (в том смысле, что общее число обрабатываемых во время слияния начальных отрезков будет минимальным), чтобы привести распределение к

$$A(DA)^{14} (DA)^{28} (DA)^{26} (DA)^{22}.$$

[Указание. Чтобы сохранить четность, необходимо вставлять A_0 и D_0 в виде соседних пар. Числа слияния для каждого начального отрезка могут быть подсчитаны, как в упр 5.4.4-5; здесь появляется некоторое упрощение, так как соседние отрезки всегда имеют соседние числа слияния.]

8. [20] Из схемы А видно, что большинство схем начального распределения отрезков (за исключением начального распределения для каскадного слияния) имеет тенденцию помещать последовательные отрезки на различные ленты. Если бы последовательные отрезки попали на одну ленту, то мы могли бы сэкономить стартстопное время; можно ли поэтому считать хорошей мыслью изменить алгоритмы распределения так, чтобы они реже переключали ленты?
- >9. [22] Оцените, сколько времени занял бы многофазный алгоритм с обратным чтением из схемы А, если бы мы использовали для сортировки все $T = 6$ лент, а не $T = 5$, как в примере 7. Было ли разумно избегать использования вводной ленты?
10. [M23] Используя анализ, проведенный в п. 5.4.2 и 5.4.3, покажите, что длина каждой перемотки во время стандартного многофазного слияния с шестью лентами или каскадного слияния редко превышает 54% файла (исключая начальную и конечную перемотки, которые охватывают весь файл).
11. [23] Изменив подходящие элементы табл. 1, оцените, сколько времени заняли бы первые девять примеров схемы А, если бы мы имели двухскоростную перемотку (быструю и медленную). Считайте, что $p = 1$, если лента заполнена меньше чем на одну четверть, а для более заполненной ленты время перемотки равно приблизительно пяти секундам плюс то время, которое получилось бы при $p = 1/5$. Измените пример 8 так, чтобы он использовал каскадное слияние с копированием, поскольку перемотка и прямое чтение в этом случае медленнее копирования [Указание: используйте результат упр. 10].
12. [40] Рассмотрим разбиение шести лент на три пары лент, где каждая пара играет роль одной ленты в многофазном слиянии с $T = 3$. Одна лента каждой пары будет содержать блоки 1, 3, 5, ..., а другая — блоки 2, 4, 6, ..., таким способом мы, по существу, добиваемся того, чтобы во все время слияния две вводные и две выводные ленты оставались активными, причем эффективная скорость слияния удваивается.
- (a) Найдите подходящий способ распространить алгоритм F на этот случай.
 (b) Оцените общее время выполнения, которое получилось бы, если бы этот метод был использован для сортировки 100000 записей по 100 литер, рассмотрев случай как прямого, так и обратного чтения.
13. [20] Может ли осциллирующая сортировка с пятью лентами, в том виде как она определена в алгоритме 5.4.5В, использоваться для сортировки четырех полных бобин исходных данных до момента окончательного слияния?
14. [M19] Выведите (10).
15. [BM29] Докажите, что $g_Q(z) = h_Q(z)/(1 - z)$, где $h_Q(z)$ является рациональной функцией z , не имеющей особенностей внутри единичного круга; следовательно, $a_n^{(Q)} = h_Q(1)n + O(1)$ при $n \rightarrow \infty$. В частности, покажите, что

$$h_3(1) = \det \begin{pmatrix} 0 & -p_0 & 0 & 0 \\ 0 & 1 - p_1 & -p_0 & 0 \\ 0 & -p_2 & 1 - p_1 & -p_0 \\ 1 & 0 & 0 & 0 \end{pmatrix} / \det \begin{pmatrix} 1 & -p_0 & 0 & 0 \\ 1 & 1 - p_1 & -p_0 & 0 \\ 1 & -p_2 & 1 - p_1 & -p_0 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

15. [M46] Проанализируйте слияние с $P + Q$ буферами более тщательно, чем это было сделано в тексте, используя более точную модель.
17. [41] Проведите детальное изучение задачи сортировки 100000 записей по 100 литер, нарисуйте схемы, подобные схеме А, в предположении, что имеется 3, 4 или 5 лент.

5.4.7. *Внешняя поразрядная сортировка

В предыдущих пунктах мы рассмотрели процесс ленточной сортировки слиянием; но существует и другой способ сортировки на лентах, основанный на принципе поразрядной сортировки, используемой в механических сортировальных машинах для перфокарт (ср. с п. 5.2.5). Этот метод иногда называют распределяющей сортировкой, поколонной сортировкой, карманной сортировкой, цифровой сортировкой, сортировкой разделением и т. д. Он, как оказывается, по существу, *противоположен* слиянию!

Предположим, например, что в нашем распоряжении имеются четыре ленты, а ключей может быть только восемь: 0, 1, 2, 3, 4, 5, 6, 7. Если исходные данные находятся на ленте $T1$, то начнем с переписи всех четных ключей на $T3$ и всех нечетных на $T4$:

	$T1$	$T2$	$T3$	$T4$
Дано	{0, 1, 2, 3, 4, 5, 6, 7}	—	—	—
Проход 1	—	—	{0, 2, 4, 6}	{1, 3, 5, 7}

Теперь перематываем ленты и читаем $T3$, а затем $T4$, помещая {0, 1, 4, 5} на $T1$ и {2, 3, 6, 7} на $T2$:

Проход 2.	{0, 4}{1, 5}	{2, 6}{3, 7}	—	—
-----------	--------------	--------------	---	---

(Строка {0, 4}{1, 5} обозначает файл, содержащий записи только с ключами 0 и 4, за которыми следуют записи только с ключами 1 и 5. Заметим, что $T1$ теперь содержит те ключи, средний двоичный разряд которых содержит 0.) После еще одной перемотки и распределения ключей 0, 1, 2, 3 на $T3$ и ключей 4, 5, 6, 7 на $T4$ мы имеем

Проход 3	—	{0}{1}{2}{3} {4}{5}{6}{7}		
----------	---	---------------------------	--	--

Теперь копирование $T4$ в конец $T3$ завершает работу. В общем случае для ключей в диапазоне от 0 до $2^k - 1$ можно отсортировать файл аналогичным образом, используя k проходов, за которыми следует фаза окончательной "сборки", копирующая примерно половину данных с одной ленты на другую. Имея шесть лент, мы можем аналогичным образом использовать представления по основанию 3 для сортировки ключей от 0 до $3^k - 1$ за k проходов.

Используются также методы с частичными проходами. Предположим, например, что допускается десять ключей {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, и рассмотрим следующую процедуру, принадлежащую Р. Л. Эшенхерсту [*Theory of Switching*, 7 (Harvard Univ. Comp. Laboratory: May, 1954), I.1–I.76]:

	$T1$	$T2$	$T3$	$T4$	
Дано	{0, 1, ..., 9}	—	—	—	
Проход 1.	—	{0, 2, 4, 7}	{1, 5, 6}	{3, 8, 9}	1.0 прохода
Проход 2.	{0}	—	{1, 5, 6}{2, 7}	{3, 8, 9}{4}	0.4 прохода
Проход 3.	{0}{1}{2}	{6}{7}	—	{3, 8, 9}{4}{5}	0.5 прохода
Проход 4.	{0}{1}{2}{3}	{6}{7}{8}	{9}	{4}{5}	0.3 прохода
Сборка	{0}{1}{2}{3}{4}...{9}				0.6 прохода
					<u>2.8 прохода</u>

Если каждое значение ключа встречается примерно в одной десятой случаев, то эта процедура для сортировки десяти ключей затрачивает только 2.8 прохода, в то время как первый пример требует 3.5 прохода для сортировки всего восьми ключей. Таким образом, мы видим, что искусная схема распределения может вызвать значительное различие для поразрядной сортировки точно так же, как для слияния.

Схемы распределения предыдущих примеров представим, как и обычно, древовидными структурами:

Picture: рис. на стр. 412

Круглые внутренние узлы этих деревьев занумерованы 1, 2, 3, ... в соответствии с шагами процесса 1, 2, 3, ... Имена лент A, B, C, D (вместо $T1, T2, T3, T4$) помещены рядом с ребрами дерева, чтобы указать, куда попадают записи. Квадратные внешние узлы изображают части файла, содержащие только один ключ, и этот ключ изображен жирным шрифтом под соответствующим узлом. Ребра над квадратными узлами все помечены именем выводной ленты (C в первом примере, A во втором).

Таким образом, шаг 3 примера 1 состоит из чтения с ленты D и записи ключей 1 и 5 на ленту A и ключей 3 и 7 на ленту B . Нетрудно видеть, что число выполняемых проходов равно длине внешнего пути дерева, деленной на число внешних узлов, если мы принимаем, что все ключи равновероятны.

В силу последовательной природы ленты и дисциплины "первым включается—первым исключается", которой подчиняется прямое чтение, нельзя взять за основу схемы распределения *любое*

помеченное дерево. В дереве примера 1 данные записываются на ленту A на шаге 2 и шаге 3; данные, записанные в течение шага 2, необходимо использовать раньше данных, записанных в течение шага 3. В общем случае, если мы записываем на ленту в течение шагов i и j , где $i < j$, первыми следует использовать данные, записанные в течение шага i ; если дерево содержит две ветви вида

Picture: рис. стр. 413

то должно выполняться условие $k < l$. Кроме того, мы не можем ничего записывать на ленту A между шагами k и l , поскольку между чтением и записью необходима перемотка.

Те читатели, которые проработали упражнения п. 5.4.4, немедленно поймут, что допустимые деревья для поразрядной сортировки с прямым чтением на T лентах — это в точности "сильные T -fifo" деревья, описывающие сортировку *слиянием* на T лентах с прямым чтением! (См. упр. 5.4.4-20.) Единственное различие заключается в том, что все внешние узлы рассматриваемых здесь деревьев помечены одной и той же лентой. Мы могли бы снять это ограничение, предположив, что существует окончательная фаза "сборки", переносящая все записи на выводную ленту, или могли бы добавить это ограничение к правилам для T -fifo деревьев, потребовав, чтобы начальный распределительный проход сортировки *слиянием* был явно выражен в соответствующем дереве *слияния*.

Иными словами, *каждой схеме слияния соответствует схема распределения и каждой схеме распределения соответствует схема слияния*. По некотором размышлении это становится понятным. Рассмотрим сортировку *слиянием*, делающую все наоборот, т. е. "разливающую" окончательный выводной файл в подфайлы, которые "разливаются" в другие, и т. д.; наконец, мы разольем файл в S отрезков. Подобная схема возможна с лентами тогда и только тогда, когда возможна соответствующая схема распределения для поразрядной сортировки S ключей. Эта двойственность *слияния* и *распределения* почти точна; она не выполняется только в одном отношении: вводная лента должна сохраняться в разные моменты времени.

Пример с восемью ключами, рассмотренный в начале этого пункта, очевидно, двойствен сбалансированному *слиянию* на четырех лентах. Пример с десятью ключами и частичными проходами соответствует следующей схеме *слияния* десяти отрезков (если мы скроем фазы копирования — шаги 6–11 в дереве):

	$T1$	$T2$	$T3$	$T4$
Начальное распределение	1^4	1^3	1^1	1^2
Шаг дерева 5	1^3	1^2	—	$1^2 3^1$
Шаг дерева 4	1^2	1^1	2^1	$1^2 3^1$
Шаг дерева 3	1^1	—	$2^1 3^1$	$1^1 3^1$
Шаг дерева 2	—	4^1	3^1	3^1
Шаг дерева 1	10^1	—	—	—

Если сравнить ее с поразрядной сортировкой, то видно, что оба метода имеют, в сущности, одну и ту же структуру, но обратны во времени и имеют обратное расположение содержимого на лентах: $1^2 3^1$ (два отрезка длины 1 каждый, за которыми находится один отрезок длины 3) соответствует $\{3, 8, 9\}$ (два подфайла, содержащие каждый по 1 ключу, перед которыми расположен один подфайл, содержащий 3 ключа).

Двигаясь в другую сторону, мы можем в принципе построить поразрядную сортировку, двойственную многофазному *слиянию*, каскадному *слиянию* и т. д. Например, многофазному *слиянию* 21 отрезка на трех лентах, изображенному в начале п. 5.4.2, соответствует следующая интересная поразрядная сортировка:

	$T1$	$T2$	$T3$
Дано	$\{0, 1, \dots, 20\}$	—	—
Проход 1	—	$\{0, 2, 4, 5, 7, 9, 10, 12, 13, 15, 17, 18, 20\}$	$\{1, 3, 6, 8, 11, 14, 16, 19\}$
Проход 2	$\{0, 5, 10, 13, 18\}$	—	$\{1, 3, 6, 8, 11, 14, 16, 19\} \{2, 4, 7, 9, 12, 15, 17, 20\}$
Проход 3	$\{0, 5, 10, 13, 18\} \{1, 6, 11, 14, 19\} \{2, 7, 12, 15, 20\}$	$\{3, 8, 16\} \{4, 9, 17\}$	—
Проход 4	—	$\{3, 8, 16\} \{4, 9, 17\} \{5, 10, 18\} \{6, 11, 19\} \{7, 12, 20\}$	$\{0, 13\} \{1, 14\} \{2, 15\}$
Проход 5	$\{8\} \{9\} \{10\} \{11\} \{12\}$	—	$\{0, 13\} \{1, 14\} \{2, 15\} \{3, 16\} \dots \{7, 20\}$
Проход 6	$\{8\} \{9\} \{10\} \{11\} \{12\} \{13\} \dots \{20\}$	$\{0\} \{1\} \dots \{7\}$	—

Правило распределения, согласно которому ключи располагаются на лентах на каждом шаге, кажется магическим, но на самом деле оно имеет простую связь с системой счисления, использующей числа Фибоначчи! (См. упр. 2.)

Обратное чтение. Двойственность поразрядной сортировки и *слияния* применима также и к алгоритмам, читающим ленту в обратном направлении. Мы определили " T -lifo деревья" в п. 5.4.4, и

нетрудно видеть, что они подходят для поразрядной сортировки в той же мере, что и для сортировки слиянием.

Поразрядная сортировка с обратным чтением была фактически рассмотрена Джоном Мочли еще в 1946 г. в одной из первых опубликованных работ по сортировке вообще (см. § 5.5); Мочли на самом деле дал следующую конструкцию:

	$T1$	$T2$	$T3$	$T4$
Дано	—	$\{0, 1, \dots, 9\}$	—	—
Проход 1	$\{4, 5\}$	—	$\{2, 3, 6, 7\}$	$\{0, 1, 8, 9\}$
Проход 2	$\{4, 5\}\{2, 7\}$	$\{3, 6\}$	—	$\{0, 1, 8, 9\}$
Проход 3	$\{4, 5\}\{2, 7\}\{0, 9\}$	$\{3, 6\}\{1, 8\}$	—	—
Проход 4	$\{4, 5\}\{2, 7\}$	$\{3, 6\}\{1, 8\}$	$\{9\}$	$\{0\}$
...				
Проход 8	—	—	$\{9\}\{8\}\{7\}\{6\}\{5\}$	$\{0\}\{1\}\{2\}\{3\}\{4\}$
Окончательная сборка	—	—	$\{0\}\{1\}\{2\}\{3\}\{4\}\{5\} \dots \{9\}$	

Эта схема не является наиболее эффективной среди всех возможных, но она интересна тем, что показывает, что методы с частичными проходами рассматривались для поразрядной сортировки еще в 1946 г., хотя в литературе по слиянию они появились лишь около 1960 г.

Эффективная конструкция схем распределения с обратным чтением была предложена Э. Байесом [JACM, 11 (1968), 491–493]. Пусть дано $P + 1$ лент и S ключей; разделите ключи на P подфайлов, каждый из которых содержит $\lfloor S/P \rfloor$ или $\lceil S/P \rceil$ ключей, и применяйте эту процедуру рекурсивно к каждому подфайлу. Если $S < 2P$, то один подфайл должен состоять из единственного наименьшего ключа; его и следует записать на выводную ленту. (Общая конструкция с прямым порядком Р. М. Карпа, описанная в конце п. 5.4.4, включает этот метод как частный случай.)

Обратное чтение несколько усложняет слияние, поскольку оно обращает порядок отрезков. Соответствующий эффект имеется и в поразрядной сортировке. Результат оказывается устойчивым или "антиустойчивым" в зависимости от того, какой уровень достигнут в дереве. После поразрядной сортировки с обратным чтением, когда некоторые внешние узлы находятся на четных уровнях, а некоторые — на нечетных, для одних ключей относительный порядок различных записей с одинаковыми ключами будет *совпадать* с первоначальным порядком, но для других он будет *противоположен* исходному. (Ср. с упр. 6.)

Осциллирующая сортировка слиянием также имеет свою пару в этой двойственности. В *осциллирующей поразрядной сортировке* мы продолжаем разделять ключи, пока не достигнем подфайлов, содержащих только один ключ или достаточно малых, чтобы поддаваться внутренней сортировке; такие подфайлы сортируются и записываются на выводную ленту, затем процесс разделения возобновляется. Например, если имеются три рабочие ленты и одна выводная и если ключи являются двоичными числами, мы можем начать с того, что поместим ключи вида '0x' на ленту T1, а ключи '1x' на ленту T2. Если на ленте T1 окажется больше записей, чем емкость памяти, то вновь просматриваем ее и помещаем '00x' на T2 и '01x' на T3. Теперь, если подфайл '00x' достаточно короткий, производим внутреннюю сортировку его и выводим результат, а затем начинаем обработку подфайла '01x'. Подобный метод был назван Э. Х. Фрэндом "каскадной псевдопоразрядной сортировкой" [JACM, 3 (1956), 157–159]; более подробно его разработали Х. Нэглер [JACM, 6 (1959), 459–468], который дал ему красочное имя "метод двуглавого змия", и К. Х. Годетт [IBM Tech. Disclosure Bull., 12 (April, 1970), 1849–1853].

Превосходит ли поразрядная сортировка слияние? Одним важным следствием принципа двойственности является то, что *поразрядная сортировка обычно хуже сортировки слиянием*. Это связано с тем, что метод выбора с замещением дает сортировке слиянием определенное преимущество: нет очевидного пути так построить поразрядную сортировку, чтобы можно было использовать внутренние сортировки, включающие более одной емкости памяти за один раз. На самом деле осциллирующая поразрядная сортировка часто будет порождать подфайлы, несколько меньшие емкости памяти, так что ее схема распределения соответствует дереву со значительно большим числом внешних узлов, чем было бы при использовании слияния и выбора с замещением. Соответственно возрастает длина внешнего пути дерева (т. е. время сортировки). (См. упр. 5.3.1–33.)

Для внешней поразрядной сортировки существует, однако, одно важное применение. Предположим, например, что имеется файл, содержащий фамилии всех сотрудников большого предприятия в алфавитном порядке; предприятие состоит из 10 отделений, и требуется отсортировать этот файл по отделениям, *сохраняя* алфавитный порядок сотрудников в каждом отделении. Если файл длинный, то мы имеем дело именно с той ситуацией, где следует применять стабильную поразрядную сортировку, так как число записей, принадлежащих каждому из 10 отделений, будет, вероятно, больше, чем

число записей, которое было бы в начальных отрезках, полученных выбором с замещением. Вообще говоря, если диапазон ключей так мал, что набор записей с одинаковыми ключами более чем вдвое превысит оперативную память, то разумно использовать поразрядную сортировку.

Мы видели в п. 5.2.5, что на некоторых высокоскоростных ЭВМ *внутренняя* поразрядная сортировка предпочтительнее слияния, поскольку "внутренний цикл" поразрядной сортировки обходится без сложных переходов. Если внешняя память очень быстрая, то для таких машин может оказаться проблемой проводить слияние данных с такой скоростью, чтобы успеть за оборудованием ввода/вывода. Поэтому в подобной ситуации поразрядная сортировка, возможно, лучше слияния, особенно если известно, что ключи равномерно распределены.

Упражнения

- [20] Ближе к началу п. 5.4 было определено общее сбалансированное слияние на T лентах с параметром P , $1 \leq P < T$. Покажите, что оно соответствует поразрядной сортировке, использующей систему счисления со смешанным основанием.
- [M28] В тексте схематически представлена многофазная поразрядная сортировка 21 ключа! Обобщите ее на случай F_n ключей; объясните, какие ключи и на какой ленте оказываются в конце каждой фазы. [Указание: рассмотрите систему счисления, использующую числа Фибоначчи; упр. 1.2.8-34.]
- [M40] Распространите результаты упр. 2 на многофазную поразрядную сортировку с четырьмя или большим количеством лент (ср. с упр. 5.4.2-10).
- [M20] Докажите, что схема распределения Эшенхерста служит наилучшим способом сортировки 10 ключей на четырех лентах без обратного чтения в том смысле, что соответствующее дерево имеет минимальную длину внешнего пути среди всех "сильных 4-fifo деревьев". (Таким образом, это, по существу, наилучший метод, если не учитывать время перемотки.)
- [15] Нарисуйте 4-lifo дерево, соответствующее поразрядной сортировке Мочли с обратным чтением 10 ключей.
- >6. [20] Некоторый файл содержит двухразрядные ключи 00, 01, ..., 99. После выполнения поразрядной сортировки Мочли по цифре единиц мы можем повторить ту же схему по цифре десятков, поменяв ролями ленты $T2$ и $T4$. В каком порядке в конце концов окажутся ключи на $T2$?
- [21] Применим ли принцип двойственности также и к файлам на нескольких бобинах?

5.4.8. Сортировка с двумя лентами

Для того чтобы при выполнении слияния не было чрезмерного движения лент, необходимы три ленты. Интересно подумать о том, как можно разумным образом выполнить внешнюю сортировку с использованием только двух лент.

В 1956 г. Г. Б. Демут предложил некий метод, представляющий собой комбинацию выбора с замещением и сортировки методом пузырька. Предположим, что исходные данные занимают ленту $T1$, и начнем с того, что прочитаем в память $P + 1$ записей. Теперь выведем запись с наименьшим ключом на ленту $T2$ и заменим ее следующей исходной записью. Продолжаем выводить записи, ключ которых в текущий момент наименьший в памяти, сохраняя дерево выбора или приоритетную очередь из $P + 1$ элементов. Когда ввод наконец исчерпается, в памяти окажутся наибольшие P ключей файла; выведем их в возрастающем порядке. Теперь перемотаем обе ленты и повторим этот процесс, читая с $T2$ и записывая на $T1$; каждый такой проход помещает еще по крайней мере P записей на свои места. В программу можно встроить простую проверку для определения момента, когда весь файл станет упорядоченным. Потребуется не более $\lceil (N - 1)/P \rceil$ проходов.

Минутное размышление показывает, что каждый проход этой процедуры эквивалентен P последовательным проходам сортировки методом пузырька (алгоритм 5.2.2B)! Если элемент имеет P или более инверсий, то при вводе он окажется меньше всех элементов в дереве и поэтому будет немедленно выведен (потеряв, таким образом, P инверсий). Если элемент имеет менее P инверсий, то он попадает в дерево выбора и будет выведен раньше всех больших ключей (потеряв, таким образом, все свои инверсии). Если $P = 1$, то происходит то же самое, что и в методе пузырька, по теореме 5.2.21.

Общее число проходов будет, следовательно, равно $\lceil I/P \rceil$, где I — максимальное число инверсий любого элемента. По теории, развитой в п. 5.2.2, среднее значение I есть $N - \sqrt{\pi N/2} + (2/3) + O(1/\sqrt{N})$.

Если файл не слишком сильно превосходит размер оперативной памяти или если он первоначально почти упорядочен, то эта сортировка методом пузырька P -го порядка будет довольно быстрой; в действительности ее можно предпочесть даже в том случае, когда имеются дополнительные лентопротяжные устройства, так как весь процесс сортировки может закончиться раньше, чем оператор успеет установить третью ленту! С другой стороны, она будет работать весьма медленно над довольно большими файлами со случайным расположением элементов, так как время ее работы приблизительно пропорционально N^2 .

Посмотрим, как реализуется этот метод для 100000 записей в примере из п. 5.4.6. Нам нужно разумно выбрать P , чтобы учесть межблочные промежутки при совмещении операций чтения и записи с вычислениями. Так как в примере предполагается, что каждая запись имеет длину 100 литер, а 100000 литер заполняют память, то у нас будет место для двух буферов ввода и двух буферов вывода размера B , если выбрать значения P и B , такие, что

$$100(P + 1) + 4B = 100000. \quad (1)$$

Если использовать обозначения п. 5.4.6, то приблизительное время работы каждого прохода выражается как

$$NC\omega\tau(1 + \rho), \quad \omega = (B + \gamma)/B. \quad (2)$$

Поскольку число проходов обратно пропорционально P , мы хотим выбрать такое B , кратное 100, которое минимизирует величину ω/P . Элементарный анализ показывает, что минимум достигается, когда B равно приблизительно $\sqrt{24975\gamma} + \gamma^2 - \gamma$. Поэтому мы выбираем $B = 3000$, $P = 879$. Положив в приведенных выше формулах $N = 100000$, получаем, что число проходов $\lceil I/P \rceil$ будет около 114, а оценка общего времени решения составляет примерно 8.57 ч (предполагая для удобства, что исходные данные и окончательный вывод также имеют $B = 3000$). Здесь представлен случай, когда данные занимают около 0.44 бобины; полная бобина потребовала бы примерно в пять раз больше времени. Можно произвести некоторые улучшения, предусмотрев в алгоритме периодические прерывания и пересылку записей с наибольшими ключами на вспомогательную ленту, которая затем снимается, несколько эти записи просто копируются туда и обратно после того, как они уже оказались на своих местах.

Применение быстрой сортировки. Еще одним методом внутренней сортировки, который проходит данные почти последовательно, является обменная сортировка с разделением или быстрая сортировка (алгоритм 5.2.2Q) Можно ли ее приспособить к двум лентам? [N. В; Yoash, *CACM*, 8 (1965), 649.]

Нетрудно увидеть как можно сделать это, воспользовавшись обратным чтением. Предположим, что две ленты помечены 0 и 1, и представим, что файл располагается следующим образом:

Picture: Расположение файла на ленте, стр. 419

Каждая лента выступает в качестве стека. Две ленты вместе, используемые как представлено здесь, дают возможность считать файл линейным списком, в котором мы можем перемещать текущую позицию влево или вправо, копируя элементы из одного стека в другой. Следующие рекурсивные подпрограммы определяют соответствующую процедуру сортировки.

- **SORT00** [отсортировать верхний подфайл с ленты 0 и вернуть его на ленту 0]. Если подфайл помещается в оперативную память, то применить к нему внутреннюю сортировку и затем вернуть его на ленту. В противном случае выбрать одну запись R из подфайла; пусть ее ключом будет K . Читая ленту 0 в обратном направлении, копировать все записи, ключи которых $> K$, получая таким образом новый "верхний" подфайл на ленте 1. Теперь, читая ленту 0 в прямом направлении, копировать все записи с ключами, равными K , на ленту 1. Затем, вновь читая ленту 0 в обратном направлении, копировать все записи с ключами $< K$ на ленту 1. Выполнить **SORT10** над ключами $< K$, затем скопировать ключи, равные K , на ленту 0 и, наконец, выполнив **SORT10** над ключами $> K$, завершить сортировку.
- **SORT01** [отсортировать верхний подфайл с ленты 0 и записать его на ленту 1]. Аналогично **SORT00**, но последнее обращение к "**SORT10**" заменено на "**SORT11**", за которым следует копирование ключей $\leq K$ на ленту 1.
- **SORT10** [отсортировать верхний подфайл с ленты 1 и записать его на ленту 0]. Такая же, как **SORT01**, но меняются местами 0 и 1, а также операторы отношений $<$ и $>$.
- **SORT11** [отсортировать верхний подфайл с ленты 1 и вернуть его на ленту 1]. Такая же, как **SORT00**, но меняются местами 0 и 1, а также отношения $<$ и $>$. Можно без труда справиться с рекурсивной природой этих процедур, записывая подходящую управляющую информацию на ленты.

Если считать, что данные находятся в случайном порядке и вероятность равных ключей пренебрежимо мала, то можно оценить время работы этого алгоритма следующим образом. Пусть M — число записей, помещающихся в оперативной памяти. Пусть X_N — среднее число записей, читаемых

во время применения **SORT00** или **SORT11** к подфайлу из N записей, где $N > M$, и пусть Y_N — соответствующая величина для **SORT01** и **SORT10**. Тогда имеем:

$$\begin{aligned} X_N &= \begin{cases} 0, & \text{если } N \leq M, \\ 3N + 1 + \frac{1}{N} \sum_{0 \leq k < N} (Y_k + Y_{N-1-k}), & \text{если } N > M, \end{cases} \\ Y_N &= \begin{cases} 0, & \text{если } N \leq M, \\ 3N + 2 + \frac{1}{N} \sum_{0 \leq k < N} (Y_k + X_{N-1-k} + k), & \text{если } N > M. \end{cases} \end{aligned} \quad (3)$$

Решение этих рекуррентных соотношений (см. упр. 2) показывает, что общий объем информации, читаемой с ленты в течение фаз внешнего разделения, в среднем равен $6\frac{2}{3}N \ln N + O(N)$ при $N \rightarrow \infty$. Мы также знаем из формулы (5.2.2–25), что среднее число фаз внутренней сортировки будет равно $2(N+1)/(M+2) - 1$.

Если мы применим этот анализ к примеру 100000 записей, рассмотренному в п. 5.4.6, причем воспользуемся буферами по 25000 литер и будем считать, что время сортировки подфайла из $n \leq M = 1000$ записей равно $2nC\omega\tau$, то получим среднее время сортировки, приблизительно равное 103 мин (включая, как в схеме А, окончательную перемотку). Итак, метод быстрой сортировки в среднем неплох; но, конечно, в *наихудшем* случае он ужасен и уступает даже методу пузырька, обсуждавшему выше.

Поразрядная сортировка. Обменную поразрядную сортировку (алгоритм 5.2.2R) можно аналогичным образом приспособить для сортировки с двумя лентами, так как он очень похож на быструю сортировку. В качестве трюка, который позволил применить оба эти метода, использовалась идея чтения файла более чем один раз — то, чего мы никогда не делали в предыдущих алгоритмах для лент.

С помощью того же трюка можно осуществить обычную поразрядную сортировку на двух лентах "сначала-по-младшей-цифре". Имея исходные данные на $T1$, копируем на $T2$ все записи, ключ которых в двоичной системе оканчивается на 0; затем после перемотки $T1$ читаем ее вновь, копируя записи с ключами, оканчивающимися на 1. Теперь перематываются обе ленты и выполняется аналогичная пара проходов, но с заменой $T1$ на $T2$ и использованием *предпоследней* двоичной цифры. В этот момент $T1$ будет содержать все записи с ключами $(\dots 00)_2$, за которыми следуют записи с ключами $(\dots 01)_2$, затем $(\dots 10)_2$, затем $(\dots 11)_2$. Если ключи имеют размер b битов, нам потребуется, чтобы завершить сортировку, только $2b$ проходов по всему файлу.

Подобную поразрядную сортировку можно применять только к *старшим* b битам ключа для некоторого разумно выбранного числа b ; таким образом, число инверсий уменьшится примерно в 2^b раз, если ключи были равномерно распределены; и тогда несколько проходов P -путевой сортировки методом пузырька позволят завершить работу.

Новый, но несколько более сложный подход к распределяющей сортировке с двумя лентами предложили А. И. Никитин и Л. И. Шолмов [*Кибернетика*, 2, 6 (1966), 79–84]. Имеются счетчики числа ключей по одному на каждую возможную конфигурацию старших битов, и на основе этих счетчиков строятся искусственные ключи $\kappa_1, \kappa_2, \dots, \kappa_M$ так, чтобы для каждого i число действительных ключей, лежащих между κ_i и κ_{i+1} , было между заранее определенными границами P_1 и P_2 . Таким образом, M лежит между $\lceil N/P_1 \rceil$ и $\lceil N/P_2 \rceil$. Если счетчики старших битов не дают достаточной информации для определения таких $\kappa_1, \kappa_2, \dots, \kappa_M$, то делается еще один или несколько проходов для подсчета частоты конфигураций менее значащих битов при некоторых конфигурациях старших битов. После того как таблица искусственных ключей построена, $2^{\lceil \log_2 M \rceil}$ проходов будет достаточно для завершения сортировки. (Этот метод требует пространства памяти, пропорционального N , и поэтому не может использоваться для внешней сортировки при $N \rightarrow \infty$. На практике мы не станем использовать этот метод для файлов на нескольких бобинах, и, следовательно, M будет сравнительно невелико, так что таблица искусственных ключей легко поместится в памяти.)

Имитация дополнительных лент. Ф. К. Хенни и Р. Э. Стирнз изобрели общий метод имитации k лент всего на двух лентах, причем таким образом, что требуемое суммарное перемещение ленты возрастает всего лишь в $O(\log L)$ раз, где L — максимальное расстояние, которое нужно пройти на любой одной ленте [*АСМ*, 13 (1966), 533–546]. Их построение в случае сортировки можно слегка упростить, что и сделано в следующем методе, предложенном Р. М. Карпом.

Будем имитировать обычное сбалансированное слияние на четырех лентах, используя две ленты: $T1$ и $T2$. На первой из них (т. е. на $T1$) содержимое имитируемых лент хранится таким способом, как изображено на рис. 86; представим себе, что данные записаны на четырех дорожках по одной для каждой имитируемой ленты. (В действительности лента не имеет таких дорожек; мы мыслим блоки 1, 5, 9, 13, ... как дорожку 1, блоки 2, 6, 10, 14, ... как дорожку 2 и т. д.) Другая лента ($T2$) используется только для вспомогательного хранения, чтобы помочь в выполнении перестановок на $T1$.

Picture: Рис. 86. Разбивка ленты $T1$ в конструкции Хенни и Стирнза

Блоки на каждой дорожке разделяются на *зоны*, содержащие соответственно $1, 2, 4, 8, \dots, 2^k, \dots$ блоков. Зона k на каждой дорожке либо заполнена точно 2^k блоками данных, либо целиком пуста. Например, на рис. 86 на дорожке 1 данные содержатся в зонах 1 и 3; на дорожке 2—в зонах 0, 1 и 2; на дорожке 3—в зонах 0 и 2; на дорожке 4—в зоне 1, а все остальные зоны пусты.

Предположим, что мы сливаем данные с дорожек 1 и 2 на дорожку 3. В оперативной памяти ЭВМ находятся два буфера, используемые двухпутевым слиянием для ввода, а также третий буфер— для вывода. Когда буфер ввода для дорожки 1 станет пустым, можно заполнить его следующим образом: найти первую непустую зону дорожки 1, скажем зону k , и скопировать первый ее блок в буфер ввода, затем скопировать остальные $2^k - 1$ блоков данных на $T2$ и переместить их в зоны $0, 1, \dots, k - 1$ дорожки 1. (Теперь зоны $0, 1, \dots, k - 1$ заполнены, зона k пуста.) Аналогичная процедура используется для заполнения буфера ввода для дорожки 2, как только он станет пустым. Когда буфер вывода подготовлен для записи на дорожку 3, мы обращаем этот процесс, т. е. просматриваем $T1$ пока не найдется первая *пустая* зона на дорожке 3, скажем зона k , и в то же время копируем данные из зон $0, 1, \dots, k - 1$ на $T2$. Данные на $T2$, к которым присоединяется содержимое буфера вывода, используются теперь для заполнения зоны k на дорожке 3.

Для этой процедуры мы должны уметь писать в середину ленты $T1$, не разрушая последующую информацию на этой ленте. Как и в случае осциллирующей сортировки с прямым чтением (п. 5.4.5), можно без опасений выполнять это действие, если принять меры предосторожности.

Поскольку просмотр до зоны k выполняется только один раз за каждые 2^k шагов, то, чтобы переписать $2^i - 1$ блоков с дорожки 1 в память, потребуется переместить ленту на $\sum_{v \leq k < l} 2^{l-1-k} \cdot c \cdot 2^k = cl2^{l-1}$, где c —некоторая константа. Таким образом, каждый проход слияния требует $O(N \log N)$ шагов. Так как в сбалансированном слиянии имеется $O(\log N)$ проходов, то общее время работы будет $O(N(\log N)^2)$, что асимптотически значительно лучше, чем наихудший случай для быстрой сортировки.

Но на самом деле этот метод оказывается почти бесполезным, если применяется для сортировки 100000 записей из примера п. 5.4.6, поскольку информация, которая должна размещаться на ленте $T1$, не уместится на одной бобине ленты. И даже если мы пренебрежем этим фактом и будем исходить из самых оптимистических предположений относительно совмещения чтения/записи/вычислений, относительно длин межблочных промежутков и т. д., то найдем, что для выполнения сортировки потребуется около 37 ч! Итак, этот метод представляет чисто академический интерес: константа в $O(N(\log N)^2)$ слишком велика, чтобы метод был удовлетворителен для практических значений N .

Одноленточная сортировка. Можно ли обойтись всего одной лентой? Нетрудно видеть, что сортировку методом пузырька P -го порядка можно преобразовать в одноленточную сортировку, но результат будет ужасен.

Г. Б. Демут [Ph. D. thesis (Stanford University, 1956), 85] сделал наблюдение, что в вычислительной машине с ограниченной оперативной памятью нельзя уменьшить число инверсий перестановки больше, чем на ограниченную величину, после просмотра ленты на ограниченное расстояние; следовательно, любой алгоритм сортировки с одной лентой потребует в среднем по крайней мере dN^2 единиц времени (где d —некоторая положительная константа, зависящая от конфигурации ЭВМ).

Р. М. Карп нашел очень интересный подход к исследованию этой темы, обнаружив то, что, по существу, является *оптимальным* способом сортировки с одной лентой. При обсуждении алгоритма Карпа удобно следующим образом переформулировать задачу: *как быстрее всего перевезти людей между этажами, если работает только один лифт?*

Рассмотрим здание с n этажами; помещение каждого этажа рассчитано на c человек. В этом здании нет ни окон, ни дверей, ни лестниц, но все же есть лифт, который может останавливаться на любом этаже. В здании находятся cn человек, и ровно c из них хотят попасть на каждый отдельный этаж. В лифт вмещается самое большее b человек, и он затрачивает одну единицу времени для перемещения с этажа i на этаж $i + 1$. Мы хотели бы найти быстрейший способ переместить всех людей на нужные этажи, если требуется, чтобы лифт начал и закончил свое движение на первом этаже.

Нетрудно заметить связь между этой задачей и одноленточной сортировкой: люди—это записи, здание—лента, этажи—отдельные блоки на ленте, а лифт—оперативная память ЭВМ. Действиям программ для ЭВМ свойственна большая гибкость, чем действиям лифтера (она может, например, создавать двойников или, разрезав человека на две части, оставить их на время на разных этажах и т. д.); но в приводимом ниже алгоритме задача решается быстрейшим мыслимым способом без выполнения таких операций. Алгоритм Карпа использует следующие два вспомогательных массива:

$$\begin{aligned} u_k, 1 \leq k \leq n : & \quad \text{число людей на этажах } \leq k, \text{ стремящихся попасть на этажи } > k \\ d_k, 1 \leq k \leq n : & \quad \text{число людей на этажах } \geq k, \text{ стремящихся попасть на этажи } < k \end{aligned} \quad (4)$$

. Когда лифт пуст, мы всегда имеем $u_k = d_{k+1}$ при $1 \leq k < n$, так как на каждом этаже находятся c человек; количество людей, направляющихся с этажей $\{1, \dots, k\}$ на этажи $\{k + 1, \dots, n\}$, должно равняться числу людей, стремящихся переправиться в обратном направлении. По определению $u_n = d_1 = 0$.

Ясно, что лифт должен сделать по крайней мере $\lceil u_k/b \rceil$ рейсов с этажа k на этаж $k + 1$ при $1 \leq k < n$, так как только b пассажиров могут подняться за один рейс. Аналогично, он должен сделать не менее $\lceil d_k/b \rceil$ рейсов с этажа k на этаж $k - 1$. Следовательно, лифту потребуется по крайней мере

$$\sum_{1 \leq k \leq n} (\lceil u_k/n \rceil + \lceil d_k/b \rceil) \tag{5}$$

единиц времени при любом правильном графике работы. Карп обнаружил, что эта нижняя граница действительно достижима, если u_1, \dots, u_{n-1} ненулевые.

Теорема К. Если $u_k > 0$ при $1 \leq k < n$, то существует график работы лифта, при котором все люди доставляются на свои этажи за минимальное время (5).

Доказательство. Предположим, что в здании имеется дополнительно b человек; первоначально они находятся в лифте, и этаж их назначения искусственно полагается нулевым. Лифт может функционировать в соответствии со следующим алгоритмом, начиная с k (текущий этаж), равного 1.

Picture: Ряс. 87. Алгоритм Карпа для лифта.

Алгоритм К. (Алгоритм Карпа для лифта)

К1 [Движение вверх.] Из $b + c$ людей, находящихся в данный момент в лифте и на этаже k , только b , имеющие самое высокое место назначения, попадают в лифт, остальные остаются на этаже k .

Пусть теперь в лифте находятся u человек с назначением $> k$ и d с назначением $\leq k$. (Окажется, что $u = \min(b, u_k)$; если $u_k < b$, мы можем, следовательно, увозить некоторых людей от их места назначения. Это—их жертва для общей пользы). Уменьшить u_k на u , увеличить d_{k+1} на d и затем увеличить k на 1.

К2 [Продолжать движение вверх?] Если $u_k > 0$, вернуться к шагу **К1**.

К3 [Движение вниз.] Из $b + c$ людей, находящихся в данный момент в лифте или на этаже k , только b , имеющие самое низкое место назначения, попадают в лифт, остальные остаются на этаже k . Пусть теперь в лифте находятся u человек с назначением $\geq k$ и d с назначением $< k$. (Всегда оказывается, что $u = 0$, а $d = b$, но алгоритм описывается здесь применительно к общим u и d , чтобы сделать доказательство несколько прозрачнее.) Уменьшить d_k на d , увеличить u_{k-1} на u и затем уменьшить k на 1.

К4 [Продолжать движение вниз?] Если $k > 1$ и $u_{k-1} > 0$, вернуться к шагу **К3**. Если $k = 1$ и $u_1 = 0$, закончить алгоритм (все люди доставлены на свое место назначения, а b "дополнительных" людей снова находятся в лифте). В противном случае вернуться к шагу **К2** ■

На рис. 88 показан пример работы этого алгоритма для здания с девятью этажами и $b = 3, c = 2$. Заметим, что одна из шестерок временно перемещается от своего места назначения, несмотря на то что лифт проходит минимально возможное расстояние. Проверка u_{k-1} на шаге **К4** является, как мы увидим, решающим моментом для правильной работы алгоритма.

Чтобы проверить правильность этого алгоритма, заметим, что шаги **К1** и **К3** всегда поддерживают массивы u и d в соответствии с текущим положением, если считать людей в лифте находящимися на текущем этаже k . Теперь можно доказать по

Picture: Рис. 88. Оптимальный способ перераспределения людей при помощи небольшого медленного лифта. (Каждый человек представлен номером этажа, на который он направляется.)

индукции, что в начале каждого шага имеют место следующие свойства:

$$u_l = d_{l+1} \quad \text{при } k \leq l < n; \tag{6}$$

$$u_l = d_{l+1} - b \quad \text{при } 1 \leq l < k; \tag{7}$$

$$\text{если } u_l = 0 \text{ и } k \leq l < n, \text{ то } u_{l+1} = 0. \tag{8}$$

Кроме того, в начале шага **К1** в лифте или на этаже k находятся $\min(u_k, b)$ человек с наивысшими назначениями среди всех людей на этажах $\leq k$ с назначениями $> k$. В начале шага **К3** в лифте или на

этаже k находятся $\min(d_k, b)$ человек с наименьшими назначениями среди всех людей на этажах $\geq k$ с назначениями $< k$. Эти условия также можно проверить индуктивно, если проследить, как мы попадаем в шаги К1 или К3 (см. упр. 5).

Из этих свойств следует, что замечания в скобках на шагах К1 и К3 справедливы. Каждое выполнение шага К1, следовательно, уменьшает $\lceil u_k/b \rceil$ на 1 и оставляет $\lceil d_{k+1}/b \rceil$ без изменений. Каждое выполнение шага К3 уменьшает $\lceil d_k/b \rceil$ на 1 и оставляет неизменным $\lceil u_{k-1}/b \rceil$. Алгоритм, следовательно, должен завершиться за конечное число шагов, и после этого в силу (6) и (8) каждый человек должен оказаться на своем этаже. ■

Если $u_k = 0$, а $u_{k+1} > 0$, мы имеем "несвязную" ситуацию; лифт должен подняться до этажа $k+1$, чтобы переместить людей вверх, хотя никому не нужно переезжать с этажей $\leq k$ на этажи $\geq k+1$. Не поступаясь общностью, можно считать $u_{n-1} > 0$; тогда любой правильный график должен включать по крайней мере

$$2 \sum_{1 \leq k < n} \max(1, \lceil u_k/b \rceil) \quad (9)$$

движений, так как мы требуем, чтобы лифт вернулся на первый этаж. График, для которого достигается эта нижняя граница, легко составить (упр. 4).

Упражнения

1. [17] В методе пузырька P -го порядка, обсуждавшемся в тексте, используется только прямое чтение и перематка. Можно ли модифицировать алгоритм так, чтобы извлечь преимущества из *обратного* чтения?
2. [M26] Найдите явные выражения в замкнутом виде для чисел X_n, Y_n , определенных в (3). [*Указание*: изучите решение уравнения (5.2.2-19).]
3. [38] Существует ли метод сортировки с двумя лентами, основанный только на сравнении ключей (а не на свойствах цифр), для которого в наихудшем случае при сортировке N записей перемещение лент составляет $O(N \log N)$? [При быстрой сортировке это значение достигается в среднем, но не в наихудшем случае, а в методе Хенни и Стирнза (рис. 86) оно равняется $O(N(\log N)^2)$.]
4. [M23] В задаче о лифте предположим, что имеются индексы p, q , причем $q \geq p+2, u_p > 0, u_q > 0$ и $u_{p+1} = \dots = u_{q-1} = 0$. Объясните, как составить график, требующий не более (9) единиц времени.
- >5. [M23] Верно ли следующее утверждение? После шага К1 алгоритма теоремы К никто в лифте не стремится попасть на более низкий этаж, чем некто из оставшихся на этажах $< k$.
6. [M30] (Р. М. Карп.) Обобщите задачу о лифте (рис. 87) на случай, когда на этаже j первоначально находится c_j пассажиров и этаж j служит назначением для c'_j пассажиров при $1 \leq j \leq n$. Покажите, что существует график работы, рассчитанный на $2 \sum_{1 \leq k < n} \max(1, \lceil u_k/b \rceil, \lceil d_{k+1}/b \rceil)$ единиц времени, причем на этаже j никогда не оказывается одновременно более $\max(c_j, c'_j)$ пассажиров. [*Указание*: введите, если необходимо, фиктивных людей, чтобы сделать $c_j = c'_j$ при всех j .]
7. [M40] (Р. М. Карп.) Обобщите задачу из упр. 6, заменив линейный путь, проходимый лифтом, на сеть дорог, по которым можно ездить на автобусе, при условии, что сеть образует любое *свободное дерево*. Автобус имеет конечную емкость, и желательно перевезти пассажиров к их местам назначения так, чтобы автобус прошел минимальное расстояние.
8. [M32] Пусть в задаче о лифте, разобранный в тексте, $c = 1$. Сколько перестановок из n человек по n этажам дадут в (4) $u_k \leq 1$ при $1 \leq k \leq n$? [Например, 3 1 4 5 9 2 6 8 7—такая перестановка.]
- >9. [M25] Найдите важную связь между шейкер-сортировкой, описанной в п. 5.2.2 (рис. 16), и числами u_1, u_2, \dots, u_n в (4) в случае $c = 1$.
10. [20] Как бы вы сортировали файлы на нескольких бобинах, имея только два лентопротяжных устройства?

5.4.9. Диски и барабаны

До сих пор мы рассматривали ленты как единственное средство для внешней сортировки, однако нередко в нашем распоряжении оказываются и другие типы устройств массовой памяти с более гибкими возможностями. Хотя такие запоминающие устройства "большого объема" или "запоминающие устройства с прямым доступом" весьма многообразны, можно выделить следующие общие свойства:

- i) Для доступа к любой определенной части хранимой информации не требуется очень много времени.
- ii) Блоки, содержащие последовательные слова, могут быстро передаваться между внутренней (оперативной) и внешней памятью.

Магнитная лента удовлетворяет (ii), но не (i), поскольку переход ленты от одного конца к другому занимает много времени. Некоторые устройства удовлетворяют (i), но не (ii); примером может служить память большого объема на ферритовых сердечниках, в которой время доступа к каждому слову примерно в десять раз превышает время доступа к внутренней памяти.

Picture: Рис. 89. Пакет дисков

Каждое внешнее запоминающее устройство имеет свои характерные особенности, которые следует тщательно изучить, прежде чем писать для него большие программы; однако технология меняется так быстро, что здесь не удастся сколько-нибудь подробно обсудить все существующие разновидности оборудования. Поэтому мы рассмотрим лишь некоторые типичные запоминающие устройства и на них проиллюстрируем продуктивные подходы к задаче сортировки.

Одним из наиболее распространенных типов внешних запоминающих устройств, удовлетворяющих (i) и (ii), является дисковый файл или модуль с пакетом дисков (рис. 89). Данные хранятся на нескольких быстро вращающихся круглых дисках, покрытых магнитным материалом; для записи или выборки информации используется держатель головок в виде гребешка, содержащий одну или несколько "головок чтения/записи" для каждой поверхности диска. Каждая поверхность делится на концентрические кольца, называемые *дорожками* или *треками*, так что за время одного оборота диска под головками чтения/записи проходит целая дорожка. Держатель головок может перемещаться в двух направлениях—внутрь или наружу, передвигая головки чтения/записи от дорожки к дорожке, но это движение требует времени. Множество дорожек, которые могут быть прочитаны или записаны без перемещения держателя головок, называется *цилиндром*. Например, на рис. 89 показан дисковый файл, который имеет по одной головке чтения/записи на каждую поверхность; пунктирными линиями обозначен один из цилиндров, состоящий из всех дорожек, просматриваемых в настоящий момент головками.

Чтобы сделать наши рассуждения более конкретными, рассмотрим гипотетическое дисковое устройство MIXTES, для которого

$$1 \text{ дорожка} = 5000 \text{ литер},$$

$$1 \text{ цилиндр} = 20 \text{ дорожек},$$

$$1 \text{ дисковое устройство} = 200 \text{ цилиндров}.$$

Такое дисковое устройство содержит 20 миллионов литер, т. е. чуть меньше того объема данных, который можно записать на одну магнитную ленту. На некоторых машинах дорожки вблизи центра содержат меньше литер, чем дорожки ближе к краю. От этого программирование значительно усложняется, но MIXTES, к счастью, не создает таких проблем.

Время, необходимое для чтения или записи на дисковый файл, представляет, по существу, сумму трех величин:

- Время поиска (время, затрачиваемое на перемещение держателя головок к нужному цилиндру).
- Время ожидания (задержка, связанная с вращением диска, пока головка чтения/записи не достигнет нужного места).
- Время передачи (задержка, связанная с вращением диска, пока данные проходят под головками).

На устройствах MIXTES время поиска для перехода от цилиндра i к цилиндру j равно $25 + \frac{1}{2}|i - j|$ мс. Если i и j —случайно выбранные целые числа между 1 и 200, то среднее значение $|i - j|$ равно $2 \binom{201}{3} / 200^2 \approx 66.7$, т. е. среднее время поиска составляет приблизительно 60 мс. Диски MIXTES совершают один оборот за 25 мс, так что время ожидания равно в среднем 12.5 мс. Время передачи n литер есть $(n/5000) \times 25 \text{ мс} = 5n \text{ мкс}$. (Это примерно в $3\frac{1}{3}$ раза быстрее, чем скорость передачи для лент MIXT, использованных в примерах п. 5.4.6.)

Таким образом, основные различия между дисками MIXTES и лентами MIXT, касающиеся сортировки, следующие:

- a) На лентах возможен только последовательный доступ к данным.
- b) Отдельная операция с диском, как правило, сопряжена со значительно большими накладными расходами (время поиска + время ожидания в сравнении со стартстопным временем).
- c) Скорость передачи у диска больше.

Используя для лент разумные схемы слияния, мы могли до некоторой степени скомпенсировать недостаток (a). Теперь у нас иная цель—нам нужно найти такие рациональные алгоритмы сортировки на дисках, в которых компенсируется недостаток (b).

Как сократить время ожидания? Рассмотрим сначала задачу минимизации задержек, вызываемых тем, что в тот момент, когда мы хотим начать команду ввода/вывода, диск не всегда находится в подходящей позиции. Нельзя заставить диск вращаться быстрее, но все-таки можно прибегнуть к разным уловкам, которые уменьшат или даже полностью устранят время ожидания. Несомненно, поможет добавление еще нескольких держателей головок, но это весьма дорогостоящая модификация оборудования. Вот несколько "программистских" идей.

- 1) Если мы читаем или записываем за один раз несколько дорожек одного цилиндра, то тем самым устраняем время ожидания (и время поиска) для всех дорожек, кроме первой. Вообще зачастую можно таким образом синхронизировать вычисления с вращением диска, что при выполнении последовательности команд ввода/вывода не будет задержек из-за ожидания.
- 2) Рассмотрим задачу чтения половины дорожки данных (рис. 90): если команда чтения выдается, когда головка находится в точке A , то задержка на ожидание отсутствует, и общее время чтения равно времени передачи, т.е. $\frac{1}{2} \times 25$ мс. Если команда начинается, когда головка находится в точке B , то требуется $\frac{1}{4}$ оборота для ожидания и $\frac{1}{2}$ для передачи; в итоге имеем $\frac{3}{4} \times 25$ мс. Наиболее интересен случай, когда головка первоначально находится в точке C : имея соответствующее оборудование и программное обеспечение, нам *не* придется терять $\frac{3}{4}$ оборота на ожидание. Можно немедленно начать чтение во вторую половину буфера ввода, затем после паузы в $\frac{1}{2} \times 25$ мс можно возобновить чтение в первую половину буфера, так что команда будет завершена, когда мы снова попадем в точку C . Поступая

Picture: Рис. 90. Анализ времени ожидания при чтении половины дорожки

таким образом, можно гарантировать, что общее время на ожидание+передачу никогда не превзойдет времени одного оборота независимо от начального положения диска. Среднее время ожидания уменьшается этой схемой с половины оборота до $\frac{1}{2}(1 - x^2)$ оборота, если читается или записывается доля x дорожки ($0 < x \leq 1$). Если читается или записывается целая дорожка ($x = 1$), то этот метод позволяет полностью устранить ожидание.

Барабаны: случай, когда поиск не нужен. На некоторых устройствах внешней памяти установлено по одной головке чтения/записи для каждой дорожки, и поэтому время поиска полностью устранено. Если на таком устройстве используется метод, продемонстрированный на рис. 90, то как время поиска, так и время ожидания сведены к нулю, при условии что мы всегда читаем или записываем всю дорожку целиком. В этой идеальной ситуации время передачи является единственным ограничительным фактором.

Рассмотрим вновь пример из п. 5.4.6, в котором сортируются 100000 записей по 100 литер каждая с внутренней памятью в 100000 литер. Весь объем сортируемых данных занимает половину диска MIXTES. Обычно невозможно одновременно читать и писать на одном дисковом устройстве; поэтому предположим, что имеются два диска, так что чтение и запись можно совместить. Пока будем считать, что "диски"—это фактически *барабаны*, содержащие 4000 дорожек по 5000 литер каждая, и никакого времени поиска не требуется.

Какой алгоритм сортировки следует использовать? Естественно выбрать метод слияния; другие методы внутренней сортировки не столь хороши в применении к дискам, за исключением, возможно, поразрядных методов, но рассуждения в п. 5.4.7 показывают, что поразрядная сортировка обычно хуже слияния, если речь идет не о каких-нибудь частных приложениях. (Нетрудно видеть, что теорема двойственности, сформулированная в этом пункте, применима к дискам точно так же, как и к лентам.)

Чтобы начать сортировку слиянием, можно использовать выбор с замещением с двумя буферами ввода по 5000 литер и двумя буферами вывода по 5000 литер, фактически можно свести их к *трем* буферам по 5000 литер, если замещать записи в текущем буфере ввода на записи, выводимые из дерева выбора. Остается еще 85000 литер (850 записей) для дерева выбора, так что один проход по данным нашего примера позволит сформировать около 60 начальных отрезков (см. формулу (5.4.6-3)). Этот проход занимает лишь около 50 с, если предположить, что скорость внутренней обработки достаточно высока, чтобы успеть за вводом/выводом (каждые 500 мкс в буфер вывода должна передаваться запись). Если же сортируемые исходные данные находятся на ленте MIXT, а не на барабане, то этот проход будет выполняться медленнее, и время будет зависеть от скорости ленты.

Нетрудно видеть, что в случае двух барабанов и при чтении/записи полными дорожками общее время передачи для P -путевого слияния уменьшается с увеличением P . К сожалению, мы не можем выполнить одно только 60-путевое слияние всех начальных отрезков, так как в памяти нет места

для 60 буферов. (Если размер буферов будет меньше 5000 литров, то появится нежелательное время ожидания.) Если мы выполняем P -путевое слияние, переписывая все данные с одного барабана на другой таким образом, что чтение и запись совмещены, то число проходов слияния равно $\lceil \log_P 60 \rceil$; следовательно, мы можем закончить работу за два прохода, если $8 \leq P \leq 59$. С уменьшением P уменьшается объем внутренних вычислений, поэтому выберем $P = 8$; если будет образовано 65 начальных отрезков, мы выберем $P = 9$. Если будет образовано 82 или больше начальных отрезков, мы можем взять $P = 10$, но так как имеется место только для 18 буферов ввода и двух буферов вывода, то появится возможность задержек при слиянии (см. алгоритм 5.4.6F); в таком случае, вероятно, лучше выполнить два частичных прохода над небольшой долей данных, сократив число начальных отрезков до 81 или меньше. При наших предположениях каждый проход слияния займет около 50 с, так что вся сортировка в этой идеальной ситуации будет завершена за $2\frac{1}{2}$ мин (плюс несколько секунд на инициализацию и другие вспомогательные действия). Это в шесть раз быстрее, чем наилучшая из сортировок с шестью лентами, рассмотренных в п. 5.4.6. Причинами этого ускорения являются увеличенная скорость передачи данных между внешней и внутренней памятью (она в $3\frac{1}{2}$ раза выше), более высокий порядок слияния (мы не можем осуществить восьмипутевое слияние на лентах не имея девяти или большего числа лент) и то, что выводные данные остаются на диске (нет необходимости в заключительной перемотке и т. д.). Если требуется, чтобы исходные данные и отсортированный результат были на лентах MIXT, а барабаны используются только для слияния, то такая сортировка займет около 8.2 мин.

Если вместо двух барабанов имеется только один, то время ввода/вывода увеличится вдвое, поскольку чтение/запись придется выполнять по отдельности. (На самом деле операции ввода/вывода займут в *три раза* больше времени, поскольку запись будет идти поверх исходных данных; в таких случаях целесообразно за каждой операцией записи выполнять операцию контрольного чтения, чтобы не потерять необратимо каких-нибудь исходных данных, если только оборудование не обеспечивает автоматической проверки записанной информации.) Однако часть этого дополнительного времени можно возратить, поскольку мы можем использовать частичные проходы, при которых одни записи обрабатываются чаще других. В случае двух барабанов требуется, чтобы все данные обрабатывались четное или нечетное число раз, но в случае одного барабана можно использовать более общие схемы слияния.

Мы видели в п. 5.4.4, что схемы слияния можно изображать с помощью деревьев и что время передачи, соответствующее схеме слияния, пропорционально длине внешнего пути дерева. В качестве схем эффективного слияния на лентах можно использовать лишь вполне определенные деревья ("T-lifo" или "сильные T-fifo"), потому что в процессе слияния некоторые отрезки оказываются "спрятанными" в середине ленты. Но *при использовании дисков или барабанов пригодны любые деревья*, если только степени их внутренних узлов не слишком велики (т. е. согласуются с наличным объемом внутренней памяти).

Следовательно, время передачи можно минимизировать, если выбрать дерево с минимальной длиной внешнего пути, такое, как полное P -арное дерево, где P —самое большое, какое возможно. По формуле (5.4.4–9) длина внешнего пути такого дерева с S внешними узлами (листьями) равна

$$qS - \lfloor (P^q - S)/(P - 1) \rfloor, \quad q = \lceil \log_P S \rceil. \quad (1)$$

Особенно просто строится алгоритм, который осуществляет слияние в соответствии со схемой полного P -арного дерева. (См., например, рис. 91, на котором показан случай $P = 3$, $S = 6$.) Сначала мы добавляем, если необходимо, "фиктивные отрезки", чтобы сделать $S \equiv 1 \pmod{P - 1}$; затем объединяем отрезки в соответствии с дисциплиной "первым включается —

Picture: Рис. 91. Полное тернарное дерево с шестью листьями...

первым исключается", сливая на каждом этапе P самых "старых" отрезков в начале очереди в один отрезок, помещаемый в конец.

Полные P -арные деревья дают оптимальную схему, если все отрезки имеют равную длину, но часто результат может быть еще лучше, если некоторые отрезки длиннее других. Оптимальную схему для этой общей ситуации можно без труда построить с помощью метода Хаффмана (упр. 2.3.4.5–10), который на языке слияния формулируется так: "сначала добавьте $(1 - S) \bmod (P - 1)$ фиктивных отрезков длины 0, затем многократно сливайте P кратчайших из имеющихся отрезков, пока не останется один отрезок". Если все начальные отрезки имеют одинаковую длину, то этот метод сводится к описанной выше дисциплине.

В нашем примере со 100000, записей мы можем выполнять 9-путевое слияние, так как в памяти поместятся 18 буферов ввода и два буфера вывода, и в алгоритме 5.4.6F будет достигнуто полное

совмещение вычислений. Полное 9-арное дерево с 60 листьями соответствует схеме слияния с $1\frac{29}{30}$ прохода, если все начальные отрезки имеют одинаковую длину. Общее время сортировки с одним барабаном и с использованием "контрольного чтения" после каждой записи становится, таким образом, равным 7.4 мин. Увеличивая P , можно немного уменьшить это время, но ситуация весьма запутанная, поскольку не исключается задержка чтения, так как буферы могут оказаться слишком полными или слишком пустыми.

Влияние времени поиска. Предыдущее обсуждение показывает, что для барабанов относительно легко сконструировать "оптимальную" схему слияния, поскольку время поиска и время ожидания можно свести на нет. Но если используются диски, то поиск информации занимает больше времени, чем ее чтение. Поэтому время поиска оказывает значительное влияние на стратегию сортировки. Уменьшение порядка слияния P дает возможность использовать большие по размеру буферы, так что реже требуется поиск; за счет этого часто компенсируется дополнительное время передачи, которое растет с уменьшением P .

Время поиска зависит от расстояния, проходимого держателем головок, и можно попытаться организовать работу таким образом, чтобы это расстояние было минимальным. Быть может, разумно сначала сортировать записи внутри цилиндров. Однако довольно большое слияние требует большого количества переходов между цилиндрами (см., например, упр. 2). Кроме того, режим мультипрограммирования в современных операционных системах означает, что пользователь лишь в редких случаях может по-настоящему контролировать положение держателя головок; блестящие схемы, минимизирующие поиск, обычно работают только по выходным дням! Таким образом, предположение о том, что каждая команда для диска требует "случайного" поиска, часто вполне оправдано.

Наша цель в том и состоит, чтобы найти такое дерево (т. е. схему слияния), которое обеспечивает наилучший баланс между временем поиска и временем передачи; для этой цели нам нужен некоторый способ, позволяющий оценить достоинства любого конкретного дерева по отношению к конкретной конфигурации оборудования. Рассмотрим, например, дерево на рис. 92; мы хотим оценить, сколько времени займет выполнение соответствующего слияния, чтобы можно было сравнить это дерево с другими.

В последующих рассуждениях мы сделаем некоторые простые предположения относительно слияния на дисках, чтобы проиллюстрировать некоторые общие идеи. Предположим, что (1) на чтение или запись n литер требуется $72.5 + 0.005n$ мс; (2) под рабочее пространство отводится 100000 литер внутренней памяти; (3) для пересылки одной литеры из буфера ввода в буфер вывода затрачивается в среднем 0.004 мс на вычисление; (4) нет совмещения чтения, записи и вычислений; (5) размер буфера, используемого для вывода, не обязательно должен быть равен размеру буфера, используемого для чтения данных на следующем проходе. Анализ задачи сортировки при этих простых предположениях будет полезен для понимания более сложных ситуаций.

Если выполняется P -путевое слияние, то мы можем разделить внутреннюю рабочую память на $P+1$ буферных областей: P —для ввода и 1—для вывода; в каждом буфере по $B = 100000/(P+1)$ литер. Предположим, что предназначенные для слияния файлы содержали в сумме L литер; тогда мы выполним приблизительно L/B операций вывода и примерно столько же операций ввода; следовательно, общее время слияния при таких предположениях будет равно (в миллисекундах) приблизительно

$$2 \left(72.5 \frac{L}{B} + 0.005L \right) + 0.004L = (0.00145P + 0.011545)L. \quad (2)$$

Иными словами, P -путевое слияние L литер занимает примерно $(\alpha P + \beta)L$ единиц времени, где α и β —некоторые константы, зависящие от времени поиска, времени ожидания, времени вычислений и размера памяти. Эта формула приводит к интересному способу построения хороших схем слияния для дисков. Рассмотрим, например, рис. 92 и будем считать, что все начальные отрезки (изображенные квадратными "листьями") имеют длину L_0 . Тогда каждое слияние в узлах 9 и 10 занимает $(2\alpha + \beta)(2L_0)$ единиц времени, слияние в узле 11 занимает $(3\alpha + \beta)(4L_0)$ единиц и окончательное слияние в узле 12 занимает $(4\alpha + \beta)(8L_0)$ единиц. Общее время слияния, следовательно, составляет $(52\alpha + 16\beta)L_0$ единиц. Коэффициент "16" нам хорошо известен: это просто длина внешнего пути дерева. Коэффициент "52" при α соответствует новому понятию, которое мы можем назвать *длиной степенного пути дерева*; она равна сумме, взятой по всем листьям, степеней внутренних узлов, лежащих на пути от листа к корню. Например, на рис. 92 длина степенного пути равна $(2+4) + (2+4) + (3+4) + (2+3+4) + (2+3+4) + (3+4) + (4) + (4) = 52$.

Picture: Рис. 92. Дерево с длиной внешнего пути 16 ...

Если \mathcal{J} —любое дерево, то пусть $D(\mathcal{J})$, $E(\mathcal{J})$ обозначают соответственно длину степенного пути и длину внешнего пути этого дерева. Анализ сводится к следующей теореме:

Теорема Н. Если время, требуемое для выполнения P -путевого слияния L литер, имеет вид $(\alpha P + \beta)L$ и если требуется слить S отрезков равной длины, то наилучшая схема слияния соответствует дереву \mathcal{J} , для которого $\alpha D(\mathcal{J}) + \beta E(\mathcal{J})$ минимально среди всех деревьев с S листьями.

(Эта теорема неявно содержалась в неопубликованной статье, которую Джордж А. Хаббэрд представил на национальную конференцию АСМ в 1963 г.)

Пусть α и β —фиксированные константы; будем говорить, что дерево *оптимально*, если оно имеет минимальное значение $\alpha D(\mathcal{J}) + \beta E(\mathcal{J})$ среди всех деревьев \mathcal{J} с тем же числом листьев. Нетрудно видеть, что *все поддеревья оптимального дерева также оптимальны*. Поэтому мы можем строить оптимальные деревья с n листьями, объединяя оптимальные деревья, у которых меньше чем n листьев.

Теорема К. Пусть последовательность чисел $A_m(n)$ определена при $1 \leq m \leq n$ правилами

$$A_1(1) = 0; \tag{3}$$

$$A_m(n) = \min_{1 \leq k \leq n/m} (A_1(k) + A_{m-1}(n - k)) \quad \text{при } 2 \leq m \leq n; \tag{4}$$

$$A_1(n) = \min_{2 \leq m \leq n} ((\alpha m n + \beta n + A_m(n))) \quad \text{при } n \geq 2. \tag{5}$$

Тогда $A_1(n)$ есть минимальное значение $\alpha D(\mathcal{J}) + \beta E(\mathcal{J})$ среди всех деревьев \mathcal{J} с n листьями.

Доказательство. Из соотношения (4) следует, что $A_m(n)$ есть минимальное значение $A_1(n_1) + \dots + A_1(n_m)$ по всем положительным числам n_1, \dots, n_m , таким, что $n_1 + \dots + n_m = n$. Требуемый результат получается теперь индукцией по n . ■

Рекуррентные соотношения (3), (4), (5) можно использовать также для построения самих оптимальных деревьев. Пусть $k_m(n)$ —значение, для которого достигается минимум в определении $A_m(n)$. Тогда можно построить оптимальное дерево с n листьями, объединяя $m = k_1(n)$ поддеревьев в корне; поддеревья являются оптимальными деревьями с $k_m(n), k_{m-1}(n - k_m(n)), k_{m-2}(n - k_m(n) - k_{m-1}(n - k_m(n))), \dots$ листьями соответственно.

Таблица 1

Характеристики оптимального дерева $A_m(n), k_m(n)$ при $\alpha = \beta = 1$												Описание дерева		
$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$	$m = 7$	$m = 8$	$m = 9$	$m = 10$	$m = 11$	$m = 12$			
$n = 1$	0, 0											—	$n = 1$	
$n = 2$	6, 2	0, 1										1 : 1	$n = 2$	
$n = 3$	12, 3	6, 1	0, 1									1 : 1 : 1	$n = 3$	
$n = 4$	20, 4	12, 1	6, 1	0, 1								1 : 1 : 1 : 1	$n = 4$	
$n = 5$	30, 5	18, 2	12, 1	6, 1	0, 1							1 : 1 : 1 : 1 : 1	$n = 5$	
$n = 6$	42, 2	24, 3	18, 1	12, 1	6, 1	0, 1						3 : 3	$n = 6$	
$n = 7$	52, 3	32, 3	24, 1	18, 1	12, 1	6, 1	0, 1					1 : 3 : 3	$n = 7$	
$n = 8$	62, 3	40, 4	30, 2	24, 1	18, 1	12, 1	6, 1	0, 1				2 : 3 : 3	$n = 8$	
$n = 9$	72, 3	50, 4	36, 3	30, 1	24, 1	18, 1	12, 1	6, 1	0, 1			3 : 3 : 3	$n = 9$	
$n = 10$	84, 3	60, 5	44, 3	36, 1	30, 1	24, 1	18, 1	12, 1	6, 1	0, 1		3 : 3 : 4	$n = 10$	
$n = 11$	96, 3	72, 4	52, 3	42, 2	36, 1	30, 1	24, 1	18, 1	12, 1	6, 1	0, 1	3 : 4 : 4	$n = 11$	
$n = 12$	108, 3	82, 4	60, 4	48, 3	42, 1	36, 1	30, 1	24, 1	18, 1	12, 1	6, 1	0, 1	4 : 4 : 4	$n = 12$
$n = 13$	121, 4	92, 4	70, 4	56, 3	48, 1	42, 1	36, 1	30, 1	24, 1	18, 1	12, 1	6, 1	3 : 3 : 3 : 4	$n = 13$
$n = 14$	134, 4	102, 5	80, 4	64, 3	54, 2	48, 1	42, 1	36, 1	30, 1	24, 1	18, 1	12, 1	3 : 3 : 4 : 4	$n = 14$
$n = 15$	147, 4	114, 5	90, 5	72, 3	60, 3	54, 1	48, 1	42, 1	36, 1	30, 1	24, 1	18, 1	3 : 4 : 4 : 4	$n = 15$
$n = 16$	160, 4	124, 7	102, 4	80, 4	68, 3	60, 1	54, 1	48, 1	42, 1	36, 1	30, 1	24, 1	4 : 4 : 4 : 4	$n = 16$
$n = 17$	175, 4	134, 8	112, 4	90, 4	76, 3	66, 2	60, 1	54, 1	48, 1	42, 1	36, 1	30, 1	4 : 4 : 4 : 5	$n = 17$
$n = 18$	190, 4	144, 9	122, 4	100, 4	84, 3	72, 3	66, 1	60, 1	54, 1	48, 1	42, 1	36, 1	4 : 4 : 5 : 5	$n = 18$
$n = 19$	205, 4	156, 9	132, 5	110, 4	92, 3	80, 3	72, 1	66, 1	60, 1	54, 1	48, 1	42, 1	4 : 5 : 5 : 5	$n = 19$
$n = 20$	220, 4	168, 9	144, 4	120, 5	100, 4	88, 3	78, 2	72, 1	66, 1	60, 1	54, 1	48, 1	5 : 5 : 5 : 5	$n = 20$
$n = 21$	236, 5	180, 9	154, 4	132, 4	110, 4	96, 3	84, 3	78, 1	72, 1	66, 1	60, 1	54, 1	4 : 4 : 4 : 4 : 5	$n = 21$
$n = 22$	252, 3	192, 10	164, 4	142, 4	120, 4	104, 3	92, 3	84, 1	78, 1	72, 1	66, 1	60, 1	4 : 9 : 9	$n = 22$
$n = 23$	266, 3	204, 11	174, 5	152, 4	130, 4	112, 3	100, 3	90, 2	84, 1	78, 1	72, 1	66, 1	5 : 9 : 9	$n = 23$
$n = 24$	282, 3	216, 12	186, 5	162, 5	140, 4	120, 4	108, 3	96, 3	90, 1	84, 1	78, 1	72, 1	5 : 9 : 10	$n = 24$
$n = 25$	296, 3	229, 12	196, 7	174, 4	150, 5	130, 4	116, 3	104, 3	96, 1	90, 1	84, 1	78, 1	7 : 9 : 9	$n = 25$

Эта конструкция при $\alpha = \beta = 1$ проиллюстрирована в качестве примера в табл. 1. Компактные описания соответствующих оптимальных деревьев имеются в правой части таблицы; элемент "4:9:9" для $n = 22$, например, означает, что оптимальное дерево \mathcal{J}_2 с 22 листьями можно получить в результате объединения \mathcal{J}_4 , \mathcal{J}_9 и \mathcal{J}_9 (рис. 93). Оптимальные деревья не единственны; например, элемент 5:8:9 был бы столь же хорошим, как и 4:9:9.

Наш вывод выражения (2) показывает, что всегда, когда используются $P + 1$ равных буферов, будет иметь место соотношение $\alpha \leq \beta$. Предельный случай $\alpha = \beta$, показанный в табл. 1 и на рис. 93, возникает тогда, когда требуется минимизировать само время поиска безотносительно ко времени передачи.

Вернемся к нашей первоначальной задаче. Мы еще не рассмотрели, как получить начальные отрезки; без совмещения чтения/записи/вычислений выбор с замещением теряет некоторые из своих преимуществ. Возможно, нам следует заполнить всю внутреннюю

Picture: Рис. 93. Оптимальный способ слияния 22 начальных отрезков равной длины, если в теореме $\alpha = \beta$. Эта схема минимизирует время поиска при предположениях, приведших к формуле (2).

память, отсортировать ее и вывести результат; каждую из таких операций ввода и вывода можно выполнить с одним поиском. Или, возможно, лучше использовать, скажем, 20% памяти как комбинированный буфер ввода/вывода и выполнять выбор с замещением. Это требует в пять раз больше поисков (дополнительно примерно 60 с!), но уменьшает число начальных отрезков со 100 до 64; уменьшение будет еще более резким, если вводный файл уже почти упорядочен.

Если мы решили не использовать выбор с замещением, то оптимальное дерево для $S = 100$, $\alpha = 0.00145$, $\beta = 0.01545$ [см. (2)] оказывается весьма прозаическим. Это просто 10-путевое слияние, выполняемое за два прохода по данным. Выделяя 30 с на внутреннюю сортировку (скажем, 100 быстрых сортировок), получаем, что начальный распределительный проход занимает примерно $2\frac{1}{2}$ мин, а каждый проход слияния занимает почти 5 мин; в итоге имеем 12.4 мин. Если мы решили использовать выбор с замещением, то оптимальное дерево для $S = 64$ оказывается одинаково неинтересным (два прохода 8-путевого слияния); начальный распределительный проход занимает примерно $3\frac{1}{2}$ мин, каждый проход слияния — около $4\frac{1}{2}$ мин, оценка общего времени составляет 12.6 мин. Вспомним, что в обоих этих методах фактически полностью исключается совмещение чтения/записи/вычислений, чтобы иметь большие буферы (с целью уменьшения времени поиска). Ни одна из этих оценок не включает время, которое может потребоваться для операций контрольного чтения.

На практике последний проход слияния оказывается отличным от остальных; например, выводные данные часто редактируются и/или записываются на ленту. В таких случаях дерево, изображающее схему слияния, следует выбирать с использованием в корневом узле иного критерия оптимальности.

***Другой способ распределения буферов.** Давид Э. Фергюсон указал [CACM, 14 (1971), 476–478], что можно уменьшить время поиска, если не делать все буферы одного размера. Та же мысль и примерно в то же время пришла в голову еще нескольким лицам [S. J. Waters, *Comp. J.*, 14 (1971), 109–112; Ewing S. Walker, *Software Age*, 4 (August-September, 1970), 16–17].

Предположим, что мы выполняем четырехпутевое слияние отрезков равной длины L_0 , имея память в M литер. Если мы разделим память на равные буферы размера $B = M/5$, то нам потребуется около L_0/B поисков для каждого вводного файла и $4L_0/B$ поисков для вывода, что в сумме дает $8L_0/B = 40L_0/M$ поисков. Но если мы используем четыре буфера ввода размера $M/6$ и один буфер вывода размера $M/3$, то нам потребуется всего лишь около $4 \times (6L_0/M) + 4 \times (3L_0/M) = 36L_0/M$ поисков! Времена передачи в обоих случаях одинаковы, так что мы ничего не потеряем от этого изменения.

В общем случае предположим, что мы хотим слить отсортированные файлы, имеющие длины L_0, \dots, L_p , в отсортированный файл длины $L_{P+1} = L_1 + \dots + L_p$, и предположим, что для k -го файла используется буфер размера B_k . Тогда

$$B_1 + \dots + B_p + B_{P+1} = M, \quad (6)$$

где M — общий объем наличной внутренней памяти. Число поисков будет приблизительно равно

$$\frac{L_1}{B_1} + \dots + \frac{L_p}{B_p} + \frac{L_{P+1}}{B_{P+1}}. \quad (7)$$

Попытаемся минимизировать эту величину при условии (6), считая для удобства, что B_k не обязаны быть целыми. Если увеличить B_j на δ и уменьшить B_k на ту же небольшую величину, то число поисков изменится на

$$\frac{L_j}{B_j + \delta} - \frac{L_j}{B_j} + \frac{L_k}{B_k - \delta} - \frac{L_k}{B_k} = \left(\frac{L_k}{B_k(B_k - \delta)} - \frac{L_j}{B_j(B_j + \delta)} \right) \delta,$$

т. е., распределение можно улучшить, если $L_j/B_j^2 \neq L_k/B_k^2$. Следовательно, мы получим минимальное число поисков, только если

$$\frac{L_1}{B_1^2} = \dots = \frac{L_P}{B_P^2} = \frac{L_{P+1}}{B_{P+1}^2}. \quad (8)$$

Так как минимум обязательно существует, он должен достигаться при

$$B_k = \sqrt{L_k M} / (\sqrt{L_1} + \dots + \sqrt{L_{P+1}}), \quad 1 \leq k \leq P+1, \quad (9)$$

поскольку это единственные значения B_1, \dots, B_{P+1} , удовлетворяющие одновременно (6) и (8). Подстановка этих значений в (7) дает весьма простую формулу для общего числа поисков:

$$(\sqrt{L_1} + \dots + \sqrt{L_{P+1}})^2 / M, \quad (10)$$

которую можно сравнить с числом $(P+1)(L_1 + \dots + L_{P+1})/M$, получающимся в том случае, если все буферы равны по длине. Выигрыш равен $\sum_{1 \leq j < k \leq P+1} (\sqrt{L_j} - \sqrt{L_k})^2 / M$ согласно упр. 1.2.3-31. К сожалению, формула (10) не годится для простого определения оптимальной схемы слияния, как в теореме К (ср. с упр. 14).

Дополнительные соображения. Если используются два дисковых устройства (одно для чтения и одно для записи), то можно совместить часть времени записи ценой дополнительного буфера вывода. Если используется метод "прогнозирования" (т. е. мы смотрим на последний ключ в каждом буфере ввода, чтобы определить, какой буфер первым освободится), то можно совместить часть времени чтения. К сожалению, простая процедура синхронизации, вроде процедуры в алгоритме 5.4.6F, не обеспечивает непрерывного ввода/вывода, так как меняющееся время чтения означает, что операции ввода и вывода не будут оканчиваться в одно время. Точный объем совмещений трудно оценить, и, возможно, лучше промоделировать процесс слияния, чтобы найти константы α и β для приближенного времени слияния, как в теореме Н.

Если мы используем "плавающие" буферы (не приписанные к конкретному файлу), то они должны быть одного размера, и тогда нельзя применять приведенную выше технику распределения буферов; можно только выбрать буферы вывода больше, чем буферы ввода. Если используется $2P$ буферов ввода (вместо P), мы совмещаем значительную долю времени ввода, однако удваиваем время поиска и, возможно, ничего не выиграем; по-видимому, лучше всего иметь $P+1$ или $P+2$ буферов ввода. Теперь читателю должно быть ясно, что никакой четкой стратегии для оптимальной сортировки на дисках не разработано; число возможных вариантов значительно превышает число стратегий, которые были проанализированы теоретически. Хотя теория, развитая в этом пункте, дает некоторые полезные представления, все еще требуется значительный объем экспериментальных исследований.

***Подробнее об оптимальных деревьях.** В теоремах Н и К интересно рассмотреть крайний случай $\beta = 0$, несмотря на то что практические ситуации обычно приводят к параметрам $0 \leq \alpha \leq \beta$. Какое дерево с n листьями имеет наименьшую возможную длину степенного пути? Любопытно, что в этом случае наилучшим оказывается трехпутевое слияние.

Теорема Л. Длина степенного пути дерева с n листьями никогда не меньше, чем

$$f(n) = \begin{cases} 3qn + 2(n - 3^q), & \text{если } 2 \cdot 3^{q-1} \leq n \leq 3^q; \\ 3qn + 4(n - 3^q), & \text{если } 3^q \leq n \leq 2 \cdot 3^q. \end{cases} \quad (11)$$

Тернарные деревья \mathcal{J}_n , определенные правилами

Picture: Рисунок на стр. 442

имеют минимальную длину степенного пути.

Доказательство. Важно заметить, что $f(n)$ — выпуклая функция, т. е.

$$f(n+1) - f(n) \geq f(n) - f(n-1) \quad \text{при всех } n \geq 2. \quad (13)$$

Это свойство существенно ввиду следующей леммы.

Лемма С. функция $g(n)$, определенная на положительных целых числах, удовлетворяет условию

$$\min_{1 \leq k \leq n} (g(k) + g(n - k)) = g(\lfloor n/2 \rfloor) + g(\lceil n/2 \rceil), \quad n \geq 2, \quad (14)$$

тогда и только тогда, когда она выпуклая.

Доказательство. Если $g(n+1) - g(n) < g(n) - g(n-1)$ при некотором $n > 2$, то имеем $g(n+1) + g(n-1) < g(n) + g(n)$, что противоречит (14). Обратно, если (13) имеет место для g и $1 \leq k < n - k$, то в силу выпуклости $g(k+1) + g(n-k-1) \leq g(k) + g(n-k)$. ■

Последнюю часть доказательства леммы С можно распространить на любое $m \geq 2$ и показать, что

$$\min_{\substack{n_1 + \dots + n_m = n \\ n_1, \dots, n_m \geq 1}} (g(n_1) + \dots + g(n_m)) = g(\lfloor n/m \rfloor) + g(\lfloor (n+1)/m \rfloor) + \dots + g(\lfloor (n+m-1)/m \rfloor), \quad (15)$$

если g выпукла. Пусть

$$f_m(n) = f(\lfloor n/m \rfloor) + f(\lfloor (n+1)/m \rfloor) + \dots + f(\lfloor (n+m-1)/m \rfloor).$$

Доказательство теоремы L завершается доказательством того, что $f_3(n) + 3n = f(n)$ и $f_m(n) + mn \geq f(n)$ при всех $m \geq 2$ (см. упр. 11). ■

Было бы очень хорошо, если бы оптимальные деревья всегда характеризовались так же четко, как в теореме L. Но результаты для $\alpha = \beta$, которые мы видим в табл. 1, показывают, что функция $A_1(n)$ не всегда выпукла. На самом деле табл. 1 достаточно, чтобы опровергнуть большинство простых предположений об оптимальных деревьях! Мы, однако, можем спасти часть теоремы L для случая произвольных α и β . М. Шлюмбергер и Ж. Вийеман показали, что *высоких* порядков слияния всегда можно избежать.

Теорема М. Если даны α и β , как в теореме H, то существует оптимальное дерево, в котором степень любого узла не превосходит

$$d(\alpha, \beta) = \left\lceil \min_{k \geq 1} \left(k + \left(1 + \frac{1}{k} \right) \left(1 + \frac{\beta}{\alpha} \right) \right) \right\rceil. \quad (17)$$

Доказательство. Пусть n_1, \dots, n_m — положительные целые числа, такие, что $n_1 + \dots + n_m = n$, $A(n_1) + \dots + A(n_m) = A_m(n)$, $n_1 \leq \dots \leq n_m$, и предположим, что $m \geq d(\alpha, \beta) + 1$. Пусть k — то значение, при котором достигается минимум в (17); покажем, что

$$\alpha n(m-k) + \beta n + A_{m-k}(n) \leq \alpha m n + \beta n + A_m(n), \quad (18)$$

и, следовательно, минимум в (4) всегда достигается при некотором $m \leq d(\alpha, \beta)$.

По определению, поскольку $m \geq k + 2$, мы должны иметь

$$\begin{aligned} A_{m-k}(n) &\leq A_1(n_1 + \dots + n_{k+1}) + A_1(n_{k+2}) + \dots + A_1(n_m) \leq \\ &\leq \alpha(n_1 + \dots + n_{k+1})(k+1) + \beta(n_1 + \dots + n_{k+1}) + \\ &\quad + A_1(n_1) + \dots + A_1(n_m) = \\ &= (\alpha(k+1) + \beta)(n_1 + \dots + n_{k+1}) + A_m(n) \leq \\ &\leq (\alpha(k+1) + \beta)(k+1)n/m + A_m(n), \end{aligned}$$

отсюда (18) легко выводится. (Тщательное рассмотрение этого доказательства показывает, что оценка (17) является "наилучшей из возможных" в том смысле, что некоторые оптимальные деревья должны иметь узлы степени $d(\alpha, \beta)$; см. упр. 13.) ■

Построение в теореме K требует $O(N^2)$ ячеек памяти и $O(N^2 \log N)$ шагов для вычисления $A_m(n)$ при $1 \leq m \leq n \leq N$, а в теореме M показано, что требуется только $O(N)$ ячеек и $O(N^2)$ шагов. Шлюмбергер и Вийеман открыли еще несколько очень интересных свойств оптимальных деревьев [Optimal disk merge patterns, *Acta Informatica*, 3 (1973), 25–36].

Использование сцепления. М. А. Готц [CACM, 6 (1963), 245–248] предложил интересный способ, при котором отдельные дорожки связываются вместе и благодаря этому исключается поиск при выводе. Для реализации его идеи требуется почти невообразимый набор программ, управляющих

дисками, но сама идея, кроме сортировки, применима во многих других задачах, и поэтому она может привести к весьма ценным методам общего назначения.

Идея проста. Вместо размещения дорожек последовательно внутри цилиндров диска мы связываем их вместе и храним списки свободного пространства по одному для каждого цилиндра. Когда приходит время вывести дорожку информации, мы записываем ее в текущий цилиндр (в тот, где оказался держатель головок), если этот цилиндр не полон. При этом время поиска обычно сводится на нет.

Здесь нас подстерегает ловушка. Дело в том, что мы не можем записывать ссылку на следующую дорожку внутри самой дорожки, поскольку необходимая для этого информация в нужный момент не известна. (Мы могли бы, если это нас устроит, записывать ссылку на предыдущую дорожку и на следующем проходе читать файл в обратном направлении.) Можно хранить отдельную таблицу адресов связи для дорожек каждого файла, возможно, частично на самом диске. Списки свободного пространства можно компактно представить с помощью битовых таблиц, где 1000 битов указывают занятость или незанятость 1000 дорожек.

Поможет ли сортировка ключей? Если записи длинные, а ключи короткие, весьма заманчиво создать новый файл, состоящий только из ключей вместе с порядковыми номерами, определяющими их первоначальное положение в файле. После сортировки этого файла ключей можно заменить ключи последовательными числами 1, 2, ... Затем этот новый файл можно отсортировать по положению в первоначальном файле, и мы получим удобное описание того, как "растасовать" записи, чтобы переразместить их в требуемом порядке.

Схематически представим процесс так:

- a) исходный файл $(K_1, I_1) (K_2, I_2) \dots (K_N, I_N)$ длинный;
- b) ключевой файл $(K_1, 1) (K_2, 2) \dots (K_N, N)$ короткий;
- c) отсортированный (b) $(K_{p_1}, p_1) (K_{p_2}, p_2) \dots (K_{p_N}, p_N)$ короткий;
- d) отредактированный (c) $(1, p_1) (2, p_2) \dots (N, p_N)$ короткий;
- e) отсортированный (d) $(q_1, 1) (q_2, 2) \dots (q_N, N)$ короткий;
- f) отредактированный (a) $(q_1, I_1) (q_2, I_2) \dots (q_N, I_n)$ длинный.

Здесь $p_j = k$ тогда и только тогда, когда $q_k = j$; два процесса сортировки в стадиях (c) и (e) сравнительно быстрые (иногда можно обойтись внутренней сортировкой), так как записи не слишком длинные. На стадии (f) мы свели задачу к сортировке файла, ключи которого—просто числа $\{1, 2, \dots, N\}$; каждая запись теперь определяет в точности то место, куда ее следует переместить.

Проблема внешнего переразмещения, остающаяся после стадии (f), кажется на первый взгляд тривиальной, но фактически она очень сложна, и все еще не найдено ни одного действительно хорошего алгоритма (существенно лучшего, чем сортировка). Мы, очевидно, могли бы выполнить переразмещение за N шагов, перемещая всякий раз по одной записи; для достаточно большого N это лучше, чем $N \log N$ в методах сортировки. Но N никогда не бывает таким большим; N поисков просто нереальны!

К отредактированным записям можно эффективно применять какой-либо метод поразрядной сортировки, так как ключи имеют строго равномерное распределение. На многих высокоскоростных ЭВМ с высокоскоростными периферийными устройствами время вычислений для восьмипутового распределения значительно меньше, чем время вычислений для восьмипутового слияния; следовательно, распределительная сортировка может быть наилучшей из всех процедур (см. п. 5.4.7, а также упр. 17).

Но, по-видимому, бессмысленно проводить сортировку ключей лишь для того, чтобы выполнить затем полную сортировку! Одна из причин неожиданной трудности проблемы внешнего переразмещения была открыта Р. У. Флойдом, который нашел нетривиальную нижнюю границу для числа поисков, требуемых, чтобы переставить записи в дисковом файле [Complexity of Computer Computations (New York, Plenum, 1972), 105–109].

Результат Флойда удобно описать, воспользовавшись аналогией с лифтом, как в п. 5.4.8; на этот раз мы хотим найти график работы лифта, минимизирующий число остановок, а не пройденное расстояние. (Минимизация числа *остановок* не вполне эквивалентна нахождению алгоритма переразмещения с минимальным числом поисков, так как одна остановка объединяет вход в лифт с выходом из него; однако разница не слишком велика, и мы воспользуемся критерием минимизации остановок, чтобы пояснить основные идеи.) Будем использовать функцию "дискретной энтропии"

$$F(n) = \sum_{1 < k \leq n} (\lceil \log_2 k \rceil + 1) = B(n) + n - 1 = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + n, \quad (19)$$

где $B(n)$ есть функция "бинарной вставки" (см. формулу (5.3.1-3)). Согласно (5.3.1-33), функция $F(n)$ есть не что иное, как минимальная длина внешнего пути бинарного дерева с n листьями, и

$$n \log_2 n \leq F(n) \leq n \log_2 n + 0.0861n. \quad (20)$$

Так как $F(n)$ выпукла и удовлетворяет условию $F(n) = n + F(\lfloor n/2 \rfloor) + F(\lceil n/2 \rceil)$, то по лемме С

$$F(n) \leq F(k) + F(n-k) + n \quad \text{при } 0 \leq k \leq n. \quad (21)$$

Это соотношение вытекает также из представления F в виде длины внешнего пути; в дальнейшем оно окажется решающим фактом.

Как и в п. 5.4.8, мы будем предполагать, что лифт вмещает b человек, каждый этаж вмещает c человек и имеется n этажей. Пусть s_{ij} — число людей, находящихся в данный момент на этаже i и стремящихся попасть на этаж j . По определению оценка совместности любого расположения людей в здании есть $\sum_{1 \leq i, j \leq n} F(s_{ij})$.

Предположим, например, что $b = c = n = 6$ и что первоначально 36 человек следующим образом распределены между этажами:

$$\begin{array}{cccccc} \text{▬▬▬▬▬} & & & & & \\ 123456 & 123456 & 123456 & 123456 & 123456 & 123456 \end{array} \quad (22)$$

Лифт пуст и находится на этаже 1; "▬" обозначает свободное место. На каждом этаже находится человек, стремящийся попасть на произвольный этаж, поэтому величины s_{ij} равны 1 и оценка совместности равна 0. Если теперь лифт отвезет 6 человек на этаж 2, то мы получим расположение

$$\begin{array}{cccccc} & 123456 & & & & \\ \text{▬▬▬▬▬} & 123456 & 123456 & 123456 & 123456 & 123456 \end{array} \quad (23)$$

и оценка совместности станет равна $6F(0) + 24F(1) + 6F(2) = 12$. Допустим, лифт перевезет 1, 1, 2, 3, 3, 4 на этаж 3:

$$\begin{array}{cccccc} & & 112334 & & & \\ \text{▬▬▬▬▬} & 245566 & 123456 & 123456 & 123456 & 123456 \end{array} \quad (24)$$

Оценка совместности изменилась до $4F(2) + 2F(3) = 18$. Когда каждый человек будет в конце концов перевезен к своему месту назначения, оценкой совместности будет $6F(6) = 96$.

Флойд заметил, что оценка совместности никогда не увеличивается больше, чем на $b + c$ за одну остановку, так как множество из s людей с равным назначением, объединяясь с аналогичным множеством мощности s' , увеличивает оценку на $F(s + s') - F(s) - F(s') \leq s + s'$. Таким образом, имеем следующий результат.

Теорема F. Пусть t — определенная выше оценка совместности начального расположения людей. Лифт должен сделать по крайней мере

$$\lceil (nF(c) - t)/(b + c) \rceil$$

остановок, чтобы доставить всех людей к их месту назначения. ■

Сформулируем этот результат применительно к дискам. Допустим, имеется N записей по B в каждом блоке, и предположим, что во внутренней памяти одновременно помещается M записей. При каждом чтении с диска один блок переносится в память, а при каждой записи один блок помещается на диск, и s_{ij} есть число записей в блоке i , принадлежащих блоку j . Если $N \geq B^2$, то существует начальное расположение, для которого все $s_{ij} \leq 1$, и, следовательно, для перераспределения записей необходимо прочитать по крайней мере $(N/B)F(B)/M \approx (N \log_2 B)/M$ блоков. (Если B велико, то множитель $\log_2 B$ делает эту нижнюю оценку нетривиальной, но если $B = 1$, то нижнюю оценку, очевидно, можно улучшить.)

Упражнения

- [M22] В тексте объясняется метод, с помощью которого среднее время ожидания, требуемое для чтения доли x дорожки, уменьшается с $\frac{1}{2}$ до $\frac{1}{2}(1 - x^2)$ оборотов. Это минимально возможное время, когда имеется один держатель головок. Каково соответствующее минимальное время ожидания,

если имеются два держателя головок, разнесенные на 180° (в предположении, что в любой момент данные может передавать только один из держателей)?

2. [М30] (Э. Г. Конхейм.) Цель этой задачи — исследовать, на какое расстояние должен переместиться держатель головок во время слияния файлов, расположенных "ортогонально" к цилиндрам. Допустим, имеется P файлов, каждый из которых содержит L блоков записей, и предположим, что первый блок каждого файла находится на цилиндре 1, второй — на цилиндре 2 и т.д. Движением держателя головок во время слияния управляет относительный порядок последних ключей каждого блока; следовательно, можно изобразить ситуацию следующим способом, удобным для математической интерпретации. Рассмотрим множество из PL упорядоченных пар

$$\begin{array}{cccc} (a_{11}, 1) & (a_{21}, 1) & \dots & (a_{P1}, 1) \\ (a_{12}, 2) & (a_{22}, 2) & \dots & (a_{P2}, 2) \\ \vdots & \vdots & & \vdots \\ (a_{1L}, L) & (a_{2L}, L) & \dots & (a_{PL}, L) \end{array}$$

где множество $\{a_{ij} \mid 1 \leq i \leq P, 1 \leq j \leq L\}$ соответствует множеству чисел $\{1, 2, \dots, PL\}$ в некотором порядке и $a_{ij} < a_{i,j+1}$ при $1 \leq j < L$ (строки представляют цилиндры, столбцы — исходные файлы). Отсортируем пары по их первым компонентам, и пусть получившейся последовательностью будет $(1, j_1) (2, j_2) \dots (PL, j_{PL})$. Покажите, что если все $(PL)!/L!^P$ возможных наборов a_{ij} равновероятны, то среднее значение

$$|j_2 - j_1| + |j_3 - j_2| + \dots + |j_{PL} - j_{PL-1}|$$

равно

$$(L-1) \left(1 + (P-1) 2^{2L-2} / \binom{2L}{L} \right).$$

[Указание: см. упр. 5.2.1-14.] Заметим, что при $L \rightarrow \infty$ это значение асимптотически равно $\frac{1}{4}(P-1)L\sqrt{\pi L}$.

3. [М15] Предположим, что ограниченный размер внутренней памяти делает нежелательным 10-путевое слияние. Как можно изменить рекуррентные соотношения (3), (4), (5), чтобы $A_1(n)$ стало минимальным значением $\alpha D(\mathcal{J}) + \beta E(\mathcal{J})$ среди всех деревьев \mathcal{J} с n листьями, не имеющих узлов степени, большей чем 9?
- >4. [М21] (а) Выведите формулу, аналогичную (2), для времени выполнения P -путевого слияния L литер, если используется видоизмененная форма квадратичного распределения буферов¹⁵: все P буферов ввода должны иметь равные длины, а размер буфера вывода следует выбрать так, чтобы минимизировать время поиска.
(б) Покажите, что построение в теореме К можно изменить так, что получится схема слияния, которая будет оптимальной в смысле вашей формулы из части (а).
5. [М20] Если используются два диска, причем чтение с одного совмещается с записью на другой, то мы не можем применять схему слияния, подобную представленной на рис. 93, так как некоторые листья находятся на четных уровнях, а другие на нечетных. Покажите, как изменить построение в теореме К, чтобы получались оптимальные деревья с учетом ограничения, что все листья находятся на четных уровнях или все на нечетных.
- >6. [22] Найдите дерево, оптимальное в смысле упр. 6, если $n = 23$ и $\alpha = \beta = 1$. (Возможно, у вас появится желание прибегнуть к помощи ЭВМ.)
- >7. [М24] Если длины начальных отрезков не равны, то наилучшая схема слияния (в смысле теоремы Н) минимизирует $\alpha D(\mathcal{J}) + \beta E(\mathcal{J})$, где $D(\mathcal{J})$ и $E(\mathcal{J})$ теперь представляют собой взвешенные длины путей: каждому листу дерева приписываются веса w_1, \dots, w_n (соответствующие длинам начальных отрезков), а суммы степеней и длины путей умножаются на соответствующие веса. Например, если \mathcal{J} — дерево, представленное на рис. 92, то получим $D(\mathcal{J}) = 6w_1 + 6w_2 + 7w_3 + 9w_4 + 9w_5 + 7w_6 + 4w_7 + 4w_8$, $E(\mathcal{J}) = 2w_1 + 2w_2 + 2w_3 + 3w_4 + 3w_5 + 2w_6 + w_7 + w_8$. Докажите, что при некотором k всегда существует оптимальная схема, в которой первыми сливаются k самых коротких отрезков.
8. [49] Существует ли алгоритм, который при заданных α, β и весах w_1, \dots, w_n находит оптимальные в смысле упр. 7 деревья, затрачивая только $O(n^c)$ шагов при некотором c ?
9. [М49] (Шлюмбергер и Вийеман.) Верно ли, что для всех n, α и β существуют оптимальные в смысле теоремы Н деревья с n листьями, у которых все листья находятся самое большее на

¹⁵ См. также формулу (9). — Прим. ред.

двух соседних уровнях? Если α и β фиксированы, существует ли число m , такое, что для всех достаточно больших n имеет место $A_1(n) = \alpha mn + \beta n + A_m(n)$? (Оба эти предположения были проверены в случае $\alpha/\beta \geq 4$.)

10. [BM49] Каково асимптотическое поведение $A_1(n)$ при $n \rightarrow \infty$ при заданных α и β ?
11. [M29] В обозначениях (11) и (16) докажите, что $f_m(n) + mn \geq f(n)$ при всех $m, n \geq 2$, и определите все m и n , при которых имеет место равенство.
12. [25] Докажите, что при всех $n > 0$ существует дерево с n листьями и минимальной длиной степенного пути (11), все листья которого расположены на одном уровне.
13. [M24] Покажите, что при $2 \leq n \leq d(\alpha, \beta)$ единственной наилучшей схемой слияния в смысле теоремы Н является n -путевое слияние. (Ср. с формулой (17).)
14. [40] При использовании квадратичного метода распределения буферов время поиска для схемы слияния на рис. 92 будет пропорционально $(\sqrt{2} + \sqrt{4} + \sqrt{1} + \sqrt{1} + \sqrt{8})^2 + (\sqrt{1} + \sqrt{1} + \sqrt{2})^2 + \sqrt{1} + \sqrt{2} + (\sqrt{1} + \sqrt{4})^2 + (\sqrt{1} + \sqrt{1} + \sqrt{2})^2$; это значение представляет собой сумму величин $(\sqrt{n_1} + \dots + \sqrt{n_m} + \sqrt{n_1 + \dots + n_m})^2$ по всем внутренним узлам, где поддеревья этих узлов имеют (n_1, \dots, n_m) листьев. Напишите программу для ЭВМ, которая порождает деревья с минимальным временем поиска, имеющие 1, 2, 3, ... листьев, используя для оценки времени поиска эту формулу.
15. [M22] Покажите, что можно несколько усилить теорему F, если лифт первоначально пуст и если $nF(c) \neq t$: в этом случае необходимо не менее $\lceil (nF(c) + b - t)/(b + c) \rceil$ остановок.
16. [23] (Р. У. Флойд.) Найдите график работы лифта, перевозящий всех людей из (22) к их местам назначения за 12 или меньшее число остановок. (В (23) показано положение после одной, а не после двух остановок.)
17. [25] (Б. Т. Беннет и А. Ч. Мак-Келлар, 1971.) Рассмотрим следующий метод с сортировкой ключей, продемонстрированный на примере файла с 10 ключами:
 - a) Исходный файл: $(50, I_0) (08, I_1) (51, I_2) (06, I_3) (90, I_4) (17, I_5) (89, I_6) (27, I_7) (65, I_8) (42, I_9)$.
 - b) Файл ключей: $(50, 0) (08, 1) (51, 2) (06, 3) (90, 4) (17, 5) (89, 6) (27, 7) (65, 8) (42, 9)$.
 - c) Отсортированный (b): $(06, 3) (08, 1) (17, 5) (27, 7) (42, 9) (50, 0) (51, 2) (65, 8) (89, 6) (90, 4)$.
 - d) Присваивание номеров ящиков: $(3, 3) (3, 9) (2, 1) (2, 5) (2, 7) (2, 8) (1, 0) (1, 2) (1, 4) (1, 6)$.
 - e) Отсортированный (d): $(1, 0) (2, 1) (1, 2) (3, 3) (1, 4) (2, 5) (1, 6) (2, 7) (2, 8) (3, 9)$.
 - f) (a), распределенный по ящикам с использованием (e):

Ящик 1: $(50, I_0) (51, I_2) (90, I_4) (89, I_6)$.

Ящик 2: $(08, I_1) (17, I_5) (27, I_7) (65, I_8)$.

Ящик 3: $(06, I_3) (42, I_9)$.

- g) Результат выбора с замещением (сначала читается ящик 3, затем ящик 2, затем ящик 1): $(06, I_3) (08, I_1) (17, I_5) (27, I_7) (42, I_9) (50, I_0) (51, I_2) (65, I_8) (89, I_6) (90, I_4)$.

Номера ящиков на шаге (d) присваиваются путем применения к (c) выбора с замещением справа налево в убывающем порядке по второй компоненте. Номер ящика—это номер отрезка. В приведенном примере используется выбор с замещением только с двумя элементами в дереве; для выбора с замещением в (d) и (g) должен использоваться одинаковый размер дерева выбора. Заметим, что содержимое ящиков не обязательно упорядочено.

Докажите, что этот метод будет сортировать, т. е. что выбор с замещением в (g) образует лишь один отрезок. (Этот метод уменьшает число ящиков, требуемое обычной сортировкой ключей посредством распределения, особенно если исходные данные уже в сильной степени упорядочены.)

- >18. [25] Некоторые системы предоставляют программистам виртуальную память: можно писать программы, исходя из предположения, что имеется очень большая внутренняя память, способная вместить все данные. Эта память разделена на страницы, и лишь немногие из них действительно находятся во внутренней памяти в любой момент времени, остальные же хранятся на дисках или барабанах. Программисту не нужно вникать во все эти подробности, так как система сама обо всем заботится: новые страницы автоматически вызываются в память, когда они нужны.

Может показаться, что появление виртуальной памяти приводит к отмиранию методов внешней сортировки, так как эта работа может быть выполнена просто с помощью методов, разработанных для внутренней сортировки. Разберите эту ситуацию. Почему специально разработанный метод внешней сортировки может оказаться лучше применения общей техники подкачки страниц к методу внутренней сортировки?

5.5. РЕЗЮМЕ. ИСТОРИЯ И БИБЛИОГРАФИЯ

Теперь, когда мы почти достигли конца этой очень длинной главы, стоит "отсортировать" наиболее важные факты из числа изученных.

Алгоритм сортировки—это процедура, которая реорганизует файл записей таким образом, чтобы ключи оказались в возрастающем порядке. Благодаря такому упорядоченному расположению

группируются записи с равными ключами, становится возможной эффективная обработка многих файлов, отсортированных по одному ключу, создается основа для эффективных алгоритмов выборки и более убедительно выглядят документы, подготовленные с помощью ЭВМ.

Внутренняя сортировка используется в тех случаях, когда все записи помещаются в быстрой внутренней памяти машины. Мы изучили с разной степенью детализации более двух десятков алгоритмов внутренней сортировки, но нам, наверное, было бы куда спокойнее, если бы мы не знали так много различных подходов к этой задаче! Изучение всех этих методов было приятным времяпровождением. Но теперь перед нами безрадостная перспектива—предстоит на деле решить, какой метод следует использовать в той или иной конкретной ситуации.

Было бы прекрасно, если бы только один или два метода сортировки превосходили все остальные безотносительно к приложению или используемой машине. На самом же деле каждый метод имеет свои собственные, одному ему присущие достоинства. Например, метод пузырька (алгоритм 5.2.2B) не имеет ярко выраженных преимуществ, так как всегда можно найти лучший способ сделать то, что он делает; но даже этот метод после соответствующего обобщения оказывается полезным для сортировки с двумя лентами (см. п. 5.4.8). Итак, приходим к заключению, что почти все алгоритмы заслуживают того, чтобы о них помнили, так как существуют приложения, в которых они оказываются весьма хорошими.

В следующем кратком обзоре освещаются основные аспекты наиболее важных алгоритмов внутренней сортировки, с которыми мы встречались. (Как обычно, N означает число записей в файле.)

1. *Распределяющий подсчет.* Алгоритм 5.2D очень полезен, если диапазон ключей невелик. Метод устойчив (не изменяется порядок записей с равными ключами), но требуется память для счетчиков и $2N$ записей. Одна из модификаций, экономящая N из этих записей ценой устойчивости, встречается в упр. 5.2-13.

2. *Простые вставки.* Алгоритм 5.2.1S наиболее прост для программирования, не требует дополнительного пространства и вполне эффективен при малых N (скажем, при $N \leq 25$). При больших N он становится невыносимо медленным, если только исходные данные не окажутся сразу почти упорядоченными.

3. *Сортировка с убывающим шагом.* Алгоритм 5.2.1D (метод Шелла) так же довольно просто программируется, использует минимальный объем памяти и довольно эффективен при умеренно больших N (скажем, при $N \leq 1000$).

4. *Вставки в список.* Алгоритм 5.2.1L основан на той же идее, что и простые вставки, и поэтому годится только при небольших N . Как и в других методах сортировки списков, в нем благодаря операциям со ссылками экономится стоимость перемещения длинных записей; это особенно удобно, когда записи имеют переменную длину или являются частью других структур данных.

5. *Сортировка с вычислением адреса* эффективна, если ключи подчиняются известному (обычно равномерному) закону распределения; важнейшими вариантами этого подхода являются *вставки в несколько списков* (алгоритм 5.2.1M) и комбинированная поразрядная сортировка со вставками Макларена (рассмотренная в конце п. 5.2.5). Для последнего метода достаточно иметь $O(\sqrt{N})$ дополнительных ячеек памяти.

6. *Обменная сортировка со слиянием.* Алгоритм 5.2.2M (метод Бэтчера) и родственный ему алгоритм *битонной сортировки* (упр. 5.3.4-10) полезны, если можно одновременно выполнять большое число сравнений.

7. *Обменная сортировка с разделением* (метод Хоара, широко известный как *быстрая сортировка*). Алгоритм 5.2.2Q, вероятно, самый полезный универсальный алгоритм внутренней сортировки, поскольку он требует очень мало памяти и опережает своих конкурентов по среднему времени выполнения на большинстве вычислительных машин. Однако в наихудшем случае он может работать очень медленно. Поэтому, если вероятность неслучайных данных достаточно велика, приходится тщательно выбирать разделяющий элемент. Если выбирается медиана из трех элементов (как предлагается в упр. 5.2.2-55), то такое поведение, как в наихудшем случае, становится крайне маловероятным и, кроме того, несколько уменьшается среднее время работы.

8. *Простой выбор.* Алгоритм 5.2.3S довольно прост и особенно подходит в случае, когда имеется специальное оборудование для быстрого поиска наименьшего элемента в списке.

9. *Пирамидальная сортировка.* Алгоритм 5.2.4H при минимальных требованиях памяти обеспечивает достаточно высокую скорость сортировки; как среднее время работы, так и максимальное время примерно вдвое больше среднего времени быстрой сортировки.

10. *Слияние списков.* Алгоритм 5.2.4L осуществляет сортировку списков, при которой, так же как и при пирамидальной сортировке, обеспечивается весьма высокая скорость даже в наихудшем случае, кроме того, этот метод устойчив по отношению к равным ключам.

11. *Поразрядная сортировка с использованием алгоритма 5.2.5R*—это не что иное, как сортировка списков, которая приемлема для ключей, либо очень коротких, либо имеющих необычный порядок лексикографического сравнения. Вместо ссылок можно применить распределяющий подсчет (пункт 1 нашего обзора); такая процедура потребует пространства для $2N$ записей и для таблицы, счетчиков, но благодаря простоте внутреннего цикла она особенно хороша для сверхбыстрых ЭВМ—”пожирательниц чисел”, имеющих опережающее управление. (*Предостережение.* Поразрядную сортировку не следует использовать при малых N .)

12. *Сортировка вставками и слиянием* (см. п. 5.3.1) наиболее приемлема при очень малых N в ”прямолинейных” программах. Например, этот метод оказался бы подходящим в тех приложениях, где требуется сортировать много групп из пяти или шести записей.

13. Могут существовать и гибридные методы, объединяющие один или более из приведенных выше методов. Например, короткие подфайлы, возникающие при быстрой сортировке, можно сортировать слиянием и вставками.

14. И наконец, для реализации безымянного метода, встретившегося в ответе к упр. 5.2.1-3, требуется, по-видимому, кратчайшая из возможных программ сортировки. Но среднее время работы такой программы пропорционально N^3 , т. е. это самая медленная программа сортировки в книге!

Пространственные и временные характеристики многих из этих методов, запрограммированных для MIX, сведены в табл. 1.

Picture: Таблица 1, стр. 454

Важно иметь в виду, что числа в этой таблице являются лишь грубыми показателями относительного времени сортировки. Они применимы только к одной ЭВМ, и предположения, касающиеся исходных данных, далеко не для всех программ абсолютно правомерны. Сравнительные таблицы, подобные этой, приводились во многих работах, но не найдется таких двух авторов, которые пришли бы к одинаковым выводам! Тем не менее указания о времени работы позволяют оценить хотя бы порядок скорости, которую следует ожидать от каждого алгоритма при сортировке записей из одного слова, так как MIX—довольно типичная машина.

Столбец ”пространство” в табл. 1 содержит некоторую информацию о количестве вспомогательной памяти, используемой каждым алгоритмом, в единицах длины записи. Здесь буквой ε обозначена доля записи, требуемая для одного поля связи; так, например, $N(1 + \varepsilon)$ означает, что метод требует пространство для N записей и N полей связи.

В асимптотических средних и максимальных временах, содержащихся в табл. 1, учитываются только главные члены, доминирующие при больших N в предположении случайных исходных данных; c обозначает произвольную константу. Эти формулы могут иногда ввести в заблуждение, поэтому указано также фактическое время работы программы для двух конкретных последовательностей исходных данных. Случай $N = 16$, относится к шестнадцати ключам, так часто появлявшимся в примерах § 5.2, а случай $N = 1000$ относится к последовательности $K_1, K_2, \dots, K_{1000}$, определенной формулами

$$K_0 = 0; \quad K_{n+1} = (3141592621K_n + 2113148651) \bmod 10^{10}.$$

Для получения характеристик каждого алгоритма, представленного в таблице, использовалась вполне ”высококачественная” MIX-программа, часто с учетом усовершенствований, предложенных в упражнениях. Размер байта при выполнении этих программ принят равным 100.

Для *внешней сортировки* необходимы методы, отличающиеся от методов внутренней сортировки, потому что в этом случае предполагается использование сравнительно простых структур данных и большое внимание уделяется уменьшению времени ввода/вывода. В п. 5.4.6 рассматриваются интересные методы, развитые для ленточной сортировки, а в п. 5.4.9 обсуждается использование дисков и барабанов.

Конечно, сортировка—не единственное содержание этой главы. Попутно мы узнали много о том, как работать со структурами данных, как обращаться с внешней памятью и как анализировать алгоритмы, и, наверное, мы узнали немножко и о том, как открывать новые алгоритмы.

Ранние разработки. Поиск источника современных методов сортировки возвращает нас в 19-е столетие, когда были изобретены первые машины для сортировки. В Соединенных Штатах перепись всех граждан проводится каждые 10 лет, и уже к 1880 г. проблема обработки огромных по объему данных переписи стала очень острой; и в самом деле, число одиноких (в отличие от числа состоящих в браке) в 1880 г. в таблицах отсутствует, хотя

Picture: Рис. 94. Табулятор Холлерита. (Фотография любезно предоставлена архивом ИВМ.)

вся необходимая, информация была собрана. Герман Холлерит, двадцатилетний служащий Бюро переписи, изобрел остроумный электрический табулятор, отвечающий нуждам сбора статистики, и около 100 его машин успешно использовались при обработке списков переписи 1890 г.

На рис. 94 изображен первый аппарат Холлерита, приводимый в действие от аккумуляторных батарей. Для нас основной интерес представляет "сортировальный ящик" справа, который открыт с целью показать половину из 26 внутренних отделений. Оператор вставляет перфокарту размера $6\frac{5}{8} \times 3\frac{1}{4}$ дюймов в "пресс" и опускает рукоятку; это приводит к тому, что закрепленные на пружинах штыри на верхней панели в тех местах, где на карте отперфорированы отверстия, входят в контакт с ртутью на нижней панели.

В результате замыкания соответствующей цепи показание связанного с ней циферблата изменяется на 1 и, кроме того, одна из 26 крышек сортировального ящика открывается. В этот момент оператор отпускает пресс, кладет карту в открытое отделение и закрывает крышку. По сообщениям, как-то через эту машину пропустили 19071 карту за один 6.5-часовой рабочий день; в среднем примерно 49 карт в минуту! (Средний оператор работал примерно втрое медленней.)

Население продолжало неуклонно расти, и первые табуляторы-сортировщики оказались недостаточно быстрыми, чтобы справиться с обработкой переписи 1900 г., поэтому Холлерит изобрел еще одну машину, чтобы предотвратить еще один кризис в обработке данных. Его новое устройство (запатентованное в 1901 и 1904 гг.) имело автоматическую подачу карт и выглядело, в сущности, почти так же, как современные карточные сортировальные машины. История ранних машин Холлерита с интересными подробностями изложена Леоном Э. Трусделлом в *The Development of Punch Card Tabulation* (Washington: U. S. Bureau of the Census, 1965); см. также сообщения современников Холлерита: *Columbia College School of Mines Quarterly*, 10 (1889), 238–255; *J. Franklin Inst.*, 129 (1890), 300–306; *The Electrical Engineer*, 12 (Nov. 11, 1891), 521–530; *J. Amer. Statistical Assn.*, 2 (1891), 330–341; 4 (1895), 365; *J. Royal Statistical Soc.*, 55 (1892), 326–327; *Alegemeines Statistisches Archiv*, 2 (1892), 78–126; *J. Soc. Statistique de Paris*, 33 (1892), 87–96; U. S. Patents 395781 (1889), 685608 (1901), 777209 (1904). Холлерит и другой бывший служащий Бюро переписи Джеймс Пауэрс в дальнейшем основали конкурирующие компании, которые в конце концов вошли соответственно в корпорации IBM и Remington Rand.

Сортировальная машина Холлерита—это, конечно, основа методов поразрядной сортировки, используемых в цифровых ЭВМ. В его патенте упоминается, что числовые элементы, содержащиеся два столбца, должны сортироваться "по отдельности для каждого столбца", но он не говорит, какой столбец (единиц или десятков) должен рассматриваться первым. Далеко не очевидная идея сортировки сначала по столбцу единиц была, по-видимому, открыта каким-то неизвестным оператором и передана остальным (см. п. 5.2.5); она имеется в самом раннем сохранившемся руководстве IBM по сортировке (1936 г.). Первым известным упоминанием этого метода "справа налево" является случайное замечание, встретившееся в статье Л. Дж. Комри, *Trans. of the Office Machinery Users' Assoc.* (London, 1930), 25–37. Нечаянно Комри оказался первым, кто сделал важное наблюдение, что табуляторы можно плодотворно применять в научных вычислениях, хотя первоначально они создавались для статистических и бухгалтерских приложений. Его статья особенно интересна, поскольку, содержит подробное описание табуляторов, имевшихся в Англии в 1930 г. Сортировальные машины в то время обрабатывали от 360 до 400 карт в минуту и сдавались в аренду за 9 фунтов стерлингов в месяц.

Идея слияния восходит к другому устройству для обработки карт—*подборочной машине*, которая была изобретена значительно позднее (в 1938 г.). Оборудованная двумя подающими механизмами, она могла слить две отсортированные колоды карт в одну всего за один проход; метод выполнения этого слияния хорошо описан в первом руководстве IBM по методам подборки (апрель 1939 г.). [Ср. с James W. Bryce. U. S. Patent 2189024 (1940).]

Затем на сцене появились ЭВМ и разработка методов сортировки тесно переплелась с их развитием. На самом деле имеются свидетельства того, что программа сортировки была первой когда-либо написанной для вычислительных машин с запоминаемой программой. Конструкторы вычислительной машины EDVAC особенно интересовались сортировкой, поскольку она выступала как наиболее характерный представитель потенциальных нечисленных приложений ЭВМ. Они понимали, что удовлетворительная система команд должна годиться не только для составления программы решения разностных уравнений; в ней должна быть достаточная гибкость, чтобы справиться с комбинаторными аспектами "выбора решений" в алгоритмах. Поэтому Джон фон Нейман подготовил в 1945 г. программы для внутренней сортировки слиянием, чтобы убедиться в необходимости некоторых кодов команд, которые он предлагал для машины EDVAC; существовали эффективные сортировальные машины специального назначения, и они служили тем естественным стандартом, в сопоставлении с которым можно было оценить достоинства предлагаемой организацией вычислительной машины.

Подробно это интересное исследование описано в статье Д. Э. Кнута [*Computing Surveys*, 2 (1970), 247–260]; первую программу сортировки фон Неймана в окончательном, ”отполированном” виде см. в его *Collected Works*, 5 (New York, Macmillan, 1963), 196–214.

Из-за ограниченного объема памяти в ранних машинах приходилось думать о внешней сортировке наравне с внутренней, и в докладе ”Progress Report on the EDVAC”, подготовленном Дж. П. Эккертом и Дж. У. Мочли для школы Мура по электротехнике [Moore school of Electrical Engineering (September 30, 1945)], указывалось, что ЭВМ, оснащенная устройством с магнитной проволокой или лентой, могла бы моделировать действия карточного оборудования, достигая при этом большей скорости сортировки. Этот доклад описывал сбалансированную двухпутевую поразрядную сортировку и сбалансированное двухпутевое слияние с использованием четырех устройств с магнитной проволокой или лентой, читающих или записывающих ”не менее 5000 импульсов в секунду”.

Джон Мочли выступил с лекцией о ”сортировке и слиянии” на специальной сессии по вычислениям, созывавшейся в школе Мура в 1946 г., и в записях его лекции содержится первое опубликованное обсуждение сортировки с помощью вычислительных машин [Theory and techniques for the design of electronic digital computers, ed. by G. W. Patterson, 3 (1946), 22.1–22.20]. Мочли начал свое выступление с интересного замечания: ”Требование, чтобы одна машина объединяла возможности вычислений и сортировки, может выглядеть как требование, чтобы один прибор использовался как ключ для консервов и как авторучка”. Затем он заметил, что машины, способные выполнять сложные математические процедуры, должны также иметь возможность сортировать и классифицировать данные; он показал, что сортировка может быть полезна даже в связи с численными расчетами. Он описал простые вставки и бинарные вставки, заметив, что в первом методе в среднем требуется около $N^2/4$ сравнений, в то время как в последнем их никогда не требуется более $N \log_2 N$. Однако бинарные вставки требуют весьма сложной структуры данных, и Мочли затем показал, что при двухпутевом слиянии достигается столь же малое число сравнений, но используется только последовательное прохождение списков. Последняя часть записей его лекций посвящена разбору методов поразрядной сортировки с частичными проходами, которые моделируют цифровую карточную сортировку на четырех лентах, затрачивая менее четырех проходов на цифру (ср. с п. 5.4.7).

Вскоре после этого Эккерт и Мочли организовали компанию, которая выпускала некоторые из самых ранних электронных вычислительных машин BINAC (для военных приложений) и UNIVAC (для коммерческих приложений). Вновь Бюро переписи США сыграло роль в этом развитии, приобретя первый UNIVAC. В это время вовсе не было ясно, что ЭВМ станут экономически выгодными: вычислительные машины могли сортировать быстрее, но они дороже стоили. Поэтому программисты UNIVAC под руководством Франсис Э. Гольбертон приложили значительные усилия к созданию программ внешней сортировки, работающих с высокой скоростью, и их первые программы повлияли также на разработку оборудования. По их оценкам, 100 миллионов записей по 10 слов могли быть отсортированы на UNIVAC за 9000 ч (т. е. 375 дней).

UNIVAC I, официально объявленная в июле 1951 г., имела внутреннюю память в 1000 12-литерных (72-битовых) слов. В ней предусматривалось чтение и запись на ленту блоков по 60 слов со скоростью 500 слов в секунду; чтение могло быть прямым или обратным, допускалось одновременное чтение /запись/ вычисления. В 1948 г. миссис Гольбертон придумала интересный способ выполнения двухпутевого слияния с полным совмещением чтения, записи и вычислений с использованием шести буферов ввода. Пусть для каждого вводного файла имеются один ”текущий буфер” и два ”вспомогательных буфера”; сливать можно таким образом, что всякий раз, когда приходит время вывести один блок, два текущих буфера ввода содержат вместе количество данных, равное одному блоку. Таким образом, за время формирования каждого выводного блока ровно один буфер ввода становится пустым, и мы можем устроить так, чтобы три или четыре вспомогательных буфера были заполнены всякий раз, как мы читаем в оставшийся буфер. Этот метод чуть быстрее метода прогнозирования алгоритма 5.4.6F, так как нет необходимости проверять результат одного ввода перед началом следующего. [Ср. с Collation Methods for the UNIVAC System (Eckert-Mauchly Computer Corp., 1950) vol. 1,2.]

Кульминационной точкой в этой работе стал генератор программ сортировки, который был первой крупной программой, разработанной для автоматического программирования. Пользователь указывал размер записи, позиции ключей (до пяти) в частичных полях каждой записи и ”концевые” ключи, отмечающие конец файла, и генератор сортировки порождал требуемую программу сортировки для файлов на одной бобине. Первым проходом этой программы была внутренняя сортировка блоков по 60 слов с использованием метода сравнения и подсчета (алгоритм 5.2С); затем выполнялся ряд сбалансированных двухпутевых проходов слияния с обратным чтением, исключаящих сцепление лент, как описано выше. [См. Master Generating Routine for 2-way Sorting (Eckert—Mauchly Div. of Remington Rand, 1952). Первый набросок этого доклада был озаглавлен ”Основная составляющая

программа двухпутевого слияния” (Master Prefabrication Routine for 2-way Collation)! См. также F. E. Holberton, Symposium on Automatic Programming (Office of Naval Research, 1954), 34–39.]

К 1952 г. многие методы внутренней сортировки прочно вошли в программистский фольклор, но теория была развита сравнительно слабо. Даниэль Голденберг [Time analyses of various methods of sorting data, Digital Computer Lab. memo M-1680 (Mass. Inst. of Tech., October 17, 1952)] запрограммировал для машины Whirlwind пять различных методов и провел анализ наилучшего и наихудшего случаев для каждой программы. Он нашел, что для сортировки сотни 15-битовых записей по 8-битовому ключу наилучшие по скорости результаты получаются в том случае, если используется таблица из 256 слов и каждая запись помещается в единственную соответствующую ее ключу позицию, а затем эта таблица сжимается. Однако этот метод имел очевидный недостаток, ибо он уничтожал запись, если последующая имела тот же ключ. Остальные четыре проанализированных метода были упорядочены следующим образом: прямое двухпутевое слияние лучше поразрядной сортировки с основанием 2, которая лучше простого выбора, который в свою очередь лучше метода пузырька.

Эти результаты получили дальнейшее развитие в диссертации Гарольда Х. Сьюворда в 1954 г. [Information sorting in the application of electronic digital computers to business operations, Digital Computer Lab. report R-232 (Mass. Inst. of Tech., May 24, 1954), 60 pp.]. Сьюворд высказал идеи распределяющего подсчета и выбора с замещением. Он показал, что первый отрезок случайной перестановки имеет среднюю длину $e - 1$, и анализировал наряду с внутренней сортировкой и внешнюю как на различных типах массовой памяти, так и на лентах.

Еще более достойная внимания диссертация — на этот раз докторская — была написана Говардом Б. Демуттом в 1956 г. [Electronic Data Sorting (Stanford University, October, 1956), 92 pp.]. Эта работа помогла заложить основы теории сложности вычислений. В ней рассматривались три абстрактные модели задачи сортировки: с использованием циклической памяти, линейной памяти и памяти с произвольным доступом; для каждой модели были разработаны оптимальные или почти оптимальные методы. (Ср. с упр. 5.3.4–62.) Хотя непосредственно из диссертации Демутта не вытекает никаких практических следствий, в ней содержатся важные идеи о том, как связать теорию с практикой.

Таким образом, история сортировки была тесно связана со многими исхоженными тропами в вычислениях: с первыми машинами для обработки данных, первыми запоминаемыми программами, первым программным обеспечением, первыми методами буферизации, первой работой по анализу алгоритмов и сложности вычислений.

Ни один из документов относительно ЭВМ, упомянутых до сих пор, не появлялся в ”открытой литературе”. Так уж случилось, что большая часть ранней истории вычислительных машин содержится в сравнительно недоступных докладах, поскольку относительно немногие лица были в то время связаны с ЭВМ. Наконец в 1955–1956 гг. литература о сортировке проникает в печать в виде трех больших обзорных статей. Первая статья была подготовлена Дж. К. Хоскеном [Proc. Eastern Joint. Computer Conference, 8 (1955), 39–55]. Он начинает с тонкого наблюдения: ”Чтобы снизить стоимость единицы результата, люди обычно укрупняют операции. Но при этих условиях стоимость единицы сортировки не уменьшается, а возрастает”. Хоскен описал все оборудование специального назначения, имевшееся в продаже, а также методы сортировки на ЭВМ. Его библиография из 54 пунктов основана большей частью на брошюрах фирм-изготовителей.

Подробная статья Э. Х. Фрэнда [Sorting on Electronic Computer Systems, *JACM*, 3 (1956), 134–168] явилась важной вехой в развитии сортировки. Хотя за прошедшее с 1956 г. время были разработаны многочисленные методы, эта статья все еще необычно современна во многих отношениях. Фрэнд дал тщательное описание весьма большого числа алгоритмов внутренней и внешней сортировки и обратил особое внимание на методы буферизации и характеристики магнитных лентопротяжных устройств. Он ввел некоторые новые методы (например, выбор из дерева, метод двуглавого змия и прогнозирование) и разработал некоторые математические свойства старых методов.

Третий обзор по сортировке, который появился в то время, был подготовлен Д. У. Дэвайсом [Proc. Inst. Elect. Engineers, 103B, supplement 1 (1956), 87–93]. В последующие годы еще несколько выдающихся обзоров было опубликовано Д. А. Беллом [Comp. J., 1 (1958), 71–77]; А. Ш. Дугласом [Comp. J., 2 (1959), 1–9]; Д. Д. Мак-Кракеном, Г. Вейссом и Ц. Ли [Programming Business Computers (New York: Wiley, 1959), chapter 15, 298–332]; А. Флоресом [*JACM*, 8 (1961) 41–80]; К. Э. Айверсоном [A programming language (New York: Wiley, 1962), chapter 6, 176–245]; К. К. Готлибом [*CACM*, 6 (1963), 194–201]; Т. Н. Хиббардом [*CACM*, 6 (1963), 206–213]; М. А. Готцем [Digital Computer User’s Handbook ed. by M. Klerer and G. A. Korn (New York, McGraw-Hill, 1967), chapter 1.10, 1.292–1.320]. В ноябре 1962 г. АСМ организовала симпозиум по сортировке (большая часть работ, представленных на этом симпозиуме, опубликована в мае 1963 г. в выпуске *CACM*). Они дают хорошее представление о состоянии этой области в то время. Обзор К. К. Готлиба о современных генераторах сортиров-

ки, обзор Т. Н. Хиббарда о внутренней сортировке с минимальной памятью и раннее исследование Дж. А. Хаббарда о сортировке файлов на дисках — наиболее заслуживающие внимания статьи в этом собрании.

За прошедший период были открыты новые методы сортировки: вычисление адреса (1956), слияние с вставкой (1959), обменная поразрядная сортировка (1959), каскадное слияние (1959), метод Шелла с убывающим шагом (1959), многофазное слияние (1960), вставки в дерево (1960), осциллирующая сортировка (1962), быстрая сортировка Хоара (1962), пирамидальная сортировка Уильямса (1964), обменная сортировка со слиянием Бэтчера (1964). История каждого отдельного алгоритма прослеживается в тех разделах настоящей главы где этот метод описывается. Конец 60-х годов нашего столетия ознаменовался интенсивным развитием соответствующей теории.

Полная библиография всех работ, изученных автором при написании этой главы, имеется в [Computing Reviews, 13 (1972), 283–289].

Упражнения

- [05] Подведите итог этой главе; сформулируйте обобщение теоремы 5.4.6А.
- [20] Скажите, основываясь на табл. 1, какой из методов сортировки списков для шестизначных ключей будет наилучшим для машины MIX.
- [47] (*Устойчивая сортировка с минимальной памятью.*) Говорят, что алгоритм сортировки требует *минимальной памяти*, если он использует для своих переменных только $O((\log N)^2)$ битов памяти сверх пространства, требуемого для размещения N записей. Алгоритм должен быть общим в том смысле, что должен работать при любых N , а не только при определенном значении N , если, конечно, предполагается, что при вызове алгоритма для сортировки ему обеспечивается достаточное количество памяти с произвольным доступом. Во многих из изученных нами алгоритмов сортировки это требование минимальной памяти нарушается; в частности, запрещено использование N полей связи. Быстрая сортировка (алгоритм 5.2.2Q) удовлетворяет требованию минимальной памяти, но ее время работы в наихудшем случае пропорционально N^2 . Пирамидальная сортировка (алгоритм 5.2.3H) является единственным среди изученных нами алгоритмов типа $O(N \log N)$, который использует минимальную память, хотя можно сформулировать и еще один подобный алгоритм, если использовать идею из упр. 5.2.4–18.

Самым быстрым общим алгоритмом, изученным нами, который *устойчиво* сортирует ключи, является метод слияния списков (алгоритм 5.2.4L), однако он использует не минимальную память. Фактически единственными устойчивыми алгоритмами сортировки с минимальной памятью, которые мы видели, были методы типа $O(N^2)$ (простые вставки, метод пузырька и варианты простого выбора).

Существует ли устойчивый алгоритм сортировки с минимальной памятью, требующий менее $O(N^2)$ единиц времени в наихудшем случае или в среднем?

Я достиг своей цели, если рассортировал и привел в логический порядок хотя бы ядро того огромного материала о сортировке, который появился за последние несколько лет.

Дж. К. Хоскен (1955)

6. Поиск

Поищем запись.

Эл Смит (1928)¹

Эту главу можно было назвать иначе: или претенциозно — «Хранение и выборка информации», или просто — «Поиск по таблицам». Нас будет интересовать процесс накопления информации в памяти вычислительной машины с последующим возможно более быстрым извлечением этой информации. Иногда мы сталкиваемся со столь большим объемом данных, что реально использовать их все не можем. В этом случае самым мудрым было бы забыть и разрушить большую их часть; но нередко бывает важно сохранить и так организовать имеющиеся факты, чтобы обеспечить наибыстрейшую их выборку.

Эта глава посвящена в основном изучению очень простой поисковой задачи: как находить данные, хранящиеся с определенной идентификацией. Например, в вычислительной задаче нам может

¹ Смит, Альфред Эммануэль (1873–1944) — американский политический деятель. — Прим. перев.

понадобится найти $f(x)$, имея x и таблицу значений функции f ; в лингвистической может интересоваться английский эквивалент данного русского слова.

Вообще будем предполагать, что хранится множество из N записей и необходимо определить положение соответствующей записи. Как и в случае сортировки, предположим, что каждая запись содержит специальное поле, которое называется *ключом*, возможно, потому, что многие люди ежедневно тратят массу времени на поиск своих ключей. Мы обычно требуем, чтобы N ключей были различны, так что каждый ключ однозначно определяет свою запись. Совокупность всех записей называется *таблицей* или *файлом*, причем под таблицей, как правило, подразумевается небольшой файл, а файлом обычно называют большую таблицу. Большой файл, или группа файлов, часто называется *базой данных*.

В алгоритмах поиска присутствует так называемый *аргумент поиска* K , и задача состоит в отыскании записи, имеющей K своим ключом. Существуют две возможности окончания поиска: либо поиск оказался *удачным*, т. е. позволил определить положение соответствующей записи, содержащей K , либо он оказался *неудачным*, т. е. показал, что аргумент K не может быть найден ни в одной из записей. После неудачного поиска иногда желательно вставить в таблицу новую запись, содержащую K ; алгоритм, который делает это, называется алгоритмом "поиска с вставкой". Некоторые технические устройства, известные как "ассоциативная память", решают проблему поиска автоматически аналогично тому, как это делает человеческий мозг; мы же будем изучать методы поиска на обычной универсальной вычислительной машине.

Хотя целью поиска является информация, которая содержится в записи, ассоциированной с ключом K , в алгоритмах этой главы обычно игнорируется все, кроме собственно ключей. В самом деле, если положение K определено, соответствующие данные можно найти. Например, если K встретился в ячейке $TABLE + i$, ассоциированные данные (или указатель на них) могут находиться по адресу $TABLE + i + 1$, или $DATA + i$ и т. д. Таким образом, подробности, касающиеся того, что нужно делать, когда найден ключ K , можно спокойно опустить.

Во многих программах поиск требует наибольших временных затрат, так что замена плохого метода поиска на хороший часто ведет к существенному увеличению скорости работы. Действительно, нередко удается так организовать данные или структуру данных, что поиск полностью исключается, т. е. мы всегда знаем заранее, где находится нужная нам информация. Связанная память является общепринятым методом достижения этого; например, в списке с двумя связями нет необходимости искать элемент, следующий за данным или предшествующий ему. Другой способ избежать поиска открывается перед нами, если предоставлена свобода выбора ключей. Сделаем их числами $\{1, 2, \dots, N\}$, и тогда запись, содержащая K , может быть просто помещена в ячейку $TABLE + K$. Оба эти способа использовались для устранения поиска из алгоритма топологической сортировки, обсуждавшегося в п. 2.2.3. Тем не менее во многих случаях поиск необходим (например, когда объектами топологической сортировки являются символические имена, а не числа), так что весьма важно иметь эффективные алгоритмы поиска.

Методы поиска можно классифицировать несколькими способами. Мы могли бы разделить их на внутренний и внешний поиск в соответствии с разделением алгоритмов сортировки в гл. 5 на внутреннюю и внешнюю сортировку. Или мы могли бы различать статический и динамический методы поиска, где "статический" означает, что содержимое таблицы остается неизменным (так что важно минимизировать время поиска, пренебрегая затратами на перестройку таблицы), а "динамический" означает, что таблица является объектом частых вставок (и, может быть, удалений). Третья возможная схема классифицирует методы поиска в соответствии с тем, основаны ли они на сравнении ключей или на цифровых свойствах ключей, аналогично тому, как различаются сортировка с помощью сравнения и сортировка с помощью распределения. Наконец, мы могли бы разделить методы поиска на методы, использующие истинные ключи, и на методы, работающие с преобразованными ключами.

Организация данной главы есть, в сущности, комбинация двух последних способов классификации. В § 6.1 рассматриваются методы последовательного поиска "в лоб", а в § 6.2 обсуждаются улучшения, которые можно получить на основе сравнений между ключами с использованием алфавитного или числового порядка для управления решениями. В § 6.3 рассматривается цифровой поиск, а в § 6.4 обсуждается важный класс методов, называемых хешированием и основанных на арифметических преобразованиях истинных ключей. В каждом из этих параграфов рассматривается как внутренний, так и внешний поиск, и для статического и для динамического случаев; в каждом параграфе отмечаются сравнительные достоинства и недостатки различных алгоритмов.

Между поиском и сортировкой существует определенная взаимосвязь. Например, рассмотрим следующую задачу:

Даны два множества чисел:

$$A = \{a_1, a_2, \dots, a_m\} \text{ и } B = \{b_1, b_2, \dots, b_n\};$$

определить, является ли A подмножеством B , т. е. $A \subseteq B$.

Напрашиваются три решения, а именно:

- 1) Сравнивать каждое a_i последовательно со всеми b_j до установления совпадения.
- 2) Свести все b_j в таблицу, затем искать каждое a_i по таблице.
- 3) Упорядочить A и B , затем совершить один последовательный проход по обоим файлам, проверяя соответствующие условия.

Каждое из этих решений имеет свои привлекательные стороны для различных диапазонов значений m и n . Для решения 1 потребуется приблизительно $c_1 mn$ единиц времени, где c_1 — некоторая константа, а решение 3 займет около $c_2(m \log_2 m + n \log_2 n)$ единиц, где c_2 — некоторая (большая) константа. При подходящем методе хеширования решение 2 потребует примерно $c_3 m + c_4 n$ единиц времени, где c_3 и c_4 — некоторые (еще большие) константы. Следовательно, решение 1 хорошо при очень малых m и n , а при возрастании m и n лучшим будет решение 3. Затем, пока n не достигнет размеров внутренней памяти, более предпочтительно решение 2; после этого обычно решение 3 снова становится лучше, пока n не сделается еще гораздо большим. Значит, мы имеем ситуацию, где сортировка иногда хорошо заменяет поиск, а поиск — сортировку.

Зачастую сложные задачи поиска можно свести к более простым случаям, рассматриваемым здесь. Например, предположим, что в роли ключей выступают слова, которые могли быть слегка искажены; мы хотели бы найти правильную запись, несмотря на эту ошибку. Если сделать две копии файла, в одной из которых ключи расположены в обычном алфавитном порядке, а в другой они упорядочены справа налево (как если бы слова были прочитаны наоборот), искаженный аргумент поиска в большинстве случаев совпадает до половины своей длины или дальше с записью одного из этих двух файлов. Методы поиска, содержащиеся в § 6.2 и 6.3, можно, следовательно, приспособить для нахождения ключа, который был, вероятно, искажен.

Похожие задачи привлекли к себе заметное внимание в связи с созданием систем предварительного заказа авиабилетов и в связи с другими приложениями, когда существует значительная вероятность искажения имени человека из-за неразборчивого почерка или плохой слышимости. Нужно было найти преобразование аргумента в некий код, собирающее вместе все варианты данного имени. Метод "Soundex", описываемый ниже в виде, в котором он применяется сейчас, был первоначально развит Маргарет К. Оуделл и Робертом К. Расселом [см. U. S. Patents 1261167 (1918), 1435663 (1922)]; он нашел широкое применение для кодирования фамилий.

- 1) Оставить первую букву; все буквы a, e, h, i, o, u, w, y, стоящие на других местах, вычеркнуть.
- 2) Оставшимся буквам (кроме первой) присвоить следующие значения:

$$\begin{array}{ll} b, f, p, v \rightarrow 1; & l \rightarrow 4; \\ c, g, j, k, q, s, x, z \rightarrow 2; & m, n \rightarrow 5; \\ d, t \rightarrow 3; & r \rightarrow 0. \end{array}$$

- 3) Если в исходном имени (перед шагом 1) рядом стояли несколько букв с одинаковыми кодами, пренебречь всеми, кроме первой из этой группы.
- 4) Дописывая в случае надобности нули или опуская лишние цифры, преобразовать полученное выражение в форму "буква, цифра, цифра, цифра".

Например, фамилии Euler, Gauss, Hilbert, Knuth, Lloyd и Lukasiewicz имеют коды соответственно E460, G200, H416, K530, L300, L222. Разумеется, такая система собирает вместе не только родственные, но и достаточно различные имена. Приведенные выше шесть кодов могли быть получены из фамилий Ellery, Ghosh, Heilbronn, Kant, Ladd и Lissajous. С другой стороны, такие родственные имена, как Rogers и Rodgers, Sinclair и St. Clair или Tchebysheff и Chebyshev, имеют разную кодировку. Но, вообще говоря, система "Soundex" намного увеличивает вероятность обнаружить имя под одной из его масок. [Для дальнейшего ознакомления, см. C. P. Bourne, D. F. Ford, *JACM*, 8 (1961), 538–552; Leon Davidson, *CACM*, 5 (1962), 169–171; Federal Population Censuses, 1790–1890 (Washington, D. C.: National Archives, 1971), 90.]

Когда мы используем системы типа "Soundex", нет необходимости предполагать, что все ключи различны; можно составить списки из записей с совпадающими кодами, рассматривая каждый список как объект поиска.

При использовании больших баз данных выборка информации намного усложняется, так как часто желательно рассматривать различные поля каждой записи как потенциальные ключи. Здесь

важно уметь находить записи по неполной ключевой информации. Например, имея большой файл актеров, режиссер мог бы пожелать найти всех незанятых актрис в возрасте 25–30 лет, хорошо танцующих и говорящих с французским акцентом; у спортивного журналиста может возникнуть желание подсчитать с помощью файла бейсбольной статистики общее число очков, набранных подающими-левшами команды "Красноногие из Цинцинатти" в течение седьмых периодов вечерних игр за 1964 г. Имея большой файл данных о чем-либо, люди любят задавать вопросы произвольной сложности. На самом деле мы могли бы рассматривать полную библиотеку как базу данных, в которой некто желает найти все публикации о выборке информации. Введение в методы *поиска по многим признакам* помещено в § 6.5.

Прежде чем переходить к детальному изучению поиска, полезно рассмотреть историю данного вопроса. В докомпьютерный период было составлено множество томов логарифмических, тригонометрических и других таблиц, так что математические вычисления могли быть заменены поиском. Потом эти таблицы были перенесены на перфокарты и использовались для научных задач посредством распознающих, сортировальных и дублирующих перфораторных машин. Однако после появления ЭВМ с запоминаемой программой стало очевидно, что дешевле каждый раз вычислять $\log x$ и $\cos x$, нежели искать ответ по таблице.

Несмотря на то, что задача сортировки привлекла пристальное внимание уже на заре развития ЭВМ, для разработки алгоритмов поиска было сделано сравнительно мало. Располагая небольшой внутренней памятью и только последовательными устройствами типа лент для хранения больших файлов, организовать поиск было либо совершенно тривиально, либо почти невозможно.

Но развитие все большей и большей памяти со случайным доступом в течение 50-х годов привело к пониманию того, что поиск как таковой является интересной задачей. После периода жалоб на ограниченные ресурсы пространства в ранних машинах программисты вдруг столкнулись с таким объемом памяти, который они не умели эффективно использовать.

Первые обзоры задач поиска были опубликованы А. И. Думи [*Computers & Automation*, 5, 12 (December 1956), 6–9], У. У. Петерсоном [*IBM J. Research & Development*, 1 (1957), 130–146], Э. Д. Буттом [*Information and Control*, 1 (1958), 159–164], А. Ш. Дугласом [*Comp. J.*, 2 (1959), 1–9]. Более подробный обзор сделан позднее К. Э. Айверсоном [*A Programming Language* (New York: Wiley, (1962)), 133–158] и В. Буххольцем [*IBM Systems J.*, 2 (1963), 86–111].

В начале 60-х годов было разработано несколько новых алгоритмов поиска, основанных на использовании древовидных структур; с ними мы познакомимся ниже. И в наше время ведутся активные исследования по проблемам поиска.

6.1. Последовательный поиск

"Начни с начала и продвигайся, пока не найдешь нужный ключ; тогда остановись". Такая последовательная процедура является очевидным способом поиска; удобно начать наши рассуждения с нее, так как на ней основаны многие более сложные алгоритмы. Несмотря на свою простоту, последовательный поиск содержит ряд очень интересных идей.

Сформулируем алгоритм более точно.

Алгоритм S. (Последовательный поиск.) Имеется таблица записей R_1, R_2, \dots, R_N , снабженных соответственно ключами K_1, K_2, \dots, K_N . Алгоритм предназначен для поиска записи с данным ключом K . Предполагается, что $N \geq 1$.

S1 [Начальная установка.] Установить $i \leftarrow 1$.

S2 [Сравнение.] Если $K = K_i$, алгоритм оканчивается удачно.

S3 [Продвижение.] Увеличить i на 1.

S4 [Конец файла?] Если $i \leq N$, то вернуться к шагу S2. В противном случае алгоритм оканчивается неудачно. ■

Заметим, что у этого алгоритма может быть два разных исхода: *удачный* (когда найдено положение нужного ключа) и

Picture: Последовательный поиск

неудачный (когда, установлено, что искомого аргумента нет в таблице). Это справедливо для большинства алгоритмов данной главы.

Реализация в виде программы для машины MIX очевидна.

Программа S. (Последовательный поиск.) Предположим, что K_i хранится по адресу $KEY + i$, а оставшаяся часть записи R_i — по адресу $INFO + i$. Приводимая ниже программа использует $rA \equiv K$, $rI1 \equiv i - N$.

```

START   LDA    K           1   S1. Начальная установка.
        ENT1   1           1    $i \leftarrow 1$ .
2H      CMPA  KEY+N, 1     C   S2. Сравнение.
        JE    SUCCESS     C   Выход, если  $K = K_i$ .
        INC1  1           C - S S3. Продвижение.
        J1NP  2B         C - S S4. Конец файла?
FAILURE EQU   *           1 - S Выход, если нет в таблице.

```

По адресу SUCCESS расположена команда "LDA INFO+N, 1"; она помещает нужную информацию в rA. ■

Анализ данной программы не представляет труда; видно, что время работы алгоритма S зависит от двух параметров:

$$C = (\text{количество сравнений ключей});$$

$$S = 1 \text{ при удаче и } 0 \text{ при неудаче.} \quad (1)$$

Программа S требует $5C - 2S + 3$ единиц времени. Если мы нашли $K = K_i$, то $C = i$, $S = 1$; значит, полное время равно $(5i + 1)u$. Если же поиск оказался неудачным, то $C = N$, $S = 0$, а работала программа ровно $(5N + 3)u$. Если все ключи поступают на вход с равной вероятностью, то среднее значение C при удачном поиске составляет

$$\frac{1 + 2 + \dots + N}{N} = \frac{N + 1}{2}; \quad (2)$$

среднеквадратичное отклонение C , разумеется, довольно большое—примерно $0.289N$ (см. упр. 1).

Приведенный алгоритм, несомненно, знаком всем программистам. Но мало кто знает, что этот способ реализации последовательного поиска не всегда самый лучший! Очевидное изменение убыстряет работу алгоритма (если только записей не слишком мало):

Алгоритм Q. (*Быстрый последовательный поиск*). В отличие от алгоритма S здесь еще предполагается, что в конце файла стоит фиктивная запись R_{N+1} .

Q1 [Начальная установка.] Установить $i \leftarrow 1$ и $K_{N+1} \leftarrow K$.

Q2 [Сравнение.] Если $K = K_i$, то перейти на Q4.

Q3 [Продвижение.] Увеличить i на 1 и вернуться к шагу Q2.

Q4 [Конец файла?] Если $i \leq N$, алгоритм оканчивается удачно; в противном случае—неудачно ($i = N + 1$). ■

Программа Q. (*Быстрый последовательный поиск.*) Значения регистров: rA $\equiv K$, rI1 $\equiv i - N$.

```

BEGIN   LDA    K           1   Q1. Начальная установка
        STA   KEY+N+1     1    $K_{N+1} \leftarrow K$ .
        ENT1  -N          1    $i \leftarrow 0$ .
        INC1  1           C + 1 - S Q3. Продвижение.
        CMPA  KEY+N, 1     C + l - S Q2. Сравнение.
        JNE  *-2          C + 1 - S На Q3, если  $K_i \neq K$ .
        J1NP  SUCCESS     1   Q4. Конец файла?
FAILURE EQU   *           1 - S Выход, если нет в таблице.

```

Используя параметры C и S , введенные при анализе программы S, можно заключить, что время работы программы уменьшилось до $(4C - 4S + 10)u$; это дает улучшение при $C \geq 5$ (для удачного поиска) и при $N \geq 8$ (для неудачного поиска). При переходе от алгоритма S к алгоритму Q использован важный ускоряющий принцип: если во внутреннем цикле программы проверяются два или более условия, нужно постараться оставить там только одно сравнение. Существует способ сделать программу Q еще быстрее.

Программа Q'. (*Сверхбыстрый последовательный поиск.*) Значения регистров: rA = K , rI1 $\equiv i - N$.

```

BEGIN   LDA    K           1   Q1. Начальная установка.

```

	STA	KEY + N + 1	1	$K_{N+1} \leftarrow K.$
	ENT1	-1-N	1	$i \leftarrow -1.$
3H	INC1	2	$\lfloor (C - S + 2)/2 \rfloor$	Q3. Продвижение. (Двойное.)
	CMPA	KEY+N, 1	$\lfloor (C - S + 2)/2 \rfloor$	Q2. Сравнение.
	JE	4F	$\lfloor (C - S + 2)/2 \rfloor$	На Q4, если $K = K_i.$
	CMPA	KEY+N+1, 1	$\lfloor (C - S + 1)/2 \rfloor$	Q2. Сравнение. (Следующее.)
	JNE	3B	$\lfloor (C - S + 1)/2 \rfloor$	На Q3, если $K \neq K_{i+1}.$
	INC1	1	$(C - 5) \bmod 2$	Продвинуть $i.$
4H	J1NP	SUCCESS	1	Q4. Конец файла?
FAILURE	EQU	*	$1 - S$	Выход, если нет в таблице.

■

Инструкции внутреннего цикла выписаны дважды; это исключает примерно половину операторов " $i \leftarrow i + 1$ ". Таким образом, время выполнения программы уменьшилось до

$$3.5C - 3.5S + 10 \frac{(C - S) \bmod 2}{2}$$

единиц. При поиске по большим таблицам программа Q' на 30% быстрее программы S; подобным образом могут быть улучшены многие существующие программы. Если известно, что ключи расположены в возрастающем порядке, полезно несколько изменить алгоритм.

Алгоритм Т. (Последовательный поиск в упорядоченной таблице.) Имеется таблица записей R_1, R_2, \dots, R_N , причем ключи удовлетворяют неравенствам $K_1 < K_2 < \dots < K_N$. Алгоритм предназначен для поиска данного ключа K . В целях удобства и увеличения скорости работы предполагается, что в конце таблицы расположена фиктивная запись R_{N+1} , ключ которой $K_{N+1} = \infty > K$.

T1 [Начальная установка.] Установить $i \leftarrow 1$.

T2 [Сравнение.] Если $K \leq K_i$, то перейти на T4.

T3 [Продвижение.] Увеличить i на 1 и вернуться к шагу T2.

T4 [Равенство?] Если $K = K_i$, то алгоритм оканчивается удачно. В противном случае—неудачно, нужной записи в таблице нет. ■

Если величина K с равной вероятностью принимает все возможные значения, то в случае удачного поиска алгоритм Т, по существу, не лучше алгоритма Q. Однако отсутствие нужной записи алгоритм Т позволяет обнаружить примерно в два раза быстрее.

Приведенные выше алгоритмы в целях удобства использовали индексные обозначения для элементов таблицы; аналогичные процедуры применимы и к таблицам в связанном представлении, так как в них данные тоже расположены последовательно. (См. упр. 2, 3 и 4.)

Частота обращений. До сих пор предполагалось, что с равной вероятностью может потребоваться поиск любого аргумента, однако часто такое предположение не правомерно; вообще говоря, ключ K_j будет разыскиваться с вероятностью p_j , где $p_1 + p_2 + \dots + p_N = 1$. Время удачного поиска при больших N пропорционально числу сравнений C , среднее значение которого равно

$$\bar{C}_N = p_1 + 2p_2 + \dots + Np_N. \quad (3)$$

Пусть есть возможность помещать записи в таблицу в любом порядке; тогда величина \bar{C}_N минимальна при

$$p_1 \geq p_2 \geq \dots \geq p_N, \quad (4)$$

т. е. когда наиболее часто используемые записи расположены в начале таблицы.

Посмотрим, что дает нам такое "оптимальное" расположение при различных распределениях вероятностей. Если $p_1 = p_2 = \dots = p_N = 1/N$, то формула (3) сводится к $\bar{C}_N = (N + 1)/2$, что уже было получено нами в (2). Предположим теперь, что

$$p_1 = \frac{1}{2}, p_2 = \frac{1}{4}, \dots, p_{N-1} = \frac{1}{2^{N-1}}, p_N = \frac{1}{2^{N-1}} \quad (5)$$

Согласно упр. 7, $\bar{C}_N = 2 - 2^{1-N}$; среднее число сравнений меньше двух, если записи расположены в надлежащем порядке. Другим напрашивающимся распределением является

$$p_1 = N_c, p_2 = (N - 1)c, \dots, p_N = c,$$

где

$$c = 2/N(N + 1). \quad (6)$$

Это "клиновидное" распределение дает более привычный результат

$$\bar{C}_N = c \sum_{1 \leq k \leq N} k \cdot (N + 1 - k) = \frac{N + 2}{2}; \quad (7)$$

оптимальное расположение экономит около трети поискового времени, требующегося для записей в случайном порядке.

Разумеется, распределения (5) и (6) довольно искусственны, и их нельзя считать хорошим приближением к действительности. Более типичную картину дает "закон Зипфа":

$$p_1 = c/1, p_2 = c/2, \dots, p_N = c/N, \quad \text{где } c = 1/H_N. \quad (8)$$

Это распределение получило известность благодаря Дж. К Зипфу, который заметил, что n -е наиболее употребительное в тексте на естественном языке слово встречается с частотой, приблизительно обратно пропорциональной n . [The Psycho-Biology of Language (Boston, Mass.: Houghton Miffling, 1935); Human Behavior and the Principle of Least Effort (Reading, Mass.: Addison-Wesley, 1949).] Аналогичное явление обнаружено им в таблицах переписи; там столичные районы расположены в порядке убывания населения. В случае, если частота ключей в таблице подчиняется закону Зипфа, имеем

$$\bar{C}_N = N/H_N; \quad (9)$$

поиск по такому файлу примерно в $\frac{1}{2} \ln N$ раз быстрее, чем по неупорядоченному. [Ср. с A. D. Booth et al. Mechanical Resolution of Linguistic Problems (New York: Academic Press, 1958), 79.]

Другое распределение, близкое к реальному, дает правило "80–20", которое часто встречается в коммерческих приложениях [ср. с W. P. Heising, IBM Systems J., 2 (1963), 114–115]. Это правило гласит, что 80% работы ведется над наиболее активной частью файла величиной 20%; оно применимо и к этим 20%, так что 64% работы ведется с наиболее активными 4% файла, и т. д. Иными словами,

$$\frac{p_1 + p_2 + \dots + p_{.20n}}{p_1 + p_2 + p_3 + \dots + p_n} \approx 0.80 \quad \text{для всех } n. \quad (10)$$

Вот одно из распределений, точно удовлетворяющих приведенному правилу при n , кратных 5:

$$p_1 = c, p_2 = (2^\theta - 1)c, p_3 = (3^\theta - 2^\theta)c, \dots, p_N = (N^\theta - (N - 1)^\theta)c, \quad (11)$$

где

$$c = 1/N^\theta; \theta = \frac{\log 0.80}{\log 0.20} = 0.1386. \quad (12)$$

Действительно, $p_1 + p_2 + \dots + p_n = cn^\theta$ при любом n . С вероятностями (11) не так просто работать; имеем, однако, $n^\theta - (n - 1)^\theta = \theta n^{\theta-1}(1 + O(1/n))$, т.е. существует более простое распределение, приближенно удовлетворяющее правилу "80–20":

$$p_1 = c/1^{1-\theta}, p_2 = c/2^{1-\theta}, \dots, p_N = c/N^{1-\theta}, \quad \text{где } c = 1/H_N^{1-\theta}. \quad (13)$$

Здесь, как и раньше, $\theta = \log 0.80 / \log 0.20$, а $H_N^{(s)}$ есть N -е гармоническое число порядка s , т. е. $1^{-s} + 2^{-s} + \dots + N^{-s}$. Заметим, что это распределение очень напоминает распределение Зипфа (8); когда θ изменяется от 1 до 0, вероятности меняются от равномерно распределенных к зипфовским. (В самом деле, Зипф нашел, что $\theta \approx \frac{1}{2}$ в распределении личных доходов.) Применяя (3) к (13), получаем для правила "80–20" среднее число сравнений

$$\bar{C}_N = H_N^{(-\theta)} / H_N^{(1-\theta)} = \frac{\theta N}{\theta + 1} + O(N^{(1-\theta)}) \approx 0.122N \quad (14)$$

(см. упр. 8).

Ю. С. Шварц, изучавший частоты появления слов [см. интересный график на стр. 422 в JАСМ, 10 (1963)], предложил более подходящее выражение для закона Зипфа:

$$p_1 = c/1^{1+\theta}, p_2 = c/2^{1+\theta}, \dots, p_N = c/N^{1+\theta}, \quad \text{где } c = 1/H_N^{1+\theta}, \quad (15)$$

при малых *положительных* Q . [По сравнению с (13) здесь изменен знак θ .] В этом случае

$$\bar{C}_N = H_N^\theta / H_N^{(1+\theta)} = N^{1-\theta} / (1-\theta)\zeta(1+\theta) + O(N^{1-2\theta}), \quad (16)$$

что значительно меньше, чем (9) при $N \rightarrow \infty$.

”Самоорганизующийся” файл. Предыдущие вычисления хороши, но в большинстве случаев распределение вероятностей нам не известно. Мы могли бы в каждой записи завести счетчик числа обращений, чтобы на основании показаний счетчиков переупорядочить записи. Выведенные формулы показывают, что такая процедура может привести к заметной экономии. Но не всегда желательно отводить много памяти под счетчики, так как ее можно использовать более рационально (например, применяя другие методы поиска, излагаемые ниже в данной главе).

Простая схема, происхождение которой не известно, используется уже многие годы. Она позволяет довольно хорошо упорядочить записи без вспомогательных полей для счетчиков: когда мы находим нужную запись, мы помещаем ее в начало таблицы. Подобный метод легко реализовать, когда таблица представлена в виде связанного линейного списка: ведь часто нам бывает нужно значительно изменить найденную запись.

Идея ”самоорганизующегося” метода состоит в том, что часто используемые элементы будут расположены довольно близко к началу таблицы. Пусть N ключей разыскиваются с вероятностями $\{p_1, p_2, \dots, p_N\}$ соответственно, причем каждый поиск совершается абсолютно *независимо* от предыдущих. Можно показать, что среднее число сравнений при нахождении записи в таком самоорганизующемся файле стремится к предельному значению

$$\bar{C}_N = 1 + 2 \sum_{1 \leq i < j \leq N} \frac{p_i p_j}{p_i + p_j} = \frac{1}{2} + \sum_{i,j} \frac{p_i p_j}{p_i + p_j}. \quad (17)$$

(См. упр. 11.) Например, если $p_i = 1/N$ при $1 \leq i \leq N$, самоорганизующаяся таблица всегда находится в случайном порядке, а (17) сводится к знакомому выражению $(N+1)/2$, полученному ранее.

Посмотрим, как самоорганизующаяся процедура работает при распределении вероятностей по закону Зипфа (8). Имеем

$$\begin{aligned} \bar{C}_N &= \frac{1}{2} + \sum_{1 \leq i, j \leq N} \frac{(c/i)(c/j)}{c/i + c/j} = \frac{1}{2} + c \sum_{1 \leq i, j \leq N} \frac{1}{i+j} = \\ &= \frac{1}{2} + c \sum_{1 \leq i \leq N} (H_{N+i} - H_i) = \frac{1}{2} + c \sum_{1 \leq i \leq 2N} H_i - 2c \sum_{1 \leq i \leq N} H_i = \\ &= \frac{1}{2} + c((2N+1)H_{2N} - 2N - 2(N+1)H_N + 2N) = \\ &= \frac{1}{2} + c(N \ln 4 - \ln N + O(1)) \approx \\ &\approx 2N / \log_2 N. \end{aligned}$$

(см. формулы (1.2.7–8,3)). Это гораздо лучше, чем у N при достаточно больших N , и лишь в $\ln 4 \approx 1.386$ раз хуже, чем число сравнений при оптимальном расположении записей (ср. с (9)).

Эксперименты, проводившиеся с таблицами символов в компиляторах, показали, что самоорганизующийся метод работает даже лучше, чем предсказывает (18), так как удачные поиски не независимы (небольшие группы ключей часто появляются вместе).

Схему, подобную самоорганизующейся, изучили Г. Шай и Ф. В. Дауэр [*SIAM J. Appl. Math.*, **15** (1967), 874–888.]

Поиск на ленте среди записей различной длины. Рассмотрим теперь нашу задачу в ином ракурсе. Пусть таблица, по которой производится поиск, хранится на ленте и записи имеют различные длины. Лента с системной библиотекой в старых операционных системах служит примером такого файла. Стандартные программы системы, такие, как компиляторы, компоновщики, загрузчики, генераторы отчетов и т. п., являлись ”записями” на ленте, а большинство пользовательских работ должно было начинаться с поиска нужной программы. Такая постановка задачи делает неприменимым предыдущий анализ алгоритма S: теперь шаг S3 выполняется за различные промежутки времени. Значит, нас будет интересовать не только число сравнений.

Обозначим через L_i длину записи R_i , а через p_i , вероятность того, что эта запись будет отыскиваться. Время поиска теперь примерно пропорционально величине

$$p_1 L_1 + p_2 (L_1 + L_2) + \dots + p_N (L_1 + L_2 + \dots + L_N). \quad (19)$$

При $L_1 = L_2 = \dots = L_N = 1$ это, очевидно, сводится к изученному случаю (3).

Кажется логичным поместить наиболее нужные записи в начало ленты, но здесь здравый смысл подводит нас. Действительно, пусть на ленте записаны ровно две программы— A и B . Программа A требуется в два раза чаще B , но длиннее B в четыре раза, т. е. $N = 2$, $p_A = \frac{2}{3}$, $L_A = 4$, $p_B = \frac{1}{3}$, $L_B = 1$. Если мы, следуя "логике", поставим A на первое место, то среднее время поиска составит $\frac{2}{3} \cdot 4 + \frac{1}{3} \cdot 5 = \frac{13}{3}$; но если поступить "нелогично", расположив в начале B , то получится $\frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 5 = \frac{11}{3}$. Следующая теорема позволяет определить оптимальное расположение библиотечных программ на ленте.

Теорема S. Пусть L_i и p_i , определены, как и раньше. Порядок записей на ленте оптимален тогда и только тогда, когда

$$p_1/L_1 \geq p_2/L_2 \geq \dots \geq p_N/L_N. \quad (20)$$

Иными словами, минимум выражения

$$p_{a_1}L_{a_1} + p_{a_2}(L_{a_1} + L_{a_2}) + \dots + p_{a_N}(L_{a_1} + \dots + L_{a_N})$$

по всем перестановкам $a_1 a_2 \dots a_N$ чисел $\{1, 2, \dots, N\}$ равен (19) тогда и только тогда, когда выполняется (20).

Доказательство. Предположим, что R_i и R_{i+1} поменялись местами; величина (19), ранее равная

$$\dots + p_i(L_1 + \dots + L_{i-1} + L_i) + p_{i+1}(L_1 + \dots + L_{i+1}) + \dots,$$

теперь заменится на

$$\dots + p_{i+1}(L_1 + \dots + L_{i-1} + L_{i+1}) + p_i(L_1 + \dots + L_i) + \dots.$$

Изменение равно $p_iL_{i+1} - p_{i+1}L_i$. Так как расположение (19) оптимально, то $p_iL_{i+1} - p_{i+1}L_i \geq 0$. Значит, $p_i/L_i \geq p_{i+1}/L_{i+1}$, т. е. (20) выполняется.

Докажем теперь достаточность условия (20). Разумеется, "локальная" оптимальность расположения (19) видна сразу: если мы поменяем местами две записи, время поиска изменится на $p_iL_{i+1} - p_{i+1}L_i \geq 0$. Однако "глобальная" оптимальность требует обоснования. Мы рассмотрим два доказательства: одно из них использует дискретную математику, а другое предполагает некоторую математическую изворотливость.

Доказательство. 1. Пусть (20) выполняется. Мы знаем, что любую перестановку записей можно "отсортировать", т. е. привести к расположению R_1, R_2, \dots, R_N , последовательно меняя местами лишь соседние элементы. Каждое такое изменение переводит $\dots R_j R_i \dots$ в $\dots R_i R_j \dots$ для некоторых $i < j$, т. е. уменьшает время поиска на неотрицательную величину $p_iL_j - p_jL_i$. Значит, порядок $R_1 R_2 \dots R_N$ оптимален.

Доказательство. 2. Заменяем вероятности p_i на

$$p_i(\varepsilon) = p_i + \varepsilon^i - (\varepsilon^1 + \varepsilon^2 + \dots + \varepsilon^N)/N,$$

где ε —малая положительная величина. Равенство $x_1p_1(\varepsilon) + \dots + x_Np_N(\varepsilon) = y_1p_1(\varepsilon) + \dots + y_Np_N(\varepsilon)$ выполняется для достаточно малых ε лишь при $x_1 = y_1, \dots, x_N = y_N$; в частности, в (20) станут невозможными равенства $\frac{p_i}{L_i} = \frac{p_j}{L_j}$. Рассмотрим $N!$ перестановок записей. Среди них есть по крайней мере одна оптимальная; в силу первой части теоремы она удовлетворяет (20), но в силу отсутствия равенств условию (20) удовлетворяет лишь одна перестановка. Таким образом, (20) однозначно определяет оптимальное расположение записей в таблице для вероятностей $p_i(\varepsilon)$ при достаточно малых ε . По непрерывности этот же порядок оптимален и при $\varepsilon = 0$. (Такой метод доказательств часто используется в комбинаторной оптимизации.) ■

Теорема S принадлежит У. Э. Смиты [Naval Research Logistics Quarterly, 3 (1956), 59–66]. Упражнения содержат дополнительные результаты по оптимальной организации таблиц.

Уплотнение файлов. Последовательный поиск на ленте и других внешних запоминающих устройствах протекает быстрее; если данные занимают меньше места, поэтому полезно рассмотреть несколько различных способов представления данных в таблице. Не всегда нужно запоминать ключи в явном виде.

Пусть, например, требуется таблица простых чисел до миллиона для разложения на множители 12-значных чисел. (Ср. с п. 4.5.4.) Таких чисел имеется 78498; используя 20 битов для каждого из них,

мы получим файл длины 1 569 960 битов. Это явное расточительство, так как можно иметь миллион-битовую таблицу, в которой разряды, соответствующие простым числам, равны 1. Поскольку все простые числа (кроме двойки) нечетны, достаточно иметь таблицу в 500000 битов.

Другим способом уменьшения длины файла является хранение не самих простых чисел, а интервалов между ними. В соответствии с табл. 1 величина $(p_{k+1} - p_k)/2$ меньше 64 для всех простых чисел в пределах 1 357 201, т. е. нам достаточно запомнить 78 496 шестизначных чисел (размер интервала)/2. Такая таблица имеет длину примерно 471 000 битов. Дальнейшего уплотнения можно добиться, используя для представления интервалов двоичные коды переменной длины (ср. с п 6.2.2).

Таблица 1
Таблица интервалов между последовательными простыми числами

Интервал $(p_{k+1} - p_k)$	p_k	Интервал $(p_{k+1} - p_k)$	p_k
1	2	52	19609
2	3	72	31397
4	7	86	155921
6	23	96	360653
8	89	112	370261
14	113	114	492113
18	523	118	1349533
20	887	132	1357201
23	1129	148	2010733
34	1327	154	4652353
36	9551	180	17051707
44	15683	310	20831323

В таблице помещены интервалы $p_{k+1} - p_k$, превышающие $p_{j+1} - p_j$ для всех $j < k$. Более подробно см. F. Gruenberger, G. Armerding, "Statistics on the first six million prime numbers", RAND Corp. report P-2460 (October, 1961).

Упражнения

- [M20] Пусть все ключи, по которым проводится поиск, равновероятны. Определите среднеквадратичное отклонение числа сравнений при удачном поиске в таблице из N записей.
- [16] Измените алгоритм S, приспособив его для работы с таблицами в связанном представлении. (Если P указывает на запись в таблице, то предполагается, что KEY(P) есть ключ, INFO(P)—ассоциированная информация и LINK(P)—указатель на следующую запись. Кроме того, предполагается, что FIRST указывает на первую запись, а последняя запись указывает на Λ.)
- [16] Напишите MIX-программу для алгоритма упр. 2. Выразите время работы вашей программы через величины C и S , определенные в (1).
- [17] Применима ли идея алгоритма Q к таблицам в связанном представлении? (Ср. с упр. 2.)
- [20] Программа Q' , разумеется, заметно быстрее программы Q при больших C . Существуют ли малые величины C и S , при которых программа Q' требует больше времени, чем Q?
- [20] Увеличив программу Q' на три команды, сведите время ее работы к $(3.33C + \text{const})u$.
- [M20] Определите среднее число сравнений (3), используя "бинарное" распределение (5).
- [BM22] Найдите асимптотический ряд для $H_n^{(x)}$ при $n \rightarrow \infty$; $x \neq 1$.
- [M23] В тексте замечено, что распределения вероятностей (11) и (13) приблизительно эквивалентны и что среднее число сравнений в (13) равно $\theta N/(\theta + 1) + O(N^{1-\theta})$. Равно ли этому же числу среднее число сравнений при использовании распределения (11)?
- [M20] Наилучшее расположение записей в таблице определяется формулой (4); каково *наихудшее* расположение? Покажите, что средние числа сравнений при наилучшем и наихудшем расположениях связывает простое соотношение.
- [M30] Целью этого упражнения является анализ поведения самоорганизующегося файла. Сначала необходимо ввести довольно сложные обозначения. Положим $f_m(x_1, x_2, \dots, x_m)$ равным бесконечной сумме всех различных упорядоченных произведений $x_{i_1} x_{i_2} \dots x_{i_k}$, где $1 \leq i_1, \dots, i_k \leq m$ и каждое x_i , $1 \leq i \leq m$, входит во все произведения. Например,

$$f_2(x, y) = \sum_{j,k \geq 0} (x^{1+j} y (x+y)^k + y^{1+k} x (x+y)^k) = \frac{xy}{1-x-y} \left(\frac{1}{1-x} + \frac{1}{1-y} \right).$$

Имея множество $X = \{x_1, \dots, x_n\}$ из n переменных, положим

$$P_{nm} = \sum_{1 \leq j_1 < \dots < j_m \leq n} f_m(x_{j_1}, \dots, x_{j_m});$$

$$Q_{nm} = \sum_{1 \leq j_1 < \dots < j_m \leq n} \frac{1}{1 - x_{j_1} - \dots - x_{j_m}}.$$

Например, $P_{32} = f_2(x_1, x_2) + f_2(x_1, x_3) + f_2(x_2, x_3)$, а $Q_{32} = 1/(1-x_1-x_2) + 1/(1-x_1-x_3) + 1/(1-x_2-x_3)$. По определению полагаем $P_{n0} = Q_{n0} = 1$.

- a) Предположим, что в самоорганизующейся файл запросы на элемент R_i поступают с вероятностью p_i . Покажите, что после длительной работы системы элемент R_i окажется на m -м месте от начала файла с предельной вероятностью $p_i P_{N-1, m-1}$, где $X = \{p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_N\}$.
- b) Суммируя результат (a) при $m = 1, 2, \dots$, получаем тождество

$$P_{nn} + P_{n, n-1} + \dots + P_{n0} = Q_{nn}.$$

Выведите отсюда, что

$$P_{nm} + \binom{n-m+1}{1} P_{n, m-1} + \dots + \binom{n-m+m}{m} P_{n0} = Q_{nm};$$

$$Q_{nm} - \binom{n-m+1}{1} Q_{n, m-1} + \dots + (-1)^m \binom{n-m+m}{m} Q_{n0} = P_{nm}.$$

- c) Подсчитайте предел среднего расстояния $d_i = \sum_{m \geq 1} m p_i P_{N-1, m-1}$ элемента R_i от начала таблицы; затем найдите $\bar{C}_N = \sum_{1 \leq i \leq n} p_i d_i$.
12. [M23] Используя (17), определите среднее число сравнений, требуемых при поиске в самоорганизующемся файле, если искомые ключи имеют бинарное распределение (5).
13. [M27] Используя (17), определите \bar{C}_N в случае распределения (6).
14. [M21] Имеются две последовательности действительных чисел $\langle x_1, x_2, \dots, x_n \rangle$ и $\langle y_1, y_2, \dots, y_n \rangle$. Какая перестановка индексов a_1, a_2, \dots, a_n делает сумму $\sum x_i y_{a_i}$ максимальной? минимальной?
15. [M22] В тексте показано, как оптимально расположить на ленте программы системной библиотеки, когда разыскивается только одна программа. При исследовании работы с лентой, содержащей библиотеку *подпрограмм*, нужны другие предположения, так как требуется загружать различное число подпрограмм, вызываемых из программы пользователя.

В этом случае предположим, что к элементу i обращаются с вероятностью P_i независимо от того, требуются или нет другие подпрограммы. Например, вероятность того, что нет ни одного обращения, равна $(1 - P_1)(1 - P_2) \dots (1 - P_N)$; вероятность окончания поиска сразу после загрузки i -го элемента составляет $P_i(1 - P_{i+1}) \dots (1 - P_N)$. Если через L_i обозначить длину i -й подпрограммы, то среднее время поиска будет пропорционально величине

$$L_1 P_1 (1 - P_2) \dots (1 - P_N) + (L_1 + L_2) P_2 (1 - P_3) \dots (1 - P_N) + \dots + (L_1 + \dots + L_N) P_N$$

Каким будет в этих предположениях оптимальное расположение подпрограмм на ленте?

16. [M22] (Г. Ризель.) Программист хочет проверить, выполняются ли одновременно n данных условий. (Например, он желает знать, верно ли, что $x > 0$ и $y < z^2$, но сразу не ясно, какое из условий нужно проверить первым.) Предположим, что проверка условия i занимает T_i единиц времени, вероятность выполнения условия i равна p_i и не зависит от исходов других проверок. В каком порядке должны производиться проверки?
17. [M23] (У. Э. Смит.) Предположим, что имеется n работ, причем i -я требует T_i единиц времени и имеет *крайний срок окончания* D_i . Другими словами, предполагается, что i -я работа должна быть закончена не позднее времени D_i . Составьте расписание работ, минимизирующее *максимальное запаздывание*, т. е.

$$\max(T_{a_1} - D_{a_1}, T_{a_1} + T_{a_2} - D_{a_2}, \dots, T_{a_1} + T_{a_2} + \dots + T_{a_n} - D_{a_n}).$$

17. [M30] (Сцепленный поиск.) Предположим, что N записей расположены в виде линейного массива R_1, \dots, R_N и запись R_i будет разыскиваться с вероятностью p_i . Поисковый процесс называется "сцепленным", если каждый следующий поиск начинается с места окончания предыдущего. Если идущие друг за другом поиски независимы, то в среднем требуется $\sum_{1 \leq i, j \leq N} p_i p_j d(i, j)$ единиц

времени, где через $d(i, j)$ обозначен промежуток времени, требующийся на поиск от i -й до j -й позиции. Эта модель применима, например, к поиску на диске [$d(i, j)$ — время перемещения между i -м и j -м цилиндрами].

Результаты данного упражнения позволяют охарактеризовать оптимальное расположение записей при сцепленном поиске. Рассмотрим случай, когда $d(i, j)$ есть возрастающая функция от $|i - j|$, т. е. $d(i, j) = d_{|i-j|}$; $d_1 < d_2 < \dots < d_{N-1}$. (Значение d_0 безразлично.) Докажите, что в данном случае расположение записей $R_1 \dots R_N$ будет наилучшим среди всех $N!$ перестановок тогда и только тогда, когда или $p_1 \leq p_N \leq p_2 \leq p_{N-1} \leq \dots \leq p_{\lfloor N/2 \rfloor + 1}$,

Picture: Рис. 2. Расположение в виде органных труб минимизирует среднее время сцепленного поиска.

или $p_N \leq p_1 \leq p_{N-1} \leq p_2 \leq \dots \leq p_{\lfloor N/2 \rfloor}$. (Значит, показанное на рис. 2 расположение в виде "органных труб" оптимально.) [Указание. Рассмотрите произвольное расположение, которому соответствуют вероятности $q_1 q_2 \dots q_k s r_k \dots r_2 r_1 t_1 \dots t_m$, $m \geq 0, k > 0$; $N = 2k + m + 1$. Покажите, что другое расположение $q'_1 q'_2 \dots q'_k s r'_k \dots r'_2 r'_1 t_1 \dots t_m$ лучше, если $q'_i = \min(q_i, r_i)$ и $r'_i = \max(q_i, r_i)$ (исключая случай $q'_i = q_i$ и $r'_i = r_i$ для всех i и случай $q'_i = r_i, r'_i = q_i, t_j = 0$ для всех i и j). Утверждение верно и при отсутствии s , когда $N = 2k + m$.]

19. [M20] (Продолжение упр. 18.) Пусть функция $d(i, j)$ обладает свойством $d(i, j) + d(j, i) = c$ для всех i и j . Найдите оптимальное расположение для сцепленных поисков. [Такая ситуация встречается при поиске на лентах, когда не предусмотрена возможность читать в обратном направлении и мы не знаем нужного направления поиска; при $i < j$ имеем $d(i, j) = a + b(L_{i+1} + \dots + L_j)$ и $d(j, i) = a + b(L_{j+1} + \dots + L_N) + r + b(L_1 + \dots + L_i)$, где r — время перемотки ленты.]
20. [M28] (Продолжение упр. 18.) Найдите оптимальный порядок записей для сцепленных поисков, если $d(i, j) = \min(d_{|i-j|}, d_{n-|i-j|})$ и $d_1 < d_2 < \dots$. [Такая модель применима для исследования поиска в циклическом списке с двумя связями или запоминающем устройстве с возможностью сдвига в обе стороны.]
21. [M28] Рассмотрим n -мерный куб, вершины которого имеют координаты (d_n, \dots, d_1) , $d_i = 0$ или 1 . Две вершины называются *соседними*, если у них различаются точно по одной координате. Предположим, что 2^n чисел $x_0 \leq x_1 \leq \dots \leq x_{2^n-1}$ должны быть поставлены в соответствие 2^n вершинам так, чтобы сумма $\sum |x_i - x_j|$ была минимальна; сумма берется по всем i и j , таким, что x_i и x_j поставлены в соответствие соседним вершинам. Докажите, что минимум достигается, когда при всех i значение x_i присвоено вершине, координаты которой являются двоичным представлением i .
- >22. [20] Предположим, что в большом файле нужно найти 1000 *ближайших* к данному ключу записей, т. е. 1000 записей, придающих наименьшие значения некоторой функции расстояния $d(K_i, K)$. Какая структура данных лучше всего подходит для последовательного поиска в этом случае?

6.2. ПОИСК ПОСРЕДСТВОМ СРАВНЕНИЯ КЛЮЧЕЙ

В этом параграфе мы рассмотрим методы поиска, основанные на линейном упорядочении ключей (например, порядок может быть числовым или алфавитным). После сравнения данного аргумента K с ключом K_i из таблицы поиск продолжается одним из трех способов в зависимости от того, какое из соотношений верно: $K < K_i$, $K = K_i$, $K > K_i$. При последовательном поиске мы, в сущности, должны выбирать одно из двух продолжений ($K = K_i$ или $K \neq K_i$), но если мы освободимся от ограничения иметь лишь последовательный доступ к таблице, то получим возможность эффективно использовать отношение порядка.

6.2.1. Поиск в упорядоченной таблице

Что вы станете делать, если вам вручат большую телефонную книгу и попросят найти имя человека, номер телефона которого 795-68-41? За неимением лучшего придется воспользоваться методами последовательного поиска, изложенными в § 6.1. (Правда, ловкий частный детектив мог бы попытаться набрать номер и спросить, кто говорит; не исключено также, что у него есть друзья на телефонной станции, имеющие доступ к справочникам, где телефоны расположены в порядке возрастания номеров.) Дело в том, что гораздо легче найти запись по фамилии абонента, а не по номеру, хотя телефонная книга содержит всю информацию, нужную в обоих случаях. Если необходимо найти запись в большом файле, о последовательном поиске почти не может быть речи, но использование отношения порядка в огромной степени облегчает работу.

В нашем распоряжении есть много методов сортировки (гл. 5), поэтому для нас не составит труда упорядочить файл, чтобы затем быстрее произвести поиск. Разумеется, если нужен лишь однократный просмотр, то быстрее произвести его последовательно, без предварительной сортировки. Но если

в одной и той же таблице приходится часто производить поиск, то эффективнее упорядочить ее. В этом пункте мы изучим методы поиска в таблицах со случайным доступом и упорядоченными ключами

$$K_1 < K_2 < \dots < K_N.$$

После сравнения K и K_i мы имеем или

- $K < K_i$ [R_i, R_{i+1}, \dots, R_N исключаются из рассмотрения],
- или • $K = K_i$ [поиск закончен],
- или • $K > K_i$ [R_1, R_2, \dots, R_i исключаются из рассмотрения].

Умело действуя в каждом из этих случаев, мы сэкономим много времени по сравнению с последовательным поиском, если только i не слишком близко к концам таблицы. Таким образом, упорядочение ведет к эффективным алгоритмам.

Бинарный поиск. Пожалуй, первым приходит в голову следующий очевидный метод: сначала сравнить K со средним ключом в таблице. Результат сравнения позволит определить, в какой половине файла продолжить поиск, применяя к ней ту же процедуру, и т. д. После не более чем примерно $\log_2 N$ сравнений либо ключ будет найден, либо будет установлено его отсутствие. Такая процедура иногда называется "логарифмическим поиском" или "методом деления пополам", но наиболее употребительный термин—*бинарный поиск*.

Основная идея бинарного поиска довольно проста, детали же нетривиальны, и для многих хороших программистов не одна попытка написать правильную программу закончилась неудачей. Одна из наиболее популярных реализации метода использует два указателя— l и u , соответствующие верхней и нижней границам поиска, и состоит в следующем.

Алгоритм В. (Бинарный поиск.) С помощью данного алгоритма разыскивается аргумент K в таблице записей R_1, R_2, \dots, R_N , ключи которых расположены в возрастающем порядке: $K_1 < K_2 < \dots < K_N$.

В1 [Начальная установка.] Установить $l \leftarrow 1, u \leftarrow N$.

В2 [Нахождение середины.] (В этот момент мы знаем, что если K есть в таблице, то выполняются неравенства $K_l \leq K \leq K_u$. Более точное описание ситуации приводится ниже, в упр. 1.) Если $u < l$, то алгоритм заканчивается неудачно; в противном случае установить $i \leftarrow \lfloor (l + u)/2 \rfloor$: теперь i указывает примерно в середину рассматриваемой части таблицы.

В3 [Сравнение.] Если $K < K_i$, то перейти на **В4**; если $K > K_i$, то перейти на **В5**; если $K = K_i$, алгоритм заканчивается удачно.

В4 [Корректировка u .] Установить $u \leftarrow i - 1$ и вернуться к шагу **В2**.

В5 [Корректировка l .] Установить $l \leftarrow i + 1$ и вернуться к шагу **В2**. ■

Picture: Рис. 3. Бинарный поиск.

Рисунок 4 иллюстрирует поведение алгоритма В в двух случаях: когда разыскивается аргумент, равный имеющемуся в таблице числу 653, и когда разыскивается отсутствующий аргумент 400. Скобки соответствуют указателям l и u , подчеркнутый ключ представляет K_i . В обоих случаях поиск кончается после четырех сравнений.

Программа В. (Бинарный поиск.) Как и в программах § 6.1, предполагается, что K_i занимает полное слово по адресу $KEY + i$. Приводимая ниже программа использует $rI1 \equiv l, rI2 \equiv u, rI3 \equiv i$.

а) Поиск числа 653:

```
[061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908]
061 087 154 170 275 426 503 509 [512 612 653 677 703 765 897 908]
061 087 154 170 275 426 503 509 [512 612 653] 677 703 765 897 908
061 087 154 170 275 426 503 509 512 612 [653] 677 703 765 897 908
```

б) Поиск числа 400:

```
[061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908]
[061 087 154 170 275 426 503] 509 512 612 653 677 703 765 897 908
061 087 154 170 [275 426 503] 509 512 612 653 677 703 765 897 908
061 087 154 170 [275] 426 503 509 512 612 653 677 703 765 897 908
061 087 154 170 275][426 503 509 512 612 658 677 703 765 897 908
```

Рис. 4. Примеры бинарного поиска.

START	ENT1	1	1	В1. Начальная установка. $l \leftarrow 1$.
	ENT2	N	1	$u \leftarrow N$.
	JMP	2F	1	На В2.
5H	JE	SUCCESS	$C1$	Переход, если $K = K_i$.
	ENT1	1, 3	$C1 - S$	В5. Корректировка l . $l \leftarrow l + 1$.
2H	ENTA	0, 1	$C + 1 - S$	В2. Нахождение середины.
	INCA	0, 2	$C + 1 - S$	$rA \leftarrow l + u$.
	SRB	1	$C + 1 - S$	$rA \leftarrow \lfloor rA/2 \rfloor$. (Меняется и rX .)
	STA	TEMP	$C + 1 - S$	
	CMP1	TEMP	$C + 1 - S$	
	JG	FAILURE	$C + 1 - S$	Переход, если $u < l$.
	LD3	TEMP	C	$i \leftarrow$ середина.
3H	LDA	K	C	В3. Сравнение.
	CMPA	KEY, 3	C	
	JGE	5B	C	Переход, если $K \geq K_i$.
	ENT2	-1, 3	$C2$	В4. Корректировка u .
	JMP	2B	$C2$	На В2.

Данная процедура "реализуется" на машине MIX менее удачно, чем предыдущие, так как регистровая арифметика в MIX небогата. Время, работы программы равно $(18C - 10S + 12)u$, где

Picture: Рис. 5. Бинарное дерево, соответствующее бинарному поиску ($N = 16$).

$C = C1 + C2$ —количество произведенных сравнений (сколько раз выполняется шаг В3), $S = 1$ или 0 в зависимости от удачного или неудачного исхода. Заметим, что строка 08 программы расшифровывается как "сдвиг вправо на 1" (shift right binary 1), что допустимо лишь в двоичных версиях MIX; в общем случае ее следует заменить на "MUL = $1 // 2^{4+1}$ ", тогда время работы программы увеличится до $(26C - 18S + 20)u$.

Представление в виде дерева. Чтобы досконально разобраться в алгоритме В, лучше всего представить его в виде бинарного дерева решений. (На рис. 5 показано такое дерево для случая $N = 16$.)

При $N = 16$ первым производится сравнение $K : K_8$, что представлено на рисунке корневым узлом (8). Далее, если $K < K_8$, алгоритм обрабатывает левое поддерево, сравнивая K и K_4 , аналогично, если $K > K_8$, используется правое поддерево. Неудачный поиск ведет в один из "внешних" квадратных узлов, занумерованных от $\boxed{0}$ до \boxed{N} ; например, мы достигнем узла $\boxed{6}$ тогда и только тогда, когда $K_6 < K < K_7$.

Бинарное дерево, соответствующее бинарному поиску среди N записей, можно построить следующим образом: при $N = 0$ дерево сводится к узлу $\boxed{0}$. В противном случае корневым узлом является

Picture: Рис. стр. 487

левое поддерево соответствует бинарному дереву с $\lceil N/2 \rceil - 1$ узлами, а правое—дереву с $\lfloor N/2 \rfloor$ узлами и числами в узлах, увеличенными на $\lceil N/2 \rceil$.

Аналогичным образом *любой* алгоритм поиска в упорядоченной таблице длины N , производимого путем сравнений, можно представить бинарным деревом с узлами, помеченными числами от 1 до N (предполагается, что алгоритм не содержит лишних сравнений). Обратное, любое бинарное дерево соответствует некоторому методу поиска в упорядоченной таблице; мы просто помечаем узлы

Picture: Рис. стр. 487

в симметричном порядке слева направо.

Если аргумент поиска в алгоритме В есть K_{10} , то производятся сравнения $K > K_8$, $K < K_{12}$, $K = K_{10}$. На рис. 5 это соответствует пути от корня к узлу (10). Аналогично поведение алгоритма В при других K соответствует иным путям, ведущим из корня дерева. С помощью метода построения бинарных деревьев, соответствующих алгоритму В, и индукции по N легко доказывается

Теорема В. При $2^{k-1} \leq N < 2^k$ удачный поиск, использующий алгоритм В, требует $(\min 1, \max k)$ сравнений. Неудачный поиск требует k сравнений при $N = 2^k - 1$ либо $k - 1$ или k сравнений при $2^{k-1} \leq N < 2^k - 1$. ■

Дальнейший анализ бинарного поиска. [Рекомендуем не интересующимся математикой перейти сразу к соотношениям (4).] Представление в виде дерева позволяет легко подсчитать *среднее* число сравнений. Через C_N обозначим среднее число сравнений при удачном поиске в предположении, что каждый из N ключей с равной вероятностью является аргументом поиска. Среднее число сравнений C'_N соответствует неудачному поиску; предполагается, что все интервалы (их $N + 1$) между ключами и вне крайних значений равновероятны. Имеем по определению длин внутреннего и внешнего пути

$$C_N = 1 + \frac{\text{Длина внутреннего пути дерева}}{N};$$

$$C'_N = \frac{\text{Длина внешнего пути дерева}}{N + 1}.$$

Из формулы (2.3.4.5-3) видно, что длина внешнего пути на $2N$ больше длины внутреннего пути; отсюда следует довольно неожиданное соотношение между C_N и C'_N

$$C_N = \left(1 + \frac{1}{N}\right) C'_N - 1. \quad (2)$$

Эта формула, полученная Хиббардом [JACM, 9 (1962), 16–17], справедлива для всех методов поиска, соответствующих бинарным деревьям, т. е. для всех методов, не содержащих лишних сравнений. Дисперсия C_N также может быть выражена через дисперсию C'_N (см. упр. 25).

Из приведенных формул видно, что "наилучшему" способу поиска путем сравнений соответствует дерево с минимальной длиной внешнего пути среди всех бинарных деревьев, содержащих N внутренних узлов. К счастью, можно доказать, что алгоритм В оптимален в этом смысле, так как бинарное дерево имеет минимальную длину пути тогда и только тогда, когда все внешние узлы находятся на одном или двух соседних уровнях. (См. упр. 5.3.1-20.) Следовательно, длина внешнего пути бинарного дерева, соответствующего алгоритму В, равна

$$(N + 1)(\lfloor \log_2 N \rfloor + 2) - 2^{\lfloor \log_2 N \rfloor + 1}. \quad (3)$$

(См. (5.3.1-33).) Используя (3) и (2), можно точно вычислить среднее число сравнений, если предположить, что все аргументы поиска равновероятны:

$N =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N =$	1	$1\frac{1}{2}$	$1\frac{2}{3}$	2	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$2\frac{5}{8}$	$2\frac{7}{9}$	$2\frac{9}{10}$	3	$3\frac{1}{12}$	$3\frac{2}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$3\frac{6}{16}$
$C'_N =$	1	$1\frac{2}{3}$	2	$2\frac{2}{5}$	$2\frac{4}{6}$	$2\frac{6}{7}$	3	$3\frac{4}{9}$	$3\frac{6}{10}$	$3\frac{8}{11}$	$3\frac{8}{12}$	$3\frac{10}{13}$	$3\frac{12}{14}$	$3\frac{14}{15}$	4	$4\frac{2}{17}$

В общем случае, если $k = \lfloor \log_2 N \rfloor$, имеем (ср. с (5.3.1-34))

$$\begin{aligned} C_N &= k + 1 - (2^{k+1} - k - 2)/N = \log_2 N - 1 + \varepsilon + (k + 2)/N; \\ C'_N &= k + 2 - 2^{k+1}/(N + 1) = \log_2 N + \varepsilon', \end{aligned} \quad (4)$$

где $0 \leq \varepsilon, \varepsilon' < 0.0861$.

Итак, алгоритм В требует максимум $\lfloor \log_2 N \rfloor + 1$ сравнений; среднее число сравнений при удачном поиске приближенно равно $\log_2 N - 1$. Ни один метод, основанный на сравнении ключей, не может дать лучших результатов. Среднее время работы программы В составляет примерно

$$\begin{aligned} (18 \log_2 N - 15)u & \quad \text{для удачного поиска;} \\ (18 \log_2 N + 13)u & \quad \text{для неудачного поиска} \end{aligned} \quad (5)$$

(предполагается, что все исходы поиска равновероятны).

Одна важная модификация. Соблазнительно вместо трех указателей l, i, u использовать лишь два: текущее положение i и величину его изменения δ ; после каждого сравнения, не давшего равенства, мы могли бы установить $i \leftarrow i \pm \delta$ и $\delta \leftarrow \delta/2$ (приблизительно). Этот путь реализуем, но он требует особой аккуратности в деталях, как в приведенном ниже алгоритме; более простые подходы обречены на неудачу!

Алгоритм У. (Однородный бинарный поиск.) Алгоритм служит для отыскания аргумента K в таблице записей R_1, R_2, \dots, R_N , ключи которых расположены в возрастающем порядке: $K_1 < K_2 <$

... $< K_N$. При четном N иногда происходит обращение к фиктивному ключу K_0 , который необходимо установить равным $-\infty$ (или любой величине, меньшей K . Предполагается, что $N \geq 1$).

U1 [Начальная установка.] Установить $i \leftarrow \lceil N/2 \rceil$, $m \leftarrow \lfloor N/2 \rfloor$.

U2 [Сравнение.] Если $K < K_i$, то перейти на U3; если $K > K_i$, то перейти на U4; при $K = K_i$ алгоритм оканчивается удачно.

U3 [Уменьшение i .] (Мы определили положение интервала, где нужно продолжать поиск. Он содержит m или $m - 1$ записей; i указывает на первый элемент справа от интервала.) Если $m = 0$, то алгоритм оканчивается неудачно. В противном случае установить $i \leftarrow i - \lceil m/2 \rceil$; $m \leftarrow \lfloor m/2 \rfloor$ и вернуться на U2.

U4 [Увеличение i .] (Ситуация та же, что и в шаге U3, только i указывает на первый элемент *слева* от интервала.) Если $m = 0$, то алгоритм оканчивается неудачно. В противном случае установить $i \leftarrow i + \lceil m/2 \rceil$; $m \leftarrow \lfloor m/2 \rfloor$ и вернуться на U2. ■

На рис. 6 представлено бинарное дерево, соответствующее поиску при $N = 10$. При неудачном поиске как раз перед окончанием работы алгоритма может производиться лишнее сравнение; узлы, отвечающие этим сравнениям, заштрихованы. Данный процесс поиска можно назвать *однородным*, так как разность между числом в узле уровня ℓ и числом в узле-предшественнике уровня $\ell - 1$ есть постоянная величина δ для всех узлов уровня ℓ . Обосновать правильность алгоритма U можно следующим образом. Предположим, что поиск нужно произвести в интервале

Picture: Рис. 6. Бинарное дерево для "однородного" бинарного поиска ($N = 10$).

длины $n - 1$; сравнение со средним элементом (если n четно) или с одним из двух средних (если n нечетно) выделяет два интервала длины $\lfloor n/2 \rfloor - 1$ и $\lceil n/2 \rceil - 1$. После повторения этой процедуры k раз мы получим 2^k интервалов с минимальной длиной $\lfloor n/2^k \rfloor - 1$ и максимальной длиной $\lceil n/2^k \rceil - 1$. Следовательно, на каждом этапе длины двух интервалов различаются самое большее на 1, что делает возможным выбор подходящего "среднего" элемента без запоминания последовательности точных значений длин.

Важное преимущество алгоритма U состоит в том, что нам совсем не нужно сохранять значение m ; нужно лишь сослаться на коротенькую таблицу значений δ для каждого уровня. Таким образом, алгоритм сводится к следующей процедуре, одинаково хорошей и для двоичных, и для десятичных ЭВМ.

Алгоритм С. (Однородный бинарный поиск.) Алгоритм аналогичен алгоритму U, но вместо вычислений, относящихся к m , использует вспомогательную таблицу величин

$$\text{DELTA}[j] = \left\lfloor \frac{N + 2^{j-1}}{2^j} \right\rfloor = \left(\frac{N}{2^j} \right) \text{ округленное; } \quad 1 \leq j \leq \lfloor \log_2 N \rfloor + 2. \quad (6)$$

C1 [Начальная установка.] Установить $i \leftarrow \text{DELTA}[1]$, $j \leftarrow 2$.

C2 [Сравнение:] Если $K < K_i$, то перейти на C3; если $K > K_i$, то перейти на C4. При $K = K_i$ алгоритм оканчивается удачно.

C3 [Уменьшение i .] Если $\text{DELTA}[j] = 0$, то алгоритм оканчивается неудачно. В противном случае установить $i \leftarrow i - \text{DELTA}[j]$, $j \leftarrow j + 1$ и вернуться на C2.

C4 [Увеличение i .] Если $\text{DELTA}[j] = 0$, алгоритм оканчивается неудачно. В противном случае установить $i \leftarrow i + \text{DELTA}[j]$, $j \leftarrow j + 1$ и вернуться на C2. ■

В упр. 8 будет показано, что алгоритм ссылается на вспомогательный ключ $K_0 = -\infty$ лишь при четных N .

Программа С. (Однородный бинарный поиск.) Эта программа на базе алгоритма С прodelывает ту же работу, что и программа В. Используются $rA \equiv K$, $rI1 \equiv i$, $rI2 \equiv j$, $rI3 \equiv \text{DELTA}[j]$.

START	ENT1	N+1/2	1	C1. Начальная установка.
	ENT2	2	1	$j \leftarrow 2$.
	LDA	K	1	
	JMP	2F	1	
3H	JE	SUCCESS	C1	Переход, если $K = K_i$.
	J3Z	FAILURE	C1 - S	Переход, если $\text{DELTA}[j] = 0$.
	DEC1	0,3	C1 - S - A	C3. Уменьшение i .
5H	INC2	1	C - 1	$j \leftarrow j + 1$.

2H	LD3	DELTA, 2	C	C2. Сравнение.
	CMPA	KEY. 1	C	
	JLE	3B	C	Переход, если $K \leq K_i$.
	INC1	0, 3	C2	C4. Увеличение i .
	J3NZ	5B	C2	Переход, если $DELTA[j] \neq 0$.
FAILURE	EQU	*	1 - S	Выход, если нет в таблице.

При удачном поиске этот алгоритм соответствует бинарному дереву с той же длиной внутреннего пути, что и алгоритм В, поэтому среднее число сравнений C_N дается формулой (4). При неудачном поиске алгоритм С всегда совершает ровно $\lfloor \log_2 N \rfloor + 1$ сравнений. Полное время работы программы С не вполне симметрично по отношению к левым и правым ветвям, так как C1 имеет вес, больший чем C2, но в упр. 9 будет показано, что случай $K > K_i$ встречается примерно так же часто, как и $K < K_i$; следовательно, программа С требует приблизительно

$$\begin{aligned} (8.5 \log_2 N - 6)u & \quad \text{на удачный поиск;} \\ (8.5 \lfloor \log_2 N \rfloor + 12)u & \quad \text{на неудачный поиск.} \end{aligned} \quad (7)$$

Это более чем в два раза быстрее программы В, причем не используются специальные свойства двоичных ЭВМ, в то время как в формуле (5) предполагается, что MIX имеет команду "двоичный сдвиг вправо".

Другая модификация бинарного поиска, предложенная в 1971 г. Л. Э. Шером, на некоторых ЭВМ допускает еще более быструю реализацию, так как она однородна после первого

Picture: Рис. 7. Бинарное дерево для почти однородного поиска Шера ($N = 10$).

шага и не требует вспомогательной таблицы. Сначала мы сравниваем K и K_i , где $i = 2^k$, $k = \lfloor \log_2 N \rfloor$. Если $K < K_i$, мы используем однородный поиск с последовательностью $\delta = 2^{k-1}, 2^{k-2}, \dots, 1, 0$. С другой стороны, при $K > K_i$ и $N > 2^k$ устанавливаем $i = i' = N + 1 - 2^\ell$, где $\ell = \lfloor \log_2(N - 2^k) \rfloor + 1$, и, делая вид, что первым сравнением было $K > K_{i'}$, используем однородный поиск с $\delta = 2^{\ell-1}, 2^{\ell-2}, \dots, 1, 0$.

Рисунок 7 иллюстрирует метод Шера при $N = 10$. В методе Шера никогда не требуется более $\lfloor \log_2 N \rfloor + 1$ сравнений; следовательно, он дает одно из лучших средних чисел сравнения, несмотря на то что иногда проходит через несколько последовательных избыточных шагов (ср. с упр. 12).

Еще одна модификация бинарного поиска, улучшающая все рассмотренные методы при очень больших N , обсуждается в упр. 23. В упр. 24 изложен еще более быстрый метод!

Фибоначчиев поиск. При рассмотрении многофазного слияния (п. 5.4.2) мы видели, что числа Фибоначчи могут играть роль, аналогичную степеням 2. Похожее явление имеет место и в случае поиска, когда с помощью чисел Фибоначчи создаются методы, способные соперничать с бинарным поиском. Предлагаемый метод для некоторых ЭВМ предпочтительнее бинарного, так как он включает лишь сложение и вычитание; нет необходимости в делении на 2. Следует отличать процедуру, которую мы собираемся обсуждать, от известной численной процедуры "фибоначчиева поиска", которая используется для нахождения максимума одновершинной функции [ср. с Fibonacci Quarterly, 4 (1966), 265–269]; схожесть названий ведет к некоторой путанице.

На первый взгляд фибоначчиев поиск кажется весьма таинственным; если просто взять программу и попытаться объяснить ее работу, то создается впечатление, что она работает с помощью магии. Но туман таинственности рассеется, как только мы построим

Picture: Рис. 8. Дерево Фибоначчи порядка 6.

соответствующее дерево поиска. Поэтому изучение рассматриваемого метода начнем с рассказа о "фибоначчиевых деревьях".

На рис. 8 изображено дерево Фибоначчи порядка 6. Заметьте, что оно несколько больше напоминает реальный куст, чем рассматривавшиеся ранее деревья, возможно, потому, что многие природные процессы удовлетворяют закону Фибоначчи. Вообще фибоначчиево дерево порядка k имеет $F_{k+1} - 1$ внутренних (круглых) узлов и F_{k+1} внешних (квадратных) узлов; оно строится следующим образом:

Если $k = 0$ или $k = 1$, дерево сводится просто к \square .

Если $k \geq 2$, корнем является (F_k) ; левое поддерево есть дерево Фибоначчи порядка $k - 1$; правое поддерево есть дерево Фибоначчи порядка $k - 2$ с числами в узлах, увеличенными на F_k .

Заметим, что, за исключением внешних узлов, числа, соответствующие преемникам каждого внутреннего узла, отличаются от числа в этом узле на одну и ту же величину, а именно на число Фибоначчи. Так, $5 = 8 - F_4$, $11 = 8 + F_4$ (рис. 8). Если разность была равна F_j , то на следующем уровне соответствующая разность составит для левой ветви F_{j-1} , а для правой F_{j-2} . Так, например, $3 = 5 - F_3$, а $10 = 11 - F_2$.

Эти наблюдения в совокупности с подходящим механизмом распознавания внешних узлов дают

Алгоритм F. (Фибоначчиев поиск.) Алгоритм предназначен для поиска аргумента K в таблице записей R_1, R_2, \dots, R_N , расположенных в порядке возрастания ключей $K_1 < K_2 < \dots < K_N$.

Для удобства описания предполагается, что $N + 1$ есть число Фибоначчи F_{k+1} . Подходящей начальной установкой данный метод можно сделать пригодным для любого N (см. упр. 14).

F1 [Начальная установка.] Установить $i \leftarrow F_k, p \leftarrow F_{k-1}, q \leftarrow F_{k-2}$. (В этом алгоритме p и q обозначают последовательные числа Фибоначчи.)

F2 [Сравнение.] Если $K < K_i$, то перейти на **F3**; если $K > K_i$, то перейти на **F4**; если $K = K_i$, алгоритм заканчивается удачно.

F3 [Уменьшение i .] Если $q = 0$, алгоритм заканчивается неудачно. Если $q \neq 0$, то установить $i \leftarrow i - q$, заменить (p, q) на $(q, p - q)$ и вернуться на **F2**.

F4 [Увеличение i .] Если $p = 1$, алгоритм заканчивается неудачно. Если $p \neq 1$, установить $i \leftarrow i + q$, $p \leftarrow p - q, q \leftarrow q - p$ и вернуться на **F2**. ■

В приводимой ниже реализации алгоритма F для машины MIX скорость увеличивается за счет дублирования внутреннего цикла, в одном из экземпляров которого p хранится в rI2, а q — в rI3, в другом же регистры меняются ролями; это упрощает шаг F3. На самом деле программа хранит в регистрах $p - 1$ и $q - 1$, что упрощает проверку " $p = 1$?" в шаге F4.

Программа F. (Фибоначчиев поиск.) Значения регистров: rA $\equiv K$, rI1 $\equiv i$, (rI2, rI3) $\equiv p - 1$, (rI3, rI2) $\equiv q - 1$.

START	LDA	K	1	F1. Начальная установка.			
	ENT1	F_k	1	$i \leftarrow F_k$.			
	ENT2	$F_{k-1} - 1$	1	$p \leftarrow F_{k-1}$.			
	ENT3	$F_{k-2} - 1$	1	$q \leftarrow F_{k-2}$.			
	JMP	F2A	1	На F2 .			
F4A	INC1	1, 3	F4B	INC1	1, 2	$C2 - S - A$	F4. Увеличение i. $i \leftarrow i + q$.
	DEC2	1, 3		DEC3	1, 2	$G2 - S - A$	$p \leftarrow p - q$.
	DEC3	1, 2		DEC2	1, 3	$G2 - S - A$	$q \leftarrow q - p$.
F2A	CMPA	KEY, 1	F2B	CMPA	KEY, 1	G	F2. Сравнение.
	JL	F3A		JL	F3B	G	На F3 , если $K < K_i$.
	JE	SUCCESS		JE	SUCCESS	$G2$	Выход, если $K = K_i$.
	J2NZ	F4A		J3NZ	F4B	$C2 - S$	На F4 , если $p \neq 1$.
	JMP	FAILURE		JMP	FAILURE	A	Выход, если нет в таблице.
F3A	DEC1	1, 3	F3B	DEC1	1, 2	$C1$	F3. Уменьшение i. $i \leftarrow i - q$.
	DEC2	1, 3		DEC3	1, 2	$C1$	$p \leftarrow p - q$.
	J3NN	F2B		J2NN	F2A	$C1$	Смена регистров, если $q > 0$.
	JMP	FAILURE		JMP	FAILURE	$1 - S - A$	Выход, если нет в таблице.

■

В упр. 18 анализируется время работы этой программы. Рисунок 8 показывает, а анализ доказывает, что влево мы идем несколько чаще, чем вправо. Через C , $C1$ и $C2 - S$ обозначим число выполнений шагов F2, F3 и F4 соответственно. Имеем

$$\begin{aligned}
 C &= (\text{ave } \phi k / \text{sqrt}5 + O(1), \max k - 1), \\
 C1 &= (\text{ave } k / \sqrt{5} + O(1), \max k - 1), \\
 C2 - S &= (\text{ave } \phi^{-1} k / \sqrt{5} + O(1), \max [k/2]).
 \end{aligned}
 \tag{8}$$

Значит, левая ветвь выбирается примерно в $\phi = 1.618$ раза чаще правой (это можно было предвидеть, так как каждая проба делит рассматриваемый интервал на две части, причем левая часть примерно

в ϕ раз длиннее правой). Среднее время работы программы F составляет примерно

$$\begin{aligned} (6\phi k/\sqrt{5} - (2 + 22\phi)/5)u &\approx (6.252 \log_2 N - 4.6)u && \text{для удачного поиска;} \\ (6\phi k/\sqrt{5} + (58/(27\phi))/5)u &\approx (6.252 \log_2 N + 5.8)u && \text{для неудачного поиска.} \end{aligned} \quad (9)$$

Это несколько лучше, чем (7), хотя в наихудшем случае программа F работает примерно $(8.6 \log_2 N)u$, т.е. чуть-чуть медленнее программы С.

Интерполяционный поиск. Забудем на минуту о вычислительных машинах и проанализируем, как производит поиск человек. Иногда повседневная жизнь подсказывает путь к созданию хороших алгоритмов.

Представьте, что вы ищете слово в словаре. Маловероятно, что вы сначала заглянете в середину словаря, затем отступите от начала на $1/4$ или $3/4$ и т. д., как в бинарном поиске, и уж совсем невероятно, что вы воспользуетесь фибоначиевым поиском!

Если нужное слово начинается с буквы А, вы, по-видимому, начнете поиск где-то в начале словаря. Во многих словарях имеются "побуквенные высечки" для большого пальца, которые показывают страницу, где начинаются слова на данную букву. Такую пальцевую технику легко приспособить к ЭВМ, что ускорит поиск; соответствующие алгоритмы исследуются в § 6.3.

Когда найдена отправная точка для поиска, ваши дальнейшие действия мало похожи на рассмотренные методы. Если вы заметите, что нужное слово должно находиться гораздо дальше открытой страницы, вы пропустите порядочное их количество, прежде чем сделать следующую попытку. Это в корне отличается от предыдущих алгоритмов, которые не делают различия между "много больше" и "чуть больше".

Мы приходим к такому алгоритму, называемому "интерполяционным поиском": если известно, что K лежит между K_l и K_u , то следующую пробу делаем примерно на расстоянии $(u-l)(K-K_l)/(K_u-K_l)$ от l , предполагая, что ключи являются числами, возрастающими приблизительно как арифметическая прогрессия.

Интерполяционный поиск асимптотически предпочтительнее бинарного; по существу, один шаг бинарного поиска уменьшает количество "подозреваемых" записей с n до $\frac{1}{2}n$, а один шаг интерполяционного (если ключи в таблице распределены случайным образом) — с n до \sqrt{n} . Можно показать, что интерполяционный поиск требует в среднем около $\log_2 \log_2 N$ шагов (упр. 22).

К сожалению, эксперименты на ЭВМ показали, что интерполяционный поиск уменьшает число сравнений не настолько, чтобы компенсировать возникающий дополнительный расход времени, когда таблица, в которой производится поиск, хранится во внутренней ("быстрой") памяти. Разность между $\log_2 \log_2 N$ и $\log_2 N$ становится существенной лишь для весьма больших N , а типичные файлы недостаточно случайны. Интерполяция успешна до некоторой степени лишь в применении к поиску на *внешних* запоминающих устройствах. (Заметим, что ручной просмотр словаря есть, в сущности, не внутренний, а внешний поиск; это является темой последующих рассмотрений.)

История и библиография. Первым известным примером длинного перечня элементов, упорядоченных для облегчения поиска, является знаменитая вавилонская таблица обратных величин Inakibit-Anu, датируемая примерно 200 г. до н.э. Эта глиняная пластинка, очевидно, открывала серию из трех табличек, содержащих более 500 многозначных шестидесятеричных чисел и обратных им величин, отсортированных в лексикографическом порядке. Например, встречается такая последовательность:

01	13	09	34	29	08	08	53	20	49	12	27		
01	13	14	31	52	30	49	09	07	12				
01	13	43	40	48	48	49	41	15					
01	13	48	40	30	48	46	22	59	25	25	55	33	20
01	14	04	26	40	48	36							

Сортировка 500 подобных чисел средствами тех времен кажется феноменальной. [См. также D. E. Knuth, *SACM*, 15 (1972), 671–677, где содержится много дополнительных сведений.]

Довольно естественно располагать по порядку числа, но отношение порядка между буквами или словами не представляется само собой очевидным. Однако последовательности для сравнения букв присутствовали уже в наиболее древних алфавитах. Например, многие библейские псалмы содержат стихи, следующие друг за другом в строго алфавитном порядке: первый стих начинается с аефа, второй с бета и т. д.; это облегчало запоминание. Иногда стандартная последовательность букв использовалась семитами и греками для обозначения чисел; например, α , β , γ обозначали 1, 2, 3 соответственно.

Однако использование алфавитного порядка для слов целиком было, вероятно, гораздо более поздним изобретением; вещи, очевидные сейчас даже для ребенка, когда-то требовалось объяснять взрослым! Некоторые документы, датируемые примерно 300 г. до н.э., найденные на островах Эгейского моря, содержат списки членов нескольких религиозных общин; они упорядочены, но только по первой букве, т. е. представляют собой результат лишь первого прохода слева направо поразрядной сортировки. Греческие папирусы 134–135 г. н. э. содержат фрагменты счетов, в которых фамилии налогоплательщиков упорядочены по первым двум буквам. Аполлоний Софист² использовал алфавитное упорядочение по первым двум, а часто и по последующим буквам в своем "Словаре гомеровских слов" (I век н. э.). Известно несколько примеров более совершенного алфавитного упорядочения, скажем выдающиеся "Комментарии к Гиппократу" Галена³ (около 200 г.), но это было очень редким явлением. Так, в хронике Св. Исидора⁴ "Etymologiae" (около 630 г., книга X) слова упорядочены лишь по первой букве, а в наиболее раннем из известных двуязычных словарей "Corpus Glossary" (около 725 г.) — только по двум первым. Две последние работы были, вероятно, крупнейшими нечисловыми файлами данных, скомпилированными в средние века.

Только в книге Джованни Генуэзского "Catholicon" (1286 г.) мы находим подробное описание правильного алфавитного порядка. В предисловии Джованни объясняет, что

amo предшествует *bibo*
abeo предшествует *adeo*
amatus предшествует *amor*
imprudens предшествует *impudens*
iusticia предшествует *iustus*
polisintheton предшествует *polissenus*

(т. е. приводит примеры ситуаций, когда порядок определяется по 1, 2, ..., 6-й буквам) "и далее аналогично". Он замечает, что открытие этого правила потребовало значительных усилий. "Я прошу Вас, уважаемый читатель, не презирать эту мою большую работу и этот порядок как нечто ничего не стоящее". Развитие алфавитного порядка до момента изобретения книгопечатания подробно изучил Л. У. Дейли (*Collection Latomus*, 90 (1967), 100 pp.). Он обнаружил несколько интересных старинных рукописей, которые, несомненно, использовались как черновики при сортировке слов по первой букве (см. стр. 87–90 его монографии).

Первый словарь английского языка Роберта Кодри "Table Alphabeticall" (London, 1604) содержит следующие инструкции:

Если слово, которое тебе нужно найти, начинается с (a), смотри в начало этой книги, а если с (v) — то в конец. Опять если слово начинается с (ca), смотри в начало буквы (c), а если с (cu) то в конец. И так до конца слова.

Интересно заметить, что Кодри во время подготовки словаря, вероятно, сам учился расставлять слова в алфавитном порядке; на нескольких первых страницах встречается много неправильно стоящих слов, зато дальше число ошибок существенно уменьшается.

Бинарный поиск впервые упомянут Джоном Мочли в дискуссии, которая была, пожалуй, первым опубликованным обсуждением методов нечисленного программирования [см. *Theory and techniques for the design of electronic digital computers*, ed. by G. W. Patterson, 3 (1946); 22.8–22.9]. В течение 50-х годов метод становится "хорошо известным", но, кажется, никто не разрабатывал детали алгоритма для $N \neq 2^n - 1$. [См. A. D. Booth, *Nature*, 176 (1955), 565; A. I. Dumev, *Computers and Automation*, 5 (December, 1956), 7, где бинарный поиск имеет название "Двадцать вопросов"; D. D. McCracken, *Digital Computer Programming* (Wiley, 1957), 201–203; M. Halpern, *CACM* 1, 2 (February, 1958), 1–3.]

По-видимому, алгоритм бинарного поиска, пригодный для всех N , впервые опубликован Боттенбруком [*JACM*, 9 (1962), 214]. Он представил интересную модификацию алгоритма В, когда проверки на равенство отодвигаются в конец алгоритма. Используя в шаге В2 $i \leftarrow \lceil (l+u)/2 \rceil$ вместо $\lfloor (l+u)/2 \rfloor$, он устанавливает $l \leftarrow i$ при $K \geq K_i$; тогда $u-l$ уменьшается после каждого шага. В конце, когда $l = u$, имеем $K_l \leq K < K_{l+1}$, и можно проверить, был ли поиск удачным, произведя еще одно сравнение. (Он предполагал, что первоначально $K \geq K_1$.) Эта идея позволяет несколько ускорить внутренний цикл на многих ЭВМ; то же верно и для всех обсуждавшихся в данном пункте алгоритмов, однако такое изменение оправдано лишь для больших N (см. упр. 23).

К. Э. Айверсон [*A Programming Language* (Wiley, 1962), 141] привел процедуру алгоритма В, но без рассмотрения возможности неудачного поиска. Д. Кнут (*CACM*, 6 (1963), 556–558) представил

² Один из грамматиков древности, родом из Александрии. — Прим. перев.

³ Римский врач и естествоиспытатель, классик античной медицины. — Прим. перев.

⁴ Исидор Севильский — испанский епископ, выдающийся ученый и писатель. — Прим. перев.

алгоритм В как пример использования автоматической системы построения блок-схем. Однородный бинарный поиск (алгоритм С) предложил автору в 1971 г. А. К. Чандра (Стандфордский университет).

Фибоначчиев поиск изобретен Д. Фергюсоном [САСМ, 3 (1960), 648], но его блок-схема и анализ были некорректны. Дерево Фибоначчи (без меток) появилось гораздо раньше как курьез в первом издании популярной книги Гуго Штейнгауза "Математический калейдоскоп" (М., Гостехиздат, 1949) на стр. 34; оно было нарисовано корнем вниз и выглядело, как обычное дерево; правые ветви были сделаны в два раза длиннее левых, так что все листья оказались на одном уровне.

Интерполяционный поиск предложен У. У. Петерсоном [IBM J. Res. & Devel., 1 (1957), 131–132]. Он дал теоретическую оценку среднего числа сравнений для случая, когда ключи случайно выбираются из равномерно распределенной последовательности, однако эти оценки расходятся с экспериментальными данными.

Упражнения

- >1. [21] Докажите, что если $u < l$ в шаге В2 бинарного поиска, то $u = l - 1$ и $K_u < K < K_l$. (Будем считать, что $K_0 = -\infty$ и $K_{N+1} = \infty$, хотя алгоритм не использует эти добавочные ключи, т. е. они не должны присутствовать в таблице.)
- >2. [22] Будет ли алгоритм В работать правильно, если (а) заменить в шаге В5 " $l \leftarrow i + 1$ " на " $l \leftarrow i$ "? (б) Заменить в шаге В4 " $u \leftarrow i - 1$ " на " $u \leftarrow i$ "? (с) Внести оба изменения?
3. [15] Какой метод поиска соответствует дереву

Picture: Рис. стр. 500

Каково среднее число сравнений при удачном поиске? при неудачном?

4. [20] Если поиск, производимый программой S из § 6.1 (последовательный поиск), требует 638 единиц времени, то сколько будет работать программа В (бинарный поиск)?
5. [M24] При каких N программа В в среднем *медленнее* последовательного поиска (программа 6.1Q'), если предполагается, что поиск удачен?
6. [28] (К. Э. Айверсон) В силу упр. 5 лучше всего было бы иметь некий "гибридный" метод, переходящий от бинарного поиска к последовательному, когда длина остающегося интервала меньше некоторой разумно выбранной величины. Напишите эффективную программу для MIX и определите наилучший момент смены методов поиска.
- >7. [M22] Будет ли алгоритм U работать правильно, если мы изменим шаг U1 так, что (а) как i , так и m устанавливаются равными $\lfloor N/2 \rfloor$; (б) как i , так и m устанавливаются равными $\lceil N/2 \rceil$? [Указание: предположим, что первый шаг выглядел бы так: "Установить $i \leftarrow 0$, $m \leftarrow N$ (или $N + 1$), перейти на U4".]
8. [M20] (а) Чему равна сумма $\sum_{0 \leq j \leq \lfloor \log_2 N \rfloor + 2} \text{DELTA}[j]$ приращений в алгоритме С? (б) Каковы минимальное и максимальное значения i , которые могут появиться в шаге С2?
9. [M26] Проведите частотный анализ для программы С и найдите точную зависимость средних значений $C1$, $C2$ и A от N и S .
10. [20] Существуют ли $N > 1$, при которых алгоритмы В и С абсолютно эквивалентны в том смысле, что они совершают одинаковые, последовательности сравнений при всех аргументах поиска?
11. [21] Объясните, как написать MIX-программу для алгоритма С, содержащую примерно $7 \log_2 N$ команд, время работы которой составит приблизительно $4.5 \log_2 N$ единиц?
12. [20] Нарисуйте дерево бинарного поиска, соответствующее методу Шера при $N = 12$.
13. [M24] Затабулируйте средние числа сравнений в методе Шера для $1 \leq N < 16$, рассматривая как удачные, так и неудачные поиски.
14. [21] Объясните, как приспособить алгоритм F для работы с любым $N \geq 1$.
15. [21] На рис. 9 изображена диаграмма размножения кроликов в исходной задаче Фибоначчи (ср. с п. 1.2.8). Существует ли простая связь между этой диаграммой и фибоначчиевым деревом решений?
16. [M19] При каких значениях k фибоначчиево дерево порядка k определяет оптимальную процедуру поиска, т. е. такую, при которой среднее число сравнений минимально?
17. [M21] Из упр. 1.2.8-34 (или упр. 5.4.2-10) мы знаем, что любое натуральное число n единственным образом представляется в виде сумму чисел Фибоначчи: $n = F_{a_1} + F_{a_2} + \dots + F_{a_r}$, где $r \geq 1$, $a_j \geq a_{j+1} + 2$ при $1 \leq j < r$ и $a_r \geq 2$. Докажите, что в фибоначчиевом дереве порядка k путь от корня до узла (n) имеет длину $k + 1 - r - a_r$.

Picture: Рис. 9. Пары кроликов, размножающиеся по правилу Фибоначчи.

18. [M30] Проведите частотный анализ для программы F и найдите точные формулы для средних значений C_1 , C_2 и A как функций от k , F_k , F_{k+1} и S .
19. [M42] Проведите детальный анализ среднего времени работы алгоритма, предложенного в упр. 14.
20. [M22] Число сравнений, требующихся при бинарном поиске, приближенно равно $\log_2 N$, при фибоначчиевом — $(\phi/\sqrt{5}) \log_\phi N$. Цель этого упражнения — показать, что эти формулы являются частными случаями более общего результата.

Пусть p и q — положительные числа и $p + q = 1$. Рассмотрим алгоритм поиска по таблице из N записей с возрастающими ключами, который, начиная со сравнения аргумента с (pN) -м ключом, повторяет эту процедуру для меньших блоков. (Для бинарного поиска $p = q = 1/2$; для фибоначчиева $p = 1/\phi$, $q = 1/\phi^2$.)

Обозначим среднее число сравнений, требуемых для поиска в таблице размера N , через $C(N)$; оно приближенно удовлетворяет соотношениям

$$C(1) = 0, \quad C(N) = 1 + pC(pN) + qC(qN) \quad \text{для } N > 1.$$

Это происходит потому, что после первого сравнения поиск примерно с вероятностью p сводится к поиску среди pN элементов и с вероятностью q — к поиску среди qN элементов. При больших N можно пренебречь эффектом низшего порядка, связанным с тем, что числа pN и qN не целые.

- a) Покажите, что $C(N) = \log_b N$ точно удовлетворяет указанным соотношениям при некотором b . Для бинарного и фибоначчиева поисков величина b получается из выведенных ранее формул.
- b) Некто рассуждал так: "С вероятностью p длина рассматриваемого интервала делится на $1/p$, с вероятностью q — на $1/q$. Следовательно интервал делится в среднем на $p(1/p) + q(1/q) = 2$, так что алгоритм в точности так же хорош, как и бинарный поиск, независимо от p и q ". Есть ли ошибка в этих рассуждениях?
21. [20] Нарисуйте бинарное дерево, соответствующее интерполяционному поиску при $N = 10$.
22. [M43] (Э. К. Яо и Ф. Ф. Яо.) Покажите, что должным образом оформленный интерполяционный поиск в среднем требует (асимптотически) $\log_2 \log_2 N$ сравнений, если N отсортированных ключей имеют независимые равномерные распределения. Более того, все алгоритмы поиска по таким таблицам должны совершать в среднем $\log_2 \log_2 N$ сравнений (оценка асимптотическая).
- >23. [25] Алгоритм бинарного поиска Г. Боттенбрука, упомянутый в конце пункта, "откладывает" проверки на равенство до самого конца поиска. (Во время работы алгоритма мы знаем, что $K_l \leq K < K_{u+1}$, проверка на равенство проводится лишь при $l = u$.) Такой трюк сделал бы программу В чуть быстрее при больших N , так как мы избавились бы от команды "JF," во внутреннем цикле. (Однако эта идея практически нереальна, так как $\log_2 N$ обычно мал; лишь при $N > 2^{36}$ компенсируется необходимость дополнительной итерации!)
- Покажите, что *любой* алгоритм поиска, соответствующий бинарному дереву и разветвляющийся по трем направлениям ($<$, $=$, или $>$), можно переделать в алгоритм, разветвляющийся во внутренних узлах лишь по двум направлениям ($<$ или \geq). В частности, модифицируйте таким способом алгоритм С.
- >24. [23] Полное бинарное дерево является удобным способом представления в последовательных ячейках дерева с минимальной длиной пути. (Ср. с п. 2.3.4.5.) Придумайте эффективный метод поиска, основанный на таком представлении. [Указание: можно ли в бинарном поиске использовать умножение на 2 вместо деления на 2?]
- >25. [M25] Предположим, что у бинарного дерева имеется a_k внутренних и b_k внешних узлов на уровне k , $k = 0, 1, \dots$ (Корень находится на нулевом уровне.) Так, для рис. 8 имеем $(a_0, a_1, \dots, a_6) = (1, 2, 4, 4, 1, 0)$ и $(b_0, b_1, \dots, b_6) = (0, 0, 0, 4, 7, 2)$. (a) Покажите, что существует простое алгебраическое соотношение, связывающее производящие функции $A(z) = \sum_k a_k z^k$ и $B(z) = \sum_k b_k z^k$. (b) Распределение вероятностей при удачном поиске по бинарному дереву имеет производящую функцию $g(z) = zA(z)/N$, а при неудачном поиске производящая функция есть $h(z) = B(z)/(N + 1)$. (В обозначениях п. 6.2.1 $C_N = \text{mean}(g)$, $C'_N = \text{mean}(h)$, а соотношение (2) связывает эти величины.) Найдите зависимость между $\text{var}(g)$ и $\text{var}(h)$.
26. [22] Покажите, что дерево Фибоначчи связано с сортировкой многофазным слиянием на трех лентах.
27. [M30] (Х. С. Стоун и Дж. Лини.) Рассмотрим процесс поиска, основанный только на сравнениях ключей и использующий одновременно k процессоров. При каждом шаге поиска определяются k индексов i_1, i_2, \dots, i_k и мы совершаем k одновременных сравнений: если $K = K_j$ для некоторого j , поиск кончается удачно, в противном случае переходим к следующему шагу поиска, основываясь на 2^k возможных исходах $K < K_{i_j}$ или $K > K_{i_j}$, $1 \leq j \leq k$.

Покажите, что такой процесс при $N \rightarrow \infty$ должен совершать в среднем по крайней мере $\log_{k+1} N$ шагов. (Предполагается, что все ключи в таблице, также как и аргумент поиска, равновероятны.) (Значит, по сравнению с однопроцессорным бинарным поиском мы выигрываем в скорости не в k раз, как можно было бы ожидать, а лишь в $\log_2(k+1)$ раз. В этом смысле выгоднее каждый процессор использовать для отдельного поиска, а не объединять их.)

6.2.2. Поиск по бинарному дереву

В предыдущем пункте мы видели, что использование неявной структуры бинарного дерева облегчает понимание бинарного и фибоначчиева поисков. Рассмотрение соответствующих деревьев позволило заключить, что при данном N среди всех методов поиска путем сравнения ключей бинарный поиск совершает минимальное число сравнений. Но методы предыдущего пункта предназначены главным образом для таблиц фиксированного размера, так как последовательное расположение записей делает вставки и удаления довольно трудоемкими. Если таблица динамически изменяется, то экономия от использования бинарного поиска не покрывает затрат на поддержание упорядоченного расположения ключей.

Явное использование структуры бинарного дерева позволяет быстро вставлять и удалять записи и производить эффективный поиск по таблице. В результате мы получаем метод, полезный как для поиска, так и для сортировки. Такая гибкость достигается

Picture: Рис. 10. Бинарное дерево поиска.

путем добавления в каждую запись двух полей для хранения ссылок.

Методы поиска по растущим таблицам, часто называют *алгоритмами таблиц символов*, так как ассемблеры, компиляторы и другие системные программы обычно используют их для хранения определяемых пользователем символов. Например, ключом записи в компиляторе может служить символический идентификатор, обозначающий переменную в некоторой программе на Фортране или Алголе, а остальные поля записи могут содержать информацию о типе переменной и ее расположении в памяти. Или же ключом может быть символ программы для MIX, а оставшаяся часть записи может содержать эквивалент этого символа. Программы поиска с вставкой по дереву, которые будут описаны в этом пункте, отлично подходят для использования в качестве алгоритмов таблиц символов, особенно если желательно распечатывать символы в алфавитном порядке. Другие алгоритмы, таблиц символов описаны в § 6.3 и 6.4.

На рис. 10 изображено бинарное дерево поиска, содержащее названия одиннадцати знаков зодиака⁵. Если теперь, отправляясь от корня дерева, мы будем искать двенадцатое название, SAGITTARIUS, то увидим, что оно больше, чем CAPRICORN, поэтому нужно идти вправо; оно больше, чем PISCES, — снова идем вправо; оно меньше, чем TAURUS, — идем влево; оно меньше, чем SCORPIO, — и мы попадаем во внешний узел $\boxed{8}$. Поиск был неудачным; теперь по окончании поиска мы можем *вставить* SAGITTARIUS, "подвязывая" его к дереву вместо внешнего узла $\boxed{8}$. Таким образом, таблица может расти без перемещения существующих записей. Рисунок 10 получен последовательной вставкой, начиная с пустого дерева, ключей CAPRICORN, AQUARIUS, PISCES, ARIES, TAURUS, GEMINI, CANCER, LEO, VIRGO, LIBRA, SCORPIO в указанном порядке.

На рис. 10 все ключи левого поддерева корня предшествуют по алфавиту слову CAPRICORN, а в правом поддерева стоят после него. Аналогичное утверждение справедливо для левого и правого поддерева любого узла. Отсюда следует, что при обходе дерева в *симметричном порядке* ключи располагаются строго в алфавитном порядке:

AQUARIUS, ARIES, CANCER, CAPRICORN, GEMINI,
LEO, . . . , VIRGO,

так как симметричный порядок основан на прохождении сначала левого поддерева каждого узла, затем самого узла, а затем его правого поддерева (ср. с п. 2.3.1).

Ниже дается подробное описание процесса поиска с вставкой.

Алгоритм Т. (*Поиск с вставкой по дереву.*) Дана таблица записей, образующих бинарное дерево. Производится поиск заданного аргумента K . Если его нет в таблице, то в подходящем месте в дерево вставляется новый узел, содержащий K .

⁵ Знаки зодиака, упорядоченные по месяцам: Козерог, Водолей, Рыбы, Овен, Телец, Близнецы, Рак, Лев, Дева, Весы, Скорпион, Стрелец, — Прим. перев.

Предполагается, что узлы дерева содержат по крайней мере следующие поля:

KEY(P) = ключ, хранящийся в узле NODE(P);
 LLINK(P) = указатель на левое поддереву узла NODE(P);
 RLINK(P) = указатель на правое поддереву узла NODE(P).

Пустые поддеревья (внешние узлы на рис. 10) представляются пустыми указателями Λ . Переменная ROOT указывает на корень дерева. Для удобства предполагается, что дерево не пусто ($ROOT \neq \Lambda$), так как при $ROOT = \Lambda$ операции становятся тривиальными.

- T1** [Начальная установка.] Установить $P \leftarrow ROOT$. (Указатель P будет продвигаться вниз по дереву.)
T2 [Сравнение.] Если $K < KEY(P)$, то перейти на **T3**; если $K > KEY(P)$, то перейти на **T4**; если $K = KEY(P)$, поиск завершен удачно.
T3 [Шаг влево.] Если $LLINK(P) \neq \Lambda$, установить $P \leftarrow LLINK(P)$ и вернуться на **T2**. В противном случае перейти на **T5**.
T4 [Шаг вправо.] Если $RLINK(P) \neq \Lambda$, установить $P \leftarrow RLINK(P)$ и вернуться на **T2**.
T5 [Вставка в дерево.] (Поиск неудачен; теперь мы поместим K в дерево.) Выполнить $Q \leftarrow AVAIL$; Q теперь указывает на новый узел. Установить $KEY(Q) \leftarrow K$, $LLINK(Q) \leftarrow RLINK(Q) \leftarrow \Lambda$. (На самом деле нужно произвести начальную установку и других полей нового узла.) Если K было меньше $KEY(P)$, установить $LLINK(P) \leftarrow Q$; в противном случае установить $RLINK(P) \leftarrow Q$. (В этот момент мы могли бы присвоить $P \leftarrow Q$ и удачно завершить работу алгоритма.) ■

Алгоритм сам подсказывает реализацию на языке MIXAL. Предположим, например, что узлы дерева имеют следующую структуру:

Picture: Рис. стр. 505

Возможно, далее расположены дополнительные слова INFO. Как и в гл. 2, AVAIL есть список свободной памяти. Итак, получается следующая программа для MIX.

Программа Т. (Поиск с вставкой по дереву.) $rA \equiv K$, $rI1 \equiv P$, $rI2 \equiv Q$.

LLINK	EQU	2:3		
RLINK	EQU	4:5		
START	LDA	K	1	T1. Начальная установка.
	LD1	ROOT	1	$P \leftarrow ROOT$.
	JMP	2F	1	
4H	LD2	0,1 (RLINK)	C2	T4. Шаг вправо. $Q \leftarrow RLINK(P)$.
	J2Z	5F	C2	На T5 , если $Q = \Lambda$.
1H	ENT1	0,2	C-l	$P \leftarrow Q$.
2H	CMPA	1,1	C	T2. Сравнение.
	JG	4B	C	На T4 , если $K > KEY(P)$.
	JE	SUCCESS	C1	Выход, если $K = KEY(P)$.
	LD2	0,1 (LLINK)	C1-S	T3. Шаг влево. $Q \leftarrow LLINK(P)$.
	J2NZ	1B	C1-S	На T2 , если $Q \neq \Lambda$.
5H	LD2	AVAIL	1-S	T5 Вставки в дерево.
	J2Z	OVERFLOW	1-S	
	LDX	0,2 (RLINK)	1-S	
	STX	AVAIL	1-S	$Q \leftarrow AVAIL$.
	STA	1,2	1-S	$KEY(Q) \leftarrow K$.
	STZ	0,2	1-S	$LLINK(Q) \leftarrow RLINK(Q) \leftarrow \Lambda$.
	JL	1F	1-S	K был меньше $KEY(P)$?
	ST2	0,1 (RLINK)	A	$RLINK(P) \leftarrow Q$.
	JMP	*+2	A	
1H	ST2	0,1 (LLINK)	1-S-A	$LLINK(P) \leftarrow Q$.
DONE	EQU	*	1-S	Выход после вставки.

■

Первые 13 строк программы осуществляют поиск, последние 11 строк—вставку. Время работы поисковой фазы равно $(7C + C1 - 3S + 4)u$, где

- C = число произведенных сравнений;
 $C1$ = число случаев, когда $K \leq KEY(P)$;
 S = 1 при удачном поиске и 0 в противном случае.

В среднем имеем $C1 = \frac{1}{2}(C + S)$; действительно, $C1 + C2 = C$ и $C1 - S \approx C2$. Поэтому время поиска примерно равно $(7.5C - 2.5S + 4)u$.

Picture: Рис 11. Поиск с вставкой по дереву.

Программы бинарного поиска, использующие неявные деревья, работают дольше! (Ср. с программой 6.2.1С.) Продублировав команды, как и в программе 6.2.1F, можно избавиться от строки 08 в программе Т, уменьшив тем самым время поиска до $(6.5C - 2.5S + 5)u$. Если поиск неудачен, фаза вставки займет еще время $14u$ или $15u$.

Алгоритм Т легко приспособить к работе с ключами и записями *переменной длины*. Например, если мы распределяем имеющееся в наличии пространство последовательно, способом "последним включается—первым исключается" (LIFO), то легко сможем создавать узлы различного размера; первое слово дерева (1) может указывать длину. Такое эффективное использование памяти делает алгоритмы таблиц символов, основанные на древовидной структуре, особенно привлекательными для разработчиков компиляторов, ассемблеров и загрузчиков.

А как насчет наихудшего случая? Многие программисты сначала скептически воспринимают алгоритм Т. Если бы ключи рис. 10 поступали в алфавитном порядке AQUARIUS, ..., VIRGO, а не в календарном CAPRICORN, ..., SCORPIO, то алгоритм выстроил бы вырожденное дерево, которое, в сущности, определяет *последовательный* поиск. (Все поля LLINK равнялись бы Λ.) Аналогично, если ключи поступают в необычном порядке:

AQUARIUS, VIRGO, ARIES, TAURUS, CANCER, SCORPIO,
CAPRICORN, PISCES, GEMINI, LIBRA, LEO,

мы получим зигзагообразное дерево, что ничуть не лучше. (Постройте это дерево!)

С другой стороны, для удачного поиска по дереву рис. 10 требуется в среднем всего лишь $3\frac{2}{11}$ сравнений, что немногим больше оптимального среднего числа сравнений 3, которое достигается при поиске по наилучшему из возможных бинарных деревьев.

Для достаточно хорошо сбалансированного дерева время поиска примерно пропорционально $\log_2 N$, а для вырожденного дерева— N . В упр. 2.3.4.5-5 доказывается, что, если считать все бинарные деревья из N узлов равновероятными, среднее время поиска приблизительно пропорционально \sqrt{N} . Чего же нам ожидать от алгоритма Т?

К счастью, можно доказать, что поиск по дереву требует лишь около $2 \ln N \approx 1.386 \log_2 N$ сравнений, если ключи добавляются к дереву в случайном порядке; хорошо сбалансированные деревья—правило, а вырожденные—исключение.

Доказать этот факт удивительно просто. Предположим, что каждая из $N!$ перестановок N ключей с равной вероятностью используется для построения дерева путем вставок. Число сравнений, нужных для нахождения ключа, ровно на единицу больше числа сравнений, потребовавшихся при вставке его в дерево. Следовательно, если через C_N обозначить среднее число сравнений при удачном поиске, а через C'_N —при неудачном, мы получим

$$C_N = 1 + \frac{C'_0 + C'_1 + \dots + C'_{N-1}}{N}. \quad (2)$$

Но согласно соотношению между длинами внутреннего и внешнего путей (6.2.1-2)

$$C_N = \left(1 + \frac{1}{N}\right) C'_N - 1. \quad (3)$$

Подставляя C_N из (3) в (2), имеем

$$(N + 1)C'_N = 2N + C'_0 + C'_1 + \dots + C'_{N-1}. \quad (4)$$

Из этого рекуррентного соотношения C'_N находится легко. Вычитание равенства

$$NC'_{N-1} = 2(N - 1) + C'_0 + C'_1 + \dots + C'_{N-2}$$

дает

$$(N + 1)C'_N - NC'_{N-1} = 2 + C'_{N-1}, \\ C'_N = C'_{N-1} + 2/(N + 1).$$

Так как $C'_0 = 0$, то

$$C'_N = 2H_{N+1} - 2. \quad (5)$$

Применив (3), после упрощений получаем

$$C_N = 2 \left(1 + \frac{1}{N} \right) H_N - 3. \quad (3)$$

Упражнения 6–8 дают более детальную информацию: можно найти не только средние значения C_N и C'_N , но и их вероятностные распределения.

Сортировка вставкой в дерево. Алгоритм Т предназначался для поиска, но его можно принять за основу алгоритма внутренней *сортировки*, так как он является естественным обобщением алгоритма вставок в список 5.2.1L. Умело запрограммированный, он будет работать лишь немного медленнее лучших алгоритмов, обсуждавшихся в гл. 5. Когда построение дерева завершено, остается симметрично обойти его (алгоритм 2.3.1Т)—мы “посетим” записи в отсортированном порядке.

Однако необходимо быть осторожным. Ясно, что в шаге Т2 при $K = \text{KEY}(P)$ надо действовать по-другому, так как мы сортируем, а не ищем. Можно, например, реагировать на $K = \text{KEY}(P)$ так же, как и на $K > \text{KEY}(P)$; это даст правильный метод сортировки. (Заметим, впрочем, что равные ключи не обязательно должны находиться в смежных узлах, лишь бы они проходились последовательно при симметричном обходе.) При большом количестве повторяющихся ключей этот метод приведет к весьма несбалансированному дереву, и сортировка замедлится. Другим методом является хранение в каждом узле списка записей с одним и тем же ключом; потребуется дополнительное поле для ссылки, но это ускорит сортировку в случае, когда встречается много одинаковых ключей.

Таким образом, алгоритм Т, применяемый только для сортировки, неплох, но есть и лучшие. Если же нужно сочетать поиск и сортировку, использование древовидной структуры следует настоятельно рекомендовать.

Интересно отметить тесную связь между анализом сортировки вставкой в дерево и анализом обменной сортировки с разделением (“быстрой сортировки”), хотя на первый взгляд эти методы совершенно различны. При вставке в первоначально пустое дерево N ключей в среднем совершается то же число сравнений, что и в алгоритме 5.2.2Q (за небольшими исключениями). Например, при вставке в дерево каждый ключ сравнивается с K_1 , а каждый ключ, меньший K_1 , сравнивается с первым ключом, меньшим K_1 , и т. д.; при быстрой сортировке каждый ключ сравнивается с первым разделяющим элементом K , а затем каждый ключ, меньший K , сравнивается с определенным, меньшим K элементом и т. д. Среднее число сравнений в обоих случаях равно NC_N . (Правда, на самом деле, чтобы ускорить внутренний цикл, алгоритм 5.2.2Q совершает несколько больше сравнений.)

Удаления. Иногда бывает нужно заставить ЭВМ забыть один из элементов таблицы. Легко удалить “лист” (узел, оба поддерева которого пусты) или узел, в котором $\text{LLINK} = \Lambda$ или $\text{RLINK} = \Lambda$. Но если обе ссылки не пусты, надо действовать особым образом—ведь нельзя в одном месте хранить два указателя.

В качестве примера снова рассмотрим рис. 10. Как удалить CAPRICORN? Вот одно из решений: удалить *следующий* ближайший узел, левая ссылка которого пуста, и затем вернуть его на место узла, который действительно требуется удалить. Например, на рис. 10 сначала удаляется GEMINI, затем CAPRICORN заменяется на GEMINI. Такая операция сохраняет порядок элементов таблицы. Алгоритм D реализует эту идею.

Алгоритм D. (Удаление из дерева.) Через Q обозначим переменную, указывающую на узел бинарного дерева поиска, которое представлено как в алгоритме Т. Данный алгоритм удаляет этот узел, оставляя бинарное дерево поиска. (На самом деле мы имеем или $Q \equiv \text{ROOT}$, или $Q \equiv \text{LLINK}(P)$, или $Q \equiv \text{RLINK}(P)$ для некоторого узла P. Алгоритм изменяет в памяти значение Q в соответствии с осуществлённым удалением.)

- D1 [Ссылка RLINK пуста?] Установить $T \leftarrow Q$. Если $\text{RLINK}(T) = \Lambda$, установить $Q \leftarrow \text{LLINK}(T)$ и перейти на D4.
- D2 [Найти преемника.] Установить $R \leftarrow \text{RLINK}(T)$. Если $\text{LLINK}(R) = \Lambda$, установить $\text{LLINK}(R) \leftarrow \text{LLINK}(T)$, $Q \leftarrow R$ и перейти на D4.
- D3 [Найти пустую ссылку LLINK.] Установить $S \leftarrow \text{LLINK}(R)$. Если $\text{LLINK}(S) \neq \Lambda$, установить $R \leftarrow S$ и повторять шаг до тех пор, пока не получим $\text{LLINK}(S) = \Lambda$. (Теперь S указывает на следующий после Q элемент при симметричном обходе.) Наконец, установить $\text{LLINK}(S) \leftarrow \text{LLINK}(T)$, $\text{LLINK}(R) \leftarrow \text{RLINK}(S)$, $\text{RLINK}(S) \leftarrow \text{RLINK}(T)$, $Q \leftarrow S$.
- D4 [Освободить узел.] Выполнить $\text{AVAIL} \leftarrow T$ (т. е. вернуть узел в пул свободной памяти). ■

Читатель может испытать этот алгоритм, удаляя узлы AQUARIUS, CANCER и CAPRICORN дерева рис. 10; каждый случай имеет свои особенности. Внимательный читатель, вероятно, заметил, что отдельно не выделен случай $RLINK(T) \neq \Lambda$, $LLINK(T) = \Lambda$; мы обсудим это несколько позже, так как алгоритм в том виде, в котором он представлен здесь, обладает весьма интересными свойствами.

Поскольку в алгоритме D нет никакой симметрии между правым и левым, то кажется, что длинная последовательность случайных удалений и вставок разбалансирует дерево, и выведенные оценки эффективности потеряют силу. Но в действительности вырождения вовсе не произойдет!

Теорема Н (Т. Н. Хиббард, 1962). После удаления посредством алгоритма D случайного узла случайного дерева вновь получается случайное дерево.

[Не любящие математику, пропустите, пожалуйста, вплоть до (10)!] Такая формулировка теоремы, разумеется, весьма туманна. Опишем ситуацию более точно. Пусть \mathcal{J} — дерево из n элементов, а $P(\mathcal{J})$ — вероятность появления \mathcal{J} , если его ключи вставляются случайным образом с помощью алгоритма T. Некоторые деревья появляются с большей вероятностью, чем другие. Через $Q(\mathcal{J})$ обозначим вероятность получения \mathcal{J} после вставки в случайном порядке $n+1$ ключей (посредством алгоритма T) и последующего удаления с помощью алгоритма D одного из этих элементов, выбранного случайно. При вычислении $P(\mathcal{J})$ предполагается, что все n перестановок ключей равновероятны; при нахождении $Q(\mathcal{J})$ мы считаем, что $(n+1)(n+1)!$ перестановок ключей и выборов ключа для удаления равновероятны. Теорема утверждает, что $P(\mathcal{J}) = Q(\mathcal{J})$ для всех \mathcal{J} .

Доказательство. По условию равновероятны не деревья, а перестановки, поэтому будем доказывать теорему, рассматривая в качестве случайных объектов *перестановки*. Определим сначала удаление из перестановки и затем докажем, что "после удаления из случайной перестановки случайного элемента остается случайная перестановка".

Пусть $a_1 a_2 \dots a_{n+1}$ — перестановка множества $\{1, 2, \dots, n+1\}$; мы хотим определить операцию удаления элемента a_i , получив в результате перестановку $b_1 b_2 \dots b_n$ множества $\{1, 2, \dots, n\}$. Эта операция должна быть согласована с алгоритмами T и D, т. е. если отправиться от дерева, построенного последовательными вставками a_1, a_2, \dots, a_{n+1} , и удалить a_i с перенумерацией ключей числами от 1 до n , то получится дерево, которое могло быть построено последовательными вставками $b_1 b_2 \dots b_n$.

К счастью, определить такую операцию нетрудно. Возможны два случая:

Случай 1: $a_i = n+1$ или $a_i + 1 = a_j$ для некоторого $j < i$. (Это, в сущности, есть условие " $RLINK(a_i) = \Lambda$ ".) Удаляем a_i из последовательности и вычитаем единицу из всех элементов, больших a_i .

Случай 2: $a_i + 1 = a_j$ для некоторого $j > i$. Заменяем a_i на a_j , удаляем a_j с первоначальной позиции и вычитаем, единицу из всех элементов, больших a_i .

Например, рассмотрим перестановку 4 6 1 3 5 2. Если пометить элемент, который нужно удалить, заключив его в кружок, то получится

$$\begin{array}{l} (4) \quad 6 \quad 1 \quad 3 \quad 5 \quad 2 = 4 \quad 5 \quad 1 \quad 3 \quad 2 \quad 4 \quad 6 \quad 1 \quad (3) \quad 5 \quad 2 = 3 \quad 5 \quad 1 \quad 4 \quad 2 \\ 4 \quad (6) \quad 1 \quad 3 \quad 5 \quad 2 = 4 \quad 1 \quad 3 \quad 5 \quad 2 \quad 4 \quad 6 \quad 1 \quad 3 \quad (5) \quad 2 = 4 \quad 5 \quad 1 \quad 3 \quad 2 \\ 4 \quad 6 \quad (1) \quad 3 \quad 5 \quad 2 = 3 \quad 5 \quad 1 \quad 2 \quad 4 \quad 4 \quad 6 \quad 1 \quad 3 \quad 5 \quad (2) = 3 \quad 5 \quad 1 \quad 2 \quad 4 \end{array}$$

Поскольку всего можно сделать $(n+1)(n+1)!$ различных удалений, теорема будет установлена, если мы покажем, что каждую перестановку множества $\{1, 2, \dots, n\}$ можно получить как результат применения ровно $(n+1)^2$ удалений.

Рассмотрим перестановку $b_1 b_2 \dots b_n$ множества $\{1, 2, \dots, n\}$. Определим $(n+1)^2$ удалений, но одному для каждой пары i, j ($1 \leq i, j \leq n+1$).

Если $i < j$, искомым удалением будет

$$b'_1 \dots b'_{i-1} (b_i) b'_{i+1} \dots b'_{j-1} (b_i + 1) b'_j \dots b'_n. \quad (7)$$

Здесь и далее b'_k обозначает b_k или $b_k + 1$ в зависимости от того, меньше b_k помеченного элемента или нет. Это удаление соответствует случаю 2.

Если $i > j$, искомым удалением будет

$$b'_1 \dots b'_{i-1} (b_j) b'_i \dots b'_n, \quad (8)$$

что соответствует случаю 1.

Наконец, при $i = j$ имеем другие удаления:

Picture: Рис. стр. 512

которые тоже описываются случаем 1.

Положив $n = 4$, рассмотрим в качестве примера 25 удалений, приводящих к перестановке 3 1 4 2:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$j = 1$	(5) 3 1 4 2	4 (3) 1 5 2	4 1 (3) 5 2	4 1 5 (3) 2	4 1 5 2 (3)
$j = 2$	(3) 4 1 5 2	3 (5) 1 4 2	4 2 (1) 5 3	4 2 5 (1) 3	4 2 5 3 (1)
$j = 3$	(3) 1 4 5 2	4 (1) 2 5 3	3 1 (5) 4 2	3 1 5 (4) 2	3 1 5 2 (4)
$j = 4$	(3) 1 5 4 2	4 (1) 5 2 3	3 1 (4) 5 2	3 1 4 (5) 2	4 1 5 3 (2)
$j = 5$	(3) 1 5 2 4	4 (1) 5 3 2	3 1 (4) 2 5	4 1 5 (2) 3	3 1 4 2 (5)

Помеченный элемент всегда стоит в i -й позиции, и для фиксированного i мы построили $(n + 1)$ различных удалений, по одному для каждого j ; следовательно, для каждой перестановки $b_1 b_2 \dots b_n$ построено $(n + 1)^2$ различных удалений. Для завершения доказательства заметим, что всего имеется $(n + 1)^2!$ удалений. ■

Доказательство теоремы Н не только описывает ситуацию после удалений, но и помогает при анализе среднего времени удаления. Упражнение 12 показывает, что при удалении случайного элемента из случайной таблицы шаг D2 выполняется в среднем чуть реже, чем в половине случаев.

Рассмотрим теперь, сколько раз проходит цикл в шаге D3. Предположим, что удаляется узел, расположенный на уровне l , а *внешний* узел, непосредственно следующий за ним при симметричном обходе, находится на уровне k . Например, при удалении узла CAPRICORN (рис. 10) имеем $l = 0$ и $k = 3$, так как узел $\boxed{4}$ расположен на уровне 3. Если $k = l + 1$, то $\text{RLINK}(T) = \Lambda$ в шаге D1; если $k > l + 1$, мы будем присваивать $S \leftarrow \text{LLINK}(R)$ (шаг D3) ровно $k - l - 2$ раз. Среднее значение l равно

$$(\text{длина внутреннего пути})/N;$$

среднее значение k равно

$$(\text{длина внешнего пути})/N - (\text{расстояние до самого левого внешнего узла})/N.$$

Расстояние до самого левого внешнего узла равно количеству таких a_i из вставляемой последовательности, что $a_i < a_j$, $1 \leq j < i$ (включая a_1); среднее значение этой величины, согласно п. 1.2.10, есть H_N . Так как длина внешнего пути на $2N$ больше длины внутреннего, среднее значение $k - l - 2$ равно $-H_N/N$. Добавляя сюда среднее количество случаев, когда $k - l - 2$ есть -1 , получаем, что при случайном удалении операция $S \leftarrow \text{LLINK}(R)$ в шаге D3 выполняется в среднем лишь

$$\frac{1}{2} + \frac{\frac{1}{2} - H_N}{N} \quad (10)$$

раз. Это успокаивает, так как в наихудшем случае удаления могут производиться довольно медленно (см. упр. 11).

Выше отмечалось, что в алгоритме D нет проверки $\text{LLINK}(T) = \Lambda$, хотя удалить такую вершину легко. Можно добавить новый шаг между D1 и D2:

D1 $\frac{1}{2}$. [Ссылка LLINK пуста?] Если $\text{LLINK}(T) = \Lambda$, установить $Q \leftarrow \text{RLINK}(T)$ и перейти на D4.

Упражнение 14 показывает, что дерево после работы алгоритма D с дополнительным шагом имеет не только не большую, а иногда даже меньшую длину пути, чем после работы обычного алгоритма D. Так что последовательность вставок и удалений с помощью модификации алгоритма D приводит к деревьям, которые в действительности *лучше*, чем предсказывает теория случайных деревьев; среднее время поиска с вставкой имеет тенденцию к уменьшению.

Частота обращений. До сих пор предполагалось, что все ключи с равной вероятностью могут быть аргументами поиска. Рассмотрим теперь более общий случай, когда с вероятностью p_k отыскивается элемент, k -м вставленный в таблицу, причем $p_1 + \dots + p_N = 1$. При сохранении предположения о случайном порядке (форма дерева остается случайной) модификация выражения (2) показывает, что среднее число сравнений в процессе удачного поиска составляет

$$1 + \sum_{1 \leq k \leq N} p_k (2H_k - 2) = 2 \sum_{1 \leq k \leq N} p_k H_k - 1. \quad (11)$$

(Ср. с (5).)

Например, если вероятности подчиняются закону Зипфа (6.1-8), среднее число сравнений сводится к

$$H_N - 1 + H_N^{(2)} / H_N \quad (12)$$

(предполагается, что мы вставляем ключи в порядке уменьшения важности; см. упр. 18). Формула (6) предсказывает примерно в два раза большую величину; при бинарном поиске мы также произвели бы больше сравнений. Например, на рис. 12 изображено дерево, полученное последовательными вставками в порядке убывания частот 31 наиболее употребительного английского слова. Относительная частота помещена под каждым словом. [Ср. Н. F. Gaines, Cryptanalysis (New York: Dover, 1956), 226.] Среднее число сравнений при

Picture: Рис. 12. 31 наиболее употребительное английское слово; вставки в порядке уменьшения частот.

удачном поиске по такому дереву равно 4.042; соответствующий бинарный поиск, использующий алгоритм 6.2.1В или С, потребовал бы 4.393 сравнения.

Оптимальные бинарные деревья поиска. Уместно спросить, какое дерево является наилучшим для поиска по таблице, если ключи запрашиваются с данными частотами? Например, оптимальное дерево для 31 наиболее употребительного английского слова изображено на рис. 13; в среднем на удачный поиск требуется 3.437 сравнения.

Исследуем проблему нахождения оптимального дерева. В случае $N = 3$ и при соответственных вероятностях p, q, t ключей

Picture: Рис. 13. Оптимальное дерево поиска для 31 наиболее употребительного английского слова.

$K_1 < K_2 < K_3$ возможны пять деревьев:

Picture: Рис. стр. 516

На рис. 14 показаны диапазоны величин p, q, r , для которых каждое из деревьев оптимально; сбалансированное дерево является наилучшим примерно в 45% случаев, если p, q, r случайны (см. упр. 21).

Picture: Рис. 14. Здесь (p, q, r) суть относительные частоты (k_1, k_2, k_3) , этот рисунок показывает; какое из пяти деревьев (13) является наилучшим. Тот факт, что $p + q + r = 1$, делает рисунок двумерным, несмотря на наличие трех координат.

К сожалению, при больших N существует

$$\binom{2N}{N} / (N + 1) \approx 4^N / (\sqrt{\pi} N^{3/2})$$

бинарных деревьев, так что мы не в состоянии все их перебрать и выбрать наилучшее. Поэтому, чтобы придумать более удачный способ отыскивать оптимальные бинарные деревья поиска, нужно ближе познакомиться с их свойствами.

До сих пор мы изучали лишь вероятности при удачном поиске; оказывается, случай неудачи столь же важен. Например, 31 слово (рис. 13) составляет лишь около 36% типичного английского текста; остальные 64%, несомненно, повлияют на структуру оптимального дерева поиска.

Следовательно, поставим задачу таким образом: дано $2n + 1$ чисел p_1, p_2, \dots, p_n и q_0, q_1, \dots, q_n , где

p_i = вероятность того, что аргументом поиска является K_i ;

q_i = вероятность того, что аргумент поиска лежит между K_i и K_{i+1} .

(По определению q_0 есть вероятность того, что аргумент поиска меньше K_1 , а q_n есть вероятность того, что аргумент поиска больше K_n .) Таким образом, $p_1 + p_2 + \dots + p_n + q_0 + q_1 + \dots + q_n = 1$, и мы хотим найти бинарное дерево, минимизирующее математическое ожидание числа сравнений при поиске, а именно

$$\sum_{1 \leq j \leq n} p_j (\text{уровень}(\overline{j}) + 1) + \sum_{0 \leq k \leq n} q_k (\text{уровень}(\overline{k})), \quad (14)$$

где (j) есть j -й внутренний узел при симметричном обходе, а \boxed{k} есть $(k + 1)$ -й внешний узел; корень находится на нулевом уровне. Так, математическое ожидание числа сравнений для бинарного дерева

Picture: Рис. стр. 517.

равно $2q_0 + 2p_1 + 3q_1 + 3p_2 + 3q_2 + p_3 + q_3$. Назовем это *ценой* дерева; скажем, что дерево с минимальной ценой *оптимально*. При таком определении нет нужды требовать, чтобы p и q давали в сумме единицу, можно искать дерево с минимальной ценой для данной последовательности "весов" $(p_1, \dots, p_n; q_0, \dots, q_n)$.

В п. 2.3.4.5 мы изучили процедуру Хаффмэна для построения деревьев с минимальной взвешенной длиной пути, но этот метод требует, чтобы все p равнялись нулю, а в порожденном им дереве внешние узлы, имеющие веса (q_0, \dots, q_n) , обычно не располагаются в нужном симметричном порядке слева направо. Значит, необходимо действовать иначе.

Воспользуемся следующим принципом: *любое поддереву оптимального дерева оптимально*. Например, если (15) есть оптимальное дерево для весов $(p_1, p_2, p_3; q_0, q_1, q_2, q_3)$, то левое поддерево корня должно быть оптимальным для $(p_1, p_2; q_0, q_1, q_2)$, т. е. любое улучшение поддерева ведет к улучшению всего дерева.

Этот принцип подсказывает вычислительную процедуру, с помощью которой систематически находятся всё большие и большие оптимальные поддеревья. Мы воспользовались абсолютно той же идеей в п. 5.4.9 для построения схем оптимального слияния; общий подход, известный под названием "динамическое программирование", будет изучен в гл. 7.

Через $c(i, j)$ обозначим цену оптимального поддерева с весами $(p_{i+1}, \dots, p_j, q_i, \dots, q_j)$, и пусть $w(i, j) = p_{i+1} + \dots + p_j + q_i + \dots + q_j$ будет суммой этих весов; $c(i, j)$ и $w(i, j)$ определяются для $0 \leq i \leq j \leq n$. Так как минимально возможная цена дерева с корнем (k) равна $w(i, j) + c(i, k - 1) + c(k, j)$, имеем

$$\begin{aligned} c(i, i) &= 0; \\ c(i, j) &= w(i, j) + \min_{i \leq k < j} (c(i, k - 1) + c(k, j)) \quad \text{при } i < j. \end{aligned} \quad (16)$$

Если $i < j$, то через $R(i, j)$ обозначим множество всех k , при которых достигается минимум в (16); это множество определяет возможные корни оптимальных поддеревьев.

Соотношения (16) позволяют найти значения $c(i, j)$ для $j - i = 1, 2, 3, \dots, n$; существует примерно $\frac{1}{2}n^2$ таких значений, а операция минимизации выполняется для приблизительно $\frac{1}{6}n^3$ величин k . Это значит, что оптимальное дерево можно найти за $O(n^3)$ единиц времени, используя $O(n^2)$ ячеек памяти.

В оценке времени работы показатель степени при n можно уменьшить, если воспользоваться свойством "монотонности". Пусть $r(i, j)$ обозначает элемент из $R(i, j)$; нет нужды находить все множество $R(i, j)$, нам достаточно одного представителя. В силу результата упр. 27 после нахождения $r(i, j - 1)$ и $r(i + 1, j)$ элемент $r(i, j)$ можно искать в пределах

$$r(i, j - 1) \leq r(i, j) \leq r(i + 1, j) \quad (17)$$

при условии неотрицательности весов. Теперь в (16) вместо $i - j$ нужно проверять лишь $r(i + 1, j) - r(i, j - 1) + 1$ значений k . Полный объем работы, если $j - i = d$, оценивается суммой

$$\sum_{\substack{d \leq j \leq n \\ i = j - d}} (r(i + 1, j) - r(i, j - 1) + 1) = r(n - d + 1, n) + r(0, d - 1) + n - d + 1 < 2n,$$

поэтому время выполнения уменьшается до $O(n^2)$.

Предложенная процедура подробно описана ниже.

Алгоритм К. (Нахождение оптимальных бинарных деревьев поиска.) Даны неотрицательные веса $(p_1, \dots, p_n; q_0, q_1, \dots, q_n)$. Алгоритм позволяет построить бинарные деревья $t(i, j)$, имеющие минимальную (в указанном выше смысле) цену для весов $(p_{i+1}, \dots, p_j; q_i, \dots, q_j)$. Вычисляются три массива:

$$\begin{aligned} c[i, j] &\text{— цена } t(i, j), & 0 \leq i \leq j \leq n; \\ r[i, j] &\text{— корень } t(i, j), & 0 \leq i \leq j \leq n; \\ w[i, j] &\text{— полный вес } t(i, j), & 0 \leq i \leq j \leq n; \end{aligned}$$

Результаты алгоритма определяются массивом r : если $i = j$, то $t(i, j)$ пусто; если $i \neq j$, то левым поддеревом является $t(i, r[i, j] - 1)$, а правым — $t(r[i, j], j)$.

- К1** [Начальная установка.] Для $0 \leq i \leq n$ установить $c[i, i] \leftarrow 0$, $w[i, i] \leftarrow q_i$ и $w[i, j] \leftarrow w[i, j-1] + p_j + q_j$ при $j = i + 1, \dots, n$. Затем для $1 \leq j \leq n$ установить $c[j-1, j] \leftarrow w[j-1, j]$ и $r[j-1, j] \leftarrow j$. (Этим определяются все оптимальные деревья, состоящие из одного узла.)
- К2** [Цикл по d .] Выполнить шаг **К3** для $d = 2, 3, \dots, n$; после этого алгоритм завершает работу.
- К3** [Цикл по j .] (**К** этому, моменту найдены оптимальные деревья с менее чем d узлами. Шаг определяет все оптимальные деревья с d узлами.) Выполнить шаг **К4** для $j = d, d+1, \dots, n$.
- К4** [Найти $c[i, j]$, $r[i, j]$.] Установить $i \leftarrow j - d$, затем установить

$$c[i, j] \leftarrow w[i, j] + \min_{r[i, j-1] \leq k \leq r[i+1, j]} (c[i, k-1] + c[k, j])$$

и занести в $r[i, j]$ величину k , при которой достигается минимум. (В упр. 22 доказывается, что $r[i, j-1] \leq r[i+1, j]$.) ■

Как пример работы алгоритма **К** рассмотрим рис. 15, связанный с разработкой указателя KWIC ("key-word-in-context"—"ключевое слово в контексте"). Заголовки всех статей первых десяти томов *ACM Journal* были отсортированы таким образом, чтобы каждому слову каждого заголовка соответствовала одна строка. Правда, некоторые слова вроде "THE" и "EQUATION"

Picture: Рис 15. Оптимальное бинарное дерево поиска для применения указателя KWIC.

были признаны неинформативными и не попали в указатель. Эти специальные слова и частоты их появлений представлены на рис. 15 внутренними узлами. Заметим, что такое название, как "On the solution of an equation for a certain new problem" ("О решении уравнения для некоторой новой задачи"), было бы настолько неинформативным, что вообще не попало бы в указатель! Идея указателя KWIC принадлежит Г. П. Лану [*Amer. Documentation*, 11 (1960), 288–295]. Полностью указатель KWIC приведен в работе [W. W. Youden, *JACM*, 10 (1963), 583–646].

При подготовке указателя KWIC можно было использовать бинарное дерево поиска для проверки, нужно ли вносить то или иное слово в указатель. Частотам появления слов, включенных в указатель, соответствуют внешние узлы дерева рис. 15; так, в заголовках статей *JACM* за 1954–1963 гг. встретились ровно 277 слов, расположенных по алфавиту между "PROBLEMS" и "SOLUTION".

На рис. 15 изображено оптимальное дерево, полученное с помощью алгоритма **К** при $n = 35$. Значения $r[0, j]$, вычисленные для $j = 1, 2, \dots, 35$, равны (1, 1, 2, 3, 3, 3, 3, 8, 8, 8, 8, 8, 8, 11, 11, ..., 11, 21, 21, 21, 21, 21, 21); значения $r[i, 35]$ для $i = 0, 1, \dots, 34$ равны (21, 21, ..., 21, 25, 25, 25, 25, 25, 25, 26, 26, 26, 30, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 35, 35).

"Промежуточные частоты" q_j заметно влияют на оптимальную структуру дерева; на рис. 16 (а) приведено оптимальное дерево, которое получилось бы при нулевых q_j . Так же важны и внутренние частоты p_i ; на рис. 16 (б) изображено оптимальное дерево при нулевых p_i . Вернемся к полному набору частот. Дерево рис. 15 требует в среднем лишь 4.75 сравнения, а деревья рис. 16—соответственно 5.29 и 5.32. (Прямой бинарный поиск в данном случае был бы лучше деревьев рис. 16.)

Так как алгоритм **К** требует $O(n^2)$ времени и пространства, его использование при больших n становится нерациональным. Конечно, в этом случае можно бы использовать другие методы поиска, и далее в этой главе они будут обсуждаться. Давайте предположим, что нужно найти оптимальное или почти оптимальное дерево, если n велико.

Мы видели, что идея вставки ключей в порядке уменьшения частот в среднем ведет к довольно хорошим деревьям; но дерево может получиться и очень плохим (см. упр. 20), так как не используются веса q_j . Другой подход состоит в выборе корня k таким образом, чтобы максимальный вес получающегося поддерева $\max(w(0, k-1), w(k, n))$ был как можно меньше. Этот подход также неудачен, ибо в качестве корня может быть выбран узел с очень малым p_k . Однако Пол Бэйер доказал, что получившееся дерево будет иметь взвешенную длину пути, близкую к оптимальной.

У. Э. Уолкер и К. К. Готлиб [*Graph Theory and Computing* (Academic Press, 1972)] предложили более удовлетворительную процедуру, сочетающую оба рассмотренных метода: попытайтесь уравнять левосторонние и правосторонние веса, но будьте готовы передвинуть корень на несколько шагов вправо или влево для нахождения узла с относительно большим весом p_k . Рисунок 17

Picture: Рис. 16. Оптимальные бинарные деревья поиска, основанные на половине данных рис. 15, (а) не учитываются внешние частоты; (б) не учитываются внутренние частоты.

показывает, почему этот метод разумен: если мы изобразим $c(0, k-1) + c(k, n)$ как функцию от k для данных KWIC (см. рис. 15), то увидим, что результат очень чувствителен к величине p_k .

Такой метод "сверху вниз" можно использовать при больших n для выбора корня и для обработки левых и правых поддеревьев. Если получится достаточно маленькое поддерево, можно применить алгоритм К. Результирующий метод дает довольно хорошие деревья (согласно сообщениям), на 2–3% хуже оптимума, а требует лишь $O(n)$ единиц пространства и $O(n \log n)$ единиц

Picture: Рис. 17. Поведение цены как функции корня k (слева по оси ординат отложен минимум средней цены).

времени. На самом деле М. Фредмэн показал, что достаточно $O(n)$ единиц времени, если использовать подходящие структуры данных [АСМ. Symp. Theory of Comp., 7 (1975), 240–244].

***Алгоритм Ху-Такера.** В специальном случае, когда все p равны нулю, Т. Ч. Ху и Э. К. Такер открыли замечательный способ построения оптимальных деревьев "снизу вверх"; при использовании подходящих структур данных их метод требует $O(n)$ единиц пространства и $O(n \log n)$ единиц времени и строит дерево, не близкое к оптимальному, а *действительно* оптимальное. Алгоритм Ху-Такера можно описать так.

• **ФАЗА 1.** Комбинация. Начните с "рабочей последовательности" весов, написанных внутри *внешних* узлов

Picture: Рис. стр. 523

Затем раз за разом комбинируйте два веса q_i и q_j для $i < j$, получая единственный вес $q_i + q_j$, исключите q_j из последовательности и замените узел, содержащий q_i , на *внутренний* узел

Picture: Рис. стр. 523

Комбинировать нужно единственную пару весов (q_i, q_j) , удовлетворяющую следующим правилам:

- i) Между q_i и q_j нет внешних узлов. (Это наиболее важное правило, отличающее данный алгоритм от метода Хаффмана.)
- ii) Сумма $q_i + q_j$ минимальна среди всех (q_i, q_j) , удовлетворяющих правилу (i).
- iii) Индекс i минимален среди всех (q_i, q_j) , удовлетворяющих правилам (i), (ii).
- iv) Индекс j минимален среди всех (q_i, q_j) , удовлетворяющих правилам (i)–(iii).

• **ФАЗА 2.** Присваивание уровней. Когда фаза 1 окончена, в рабочей последовательности остается единственный узел. Пометьте его номером уровня 0. Затем проделайте шаги фазы 1 в обратном порядке, таким образом пометая узлы номерами уровней, что, если (19) имеет уровень l , узлы q_i и q_j , породившие (19), получают номер $l + 1$.

• **ФАЗА 3.** Рекомбинация. Теперь у нас есть последовательность внешних узлов и уровней

Picture: Рис. стр. 524

Внутренние узлы, использованные в фазах 1 и 2, теперь отбросьте за ненадобностью—мы будем создавать новые узлы путем комбинации весов (q_i, q_j) по следующим новым правилам:

- i') Узлы, содержащие q_i и q_j , должны быть соседними в рабочей последовательности.
- ii') Оба уровня l_i и l_j должны быть максимальными среди всех остающихся.
- iii') Индекс i должен быть минимальным среди всех (q_i, q_j) , удовлетворяющих (i') и (ii').

Новому узлу (19) присваивается уровень $l_i - 1$. Бинарное дерево, формируемое в этой фазе, имеет минимальную взвешенную длину пути среди всех бинарных деревьев, внешние узлы которых имеют веса (слева направо) q_0, q_1, \dots, q_n .

Рисунок 18 иллюстрирует работу этого алгоритма; весами являются относительные частоты букв \square, A, B, \dots, Z в английском тексте. В течение фазы 1 первым формируется узел (6), содержащий частоты букв J и K, затем узел (16) (комбинируем P и Q), затем

Picture: Рис. стр. 524

Picture: Рис. 18. Алгоритм Ху-Такера, примененный к данным о частотах букв; фазы 1 и 2.

в этот момент мы имеем рабочую последовательность

Picture: Рис. стр. 526

Правило (i) позволяет комбинировать несмежные веса, только если они разделены внутренними узлами; так что можно скомбинировать $57 + 57$, затем $63 + 51$, затем $58 + 64$ и т. д.

Номера уровней, присвоенные во время фазы 2, изображены на рис. 18 справа от каждого узла. В результате рекомбинации (фаза 3) мы получаем дерево, показанное на рис. 19; заметьте, что деревья рис. 18 и 19 существенно различны: рис. 18 не сохраняет упорядоченности слева направо. Но цены у этих деревьев одинаковы, так как внешние узлы расположены на одинаковых уровнях.

Рассмотрим простой пример, где веса равны 4, 3, 2, 4; легко показать, что единственным оптимальным деревом является

Picture: Рис. стр. 526

На этом примере видно, что в оптимальном дереве два наименьших веса 2 и 3 *не* всегда следует комбинировать в одно оптимальное дерево, даже если они приписаны смежным узлам; нужна некоторая рекомбинационная фаза.

В нашей книге не приводится доказательство справедливости алгоритма Ху-Такера; простые доказательства не известны, и очень вероятно, что они никогда не будут найдены! Чтобы проиллюстрировать внутренне присущую данной ситуации сложность, заметим, что фаза 3 должна скомбинировать все узлы в одно дерево, но возможность этого не очевидна. Например, предположим, что фазы 1 и 2 привели к дереву

Picture: Рис. стр. 526

Picture: Рис. 19. Алгоритм Ху-Такера, примененный к данным о частотах букв; фаза 3.

(т. е. были получены узлы (a), (b), (c), (d), (e) в указанном порядке); это согласуется с правилом (i). Затем после формирования узлов

Picture: Рис. стр. 528

мы не сможем продолжить фазу 3, так как узлы уровня 3 несмежные! Правило (i) само по себе не гарантирует благополучного завершения фазы 3, и необходимо доказывать, что конфигурация, подобная (22), *никогда* не получается на фазе 1.

При реализации алгоритма Ху-Такера можно хранить приоритетные очереди для множества весов в узлах, между которыми нет внешних узлов. Например, (20) можно было бы представить приоритетными очередями, содержащими соответственно

Picture: Рис. стр. 528

и информацией о том, какие из узлов внешние, а также указанием на порядок слева направо (это нужно для применения правил (iii) и (iv)). В другой, "управляющей" приоритетной очереди можно запоминать суммы двух наименьших элементов очередей. Создание нового узла $57 + 57$ вызывает слияние трех очередей. Если приоритетные очереди представлены левосторонними деревьями (см. п. 5.2.3), то каждый шаг фазы 1 требует не более $O(\log n)$ операций; значит, при $n \rightarrow \infty$ достаточно $O(n \log n)$ операций. Конечно, при небольших n более эффективным будет относительно более прямолинейный $O(n^2)$ -способ реализации.

Оптимальное бинарное дерево рис. 19 полезно не только для поиска, но и для теории кодирования: используя 0 для обозначения левой ветви дерева и 1 для правой, получаем следующие коды различной длины:

□	000	I	1000	R	11001
A	0010	J	1001000	S	1101
B	001100	K	1001001	T	1110
C	001101	L	100101	U	111100
D	00111	M	10011	V	111101
E	010	N	1010	W	111110
F	01100	O	1011	X	11111100
G	01101	P	110000	Y	11111101
H	0111	Q	110001	Z	1111111

Таким образом, сообщение вроде "RIGHT ON" можно закодировать цепочкой

110011000011010111111000010111010.

Заметим, что переменная длина кодов не затрудняет расшифровку слева направо, так как структура дерева подсказывает, когда кончается код одной буквы и начинается другой. Такой метод кодирования сохраняет алфавитный порядок и использует для кодирования одной буквы в среднем около 4.2 битов. Этот код можно использовать для уплотнения файлов данных без нарушения лексикографического порядка буквенной информации. (Число 4.2 минимально для кодирования с помощью бинарных деревьев, хотя его можно уменьшить до 4.1 битов на букву, если отказаться от сохранения алфавитного порядка. Уменьшение с сохранением алфавитного порядка достигается кодированием не отдельных букв, а пар букв.)

Интересная асимптотическая оценка минимальной взвешенной длины пути деревьев поиска выведена Э. Н. Гилбертом и Э. Ф. Муром:

Теорема Г. Если $p_1 = p_2 = \dots = p_n = 0$, то взвешенная длина пути оптимального бинарного дерева поиска заключена между

$$\sum_{0 \leq i \leq n} q_i \log_2(Q/q_i) \text{ и } 2Q + \sum_{0 \leq i \leq n} q_i \log_2(Q/q_i),$$

где $Q = 2 \sum_{0 \leq i \leq n} q_i$.

Доказательство. Для получения нижней оценки используем индукцию по n . При $n > 0$ взвешенная длина внешнего пути не меньше

$$Q + \sum_{0 \leq i < k} q_i \log_2(Q_1/q_i) + \sum_{k \leq i \leq n} q_i \log_2((Q - Q_1)/q_i) \geq \sum_{0 \leq i \leq n} q_i \log_2(Q/q_i) + f(Q_1)$$

для некоторого k , где

$$Q_1 = \sum_{0 \leq i < k} q_i$$

и

$$f(Q_1) = Q + Q_1 \log_2 Q_1 + (Q - Q_1) \log_2(Q - Q_1) - Q \log_2 Q.$$

Функция $f(Q_1)$ неотрицательна и принимает наименьшее значение 0 при $Q_1 = \frac{1}{2}Q$.

Для получения верхней границы можем считать, что $Q = 1$. Пусть e_0, \dots, e_n — целые числа, такие, что $2^{-e_i} \leq q_i < 2^{1-e_i}$, $0 \leq i \leq n$. Построим коды C_i из нулей и единиц, используя $e_i + 1$ старших значащих цифр дроби $\sum_{0 \leq k < i} q_k + \frac{1}{2}q_i$, выраженной в двоичной системе. В упр. 35 доказывается, что C_i не является начальной подцепочкой C_j при $i \neq j$; отсюда следует, что мы можем построить бинарное дерево поиска, соответствующее этим кодам. Возвращаясь к примеру рис. 19, можно показать, что $C_0 = 0001$, $C_1 = 00110$, $C_2 = 01000001$, $C_3 = 0100011$ и т. д.; дерево начинается с

Picture: Рис. стр. 530

(Избыточные биты в конце кодов часто можно отбросить.) Взвешенная длина пути бинарного дерева, построенного такой процедурой, будет

$$\leq \sum_{0 \leq i \leq n} (e_i + 1)q_i < \sum_{0 \leq i \leq n} q_i(2 + \log_2(1/q_i)). \blacksquare$$

Методом, использованным в первой части доказательства, легко установить, что взвешенная длина пути *любого* бинарного дерева не меньше, чем $\sum_{0 \leq i \leq n} q_i \log_2(Q/q_i)$, независимо от порядка весов слева направо. (Этот фундаментальный результат принадлежит Клоду Шеннону.) Таким образом, сохранение порядка весов слева направо увеличивает цену оптимального дерева не больше, чем на цену двух дополнительных уровней, т. е. на удвоенную сумму весов.

История и библиография. Рассмотренные методы поиска по дереву были открыты независимо несколькими исследователями в начале 50-х годов. В неопубликованном документе, датированном августом 1952 г., А. И. Думи описал вставки в дерево в простейшем виде:

”Пусть имеется барабан, на котором записано 2^n элементов, и каждый из них имеет бинарный адрес. Придерживайтесь такой программы:

- 1) Прочитайте первый элемент и поместите его по адресу 2^{n-1} , т. е. в середину массива памяти.
- 2) Прочитайте следующий элемент. Сравните его с первым.
- 3) Если он больше, поместите его по адресу $2^{n-1} + 2^{n-2}$.

Если же он меньше, поместите по адресу $2^{n-2} \dots$ ”

Другая ранняя форма вставки в дерево была введена Д. Дж. Уилером; он, в сущности, допускал многопутевые разветвления, подобные тем, которые будут обсуждаться в п. 6.2.4; метод вставки в бинарное дерево был также независимо придуман К. М. Бернесом-Ли [Comp. J., 2 (1959), 5].

Первые опубликованные описания вставки в дерево принадлежат П. Ф. Уиндли [Comp. J., 3 (1960), 84–88], Э. Д. Буту и Э. Колину [Information and Control, 3 (1960), 327–334] и Т. Н. Хиббарду [JACM, 9 (1962), 13–28]. Вероятно, все авторы развивали этот метод независимо, и их доказательства формулы (6) среднего числа сравнений несколько отличаются друг от друга. В последующих работах они продолжали исследовать различные аспекты этого алгоритма: Уиндли подробно разобрал сортировку вставкой в дерево, Бут и Колин обсуждали эффект предварительного построения из первых $2^n - 1$ элементов идеально сбалансированного дерева (см. упр. 4), Хиббард предложил идею удаления и выявил связь между анализом вставки в дерево и анализом быстрой сортировки.

Идея *оптимальных* бинарных деревьев поиска была впервые развита для частного случая $p_1 = \dots = p_n = 0$ в связи с бинарным кодированием букв. В очень интересной статье Э. Гилберта и Э. Мура [Bell System Tech. J., 38 (1959), 933–968] обсуждалась эта проблема и ее связь с другими задачами кодирования. Гилберт и Мур отметили, что оптимальное дерево можно построить за $O(n^3)$ шагов, используя метод, подобный алгоритму К, но без соотношения (17). К. Э. Айверсон [A Programming Language (Wiley, 1962), 142–144] независимо рассмотрел *другой* случай, когда все q равны 0. Он предположил, что дерево будет оптимальным, если корень выбрать так, чтобы как можно лучше уравнивать вероятности левого и правого поддеревьев; мы видели, что, к сожалению, эта идея не работает. Д. Э. Кнут [Acta Informatica, 1 (1971), 14–25, 270] впоследствии рассмотрел случай любых весов p и q и доказал, что алгоритм может быть сведен к $O(n^2)$ шагам; он также привел пример применения рассмотренных методов к компилятору, где ключами в дереве являются “резервированные слова” алголоподобного языка. Т. Ч. Ху несколько лет занимался своим собственным алгоритмом для случая $p = 0$; из-за сложности задачи было трудно найти строгое доказательство справедливости этого алгоритма, но в конце концов в 1969 г. Т. Ч. Ху и Э. К. Такер справились с этим. [SIAM J. Applied Math., 21 (1971), 514–532.]

Упражнения

1. [15] Алгоритм Т был сформулирован лишь для непустых деревьев. Какие нужно внести изменения, чтобы он обрабатывал и пустые деревья?
2. [20] Измените алгоритм Т так, чтобы он работал с *правопроштыми* деревьями (ср. с п. 2.3.1; по таким деревьям проще совершать симметричный обход).
- >3. [20] В § 6.1 мы видели, что небольшое изменение алгоритма последовательного поиска 6.1S делает его быстрее (алгоритм 6.1Q). Можно ли подобную хитрость использовать для ускорения работы алгоритма Т?
4. [M24] (Э. Д. Бут и Э. Колин.) Даны N ключей в случайном порядке; предположим, что мы используем первые $2^n - 1$ для построения идеально сбалансированного дерева, помещая 2^k ключей на уровень k , $0 \leq k < n$. Затем мы применяем алгоритм Т для вставки оставшихся ключей. Найдите среднее число сравнений при удачном поиске. [Указание: видеоизменить (2).]
- >5. [M25] Названия CAPRICORN, AQUARIUS и т. д. можно упорядочить $11! = 39916800$ способами для вставки в бинарное дерево поиска. (а) Сколько таких перестановок приведут к рис. 10? (б) Сколько перестановок приведут к вырожденному дереву, т. е. во всех узлах LLINK или RLINK будут равняться Λ ?
6. [M26] Обозначим через P_{nk} количество перестановок $a_1 a_2 \dots a_n$ множества $\{1, 2, \dots, n\}$, таких, что, если алгоритм Т используется для последовательной вставки a_1, a_2, \dots, a_n в первоначально пустое дерево, ровно k сравнений производится при вставке a_n . [В этой задаче мы пренебрежем сравнениями, производимыми при вставке a_1, \dots, a_{n-1} . В принятых в тексте обозначениях $C'_{n-1} = (\sum k P_{nk})/n!$, так как это есть среднее число сравнений, которые производятся при неудачном поиске по дереву из $n - 1$ элементов.]
 - а) Докажите, что $P_{n+1,k} = 2P_{n,k-1} + (n-1)P_{n,k}$. [Указание: посмотрите, оказывается ли a_{n+1} ниже a_n .]
 - б) Найдите простую формулу для производящей функции $G_n(z) = \sum_k P_{nk} z^k$ и используйте ее для выражения P_{nk} через числа Стирлинга.
 - с) Чему равна дисперсия C'_{n-1} ?
7. [M30] (С. Р. Арора и У. Т. Дэнт.) Какое среднее число сравнений потребуется для отыскания m -го по величине элемента после вставки в первоначально пустое дерево n элементов в случайном порядке?

8. [M38] Пусть $p(n, k)$ есть вероятность того, что полная длина внутреннего пути дерева, построенного алгоритмом Т из n случайно упорядоченных ключей, равна k (длина внутреннего пути дерева равна числу сравнений, производимых при сортировке вставкой в дерево по мере его построения). (a) Найдите рекуррентное соотношение, определяющее соответствующую производящую функцию. (b) Вычислите дисперсию этого распределения. [Здесь могут быть полезны некоторые упражнения п. 1.2.7!]
9. [41] Мы доказали, что поиск с вставкой по дереву требует лишь около $2 \ln N$ сравнений, если ключи вставляются в случайном порядке; но в действительности порядок может не быть случайным. Исследуйте эмпирически, насколько хорошо вставка в дерево обслуживает таблицы символов компилятора и/или загрузчика. Дадут ли идентификаторы типичной большой программы достаточно хорошо сбалансированное бинарное дерево поиска?
- >10. [22] Предположим, что программиста не интересует порядок ключей в бинарном дереве поиска, но можно ожидать, что ввод будет производиться не в случайном порядке. Обсудите методы, позволяющие тем не менее использовать поиск по дереву, имитируя случайный порядок ввода.
11. [20] Найдите максимальное количество присваивания $S \leftarrow \text{LLINK}(R)$ (в шаге D3) при удалении узла из дерева размера N .
12. [M22] Как часто (в среднем) в шаге D1 происходит передача управления на D4, если удаляется случайный узел из случайного дерева размера N ? (См. доказательство теоремы Н.)
- >13. [M23] Будет ли случайным дерево, полученное в результате удаления корня случайного дерева посредством алгоритма D?
- >14. [22] Докажите, что длина пути дерева, порожденного алгоритмом D с дополнительным шагом $D1_{\frac{1}{2}}$, не больше, чем длина пути дерева, порожденного немодифицированным алгоритмом. Найдите случай, когда шаг $D1_{\frac{1}{2}}$ действительно уменьшает длину пути.
15. [M47] Найдите точное изменение среднего времени работы алгоритма Т после длинной последовательности вставок и удалений, если к алгоритму D добавляется новый шаг $D1_{\frac{1}{2}}$.
- >16. [25] Является ли операция удаления *коммутативной*? То есть получится ли одно и то же дерево, если с помощью алгоритма D удалить узлы X и Y ; если удалить Y и X ?
17. [25] Покажите, что если в алгоритме D полностью заменить правое на левое и обратно, то его легко приспособить для удаления данного узла из *правопрошито*го дерева с сохранением необходимых нитей. (Ср. с упр. 2.)
18. [M21]. Покажите, что закон Зипфа приводит к (12).
19. [M23] Найдите приближенно среднее число сравнений (11), если вероятности p_k удовлетворяют закону "80-20", определяемому формулами (6.1-11) и (6.1-12).
20. [M20] Пусть мы вставляем в дерево ключи в порядке уменьшения частот $p_1 \geq p_2 \geq \dots \geq p_N$. Может ли это дерево быть существенно хуже оптимального дерева поиска?
21. [M20] Пусть p, q и r —случайно выбранные вероятности, подчиняющиеся условию $p + q + r = 1$. Чему равны вероятности того, что деревья I, II, III, IV или V (см. (13)) являются оптимальными? (Рассмотрите отношения площадей соответствующих областей на рис. 14.)
22. [M20] Докажите, что при выполнении шага K4 алгоритма K $r[i, j - 1] \leq r[i + 1, j]$.
- >23. [M23] Найдите (не прибегая к помощи ЭВМ) оптимальное бинарное дерево поиска, если $n = 40$, $p_1 = 5, p_2 = p_3 = \dots = p_{40} = 1, q_0 = q_1 = \dots = q_{40} = 0$.
24. [M25] Пусть $p_n = q_n = 0$, а все остальные веса неотрицательны. Докажите, что оптимальное дерево для $(p_1, \dots, p_n; q_0, \dots, q_n)$ можно получить заменой

Picture: Рис. стр. 533

в любом оптимальном для $(p_1, \dots, p_{n-1}; q_0, \dots, q_{n-1})$ дереве.

25. [M20] Пусть A и B —непустые множества действительных чисел. По определению $A \leq B$, если выполняется следующее свойство: ($a \in A, b \in B, b < a$) влечет ($a \in B$ и $b \in A$). (a) Докажите, что это отношение непустых множеств транзитивно. (b) Докажите или опровергните, что $A \leq B$ тогда и только тогда, когда $A \leq A \cup B \leq B$.
26. [M22] Пусть $(p_1, \dots, p_n; q_0, \dots, q_n)$ —неотрицательные веса и $p_n + q_n = x$. Докажите, что при x , изменяющемся от 0 до ∞ , и при постоянных $(p_1, \dots, p_{n-1}; q_0, \dots, q_{n-1})$ цена $c(0, n)$ оптимального бинарного дерева поиска является "выпуклой непрерывной кусочно линейной" функцией x с целыми угловыми коэффициентами. Другими словами, докажите, что существуют положительные целые числа $l_0 > l_1 > \dots > l_m$ и действительные постоянные $0 = x_0 < x_1 < \dots < x_m < x_{m+1} = \infty$ и $y_0 < y_1 < \dots < y_m$, такие, что $c(0, n) = y_h + l_h x$ при $x_h \leq x \leq x_{h+1}, 0 \leq h \leq m$.
27. [M33] Целью данного упражнения является доказательство того факта, что множества корней $R(i, j)$ оптимальных бинарных деревьев поиска удовлетворяют при $j - i \geq 2$ соотношению

$$R(i, j - 1) \leq R(i, j) \leq R(i + 1, j),$$

введенному в упр. 25 (веса $(p_1, \dots, p_n; q_0, \dots, q_n)$ предполагаются неотрицательными). Доказательство проводится индукцией по $j - i$; наша задача — доказать, что $R(0, n - 1) \leq R(0, n)$ при $n \geq 2$, в предположении справедливости соотношения при $j - i < n$. [В силу симметрии отсюда следует, что $R(0, n) \leq R(1, n)$.]

- а) Докажите, что $R(0, n - 1) \leq R(0, n)$, если $p_n = q_n = 0$. (См. упр. 24.)
 б) Пусть $p_n + q_n = x$. Через R_h обозначим множество $R(0, n)$ оптимальных корней, если $x_h < x < x_{h+1}$, а через R'_h — множество оптимальных корней при $x_h = x$ (см. обозначения упр. 26). Докажите, что

$$R'_0 \leq R_0 \leq R'_1 \leq R_1 \leq \dots \leq R'_m \leq R_m.$$

Отсюда, из части (а) и упр. 25 следует, что $R(0, n - 1) \leq R(0, n)$ для всех x .

[Указание. Рассмотрите случай $x = x_h$ и предположите, что оба эти дерева

Picture: Рис. стр. 534

оптимальны; $s < r$ и $l \geq l'$. Используйте предположение индукции для доказательства того, что существует оптимальное дерево с корнем (r) и узлом \square на уровне l' , а также оптимальное дерево с корнем (s) и узлом \square на уровне l .]

28. [24] Используйте некоторый макроязык для написания макроопределения "оптимальный бинарный поиск", параметром которого является вложенная спецификация оптимального бинарного дерева.
 29. [40] Каково *наихудшее* возможное бинарное дерево поиска для 31 наиболее употребительного английского слова? (Частоты слов даны на рис. 12.)
 30. [M41] Докажите или опровергните, что цена оптимальных бинарных деревьев поиска удовлетворяет соотношению $c(i, j) + c(i + 1, j - 1) \geq c(i, j - 1) + c(i + 1, j)$.
 31. [M20] (а) Если веса (q_0, \dots, q_5) в (22) равны $(2, 3, 1, 1, 3, 2)$ соответственно, то какова взвешенная длина пути дерева? (б) Чему равна взвешенная длина пути оптимального бинарного дерева поиска, имеющего ту же последовательность весов?
 >32. [M22] (Т. Ч. Ху и Э. К. Такер.) Докажите, что веса $q_i + q_j$ новых узлов, получаемые на фазе 1 алгоритма Ху-Такера, создаются в неубывающем порядке.
 33. [M41] Чтобы найти бинарное дерево поиска, минимизирующее время работы программы Т, нужно минимизировать не просто число сравнений C , а величину $7C + C1$. Придумайте алгоритм для нахождения оптимальных бинарных деревьев поиска, если левые и правые ветви дерева имеют различные цены. [Кстати, когда цена правой ветви в два раза больше цены левой ветви, а частоты всех узлов равны, оптимальными оказываются фибоначчиевы деревья. Ср. с L. E. Stanfel, JACM, 17 (1970), 508–517.]
 34. [41] Напишите программу для алгоритма Ху-Такера, требующую $O(n)$ единиц памяти и $O(n \log n)$ единиц времени.
 35. [M23] Покажите, что коды, построенные в доказательстве теоремы G, обладают тем свойством, что C_i никогда не начинается с C_j при $i \neq j$.
 36. [M40] (П. Дж. Бэйер.) Обобщив верхнюю оценку в теореме G, докажите, что цена любого оптимального бинарного дерева поиска с неотрицательными весами не превосходит полного веса $S = \sum_{1 \leq i \leq n} p_i + \sum_{0 \leq i \leq n} q_i$, умноженного на $H + 2$, где

$$H = \sum_{1 \leq i \leq n} (p_i/S) \log_2(S/p_i) + \sum_{0 \leq i \leq n} (q_i/S) \log_2(S/q_i);$$

на самом деле процедура выбора корней, минимизирующая максимальный вес поддеревя, позволяет построить дерево бинарного поиска, удовлетворяющее этой оценке. Покажите далее, что цена оптимального бинарного дерева поиска не превышает S , умноженного на $H - \log_2(2H/e)$.

- >37. [M25] (Т. Ч. Ху и К. Ч. Тань.) Пусть $n + 1 = 2^m + k$, где $0 \leq k \leq 2^m$. Существует $\binom{2^m}{k}$ бинарных деревьев, в которых все внешние узлы расположены на уровнях m и $m + 1$. Покажите, что среди всех этих деревьев мы найдем одно с минимальной взвешенной длиной пути для последовательности весов (q_0, \dots, q_n) , если применим алгоритм Ху-Такера к последовательности $(M + q_0, \dots, M + q_n)$ для достаточно большого M .
 38. [M35] (К. Ч. Тань.) Докажите, что среди всех множеств вероятностей $(p_1, \dots, p_n; q_0, \dots, q_n)$, $p_1 + \dots + p_n + q_0 + \dots + q_n = 1$ дерево с минимальной ценой наиболее дорого, если $p_i = 0$ для всех i , $q_j = 0$ для четных j и $q_j = 1/\lfloor n/2 \rfloor$ для нечетных j . [Указание. Для произвольных вероятностей $(p_1, \dots, p_n; q_0, \dots, q_n)$ положим $c_0 = q_0$, $c_i = p_i + q_i$ при $1 \leq i \leq n$ и $S(0) = \emptyset$, а для $1 \leq r \leq \lfloor n/2 \rfloor$

положим $S(r) = S(r-1) \cup \{i, j\}$, где $c_i + c_j$ минимально по всем $i < j$, таким, что $i, j \notin S(r-1)$ и $k \in S(r-1)$ при всех $i < k < j$. Построим бинарное дерево T с внешними узлами из $S(n+1-2^q)$ на уровне $q+1$ и с остальными внешними узлами на уровне q , где $q = \lfloor \log_2 n \rfloor$. Докажите, что цена этого дерева $\leq f(n)$, где $f(n)$ —цена оптимального дерева поиска для указанных "наихудших" вероятностей.]

39. [M30] (Ч. К. Уон и Ши-Кью Чан.) Рассмотрим схему построения бинарного дерева поиска с помощью алгоритма T, но с одним отличием: когда число узлов принимает вид $2^n - 1$, дерево приводится к идеально сбалансированному однородному виду с 2^k узлами на уровне k , $0 \leq k < n$. Докажите, что число сравнений, производимых во время построения такого дерева, в среднем составляет $M \log_2 N + O(N)$. (Нетрудно показать, что на такие реорганизации дерева требуется $O(N)$ единиц времени.)
40. [M50] Можно ли обобщить алгоритм Ху-Такера для нахождения оптимальных деревьев, если каждый узел имеет степень не выше t ? Например, при $t = 3$ дерево, оптимальное для последовательности весов (1, 1, 100, 1, 1), выглядит так:

Picture: Рис. стр. 535

6.2.3. Сбалансированные деревья

Только что изученный нами алгоритм вставки в дерево порождает хорошие деревья поиска при случайных исходных данных, но все же существует досадная вероятность получить вырожденное дерево. Возможно, мы могли бы изобрести алгоритм, который в любом случае дает оптимальное дерево, но, к сожалению, это далеко не просто. Другой подход состоит в хранении полной длины пути и реорганизации дерева всякий раз, когда длина его пути превышает, скажем, $5N \log_2 N$. Но тогда в процессе построения дерева потребовалось бы около $\sqrt{N/2}$ реорганизаций.

Очень остроумное решение проблемы поддержания хорошего дерева поиска было найдено в 1962 г. двумя советскими математиками—Г. М. Адельсоном-Вельским и Е. М. Ландисом [ДАН СССР, 146 (1962), 263–266]. Их метод требует лишь двух дополнительных битов на узел и никогда не использует более $O(\log N)$ операций для поиска по дереву или для вставки элемента. В дальнейшем мы увидим, что этот подход также приводит к общему методу представления произвольных линейных списков длины N , причем каждая из следующих операций требует лишь $O(\log N)$ единиц времени:

- i) Найти элемент по данному ключу.
- ii) При данном k найти k -й элемент.
- iii) Вставить в определенном месте элемент.
- iv) Удалить определенный элемент.

Если для линейных списков принято последовательное расположение, то операции (i) и (ii) будут эффективными, но операции (iii) и (iv) займут порядка N шагов; с другой стороны, при использовании связанного расположения эффективны операции (iii) и (iv), а (i) и (ii) потребуют порядка N шагов. Представление линейных списков в виде дерева позволяет сделать все четыре операции за $O(\log N)$ шагов. Можно также сравнительно эффективно производить другие стандартные операции; например, возможна конкатенация (сцепление) списка из M элементов со списком из N элементов за $O(\log(M+N))$ шагов.

Метод, дающий все эти преимущества использует так называемые "сбалансированные деревья". Предыдущий абзац служит рекламой сбалансированных деревьев—этаким панацеей от всех бед; по сравнению с ними все другие способы представления данных кажутся устаревшими. Но необходимо сбалансировать наше отношение к сбалансированным деревьям! Если требуются не все. четыре рассмотренные операции, то нас может удовлетворить значительно менее универсальный, но проще программируемый метод. Более того, сбалансированные деревья хороши лишь при достаточно больших N ; так, если есть эффективный алгоритм, требующий $20 \log_2 N$ единиц времени, и неэффективный алгоритм, требующий $2N$ единиц времени, то при $N < 1024$ следует использовать неэффективный метод. С другой стороны, N не должно быть слишком велико; сбалансированные деревья подходят главным образом для хранения данных во *внутренней* памяти, а в п. 6.2.4 мы изучим лучшие методы для внешних файлов с прямым доступом. Так как со временем размеры внутренней памяти становятся все больше и больше, сбалансированные деревья становятся все более важными.

Высота дерева определяется как его наибольший уровень, как максимальная, длина пути от корня до внешнего узла.

Picture: 20. Сбалансированное бинарное дерево

Бинарное дерево называется *сбалансированным*; если высота левого поддеревья каждого узла отличается от высоты правого поддеревья не более чем на ± 1 . На рис. 20 показано сбалансированное дерево с 17 внутренними узлами и высотой 5; *показатель сбалансированности* представлен внутри каждого узла знаками $+$, \cdot или $-$, что отвечает разности высот правого и левого поддеревьев, равной $+1$, 0 или -1 соответственно. Фибоначчиево дерево на рис. 8 (п. 6.2.1) является другим сбалансированным бинарным деревом высоты 5, имеющим только 12 внутренних узлов; большинство показателей сбалансированности равно -1 . "Зодиакальное дерево" на рис. 10 (п. 6.2.2) не сбалансировано, так как поддеревья узлов AQUARIUS и GEMINI не удовлетворяют принятым ограничениям.

Это определение сбалансированности представляет собой компромисс между *оптимальными* бинарными деревьями (все внешние узлы которых расположены на двух смежных уровнях) и *произвольными* бинарными деревьями. Поэтому уместно спросить, как далеко может отклониться от оптимальности сбалансированное дерево? Оказывается, что длина его поискового пути никогда не превысит оптимум более чем на 45%.

Теорема А. (Г. М. Адельсон-Вельский и Е. М. Ландис). *Высота сбалансированного дерева с N внутренними узлами заключена между $\log_2(N + 1)$ и $1.4404 \log_2(N + 2) - 0.328$.*

Доказательство. Бинарное дерево высоты h , очевидно, не может содержать более чем 2^h внешних узлов; поэтому $N + 1 \leq 2^h$, т.е. $h \geq \lceil \log_2(N + 1) \rceil$.

Чтобы найти максимальное значение h , поставим вопрос по-другому: каково минимальное число узлов в сбалансированном дереве высоты h ? Пусть T_h — такое дерево с наименьшим возможным количеством узлов; тогда одно поддерево корня, например левое, имеет высоту $h - 1$, а другое — или $h - 1$, или $h - 2$. В силу определения T_h можно считать, что левое поддерево корня есть T_{h-1} , а правое — T_{h-2} . Таким образом, среди всех сбалансированных деревьев высоты h наименьшее количество узлов имеет *фибоначчиево дерево* порядка $h + 1$. (См. определение деревьев Фибоначчи в п. 6.2.1.) Итак,

$$N \geq F_{h+2} - 1 > \phi^{h+2} / \sqrt{5} - 2,$$

и требуемый результат получается так же, как следствие из теоремы 4.5.3F. ■

Мы видим, что поиск в сбалансированном дереве потребует более 25 сравнений, только если дерево состоит из по крайней мере $F_{27} - 1 = 196417$ узлов.

Рассмотрим теперь, что происходит, когда новый узел вставляется в сбалансированное дерево посредством алгоритма 6.2.2Т. Дерево на рис. 20 остается сбалансированным, если новый узел займет место одного из узлов $\boxed{4}$, $\boxed{5}$, $\boxed{6}$, $\boxed{7}$, $\boxed{10}$ или $\boxed{13}$, но в других случаях потребуется некоторая корректировка. Трудности возникают, если имеется узел с показателем сбалансированности $+1$, правое поддерево которого после вставки становится выше, или если показатель сбалансированности равен -1 и выше, становится левое поддерево. Легко понять, что, в сущности, нас беспокоят лишь два случая:

Picture: Случаи вставки в AVL-дерево

(Другие "плохие" случаи можно получить, зеркально отразив эти диаграммы относительно вертикальной оси.) Большими прямоугольниками α , β , γ , δ обозначены поддеревья с соответствующими высотами. Случай 1 имеет место, если новый элемент увеличил высоту правого поддеревья узла B с h до $h + 1$, а случай 2 — когда новый элемент увеличивает высоту левого поддеревья узла B . Во втором случае мы имеем либо $h = 0$ (и тогда сам узел X является новым узлом), либо узел X имеет два поддерева с соответственными высотами $(h - 1, h)$ или $(h, h - 1)$.

Простые преобразования восстанавливают баланс в обоих случаях, сохраняя в то же время симметричный порядок узлов A , B и X .

Picture: Повороты

В случае 1 мы просто поворачиваем дерево налево, прикрепляя β к A вместо B . Это преобразование подобно применению ассоциативного закона к алгебраической формуле, когда мы заменяем $\alpha(\beta\gamma)$ на $(\alpha\beta)\gamma$. В случае 2 это проделывается дважды: сначала (X, B) поворачивается направо, затем (A, X) — налево. В обоих случаях нужно изменить в дереве лишь несколько ссылок. Далее новые деревья имеют высоту $h + 2$, в точности ту же, что и до вставки элемента; следовательно, часть дерева, расположенная над узлом A (если таковая имеется), остается сбалансированной.

Picture: Дерево рис. 20, сбалансированное после вставки нового ключа R

Например, если мы вставляем новый узел на место $\boxed{17}$ (рис. 20), то после поворота получим сбалансированное дерево, изображенное на рис. 21 (случай 1). Заметьте, что некоторые из показателей сбалансированности изменились.

Детали этой процедуры вставки можно разработать различными способами. На первый взгляд без вспомогательного стека не обойтись, так как необходимо запоминать узлы, которые будут затронуты вставкой. Ниже приводится алгоритм, в котором, прибегнув к маленькой хитрости, мы обходимся без стека, выигрывая при этом в скорости.

Алгоритм А. (Поиск, с вставкой по сбалансированному дереву.) Имеется таблица записей, образующих сбалансированное бинарное дерево. Алгоритм позволяет произвести поиск данного аргумента K . Если K в таблице нет, в подходящем месте в дерево вставляется новый узел, содержащий K . При необходимости производится балансировка дерева.

Предполагается (как и в алгоритме 6.2.2Т), что узлы содержат поля KEY, LLINK и RLINK. Кроме того, имеется новое поле $B(P) =$ показатель сбалансированности узла $NODE(P)$, т. е. разность высот правого и левого поддеревьев; это поле всегда содержит $+1, 0$ или -1 . По адресу HEAD расположен специальный головной узел; RLINK(HEAD) указывает на корень дерева, а LLINK(HEAD) используется для хранения полной высоты дерева. Для данного алгоритма высота не имеет значения, но знание ее полезно для процедуры конкатенации, обсуждающейся ниже. Мы предполагаем, что дерево *непусто*, т. е. что $RLINK(HEAD) \neq \Lambda$.

В целях удобства описания в алгоритме используется обозначение LINK(a, P) как синоним LLINK(P) при $a = -1$ и как синоним RLINK(P) при $a = +1$.

A1 [Начальная установка.] Установить $T \leftarrow HEAD, S \leftarrow P \leftarrow RLINK(HEAD)$. [Указательная переменная P будет двигаться вниз по дереву; S будет указывать на место, где может потребоваться балансировка; T всегда указывает на отца S .]

A2 [Сравнение.] Если $K < KEY(P)$, то перейти на **A3**; если $K > KEY(P)$, то перейти на **A4**; если $K = KEY(P)$, поиск удачно завершен.

A3 [Шаг влево.] Установить $Q \leftarrow LLINK(P)$. Если $Q = \Lambda$, выполнить $Q \leftarrow AVAIL$ и $LLINK(P) \leftarrow Q$; затем идти на **A5**. В противном случае, если $B(Q) \neq 0$, установить $T \leftarrow P$ и $S \leftarrow Q$. Наконец, установить $P \leftarrow Q$ и вернуться на **A2**.

A4 [Шаг вправо.] Установить $Q \leftarrow RLINK(P)$. Если $Q = \Lambda$, выполнить $Q \leftarrow AVAIL$ и $RLINK(P) \leftarrow Q$; затем идти на **A5**.

В противном случае, если $B(Q) \neq 0$, установить $T \leftarrow P$ и $S \leftarrow Q$. Наконец, установить $P \leftarrow Q$ и вернуться на **A2**. (Последнюю часть этого шага можно объединить с последней частью шага **A3**.)

A5 [Вставка.] (Мы только что присоединили новый узел $NODE(Q)$ к дереву; теперь его поля нуждаются в начальной установке.) Установить $KEY(Q) \leftarrow K, LLINK(Q) \leftarrow RLINK(Q) \leftarrow \Lambda, B(Q) \leftarrow 0$.

A6 [Корректировка показателей сбалансированности.] (Теперь нулевые показатели сбалансированности между S и Q нужно заменить на ± 1 .) Если $K < KEY(S)$, установить $R \leftarrow P \leftarrow LLINK(S)$; в противном случае установить $R \leftarrow P \leftarrow RLINK(S)$. Затем нужно 0 или более раз повторять следующую операцию, пока P не станет равным Q : если $K < KEY(P)$, установить $B(P) \leftarrow -1$ и $P \leftarrow LLINK(P)$; если $K > KEY(P)$, установить $B(P) \leftarrow +1$ и $P \leftarrow RLINK(P)$. (Если $K = KEY(P)$, значит, $P = Q$, и можно перейти к следующему шагу.)

A7 [Проверка сбалансированности.] Если $K < KEY(S)$, установить $a \leftarrow -1$; в противном случае $a \leftarrow +1$. Теперь возможны три случая:

- i) Если $B(S) = 0$ (дерево стало выше), установить $B(S) \leftarrow a, LLINK(HEAD) \leftarrow LLINK(HEAD) + 1$; алгоритм завершен.
- ii) Если $B(S) = -a$ (дерево стало более сбалансированным), установить $B(S) \leftarrow 0$; алгоритм завершен.
- iii) Если $B(S) = a$ (дерево перестало быть сбалансированным), при $B(R) = a$ идти на **A8**, при $B(R) = -a$ идти на **A9**.

(Случай (iii) соответствует ситуации, изображенной на диаграмме (1), при $a = +1$; S и R указывают соответственно на узлы A и B , а LINK($-a, S$) указывает на α и т.д.)

A8 [Однократный поворот.] Установить $P \leftarrow R, LINK(a, S) \leftarrow LINK(-a, R), LINK(-a, R) \leftarrow S, B(S) \leftarrow B(R) \leftarrow 0$. Перейти на **A10**.

A9 [Двукратный поворот.] Установить $P \leftarrow LINK(-a, R), LINK(-a, R) \leftarrow LINK(a, P), LINK(a, P) \leftarrow R, LINK(a, S) \leftarrow LINK(-a, P), LINK(-a, P) \leftarrow S$. Теперь установить

$$(B(S), B(R)) \leftarrow \begin{cases} (-a, 0), & \text{если } B(P) = a; \\ (0, 0), & \text{если } B(P) = 0; \\ (0, a), & \text{если } B(P) = -a; \end{cases} \quad (3)$$

затем $B(P) \leftarrow 0$.

A10 [Последний штрих.] [Мы завершили балансирующее преобразование от (1) к (2), P указывает на новый корень, а T — на отца старого корня.] Если $S = \text{RLINK}(T)$, то установить $\text{RLINK}(T) \leftarrow P$; в противном случае $\text{LLINK}(T) \leftarrow P$. ■

Этот алгоритм довольно длинный, но разделяется на три простые части: шаги A1–A4 (поиск), шаги A5–A7 (вставка нового узла), шаги A8–A10 (балансировка дерева, если она нужна).

Picture: 22. Поиск с вставкой по сбалансированному дереву

Мы знаем, что для работы алгоритма требуется около $C \log N$ единиц времени при некотором C , но чтобы знать, при каких N выгодно использовать сбалансированные деревья, нужно оценить величину C . Анализ следующей MIX-программы позволяет подойти к решению этого вопроса.

Программа А. (Поиск с вставкой по сбалансированному дереву.) Эта реализация алгоритма А использует следующий формат узлов дерева:

Picture: Формат узла АВЛ-дерева

$rA \equiv K$, $rI1 \equiv P$, $rI2 \equiv Q$, $rI3 \equiv R$, $rI4 \equiv S$, $rI5 \equiv T$. Программа для шагов A7–A9 дублируется, так что величина a в явном виде в программе не фигурирует.

B	EQU	0,1		
LLINK	EQU	2,3		
RLINK	EQU	4,5		
START	LDA	K	1	A1. Начальная установка.
	ENT5	HEAD	1	$T \leftarrow \text{HEAD}$.
	LD2	0,5 (RLINK)	1	$Q \leftarrow \text{RLINK}(\text{HEAD})$.
	JMP	2F	1	На A2 с $S \leftarrow P \leftarrow Q$
4H	LD2	0,1 (RLINK)	C2	A4. Шаг вправо. $Q \leftarrow \text{RLINK}(P)$
	J2Z	5F	C2	На A5, если $Q = \Lambda$.
1H	LDX	0,2 (B)	C-1	$rX \leftarrow B(Q)$.
	JXZ	**+3	C-1	Переход, если $B(Q) = 0$.
	ENT5	0,1	D-1	$T \leftarrow P$.
2H	ENT4	0,2	D	$S \leftarrow Q$.
	ENT1	0,2	C	$P \leftarrow Q$.
	CMPA	1,1	C	A2. Сравнение.
	JG	4B	C	На A4, если $K > \text{KEY}(P)$.
	JE	SUCCESS	C1	Выход, если $K = \text{KEY}(P)$.
	LD2	0,1 (LLINK)	C1-S	A3. Шаг влево. $Q \leftarrow \text{LLINK}(P)$.
	J2NZ	1B	C1-S	Переход, если $Q \neq \Lambda$.
20–29 5H (скопировать здесь строки 14–23 программы 6.2.2 Т) A5. Вставка.				
6H	CMPA	1,4	1-S	A6. Коррект. показат. сбалансир.
	JL	**+3	1-S	Переход, если $K < \text{KEY}(S)$.
	LDS	0,4 (RLINK)	E	$R \leftarrow \text{RLINK}(S)$.
	JMP	**+2	E	
	LD3	0,4 (LLINK)	1-S-E	$R \leftarrow \text{LLINK}(S)$.
	ENT1	0,3	1-S	$P \leftarrow R$.
	ENTX	-1	1-S	$rX \leftarrow -1$.
	JMP	1F	1-S	На цикл сравнения.
4H	JE	7F	F2+1-S	На A7, если $K = \text{KEY}(P)$.
	STX	0,1 (1:1)	F2	$B(P) \leftarrow +1$ (он был +0).
	LD1	0,1 (RLINK)	F2	$P \leftarrow \text{RLINK}(P)$.
1H	CMPA	1,1	F+1-S	
	JGE	4B	F+1-S	Переход, если $K \geq \text{KEY}(P)$.
	STX	0,1 (B)	F1	$B(P) \leftarrow -1$.
	LD1	0,1 (LLINK)	F1	$P \leftarrow \text{LLINK}(P)$.
	JMP	1B	F1	На цикл сравнения.
7H	LD2	0,4 (B)	1-S	A7. Проверка сбалансир. $rI2 \leftarrow B(S)$.
	STZ	0,4 (B)	1-S	$B(S) \leftarrow 0$.

	CMPA	1, 4		1 - S			
	JG	A7R		1 - S	На $a = +1$ подпрограмму, если $K > KEY(S)$.		
A7L	J2P	DONE	A7R	J2N	DONE	1 - S	Выход, если $rI2 = -a$.
	J2Z	7F		J2Z	6F	$G + J$	Переход, если $B(S)$ был нулем.
	ENT1	0, 3		ENT1	0, 3	G	$P \leftarrow R$.
	LD2	0, 3 (B)		LD2	0, 3 (B)	G	$rI2 \leftarrow B(R)$.
	J2N	A8L		J2P	A8R	G	На A8, если $rI2 = a$.
A9L	LD1	0, 3 (RLINK)	A9R	LD1	0, 3 (LLINK)	H	A9. Двукратный поворот.
	LDX	0, 1 (LLINK)		LDX	0, 1 (RLINK)	H	$LINK(a, P \leftarrow LINK(-a, R))$
	STX	0, 3 (RLINK)		STX	0, 3 (LLINK)	H	$\rightarrow LINK(-a, R)$.
	ST3	0, 1 (LLINK)		ST3	0, 1 (RLINK)	H	$LINK(a, P) \leftarrow R$.
	LD2	0, 1 (B)		LD2	0, 1 (B)	H	$rI2 \leftarrow B(P)$.
	LDX	T1, 2		LDX	T2, 2	H	$-a, 0$ или 0
	STX	0, 1 (B)		STX	0, 4 (B)	H	$\rightarrow B(S)$.
	LDX	T2, 2		LDX	T1, 2	H	$0, 0$ или a
	STX	0, 3 (B)		STX	0, 3 (B)	H	$\rightarrow B(R)$
A8L	LDX	0, 1 (RLINK)	A8R	LDX	0, 1 (LLINK)	G	A8. Однократный поворот.
	STX	0, 4 (LLINK)		STX	0, 4 (RLINK)	G	$LINK(a, S) \leftarrow LINK(-a, P)$.
	ST4	0, 1 (RLINK)		ST4	0, 1 (LLINK)	G	$LINK(-a, P) \leftarrow S$.
	JMP	A8R1	A8R1	STZ	0, 1 (B)	G	$B(P) \leftarrow 0$.
A10	CMP4	0, 5 (RLINK)		G			A10. Последний штрих.
	JNE	*+3		G			Переход, если $RLINK(T) \neq S$.
	ST1	0, 5 (RLINK)		$G2$			$RLINK(T) \leftarrow P$.
	JMP	DONE		$G2$			Выход.
	ST1	0, 5 (LLINK)		$G1$			$LLINK(T) \leftarrow P$.
	JMP	DONE		$G1$			Выход.
	CON	+1					
T1	CON	0					Таблица для (3).
T2	CON	0					
	CON	-1					
6H	ENTX	+1		$J2$			$rX \leftarrow +1$.
7H	STX	0, 4 (B)		J			$B(S) \leftarrow a$.
	LDX	HEAD (LLINK)		J			$LLINK(HEAD)$.
	INCX	1		J			+1
	STX	HEAD (LLINK)		J			$\rightarrow LLINK(HEAD)$.
DONE	EQU	*		1 - S			Вставка завершена.



Анализ вставки в сбалансированное дерево. [Читатели, не интересующиеся математикой, могут сразу перейти к формуле (10).] Чтобы вычислить время работы алгоритма А, нужно сначала ответить на следующие вопросы:

- Сколько сравнений производится во время поиска?
- Как далеко друг от друга будут находиться узлы S и Q? (Иными словами, сколько нужно произвести корректировок в шаге A6?)
- Как часто нужно производить однократный или двукратный поворот?

Воспользовавшись теоремой А, нетрудно вывести верхнюю оценку времени работы, но нас, разумеется, интересует средний уровень. До сих пор не удалось теоретически оценить, как ведет себя алгоритм в среднем, поскольку он оказался довольно сложным, однако были получены некоторые интересные эмпирические результаты.

В первую очередь нас может интересовать число B_{nh} сбалансированных бинарных деревьев с n внутренними узлами и высотой h . Для небольших h из соотношений

$$B_0(z) = 1, \quad B_1(z) = z, \quad B_{h+1}(z) = zB_h(z)(B_h(z) + 2B_{h-1}(z)) \quad (5)$$

нетрудно вычислить производящую функцию $B_h(z) = \sum_{n \geq 0} B_{nh}z^n$ (см. упр. 6). Таким образом,

$$\begin{aligned} B_2(z) &= 2z^2 + z^3, \\ B_3(z) &= 4z^4 + 6z^5 + 4z^6 + z^7, \\ B_4(z) &= 16z^7 + 32z^8 + 44z^9 + \dots + 8z^{14} + z^{15}, \end{aligned}$$

и вообще $B_h(z)$ при $h \geq 3$ имеет вид

$$2^{F_{h+1}-1} z^{F_{h+2}-1} + 2^{F_{h+1}-2} L_{h-1} z^{F_{h+2}} + \text{сложные члены} + 2^{h-1} z^{2^h-2} + z^{2^h-1}, \quad (6)$$

где $L_k = F_{k+1} + F_{k-1}$. (Эта формула обобщает теорему А.) Число всех сбалансированных деревьев высоты h равно $B_h = B_h(1)$ и удовлетворяет рекуррентному соотношению

$$B_0 = B_1 = 1, \quad B_{h+1} = B_h^2 + 2B_h B_{h-1}, \quad (7)$$

так что $B_2 = 3$, $B_3 = 3 \cdot 5$, $B_4 = 3^2 \cdot 5 \cdot 7$, $B_5 = 3^3 \cdot 5^2 \cdot 7 \cdot 23$; в общем случае

$$B_h = A_0^{F_h} \cdot A_1^{F_h-1} \dots A_{h-1}^{F_1} \cdot A_h^{F_0}, \quad (8)$$

где $A_0 = 1$, $A_1 = 3$, $A_2 = 5$, $A_3 = 7$, $A_4 = 23$, $A_5 = 347$, ..., $A_h = A_{h-1} B_{h-2} + 2$. Последовательности A_h и B_h растут очень быстро, как "экспонента экспоненты"; из упр. 7 следует, что существует действительное число $\theta \approx 1.43684$, такое, что

$$B_h = \left\lfloor \theta^{2^h} \right\rfloor - \left\lfloor \theta^{2^h-1} \right\rfloor + \left\lfloor \theta^{2^h-2} \right\rfloor - \dots + (-1)^h \left\lfloor \theta^{2^0} \right\rfloor. \quad (9)$$

Если считать, что все B_h деревьев равновероятны, то, как показывает упр. 8, среднее число узлов в дереве высоты h равно

$$B'_h(1)/B_h(1) \approx (0.70118) \cdot 2^h. \quad (10)$$

Это означает, что высота сбалансированного дерева с n узлами обычно гораздо ближе к $\log_2 n$, чем к $\log_\phi n$.

К сожалению, полученные результаты не имеют отношения к алгоритму А, так как механизм этого алгоритма делает некоторые деревья гораздо более вероятными, чем другие. Рассмотрим, например, случай $N = 7$, когда существует 17 сбалансированных деревьев. Ключи можно вставлять $7! = 5040$ различными способами, при этом идеально сбалансированное "совершенное" дерево

Picture: рис. на стр. 546

получается 2160 раз. В противоположность этому фибоначчьево дерево

Picture: рис. на стр. 546

встречается лишь 144 раза, а похожее дерево

Picture: рис. на стр. 546

встречается 216 раз. [Заменив левые поддеревья в (12) и (13) произвольными сбалансированными деревьями с четырьмя узлами и затем зеркально отразив относительно вертикальной оси, получим 16 различных деревьев; восемь деревьев, полученных из (12), встречаются по 144 раза, а полученные из (13)—по 216 раз. Довольно странно, что (13) встречается чаще, чем (12).]

Тот факт, что идеально сбалансированные деревья получаются с такой высокой вероятностью—совместно с формулой (10), соответствующей случаю равных вероятностей,—делает чрезвычайно правдоподобным соотношение: (среднее число сравнений при поиске по сбалансированному дереву) $\approx \log_2 N + c$, где c —малая константа. Действительно, эмпирические проверки показывают, что среднее число сравнений, требуемых для вставки N -го элемента, при не слишком малых N примерно равно $1.01 \log_2 N + 0.1$

Чтобы изучить поведение алгоритма А на фазах вставки и балансировки, можно классифицировать внешние узлы сбалансированного дерева, как показано на рис. 23. Путь, ведущий из внешнего узла вверх, описывается последовательностью плюсов и минусов ("+" для правой ссылки, "-" для левой); мы выписываем ее, пока не достигнем первого узла с ненулевым показателем сбалансированности или (если таких узлов нет) пока не достигнем корня. Затем мы пишем А или В в соответствии с тем, будет ли новое дерево, полученное после вставки на данное место внутреннего узла, сбалансированным или нет. Так/путь из (3) вверх кодируется ++-В, что означает "правая ссылка, правая ссылка, левая ссылка, несбалансировано". Если код оканчивается на А, после вставки нового узла не

Picture: Рис. 23. Коды классификации, определяющие поведение алгоритма А после вставки.

требуется балансировки; код, оканчивающийся на ++В или --В, требует однократного поворота, а код, оканчивающийся на +-В или -+В, требует двукратного поворота. Если путь состоит из k звеньев, в шаге А6 корректируется ровно $k - 1$ показателей сбалансированности. Таким образом, описанные последовательности дают необходимую информацию о времени работы шагов А6–А10.

Эмпирические проверки со случайными числами в диапазоне $100 \leq N \leq 2000$ дали приближенные вероятности для путей различных видов (табл. 1); очевидно, эти вероятности быстро приближаются к предельным значениям при $N \rightarrow \infty$. В табл. 2 даны соответствующие точные вероятности ($N = 10$; все $10!$ перестановок считались равновероятными.)

Из табл. 1 мы видим, что вероятность события $k \leq 2$ равна $0.144 + 0.153 + 0.144 + 0.144 = 0.585$; таким образом, почти в 60% случаев шаг А6 тривиален. Среднее число изменений коэффициентов сбалансированности на этом шаге (0 заменяется на ± 1) примерно равно 1.8. Среднее число изменений коэффициентов сбалансированности от ± 1 до 0 в шагах А7–А10 равно $0.535 + 2(0.233 + 0.232) \approx 1.5$, т. е. вставка одного нового узла добавляет в среднем 0.3 несбалансированного узла. Это согласуется с тем фактом, что около 68% всех узлов в случайных деревьях, полученных с помощью алгоритма А, оказались сбалансированными.

Таблица 1

Приближенные вероятности при вставке N -го элемента				
Длина пути	Нет балансировки	Однократный поворот	Двукратный поворот	
1	.144	.000	.000	
2	.153	.144	.144	
3	.093	.048	.048	
4	.058	.023	.023	
5	.036	.010	.010	
> 5	.051	.008	.007	
ave	2.78	.535	.232	

Таблица 2

Точные вероятности при вставке 10-го элемента				
Длина пути	Нет балансировки	Однократный поворот	Двукратный поворот	
1	1/7	0	0	
2	6/35	1/7	1/7	
3	4/21	2/35	2/35	
4	0	1/21	1/21	
ave	247/105	53/105	26/105	

Приближенная модель поведения алгоритма А была предложена К. К. Фостером [Proc. ACM Nat. Conf., 20, (1965), 192–205]. Модель эта не вполне корректна, но достаточно близка к истине, чтобы отразить существо дела. Предположим, что в большом дереве, построенном с помощью алгоритма А, показатель сбалансированности данного узла с вероятностью p равен 0; тогда этот показатель равен +1 с вероятностью $\frac{1}{2}(1-p)$ и с той же вероятностью равен -1. Предположим далее (без всяких обоснований), что показатели сбалансированности всех узлов независимы. Тогда вероятность того, что шаг А6 делает ненулевыми ровно $(k-1)$ показателей, равна $p^{k-1}(1-p)$, поэтому среднее значение k есть $1/(1-p)$. Вероятность того, что нужно повернуть часть дерева, равна $\frac{1}{2}$. В среднем вставка нового узла должна увеличить число сбалансированных узлов на p ; это число в действительности увеличивается на 1 в шаге А5, на $-p(1-p)$ в шаге А6, на $\frac{1}{2}$ в шаге А7 и на $\frac{1}{2} \cdot 2$ в шаге А8 или А9, так что мы получаем

$$p = 1 - p/(1-p) + \frac{1}{2} + 1.$$

Решение этого уравнения относительно p дает неплохое согласие с табл. 1:

$$p = \frac{9 - \sqrt{41}}{4} \approx 0.649; \quad 1/(1-p) \approx 2.851. \quad (14)$$

Время работы фазы поиска программы А (строки 01–19) равно

$$10C + C1 + 2D + 2 - 3S, \quad (15)$$

где C , $C1$, S —те же самые, что и в предыдущих алгоритмах этой главы, а D —число несбалансированных узлов, проходимых при поиске. Эмпирические проверки показывают, что можно положить

$D \approx \frac{1}{3}C$, $C1 \approx \frac{1}{2}(C + S)$, $C + S \approx 1.01 \log_2 N + 0.1$, так что среднее время поиска примерно равно $11.3 \log_2 N + 3 - 13.7S$ единиц. (Если поиск производится гораздо чаще вставки, мы могли бы, разумеется, использовать отдельную, более быструю программу поиска, так как не было бы необходимости следить за коэффициентами сбалансированности; в этом случае среднее время поиска составило бы лишь $(6.6 \log_2 N + 3)u$, а в наихудшем случае время работы будет все же меньше, чем среднее время работы программы 6.2.2Г.)

Если поиск неудачен, время работы фазы вставки в программе А (строки 20–45) равно $8F + 26 + (0, 1 \text{ или } 2)$ единиц. Данные табл. 1 показывают, что в среднем $F \approx 1.8$. Фаза балансировки (строки 46–101) требует 16.5, 8, 27.5 или $45.5(\pm 0.5)$ единиц в зависимости от того, увеличиваем ли мы полную высоту, просто ли выходим без балансировки или же производим однократный или двукратный поворот. Первый случай почти не встречается, а другие встречаются с приближенными вероятностями 0.535, 0.233, 0.232, поэтому среднее время выполнения комбинированной вставочно-балансирующей части программы А составляет примерно $63u$.

Эти числа показывают, что операции над сбалансированными деревьями довольно быстры, хотя программа и занимает много места в памяти. Если исходные данные являются случайными, то простой алгоритм вставки в дерево (п. 6.2.2) производит одну вставку примерно на $50u$ быстрее, но использование сбалансированных деревьев гарантирует хорошие результаты даже при неслучайных исходных данных.

Один из способов сравнения программы А с программой 6.2.2Г состоит в рассмотрении наихудшего для последней программы случая. Если мы заинтересуемся количеством времени, необходимым для вставки в возрастающем порядке N ключей в первоначально пустое дерево, то окажется, что программа А медленнее при $N \leq 26$ и быстрее при $N \geq 27$.

Picture: Рис. 24. Поля RANK, используемые при поиске по позиции.

Представление линейного списка. Вернемся теперь к замечанию, сделанному в начале этого пункта, о том, что сбалансированные деревья могут использоваться для представления линейных списков таким образом, что можно будет быстро вставлять элементы (преодолевая трудность последовательного расположения), сохраняя при этом случайным доступ к элементам списка (т. е. преодолевая трудность связанного расположения).

Идея состоит в том, чтобы в каждом узле ввести новое поле с именем RANK. Это поле показывает относительное положение узла в своем поддереве, т. е. оно равно единице плюс число узлов в левом поддереве. На рис. 24 изображены значения RANK для бинарного дерева на рис. 23. Мы можем полностью исключить поле KEY, или при желании можно иметь оба поля, что позволяет находить элементы как по значению ключа, так и по относительному положению в списке.

Использование поля RANK позволяет свести поиск по позиции к изученным алгоритмам.

Алгоритм В. (Поиск по позиции в дереве.) Имеется линейный список, представленный в виде бинарного дерева. Алгоритм позволяет по данному k найти k -й элемент списка (k -й узел дерева в симметричном порядке). Предполагается, что, как и в алгоритме А, имеется головной узел, в каждом узле есть поля LLINK и RLINK и, кроме того, поле RANK, описанное выше.

В1 [Начальная установка.] Установить $M \leftarrow k$, $P \leftarrow \text{RLINK}(\text{HEAD})$.

В2 [Сравнение.] Если $P = \Lambda$, алгоритм кончается неудачно. (Это может случиться, лишь если k больше числа узлов в дереве или $k \leq 0$.) В противном случае, если $M < \text{RANK}(P)$, перейти на **В3**; если $M > \text{RANK}(P)$, перейти на **В4**; а если $M = \text{RANK}(P)$, алгоритм кончается удачно (P указывает на k -й узел).

В3 [Шаг влево.] Установить $P \leftarrow \text{LLINK}(P)$ и вернуться на **В2**.

В4 [Шаг вправо.] Установить $M \leftarrow M - \text{RANK}(P)$, $P \leftarrow \text{RLINK}(P)$ и вернуться на **В2**. ■

В данном алгоритме интерес представляет лишь операция $M \leftarrow M - \text{RANK}(P)$ в шаге В4. Можно аналогичным образом модифицировать процедуру вставки, хотя есть свои тонкости.

Алгоритм С. (Вставка в сбалансированное дерево по позиции.) Имеется линейный список, представленный в виде сбалансированного бинарного дерева. Алгоритм позволяет при данном k вставить новый узел (Q — указатель на него) перед k -м элементом списка. Если $k = N + 1$, новый узел помещается в конец списка.

Кроме того, что выполнены условия алгоритма А, предполагается, что каждый узел содержит поле RANK. Этот алгоритм очень похож на алгоритм А с тем лишь отличием, что вместо поля KEY используется поле RANK.

С1 [Начальная установка.] Установить $T \leftarrow \text{HEAD}$, $S \leftarrow P \leftarrow \text{RLINK}(\text{HEAD})$, $U \leftarrow M \leftarrow k$.

С2 [Сравнение.] Если $M \leq \text{RANK}(P)$, перейти на **С3**; в противном случае перейти на **С4**.

- C3** [Шаг влево.] Установить $RANK(P) \leftarrow RANK(P) + 1$ (мы будем вставлять новый узел слева от P). Установить $R \leftarrow LLINK(P)$. Если $R = \Lambda$, установить $LLINK(P) \leftarrow Q$ и перейти на **C5**. В противном случае, если $V(R) \neq 0$, установить $T \leftarrow P$, $S \leftarrow R$ и $U \leftarrow M$. Установить $P \leftarrow R$ и вернуться на **C2**.
- C4** [Шаг вправо.] Установить $M \leftarrow M - RANK(P)$ и $R \leftarrow RLINK(P)$. Если $R = \Lambda$, установить $RLINK(P) \leftarrow Q$ и перейти на **C5**. В противном случае, если $V(R) \neq 0$, установить $T \leftarrow P$, $S \leftarrow R$, $U \leftarrow M$. Наконец, установить $P \leftarrow R$ и вернуться на **C2**.
- C5** [Вставка.] Установить $RANK(Q) + 1$, $LLINK(Q) \leftarrow RLINK(Q) \leftarrow \Lambda$, $V(Q) \leftarrow 0$.
- C6** [Корректировка показателей сбалансированности.] Установить $M \leftarrow U$. (Тем самым восстанавливается предыдущее значение M , когда P было равно S ; все поля $RANK$ подходящим образом установлены.) Если $M < RANK(S)$, установить $R \leftarrow P \leftarrow LLINK(S)$; в противном случае установить $R \leftarrow P \leftarrow RLINK(S)$ и $M \leftarrow M - RANK(S)$. Затем, пока P не станет равным Q , нужно повторять следующую операцию: если $M < RANK(P)$, установить $V(P) \leftarrow -1$ и $P \leftarrow LLINK(P)$; если $M > RANK(P)$, установить $V(P) \leftarrow +1$, $M \leftarrow M - RANK(P)$ и $P \leftarrow RLINK(P)$. (Если $M = RANK(P)$, то $P = Q$, и можно перейти к следующему шагу.)
- C7** [Проверка сбалансированности.] Если $U < RANK(S)$, установить $a \leftarrow -1$; в противном случае $a \leftarrow +1$. Теперь возможно несколько случаев:
- i) Если $V(S) = 0$, установить $V(S) \leftarrow a$, $LLINK(HEAD) \leftarrow LLINK(HEAD) + 1$; алгоритм завершен.
 - ii) Если $V(S) = -a$, установить $V(S) \leftarrow 0$; алгоритм завершен.
 - iii) Если $V(S) = a$, то при $V(R) = a$ нужно идти на **C8**, а при $V(R) = -a$ — на **C9**.
- C8** [Однократный поворот.] Установить $P \leftarrow R$, $LINK(a, S) \leftarrow LINK(-a, R)$, $LINK(-a, R) \leftarrow S$, $V(S) \leftarrow V(R) \leftarrow 0$. Если $a = +1$, установить $RANK(R) \leftarrow RANK(R) + RANK(S)$; если $a = -1$, установить $RANK(S) \leftarrow RANK(S) - RANK(R)$. Перейти на **C10**.
- C9** [Двукратный поворот.] Прodelать все операции шага **A9** (алгоритма **A**). Затем, если $a = +1$, установить $RANK(R) \leftarrow RANK(R) - RANK(P)$, $RANK(P) \leftarrow RANK(P) + RANK(S)$, если $a = -1$, установить $RANK(P) \leftarrow RANK(P) + RANK(R)$, затем $RANK(S) \leftarrow RANK(S) - RANK(P)$.
- C10** [Последний штрих.] Если $S = RLINK(T)$, то установить $RLINK(T) \leftarrow P$; в противном случае $LLINK(T) \leftarrow P$. ■

***Удаление, конкатенация и т. д.** Вообще говоря, существует много других операций, которые не нарушают сбалансированности деревьев, но соответствующие алгоритмы достаточно длинны, так что мы не будем рассматривать их подробно. Обсудим лишь основные идеи, а интересующийся читатель сможет без большого труда восстановить необходимые детали.

Задача удаления, если поставить ее корректно, решается за $O(\log N)$ шагов [C. C. Foster, A Study of AVL Trees, Goodyear Aerospace Corp. report GER-12158 (April 1965)]. Прежде всего удаление произвольного узла можно свести к простому удалению узла P , в котором $LLINK(P)$ или $RLINK(P)$ равны Λ , как в алгоритме 6.2.2D. Этот алгоритм следует модифицировать таким образом, чтобы он строил список указателей, определяющих путь к узлу P :

$$(P_0, a_0), (P_1, a_1), \dots, (P_l, a_l), \quad (16)$$

где $P_0 = HEAD$, $a_0 = +1$; $LINK(a_i, P_i) = P_{i+1}$, $0 \leq i < l$; $P_l = P$, $LINK(a_l, P_l) = \Lambda$. Элементы этого списка в процессе спуска по дереву можно заносить во вспомогательный стек. Процесс удаления узла P устанавливает $LINK(a_{l-1}, P_{l-1}) \leftarrow LINK(-a_l, P_l)$, и нужно откорректировать показатель сбалансированности узла P_{l-1} . Предположим, что мы должны откорректировать показатель сбалансированности узла P_k , потому что уменьшилась высота поддерева a_k ; можно использовать следующую процедуру корректировки: если $k = 0$, присвоить $LLINK(HEAD) \leftarrow LLINK(HEAD) - 1$; алгоритм завершен, так как уменьшилась высота всего дерева. В противном случае нужно рассмотреть показатель сбалансированности $V(P_k)$, возможны три варианта:

- i) $V(P_k) = a_k$. Установить $V(P_k) \leftarrow 0$, уменьшить k на 1 и повторить корректировку для нового значения k .
- ii) $V(P_k) = 0$. Установить $V(P_k) \leftarrow -a_k$; алгоритм завершен.
- iii) $V(P_k) = -a_k$. Требуется балансировка!

Балансировка требуется в общем в тех же случаях, что и в алгоритме вставки; полезно снова вернуться к (1), где в роли A выступает P_k , а в роли B — узел $LINK(-a_k, P_k)$, принадлежащий ветви, противоположной той, из которой производится удаление. Есть только одно отличие: узел B может быть сбалансированным, и тогда получим новый случай 3, аналогичный случаю 1, но β будет иметь высоту $h + 1$. В случаях 1 и 2 балансировка, подобно (2), означает, что мы уменьшаем высоту, устанавливая в качестве корня (2) узел $LINK(a_{k-1}, P_{k-1})$, уменьшая k на 1 и возобновляя процедуру корректировки для нового значения k . В случае 3 мы производим однократный поворот, что не изменяет общей высоты, но делает показатели сбалансированности как A , так и B ненулевыми; поэтому после занесения в $LINK(a_{k-1}, P_{k-1})$ адреса узла B алгоритм завершается.

Важная отличительная черта удаления состоит в том, что оно может потребовать до $\log N$ поворотов, в то время как для вставки всегда хватает одного. Причина этого станет ясна, если попытаться удалить самый правый узел дерева Фибоначчи (см. рис. 8, п 6.2.1).

Использование сбалансированных деревьев для представления линейных списков делает необходимым алгоритм *конкатенации*, когда целое дерево L_2 вставляется справа от дерева L_1 без нарушения баланса. Кларк Э. Крэйн придумал оригинальное решение этой задачи. Предположим, что "высота(L_1) \geq высоты(L_2)"; другой случай обрабатывается аналогичным образом. Удалим из L_2 первый узел, назвав его стыковочным узлом J , а через L'_2 обозначим новое дерево для $L_2 \setminus \{J\}$. После этого спускаемся по правым ветвям дерева L_1 , пока не достигнем узла P , такого, что

$$\text{высота}(P) - \text{высота}(L'_2) = 0 \text{ или } 1;$$

это всегда возможно, так как при переходе на более низкий уровень высота изменяется на 1 или 2. Теперь заменяем

Picture: рис. стр. 554

и приступаем к корректировке L_1 , как если бы новый узел J был вставлен посредством алгоритма А.

Крэйн решил также более трудную обратную задачу: *расщепить* список на две части, конкатенация которых дала бы первоначальный список. Рассмотрим, например, задачу расщепления списка рис. 20 для получения двух списков, состоящих из элементов $\{A, \dots, I\}$ и $\{J, \dots, Q\}$ соответственно: требуется существенная перекомпоновка поддеревьев. В общем случае, когда мы хотим расщепить дерево в данном узле P , путь к P будет похож на путь рис. 25. Мы хотели бы построить левое дерево, содержащее $\alpha_1, P_1, \alpha_4, P_4, \alpha_6, P_6, \alpha_7, P_7, \alpha, P$ в симметричном порядке, и правое дерево, содержащее $\beta, P_8, \beta_8, P_5, \beta_5, P_3, \beta_3, P_2, \beta_2$. Это можно сделать последовательностью конкатенаций: сначала вставляем P справа от α , затем конкатенируем β с β_8 , используя P_8 в качестве стыковочного узла,

Picture: Рис. 25. Задача расщепления списка.

конкатенируем α_7 с αP (узел P_7 стыковочный), α_6 с $\alpha_7 P_7 \alpha P$ (используя P_6), $\beta P_8 \beta_8$ с β_5 (используя P_5) и т. д.; узлы P_8, P_7, \dots, P_1 , лежащие на пути к P , используются как стыковочные. Крэйн доказал, что, если исходное дерево содержит N узлов, этот алгоритм расщепления требует лишь $O(\log N)$ единиц времени. Дело в том, что конкатенация с использованием данного узла в качестве стыковочного занимает $O(k)$ шагов, где k —разность высот конкатенируемых деревьев, а значения k , в сущности, образуют геометрическую прогрессию.

Все эти алгоритмы могут работать как с полями KEY, так и с полями RANK (или с обоими вместе), правда, в случае конкатенации все ключи дерева L_2 должны быть больше ключей L_1 . Для общих целей часто предпочтительнее использовать деревья с *тремя связями*, в которых наряду с полями LLINK и RLINK используется поле UP, указывающее на отца, и однобитовый признак того, является ли данный узел "левым" или "правым" сыном. Представление в виде деревьев с тремя связями немного упрощает алгоритмы и делает возможной спецификацию узла без явного указания пути к нему от корня; мы можем написать подпрограмму удаления по данному P узла NODE(P) или узла, следующего за NODE(P) в симметричном порядке (т. е. узла NODE(P\$)), или подпрограмму нахождения списка, содержащего NODE(P) и т. д. В алгоритме удаления для деревьев с тремя связями не нужно строить список (16), так как ссылки UP дают необходимую информацию. Разумеется, в этом случае при вставке, удалении или повороте придется менять несколько больше ссылок. Использование дерева с тремя связями вместо двух аналогично использованию двухсвязевого списка вместо односвязевого: отправляясь из любого места, мы можем идти как вперед, так и назад. Полное описание алгоритмов работы со списками, основанных на использовании сбалансированных деревьев с тремя связями, можно найти в докторской диссертации Кларка Э. Крэйна (Стэнфордский университет, 1972).

Конкуренты сбалансированных деревьев. Недавно было предложено несколько других способов организации бинарных деревьев, гарантирующих не более чем логарифмический рост времени поиска при увеличении числа узлов. К настоящему времени они изучены не полностью; возможно, что для некоторых ЭВМ эти способы окажутся предпочтительнее сбалансированных деревьев.

Интересное понятие дерева со *сбалансированным весом* изучено Ю. Нивергельтом, Э. Рейнгольдом и Ч. К. Уоном. Вместо высоты рассматривается вес дерева—по определению равный числу его внешних узлов—и на все узлы налагается условие

$$\sqrt{2} - 1 < \frac{\text{Вес левого поддерева}}{\text{Вес правого поддерева}} < \sqrt{2} + 1. \quad (17)$$

Можно показать, что баланс веса сохраняется после вставки, если, подобно алгоритму А, использовать для балансировки однократные и двукратные повороты (см. упр. 25). Однако во время одной вставки может потребоваться много поворотов. Если ослабить условие (17), то балансировка станет более быстрой, правда, за счет увеличения времени поиска.

На первый взгляд может показаться, что деревья со сбалансированным весом требуют больше памяти, чем обычные сбалансированные, но на самом деле иногда они требуют несколько меньше! Если в каждом узле уже есть поле RANK для представления линейного списка, то это в точности вес левого поддерева, а соответствующие правые веса можно запоминать при движении вниз по дереву. Кажется, однако, что накладные расходы, требующиеся для сохранения весового баланса, занимают больше времени, чем алгоритм А, и экономия двух битов на узел, вероятно, не является достаточной компенсацией.

Другой интересный подход к сбалансированным деревьям, названный "(3-2)-деревья", был предложен в 1970 г. Джоном Хопкрофтом [см. Aho, Hopcroft, Ullman, *The Design and Analysis of Computer Algorithms* (Reading, Mass.; Addison-Wesley, 1974), ch. 4]. Идея состоит в том, что из каждого узла могут выходить либо две, либо три ветви, но взамен требуется, чтобы все внешние узлы располагались на одном уровне. Каждый внутренний узел содержит либо один, либо два ключа, как показано на рис. 26.

Вставку в (3-2)-дерево объяснить несколько легче, чем вставку в сбалансированное дерево. Если мы хотим поместить

Picture: Рис. 26. (3-2)-дерево.

новый ключ в узел, содержащий ровно один ключ, мы просто вставляем его в качестве второго ключа. С другой стороны, если узел уже содержал два ключа, мы делим его на два одноключевых узла и вставляем третий ключ в узел-отец. Это в свою очередь может вызвать деление отца. На рис. 27 показан процесс вставки нового ключа в (3-2)-дерево рис. 26.

Хопкрофт заметил, что операции удаления, конкатенации и расщепления проводятся над (3-2)-деревьями достаточно просто и весьма похожи на соответствующие операции со сбалансированными деревьями.

Р. Бэйер [*Proc. ACM-SIGFIDET Workshop* (1971), 219–235] предложил интересное представление (3-2)-деревьев с помощью бинарных деревьев. На рис. 28 показано такое представление (3-2)-дерева рис. 26; один бит в каждом узле используется, чтобы отличить "горизонтальные" правые ссылки от "вертикальных". Заметим, что, как и в любом бинарном дереве поиска, ключи расположены слева направо в симметричном порядке. Оказывается, что при вставке нового ключа в такие деревья нужно проделывать те же операции, что и в случае сбалансированных деревьев, а именно однократные и двукратные повороты, правда, нужна лишь одна версия каждого поворота (нет отражений относительно вертикальной оси, имевшихся в алгоритмах А и С).

Picture: Рис. 27. Вставка нового ключа "M" в (3-2)-дерево рис. 26.

Picture: Рис. 28. (3-2)-дерево рис. 26, представленное как бинарное дерево поиска.

Упражнения

- [01] Почему в (1), случай 2, для восстановления баланса нельзя просто поменять местами левые поддерева узлов А и В?
- [16] Объяснить, почему дерево стало на один уровень выше, если мы достигли шага А7 с $B(S) = 0$.
- >3. [M25] Докажите, что сбалансированное дерево, имеющее N внутренних узлов, не может иметь более $(\phi - 1)N = 0.61803N$ узлов с ненулевыми показателями сбалансированности.
- [M22] Докажите или опровергните следующее: среди всех сбалансированных деревьев с $F_{n+1} - 1$ внутренними узлами дерево Фибоначчи порядка n имеет наибольшую длину внутреннего пути.
- >5. [M25] Докажите или опровергните следующее: если алгоритм А используется для вставки N ключей в первоначально пустое дерево и если эти ключи поступают в возрастающем порядке, то полученное дерево оптимально (т. е. оно имеет минимальную длину внутреннего пути среди всех бинарных деревьев с N узлами).
- [M21] Докажите, что (5) определяет производящую функцию для сбалансированных деревьев высоты h .
- [M27] (Н. Дж. А Слоан и А. В. Ахо.) Докажите замечательную формулу (9) для числа сбалансированных деревьев высоты h . [Указание: положить $C_n = B_n + B_{n-1}$ и использовать тот факт, что $\log(C_{n+1}/C_n^2)$ весьма мал при больших n .]

8. [M24] (Л. А. Хиздер.) Покажите, что существует константа β , такая, что $B'_h(l)/B_h(1) - 2^h\beta \rightarrow 1$ при $h \rightarrow \infty$.
9. [M47] Каково асимптотическое число сбалансированных бинарных деревьев с n внутренними узлами $\sum_{h>0} B_{nh}$? Какова асимптотическая средняя высота $\sum_{h>0} hB_{nh}/\sum_{h>0} B_{nh}$?
10. [M48] Верно ли, что при вставке N -го элемента алгоритм А совершает в среднем $\sim \log_2 N + c$ сравнений (c —некоторая константа)?
11. [22] Величина 0.144 трижды появляется в табл. 1: один раз при $k = l$ и дважды при $k = 2$. Величина $1/7$ встречается в тех же трех местах табл. 2. Является ли совпадением, что во всех трех местах стоят одинаковые величины, или на то есть причины?
12. [24] Чему равно максимальное время работы программы А при вставке восьмого узла в сбалансированное дерево? Каково минимально возможное время такой вставки?
13. [10] Почему поле RANK лучше использовать так, как в тексте, а не запоминать в качестве ключа номер узла ("1" в первом узле, "2" во втором и т. д.)?
14. [11] Алгоритмы сбалансированного дерева с помощью поля RANK были приспособлены для работы с линейными списками. Можно ли таким же образом приспособить алгоритмы 6.2.2Т и 6.2.2D?
15. [18] (К. Э. Крэйи.) Предположим, что упорядоченный линейный список представлен в виде бинарного дерева с полями KEY и RANK в каждом узле. Придумайте алгоритм, который разыскивает в дереве данный ключ K и определяет положение K в списке, т. е. находит число M , такое, что меньше K лишь $M - 1$ ключей.
- >16. [20] Нарисуйте сбалансированное дерево, которое получилось бы после удаления корневого узла F из дерева рис. 20 с помощью алгоритма удаления, предложенного в тексте.
- >17. [21] Нарисуйте сбалансированное дерево, которое получилось бы после конкатенации дерева Фибоначчи (12): (а) справа и (б) слева от дерева рис. 20, если использовать алгоритм конкатенации, предложенный в тексте.
18. [21] Нарисуйте сбалансированные деревья, которые получились бы после расщепления дерева рис. 20 на две части: $\{A, \dots, I\}$ и $\{J, \dots, Q\}$ —с помощью предложенного в тексте алгоритма расщепления.
- >19. [26] Найдите способ так преобразовать данное сбалансированное дерево, чтобы показатель сбалансированности корня стал отличен от -1 . Ваше преобразование должно сохранять симметричный порядок узлов; оно должно порождать искомое сбалансированное дерево за $O(1)$ единиц времени независимо от числа его узлов.
20. [40] Рассмотрите идею использования ограниченного класса сбалансированных деревьев, все узлы которых имеют показатели сбалансированности 0 или $+1$. (Тогда поле В можно свести к одному биту.) Существует ли для таких деревьев процедура вставки с разумной эффективностью?
- >21. [30] Придумайте алгоритм, который строил бы оптимальные (в смысле упр. 5) N -узловые бинарные деревья за $O(N)$ шагов. Ваш алгоритм должен быть "диалоговым" в том смысле, что он получает узлы по одному в возрастающем порядке и строит частичные деревья, не зная заранее окончательного значения N . (Такой алгоритм можно было бы использовать при перестройке плохо сбалансированного дерева или при слиянии ключей из двух деревьев в одно дерево.)
22. [M20] Каков аналог теоремы А для деревьев со сбалансированным весом?
23. [M20] (Э. Рейнгольд.) (а) Докажите, что существуют сбалансированные деревья с произвольно малым весовым отношением "(вес левого поддерева)/(вес правого поддерева)". (б) Докажите, что существуют деревья со сбалансированным весом, имеющие сколь угодно большую разность между высотами левого и правого поддеревьев.
24. [M22] (Э. Рейнгольд.) Докажите, что единственными бинарными деревьями, удовлетворяющими усиленному условию (17)

$$\frac{1}{2} < \frac{\text{Вес левого поддерева}}{\text{Вес правого поддерева}} < 2,$$

являются идеально сбалансированные деревья с $2^n - 1$ внутренними узлами. (В таких деревьях левые и правые веса в каждом узле равны между собой.)

25. [27] (Ю. Нивергельт, Э. Рейнгольд, Ч. Уон.) Покажите, что можно придумать алгоритм вставки для деревьев со сбалансированным весом, сохраняющий условие (17) и производящий не более $O(\log N)$ поворотов на вставку.
26. [40] Исследуйте свойства сбалансированных t -арных деревьев для $t > 2$.
- >27. [M23] Оцените максимальное число сравнений, необходимых для поиска в (3-2)-дереве с N внутренними узлами.
28. [41] Дайте эффективную реализацию алгоритмов (3-2)-деревя.
29. [M47] Проанализируйте поведение (3-2)-деревьев в среднем при случайных вставках.

30. [26] (Э. Мак-Крэйт.) В § 2.5 обсуждались различные стратегии динамического распределения памяти, включая "наиболее подходящий" (выбор области наименьшего размера, удовлетворяющей запросу) и "первый подходящий" (выбор области с наименьшим адресом, удовлетворяющей запросу). Покажите, что если свободное пространство связано в бинарное дерево, в некотором смысле, сбалансированное, то можно найти (а) "наиболее подходящую" и (б) "первую подходящую" области в $O(\log n)$ единиц времени, где n есть число свободных областей памяти. (Алгоритмы § 2.5 совершают порядка n шагов.)
31. [34] (М. Фредмэн.) Придумайте представление линейных списков, обладающее тем свойством, что вставка нового элемента между позициями $m - 1$ и m при данном m требует $O(\log m)$ единиц времени.

6.2.4. Сильно ветвящиеся деревья

Изученные нами методы поиска по дереву были развиты в основном для внутреннего поиска, т. е. когда исследуемая таблица целиком содержится в быстрой внутренней памяти ЭВМ. Теперь же рассмотрим задачу внешнего поиска, когда нужно выбрать информацию из очень большого файла, расположенного на внешних запоминающих устройствах с прямым доступом, таких, как магнитные диски или барабаны. (О дисках и барабанах можно прочитать в п. 5.4.9.)

Picture: Рис. 29. Большое бинарное дерево поиска можно разделить на "страницы".

Древоподобные структуры довольно удобны для внешнего поиска (разумеется, если они представлены подходящим образом). Рассмотрим большое бинарное дерево поиска (рис. 29) и предположим, что оно хранится в дисковом файле. (Поля `LLINK` и `RLINK` содержат теперь адреса на диске, а не адреса внутренней памяти.) Если мы проявим простодушие и будем буквально следовать изученным алгоритмам поиска по дереву, нам понадобится около $\log_2 N$ обращений к диску. При N , равном миллиону, их будет примерно 20. Но если разделить дерево на "страницы" по 7 узлов в каждой, как показано пунктиром на рис. 29, и если в каждый момент времени доступна лишь одна страница, то потребуется примерно в три раза меньше обращений, т. е. поиск ускорится в три раза!

Группировка узлов в страницы, по существу, преобразует бинарные деревья в октанные, разветвляющиеся в каждой странице-узле на 8 путей. Если допустимы страницы больших размеров, разветвляющиеся на 128 путей после каждого обращения к диску, то можно находить требуемый ключ в таблице из миллиона элементов, просмотрев лишь три страницы. Можно постоянно хранить корневую страницу во внутренней памяти; тогда потребуются лишь два обращения к диску, хотя в любой момент времени во внутренней памяти будет не более 254 ключей.

Разумеется, не следует делать страницы очень большими, так как размеры внутренней памяти ограничены и чтение большей страницы занимает больше времени. Предположим, например, что чтение страницы, допускающей разветвление на m путей, занимает $72.5 + 0.05m$ мс. Время на внутреннюю обработку каждой страницы составит около $a + b \log m$ мс, где a мало по сравнению с 72.5, так что полное время, требующееся на поиск в большой таблице, примерно пропорционально $\log N$, умноженному на

$$(72.5 + 0.05m) / \log m + b.$$

Эта величина достигает минимума при $m \approx 350$; в действительности этот минимум очень "широк". Значения, очень близкие к оптимальному, достигаются при m от 200 до 500. На практике получение подобного диапазона хороших значений m зависит от характеристик используемых внешних запоминающих устройств и от длины записей в таблице.

У. И. Ландауэр [IEE Trans., EC-12 (1963), 863-871] предложил строить m -арные деревья следующим образом: размещать узлы на уровне $l + 1$, лишь если уровень l почти заполнен. Эта схема требует довольно сложной системы поворотов, так как для вставки одного нового элемента может потребоваться основательная перестройка дерева; Ландауэр исходил из предположения, что поиск приходится производить гораздо чаще вставок и удалений.

Если файл хранится на диске и является объектом сравнительно редких вставок и удалений, то для его представления подходит трехуровневое дерево, где первый уровень разветвления определяет, какой цилиндр будет использоваться, следующий уровень разветвления определяет соответствующую дорожку на этом цилиндре, а третий уровень содержит собственно записи. Такой метод называется *индексно-последовательной* организацией файла [ср. JACM, 16 (1969), 569-571].

Р. Мюнц и Р. Узгалис [Proc. Princeton Conf. on Inf. Sciences and Systems, 4 (1970), 345-349] предложили модификацию алгоритма 6.2.2Т, где все вставки порождают узлы, принадлежащие той же странице, что и их предшественник (если только это возможно); если страница полностью занята, заводится новая страница (если таковая имеется). При неограниченном количестве страниц

и случайном порядке поступающих данных среднее число обращений, к диску, как можно показать, приближенно равно $H_N/(H_m - 1)$, что лишь немногим больше числа обращений в случае наилучшего возможного m -арного дерева (см. упр. 10).

***B*-деревья.** В 1970 г. Р. Бэйер и Э. Мак-Крэйт [*Acta Informatica*, (1972), 173–189] и независимо от них примерно в то же время М. Кауфман [неопубликовано] предложили новый подход к внешнему поиску посредством сильно ветвящихся деревьев. Их идея основывается на подвижности нового типа структур данных, названных *B*-деревьями, и позволяет производить поиск или изменять большой файл с "гарантированной" эффективностью в наихудшем случае, используя при этом сравнительно простые алгоритмы.

B-деревом порядка m называется дерево, обладающее следующими свойствами:

- i) Каждый узел имеет не более m сыновей.
- ii) Каждый узел, кроме корня и листьев, имеет не менее $m/2$ сыновей.
- iii) Корень, если он не лист, имеет не менее 2 сыновей.
- iv) Все листья расположены на одном уровне и не содержат информации.
- v) Нелистовой узел с k сыновьями содержит $k - 1$ ключей.

(Как и обычно, лист—концевой узел, у него нет преемников. Так как листья не содержат информации, их можно рассматривать как внешние узлы, которых в действительности нет в дереве, так что Λ —это указатель на лист.)

На рис. 30 показано *B*-дерево порядка 7. Каждый узел (кроме корня и листьев) имеет от $\lceil 7/2 \rceil$ до 7 преемников и содержит 3, 4, 5 или 6 ключей. Корневой узел может содержать от 1 до 6 ключей (в данном случае 2). Все листья находятся на третьем уровне. Заметьте, что (а) ключи расположены в возрастающем порядке слева направо, если использовать естественное обобщение понятия симметричного порядка, и (б) количество листьев ровно на единицу больше количества ключей.

B-деревья порядка 1 и 2, очевидно, неинтересны; поэтому мы рассмотрим лишь случай $m \geq 3$. Заметьте, что (3-2)-деревья,

Picture: Рис. 30. *B*-дерево порядка 7

определенные в конце п. 6.2.3, являются *B*-деревьями порядка 3; и обратно, *B*-дерево порядка 3 есть (3-2)-дерево.

Узел, содержащий j ключей и $j + 1$ указателей, можно представить в виде

Picture: Узел *B*-дерева

где $K_1 < K_2 < \dots < K_j$, а P_i указывает на поддереву ключей, больших K_i и меньших K_{i+1} . Следовательно, поиск в *B*-дереве абсолютно прямолинеен: после того как узел (1) размещен во внутренней памяти, мы ищем данный аргумент среди ключей K_1, K_2, \dots, K_j . (При больших j , вероятно, производится бинарный поиск, а при малых j наилучшим будет последовательный поиск.) Если поиск удачен, нужный ключ найден; если поиск неудачен в силу того, что аргумент лежит между K_i и K_{i+1} , мы "подкачиваем" (вызываем в оперативную память) узел, указанный P_i , и продолжаем процесс. Указатель P_0 используется, если аргумент меньше K_1 , а P_j используется, если аргумент больше K_j . При $P_i = \Lambda$ поиск неудачен.

Привлекательной чертой *B*-деревьев является исключительная простота, с которой производятся вставки. Рассмотрим, например, рис. 30; каждый лист соответствует месту возможной вставки. Если нужно вставить новый ключ 337, мы просто меняем узел

Picture: Вставка в *B*-дерево

С другой стороны, при попытке вставить новый ключ 071 мы обнаруживаем, что в соответствующем узле уровня 2 нет места—он уже "полон". Эту трудность можно преодолеть, расщепив узел на две части по три ключа в каждой и подняв средний ключ на уровень 1:

Picture: Расщепление узла при вставке

Вообще, если нужно вставить новый элемент в *B*-дерево порядка m , все листья которого расположены на уровне l , новый ключ вставляют в подходящий узел уровня $l - 1$. Если узел теперь содержит m ключей, т. е. имеет вид (1) с $j = m$, его расщепляют на два узла

Picture: Расщепление узла: общий вид

и вставляют ключ $K_{\lceil m/2 \rceil}$ в узел-отец. (Таким образом, указатель P в нем заменяется последовательностью $P, K_{\lceil m/2 \rceil}, P'$.) Эта вставка в свою очередь может привести к расщеплению узла-отца. (Ср. с рис. 27, где изображен случай $m = 3$.) Если нужно расщепить корневой узел, который не имеет отца, то просто создают новый корень и помещают в него единственный ключ $K_{\lceil m/2 \rceil}$, в таком случае дерево становится на единицу выше.

Описанная процедура вставки сохраняет все свойства B -деревьев; чтобы оценить всю красоту идеи, читатель должен выполнить упр. 1. Заметьте, что дерево мало-помалу растет вверх от верхней части, а не вниз от нижней части, так как единственная причина, способная вызвать рост дерева, — расщепление корня.

Удаления из B -деревьев лишь немногим сложнее вставок (см. упр. 7).

Гарантированная эффективность. Рассмотрим, к скольким узлам будет происходить обращение в наихудшем случае при поиске в B -дереве порядка m . Предположим, что имеется N ключей, а на уровне l расположено $N + 1$ листьев. Тогда число узлов на уровнях 1, 2, 3, ... не менее 2, $2^{\lceil m/2 \rceil}$, $2^{\lceil m/2 \rceil^2}$, ...; следовательно,

$$N + 1 \geq 2^{\lceil m/2 \rceil^{l-1}}. \quad (5)$$

Иными словами,

$$l \leq 1 + \log_{\lceil m/2 \rceil} \left(\frac{N + 1}{2} \right); \quad (6)$$

это означает, например, что при $N = 1999998$ и $m = 199$ имеем $l \leq 3$. Так как при поиске происходит обращение самое большее к l узлам, эта формула гарантирует малое время работы алгоритма.

При вставке нового ключа может понадобиться расщепить l узлов. Однако среднее число расщепляемых узлов много меньше, так как общее количество расщеплений при построении дерева равно на единицу меньше количества узлов, которое мы обозначим через p . В дереве не менее $1 + (\lceil m/2 \rceil - 1)(p - 1)$ ключей; следовательно,

$$p \leq 1 + \frac{N - 1}{\lceil m/2 \rceil - 1}. \quad (7)$$

Это означает, что среднее число расщеплений, требующихся на одну вставку, меньше $1/(\lceil m/2 \rceil - 1)$.

Улучшения и изменения. Предложенные методы можно улучшить, если слегка изменить определение B -дерева. Прежде всего заметим, что все указатели в узлах на уровне $l - 1$ равны Λ и не равны Λ на других уровнях. Это часто представляет собой значительную потерю пространства, и мы можем сэкономить как пространство, так и время, если исключим все Λ и будем использовать для всех нижних узлов другое значение m . Использование двух различных m не ухудшает алгоритма вставки, так как обе половинки расщепленного узла остаются на том же уровне, что и первоначальный узел. Можно определить обобщенное B -дерево порядков m_1, m_2, m_3, \dots , потребовав, чтобы все некорневые узлы на уровне $l - i$ имели от $m_i/2$ до m_i преемников; у подобного B -дерева свое m для каждого уровня, однако алгоритм вставки работает, в сущности, по-прежнему.

Развивая идею предыдущего абзаца, мы можем использовать совершенно разные форматы узлов для разных уровней дерева, можем, кроме того, хранить в листьях информацию. Иногда ключи составляют лишь небольшую часть записей, и в таких случаях было бы ошибкой хранить записи целиком в узлах, близких к корню; это привело бы к слишком малому значению m и не дало бы эффективного разветвления.

Уместно поэтому снова рассмотреть рис. 30, представляя себе, что все записи файла хранятся теперь в листьях и что лишь несколько ключей продублировано в остальных узлах. В таком случае самый левый лист содержит все записи с ключами ≤ 011 ; лист, отмеченный буквой A , содержит записи, ключи которых удовлетворяют соотношению $439 < K \leq 449$, и т. п. Листья растут и расщепляются, как другие узлы, но с той существенной разницей, что записи не перемещаются из листьев вверх на следующий уровень. Таким образом, листья всегда заполнены не менее чем наполовину; при их расщеплении в "нелистовую" часть дерева поступает новый ключ. Если каждый лист связан со своим преемником отношением симметричного порядка, мы получаем возможность эффективно и удобно обходить файл как последовательно, так и случайным образом.

Некоторые вычисления С. П. Гхоша и М. Х. Сенко [JACM, 16 (1969), 569–579] наводят на мысль о том, что полезно иметь довольно большие листья; скажем до 10 последовательных страниц в длину. Зная диапазон ключей для каждого листа, с помощью линейной интерполяции можно предугадать, какая из 10 страниц содержит данный аргумент поиска. Если догадка окажется неверной, мы потеряем время, но эксперименты показывают, что эта потеря может быть меньше, чем экономия времени от уменьшения размера дерева.

Т. Х. Мартин (не опубликовано) указал на то, что идея, лежащая в основе B -деревьев, полезна также при работе с ключами *переменной длины*. Нет нужды заключать число сыновей каждого узла в

пределы $[m/2, m]$; вместо этого можно просто сказать, что каждый узел должен быть заполнен данными не менее чем наполовину. Механизм вставки и расщепления работает по-прежнему превосходно, хотя точное число ключей в каждом узле зависит от их длин. Однако нельзя позволять ключам быть слишком длинными (упр. 5).

Другой важной модификацией схемы B -дерева является идея "переливания", введенная Бэйером и Мак-Крэйтом. Она состоит в том, чтобы улучшить алгоритм вставки путем уменьшения количества расщеплений; вместо них используются локальные повороты. Пусть имеется переполненный узел, содержащий m ключей и $m+1$ указателей; прежде чем расщеплять этот узел, стоит обратить внимание на его брата справа, содержащего, например, j ключей и $j+1$ указателей. В узле-отце существует ключ \bar{K}_f , разделяющий ключи этих двух братьев; схематически это можно изобразить так:

Picture: Рис. стр. 568

При $j < m-1$ простое перераспределение делает расщепление ненужным: мы оставляем $\lceil (m+j)/2 \rceil$ ключей в левом узле, заменяем в узле-отце \bar{K}_f на $K_{\lfloor (m+j)/2 \rfloor + 1}$, а остальные ключи (их $\lceil (m+j)/2 \rceil$) и соответствующие указатели помещаем в правый узел. Таким образом, избыток "переливается" в соседний узел. Если же он был полон ($j = m-1$), можно расщепить оба узла, получая три заполненных примерно на две трети узла, содержащих соответственно $\lfloor (2m-2)/3 \rfloor$, $\lfloor (2m-1)/3 \rfloor$ и $\lfloor 2m/3 \rfloor$ ключей:

Picture: Рис. стр. 568

Если у исходного узла нет правого брата, уместно описанным образом использовать левого. (Если же существуют оба брата, можно воздержаться от расщепления, пока оба не заполнятся.) Наконец, если расщепляемый узел не имеет братьев, то это обязательно корень; мы можем изменить определение B -дерева, допустив, чтобы корень содержал до $2\lfloor (2m-2)/3 \rfloor$ ключей, так что при своем расщеплении он порождает два узла, в каждом из которых $\lfloor (2m-2)/3 \rfloor$ ключей.

Приведенные рассуждения позволяют "вывести новый, лучший вид" деревьев; назовем их B^* -деревьями и определим следующим образом:

- i) Каждый узел, кроме корня, имеет не более m сыновей.
- ii) Каждый узел, кроме корня и листьев, имеет не менее $(2m-1)/3$ сыновей.
- iii) Корень имеет не менее 2 и не более $2\lfloor (2m-2)/3 \rfloor + 1$ сыновей.
- iv) Все листья расположены на одном уровне.
- v) Узлы, не являющиеся листьями и имеющие k сыновей, содержат $k-1$ ключей.

Важное изменение доставляет условие (ii), утверждающее, что мы используем по крайней мере две трети имеющегося в каждом узле пространства. Тем самым экономится и время, так как в формулах (6) и (7) величину " $\lceil m/2 \rceil$ " следует заменить на " $\lceil (2m-1)/3 \rceil$ ".

Возможно, читатель отнесся к B -деревьям скептически из-за того, что степень корня может быть равна 2. Зачем тратить целое обращение к диску ради разветвления всего лишь на два пути? Простая схема буферизации, называемая "замещением дольше всего не использовавшейся страницы", помогает преодолеть это неудобство; мы можем выделить во внутренней памяти несколько буферов, так что команда реального ввода будет не нужна, если соответствующая страница уже присутствует в памяти. При такой схеме алгоритмы поиска или вставки выполняют команды "виртуального чтения", которые на самом деле транслируются в инструкции ввода, лишь если нужной страницы нет в памяти; последующая команда "освобождения" выполняется, когда информация в буфере обработана и, возможно, изменена алгоритмом. Если требуется действительное чтение, мы выбираем буфер, освобожденный ранее других, записываем его обратно на диск, если он изменился с момента считывания, затем в выбранный буфер считываем нужную страницу.

Так как число уровней дерева обычно мало по сравнению с числом буферов, такая страничная система обеспечивает постоянное присутствие в памяти корневой страницы; то же самое можно утверждать о страницах первого уровня, если корень имеет два или три сына. Можно было бы ввести специальный механизм для обеспечения постоянного присутствия определенного минимального количества близких к корню страниц. Заметьте, что предложенная схема автоматически обеспечивает присутствие в памяти страниц, необходимых для выполнения расщеплений, которые могут понадобиться при вставках.

Судя по экспериментам, проведенным Э. Мак-Крэйтом, эта идея оказалась весьма удачной. Например, он обнаружил, что с десятью буферными страницами и $m = 121$ процесс вставки в возрастающем порядке 100000 ключей потребовал лишь 22 действительные команды чтения и 857 действительных команд записи; таким образом, большая часть действий производилась во внутренней

памяти. Далее, дерево содержало лишь 835 узлов—ровно на 1 больше минимально возможного значения $\lceil 100000/(m-1) \rceil = 834$, т. е. память использовалась почти на 100%. Для этого эксперимента Мак-Крэйт применил технику переливания, но с расщеплением на две части, как в (4), а не на три, как в (9). (См. упр. 3.)

В другом эксперименте (тоже с 10 буферами, $m = 121$ и той же техникой переливания) он вставил в первоначально пустое дерево 5000 ключей в случайном порядке. Получилось двухуровневое дерево с 48 узлами (память использовалась на 87%), и потребовались 2762 действительные команды считывания и 2739 действительных команд записи. После этого 1000 случайных поисков 786 раз производили действительные считывания. Тот же эксперимент без метода переливания после 2743 действительных считываний и 2800 действительных записей привел к двухуровневому дереву с 62 узлами (67% использования памяти); 1000 последовательных случайных поисков потребовали 836 действительных считываний. Это показывает не только эффективность страничной схемы, но также полезность локальной обработки переполнения вместо немедленного расщепления. Эндрю Яо доказал, что среднее число узлов после случайных вставок без переливания при больших N и m составит $N/(m \ln 2) + O(N/m^2)$, так что память будет использоваться примерно на $\ln 2 = 0.693$ [Acta Informatica, будет опубликовано].

Упражнения

1. [10] Какое B -дерево порядка 7 получится после вставки ключа 613 в дерево рис. 30? (Метод переливания не применять.)
2. [15] Выполните упр. 1, используя метод переливания с расщеплением на три части, как в (9).
- >3. [23] Предположим, что ключи 1, 2, 3, ... в возрастающем порядке вставляются в первоначально пустое B -дерево порядка 101. Какой ключ впервые приведет к появлению листьев на уровне 4: (а) если не использовать переливания; (б) если использовать переливания лишь с расщеплением на две части, как в (4); (с) если использовать B^* -деревья порядка 101 с расщеплением на три части, как в (9)?
4. [21] (Бэйер и Мак-Крэйт.) Объясните, как производить вставки в обобщенное B -дерево, чтобы все узлы, кроме корня и листьев, имели не менее $\frac{3}{4}m - \frac{1}{2}$ сыновей.
- >5. [21] Предположим, что под узлы отводится по 1000 байтов позиций для литер на внешней памяти. Если каждая ссылка занимает 5 байтов и если ключи имеют переменную, но кратную 5 длину от 5 до 50 байтов, то какое минимальное число байтов может быть занято в узле после его расщепления в процессе вставки? (Рассмотрите лишь простую процедуру расщепления, аналогичную описанной в тексте для B -деревьев с ключами фиксированной длины, без переливания.)
6. [22] Можно ли использовать идею B -дерева для выборки элементов линейного списка по позиции, а не по значению ключа? (Ср. с алгоритмом 6.2.3В.)
7. [23] Придумайте алгоритм удаления для B -деревьев.
8. [28] Придумайте алгоритм конкатенации B -деревьев (ср. с п. 6.2.3).
9. [30] Обдумайте, как большой файл, организованный в виде B -дерева, можно использовать для доступа и изменения со стороны большого числа одновременно работающих пользователей так, чтобы пользователи различных страниц практически не создавали помех друг другу.
10. [BM37] Рассмотрим обобщение вставки в дерево, предложенное Мюнцем и Узгалисом, когда каждая страница может содержать M ключей. Пусть мы вставили в такое дерево N случайных элементов; теперь оно имеет $N + 1$ внешних узлов. Обозначим через $b_{Nk}^{(j)}$ вероятность того, что неудачный поиск требует обращений к k страницам и что предшественник того внешнего узла, где кончается поиск, принадлежит к странице, содержащей j ключей. Пусть $B_N^{(j)}(z) = \sum b_{Nk}^{(j)} z^k$ — соответствующая производящая функция. Докажите, что

$$\begin{aligned}
 B_1^{(j)}(z) &= \delta_{j1}z; \\
 B_N^{(j)}(z) &= \frac{N-j-1}{N+1} B_{N-1}^{(j)}(z) + \frac{j+1}{N+1} B_{N-1}^{(j-1)}(z), \quad 1 < j < M; \\
 B_N^{(1)}(z) &= \frac{N-2}{N+1} B_{N-1}^{(1)}(z) + \frac{2z}{N+1} B_{N-1}^{(M)}(z); \\
 B_N^{(M)}(z) &= \frac{N-1}{N+1} B_{N-1}^{(M)}(z) + \frac{M+1}{N+1} B_{N-1}^{(M-1)}(z).
 \end{aligned}$$

Найдите асимптотическое поведение $C'_N = \sum_{1 \leq j \leq M} B_N^{(j)'}(1)$ — среднего числа страниц, к которым происходит обращение в процессе неудачного поиска. [Указание: выразить рекуррентное

соотношение с помощью матрицы

$$W(z) = \begin{pmatrix} -3 & 0 & \dots & 0 & 2z \\ 3 & -4 & \dots & 0 & 0 \\ 0 & 4 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & -M-1 & 0 \\ 0 & 0 & \dots & M+1 & -2 \end{pmatrix}$$

и сопоставить C'_N с многочленом N -й степени в $W(l)$.]

6.3. Цифровой поиск

Вместо того чтобы основывать метод поиска на сравнении ключей, можно воспользоваться их представлением в виде последовательности цифр или букв. Рассмотрим, например, имеющиеся во многих словарях "побуквенные высечки"; по первой букве данного слова мы можем немедленно нащупать страницы, содержащие все слова, начинающиеся с этой буквы.

Если развить идею "побуквенных высечек", мы придем к схеме поиска, основанной на "индексировании", как показано в табл. 1. Предположим, требуется проверить, принадлежит ли данный аргумент поиска к 31 наиболее употребительному английскому слову (ср. с рис. 12 и 13, п. 6.2.2). В табл. 1 данные представлены в виде так называемой структуры "бора"; термин введен Э. Фрэдкином [САСМ, 3 (1960), 490–500] как часть слова *выборка*⁶ (информации). Бор в сущности является M -арным деревом, узлы которого суть M -местные векторы с компонентами, соответствующими цифрам или буквам. Каждый узел уровня l представляет множество всех ключей, начинающихся с определенной последовательности из l литер; узел определяет M -путевое разветвление в зависимости от $(l+1)$ -й литеры.

Например, бор табл. 1 имеет 12 узлов; узел 1—корень, и первую букву следует искать здесь. Если первой оказалась, скажем, буква N, из таблицы видно, что нашим словом будет NOT (или же, что его нет в таблице). С другой стороны, если первая буква—W, узел (1) направит нас к узлу (9), где мы должны аналогичным образом отыскать вторую букву; узел (9) укажет, что этой буквой должна быть A, H или I.

Узлы-векторы в табл. 1 организованы в соответствии с кодом литер, принятым для MIX. Это означает, что поиск по бору будет довольно быстрым, так как мы должны просто выбирать слова из массива, используя в качестве индексов литеры нашего ключа. Методы быстрого многопутевого разветвления с помощью индексирования называются "просмотром таблиц" ("Table Look-At") в противоположность "поиску по таблицам" ("Table Look-Up") [см. P. M. Sherman, САСМ, 4 (1961), 172–173, 175].

⁶ В оригинале соответственно trie (искаженное "tree") и retrieval—Прим. перев.

Алгоритм Т. (Поиск по бору.) Алгоритм позволяет найти данный аргумент K в таблице записей, образующих M -арный бор.

Таблица 1

”Бор” для 31 наиболее употребительного английского слова

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
А	---	A	---	---	---	I	---	---	---	---	HE	---
В	(2)	---	---	---	(10)	---	---	---	WAS	---	---	THAT
С	(3)	---	---	---	---	---	---	---	---	---	---	---
Д	---	---	---	---	---	---	---	---	---	HAD	---	---
Е	---	---	BE	---	(11)	---	---	---	---	---	---	THE
Ж	(4)	---	---	---	---	---	OF	---	---	---	---	---
З	---	---	---	---	---	---	---	---	---	---	---	---
И	(5)	---	---	---	---	---	---	(12)	WHICH	---	---	---
Й	(6)	---	---	---	HIS	---	---	---	WITH	---	---	THIS
К	---	---	---	---	---	---	---	---	---	---	---	---
Л	---	---	---	---	---	---	---	---	---	---	---	---
М	---	---	---	---	---	---	---	---	---	---	---	---
Н	NOT	AND	---	---	---	IN	ON	---	---	---	---	---
О	(7)	---	---	FOR	---	---	---	TO	---	---	---	---
П	---	---	---	---	---	---	---	---	---	---	---	---
Р	---	---	---	---	---	---	---	---	---	---	---	---
С	---	---	---	---	---	---	---	---	---	---	---	---
Т	---	ARE	---	FROM	---	---	OR	---	---	---	HER	---
У	---	---	---	---	---	---	---	---	---	---	---	---
Ф	---	---	---	---	---	---	---	---	---	---	---	---
Х	---	---	---	---	---	---	---	---	---	---	---	---
Ц	---	AS	---	---	---	IS	---	---	---	---	---	---
Ч	(8)	AT	---	---	---	IT	---	---	---	---	---	---
Ш	---	---	BUT	---	---	---	---	---	---	---	---	---
Щ	---	---	---	---	---	---	---	---	---	HAVE	---	---
Ъ	(9)	---	---	---	---	---	---	---	---	---	---	---
Ы	---	---	---	---	---	---	---	---	---	---	---	---
Ь	YOU	---	BY	---	---	---	---	---	---	---	---	---
Э	---	---	---	---	---	---	---	---	---	---	---	---

Узлы бора представляют собой векторы, индексы которых изменяются от 0 до $M - 1$; каждая компонента этих векторов есть либо ключ, либо ссылка (возможно, пустая).

T1 [Начальная установка.] Установить указатель P так, чтобы он указывал на корень бора.

T2 [Разветвление.] Занести в k следующую (стоящую правее) литеру аргумента K . (Если аргумент полностью обработан, мы засылаем в k ”пробел” или символ конца слова. Литера должна быть представлена целым числом в диапазоне $0 \leq k < M$.) Через X обозначим k -й элемент в узле $\text{NODE}(P)$. Если X —ссылка, перейти на **T3**; если X —ключ, то перейти на **T4**.

T3 [Продвижение.] Если $X \neq \Lambda$, установить $P \leftarrow X$ и вернуться на **T2**; в противном случае алгоритм заканчивается неудачно.

T4 [Сравнение.] Если $K = X$, алгоритм заканчивается удачно; в противном случае он заканчивается неудачно. ■

Заметим, что, если поиск неудачен, все же будет найден элемент, лучше всего совпадающий с аргументом поиска. Это свойство полезно для некоторых приложений.

Чтобы сравнить время работы алгоритма и других алгоритмов данной главы, можно написать короткую программу для MIX, предполагая, что литера занимает один байт, а ключи— не более пяти байтов.

Программа Т. (Поиск по бору.) Предполагается, что все ключи занимают одно слово машины MIX; если ключ состоит менее чем из пяти литер, он дополняется справа пробелами. Так как мы используем для представления литер код MIX, каждый байт аргумента поиска должен содержать число, меньшее 30. Ссылки представлены как отрицательные числа в поле 0 : 2. Значения регистров: rI1 \equiv P, rX \equiv необработанная часть K.

START	LDX	K	1	T1. Начальная установка.
	ENT1	ROOT	1	P \leftarrow указатель на корень бора.
2H	SLAX	1	C	T2. Разветвление.
	STA	*+1 (2 : 2)	C	Выделение следующей литеры k.
	ENT2	0, 1	C	Q \leftarrow P + k.
	LD1N	0, 2 (0 : 2)	C	P \leftarrow LINK(Q).
	J1P	2B	C	T3. Продвижение. На T2, если P—ссылка \neq Λ .
	LDA	0, 2	1	T4. Сравнение. rA \leftarrow KEY(Q).
	CMPA	K	1	
	JE	SUCCESS	1	Удачный выход, если rA = k.
FAILURE	EQU	*	1	Выход, если K нет в бору.

Время работы программы равно $(8C + 8)u$, где C—число проверяемых литер. Так как $C \leq 5$, поиск не займет более 48 единиц времени.

Если теперь сравнить эффективность этой программы (использующей бор табл. 1) и программы 6.2.2Т (использующей *оптимальное* бинарное дерево поиска (рис. 13)), можно заметить следующее:

1. Бор занимает гораздо больше памяти: для представления 31 ключа использовано 360 слов, в то время как бинарное дерево поиска занимает 62 слова памяти. (Однако упр. 4 показывает, что, проявив известную изобретательность, можно вместить бор табл. 1 в 49 слов.)

Picture: Рис. 31. Бор табл. 1, преобразованный в "лес".

2. Удачный поиск в обоих случаях требует около 26 единиц времени. Но неудачный поиск оказывается более быстрым для бора и более медленным для бинарного дерева поиска. Для наших данных поиск чаще будет неудачным, нежели удачным, поэтому с точки зрения скорости бор более предпочтителен.

3. Если вместо 31 наиболее употребительного английского слова рассмотрим применение указателя KWIC (рис. 15), бор теряет свои преимущества вследствие характера данных. Например, бор требует 12 итераций, чтобы различить слова COMPUTATION и COMPUTATIONS. (В этом случае было бы лучше так построить бор, чтобы слова обрабатывались справа налево, а не слева направо.)

Первая публикация о боровой памяти принадлежит Рене де ла Брианде [Proc. Western Joint Computer Conf., 15 (1959), 295–298]. Он указал, что можно сэкономить память (за счет увеличения времени работы), если использовать связанный список для каждого узла-вектора, так как большинство элементов этого вектора обычно пусто. На самом деле предложенная идея ведет к замене бора табл. 1 на лес деревьев, изображенный на рис. 31. При поиске в таком лесу мы находим корень, соответствующий первой литере, затем сына этого корня, соответствующего второй литере, и т. д.

Фактически в своей статье де ла Брианде не прерывал разветвления так, как показано в табл. 1 или на рис. 31; вместо этого он представлял каждый ключ литеры за литерой до достижения признака конца слова. Таким образом, в действительности де ла Брианде вместо дерева "H" (рис. 31) использовал бы

Picture: Рис. стр. 576

Такое представление требует больше памяти, но делает обработку данных переменной длины особенно легкой. Если использовать на литеру два поля ссылок, можно легко организовать динамические вставки и удаления. Более того, во многих приложениях аргумент поиска поступает в "распакованной" форме, т. е. каждая литера занимает целое слово, и такое дерево позволяет избежать "упаковки" перед проведением поиска.

Если использовать обычный способ представления деревьев в бинарном виде, то (1) преобразуется в бинарное дерево (Для представления полного леса рис. 31 необходимо добавить указатель, ведущий из H направо, к соседнему корню I.) При поиске в этом бинарном дереве литеры аргумента сравниваются с литерами дерева; мы следуем по ссылкам RLINK, пока не найдем нужную литеру; затем берем ссылку LLINK и следующую литеру аргумента рассматриваем аналогичным образом.

Поиск в таком бинарном дереве в большей или меньшей степени сводится к сравнениям, причем разветвление происходит не по признаку "меньше-больше", а по "равно-неравно". Элементарная теория п. 6.2.1 гласит, что в среднем на то, чтобы различить N ключей, нужно по крайней мере $\log_2 N$ сравнений; среднее число сравнений, производимых при поиске по дереву, подобному изображенному на рис. 31, не меньше числа проверок при использовании методов бинарного поиска, изложенных в § 6.2.

С другой стороны, бор табл. 1 позволяет сразу производить M -путевое разветвление; мы увидим, что при больших N поиск в среднем включает лишь около $\log_M N = \log_2 N / \log_2 M$ итераций, если исходные данные — случайные числа. Мы увидим также, что применение схемы бора в "чистом" виде (как в алгоритме Т) требует порядка $N / \ln M$ узлов, если нужно различить N случайных ключей; следовательно, размер занимаемой памяти пропорционален $MN / \ln M$.

Эти рассуждения показывают, что идея бора хороша лишь для небольшого числа верхних уровней дерева. Можно улучшить характеристики, комбинируя две стратегии: для нескольких первых литер использовать бор, а затем перейти на другую методику. Например, Э. Г. Сассенгат, мл. [Е. Н. Sassen-guth, *SASM*, 6 (1963), 272–279] предложил использовать литерный анализ до достижения той части дерева, где остается, скажем, не более шести ключей, а затем произвести в этом коротком списке последовательный поиск. Далее мы увидим, что такая смешанная стратегия уменьшает количество узлов бора примерно в шесть раз без существенного изменения времени работы.

Приложение к английскому языку. Напрашивается много изменений основных стратегий бора и политерного поиска. Чтобы получить представление о некоторых из имеющихся возможностей, рассмотрим гипотетическое крупномасштабное приложение: предположим, что в памяти ЭВМ нужно запомнить довольно полный словарь английского языка. Для этого требуется достаточно большая внутренняя память, скажем на 50000 слов. Наша цель состоит в нахождении экономного способа представления словаря, обеспечивающего при этом достаточно быстрый поиск.

Реализация такого проекта — задача непростая. Требуется как хорошее знание содержания словаря, так и значительная программистская изобретательность. Давайте на некоторое время поставим себя в положение человека, приступающего к такому крупному проекту.

Обычный университетский словарь содержит свыше 100000 слов; наши запросы не столь велики, но рассмотрение подобного словаря позволяет наметить пути к решению. Если мы попытаемся применить боровую память, то вскоре заметим возможность важных упрощений. Предположим, что собственные имена и сокращения в расчет не принимаются. Тогда, если слово начинается на букву *b*, его вторая буква обязательно отлична от *b, c, d, f, g, j, k, m, n, p, q, s, t, v, w, x, z*. [Слово "bdellium" ("бделлий" — род ароматической смолы) можно и не включать в словарь!] На самом деле, те же 17 букв не могут стоять на втором месте в словах, начинающихся с *c, d, f, g, h, j, k, l, m, n, p, r, t, v, w, x, z*. Исключение составляют несколько слов, начинающихся с *ct, cz, dw, fj, gn, mn, nt, pf, pn, pt, tc, tm, ts, tz, zw*, и заметное количество слов, начинающихся с *kn, ps, tw*. Один из способов использовать этот факт состоит в кодировании букв (например, можно иметь таблицу из 26 слов и выполнять команду вроде "LD1 TABLE, 1") таким образом, чтобы согласные *b, c, d, . . . , z* переводились в специальное представление, большее, чем числовой код оставшихся букв *a, e, h, i, l, o, r, u, y*. Теперь многие узлы бора могут быть укорочены до 9-путевого разветвления; специальная ячейка выхода используется для редко встречающихся букв. Такой подход позволяет сэкономить память во многих частях бора, предназначенного для английского языка, а не только при обработке второй буквы.

В обычном университетском словаре слова начинаются лишь с 309 из $26^2 = 676$ возможных комбинаций двух букв, а из этих 309 пар 88 служат началом 15 или менее слов. (Типичные примеры таких пар: *aa, ah, aj, ak, ao, aq, ay, az, bd, bh, . . . , xr, ys, yi, ur, ut, uy, uw, za, zu, zw*; большинство людей может назвать максимум по одному слову из большинства этих категорий.) Если встречается одна из указанных редких категорий, можно от боровой памяти перейти к некоторой другой схеме, например к последовательному поиску.

Другим способом уменьшения требуемого для словаря объема памяти является использование приставок. Например, если ищется слово, начинающееся с *re-, pre-, anti-, trans-, dis-, un-* и т. д., можно отделить приставку и искать остаток слова. Таким путем можно удалить много слов типа *reapply* (повторно применять), *recompute* (снова вычислить), *redecorate* (заново декорировать), *redesign* (перепроектировать), *redeposit* (перекладывать) и т. д.; однако нужно оставить слова вроде *remainder* (остаток), *requirement* (требование), *retain* (сохранять), *remove* (удалять), *readily* (охотно) и т. д., так как их значения без приставки "re-" покрыты тайной. Итак, сначала следует искать слово и лишь затем пытаться удалить приставку, если первый поиск оказался неудачным.

Еще важнее использовать суффиксы. Не вызывает сомнений, что двукратное включение каждого существительного и глагола в единственном и множественном числе было бы слишком расточительным; имеются и другие типы суффиксов. Например, следующие окончания можно добавлять ко

многим глагольным основам, получая набор родственных слов:

-e	-es	-ing
-ed	-s	-ings
-edly	-able	-ingly
-er	-ible	-ion
-or	-ably	-ions
-ers	-ibly	-ional
-ors	-ability	-ionally
	-abilities	

(Многие из этих суффиксов сами составлены из суффиксов.) Если попытаться добавить эти суффиксы к основам

comput-
calculat-
search-
suffix-
translat-
interpret-
confus-

мы увидим, как много слов получается; это чрезвычайно увеличивает вместимость нашего словаря. Безусловно, при такой процедуре получается и много несуществующих слов, например "computation"; основа computat- оказывается столь же необходимой, как и comput-. Однако вреда в этом нет, поскольку такие комбинации и не будут служить аргументами поиска, а если бы некий автор придумал бы слово "computedly", мы имели бы для него готовый перевод. Заметим, что большинству людей понятно слово "confusability" (смущаемость), хотя оно встречается в немногих словарях; наш словарь будет давать подходящее толкование. При правильной обработке приставок и суффиксов наш словарь сможет из одной лишь глагольной основы "establish" вывести значение слова "antidisestablishmentarianism".

Разумеется, нужно заботиться о том, чтобы значение слова полностью определялось основой и суффиксом; исключения должны быть внесены в словарь таким образом, чтобы их можно было найти до обращения к поиску суффикса. Например, сходство между словами "socialism" (социализм) и "socialist" (социалист) способно только сбить с толку при анализе слов "organism" (организм) и "organist" (органист)!

Можно использовать целый ряд приемов для уменьшения объема памяти. Но как в единой системе компактно представить разнообразие методов? Ответ состоит в том, чтобы трактовать словарь как *программу*, написанную на специальном машинном языке для специальной *интерпретирующей системы* (ср. с п. 1.43); записи в узле бора можно трактовать как *инструкции*. Например, в табл. 1 мы имеем два вида "инструкций", а программа T использует знаковый бит как код операции; знак минус означает ответвление к другому узлу и переход к следующей литере за следующей инструкцией, в то время как знак плюс вызывает сравнение аргумента с определенным ключом.

В интерпретирующей системе для нашего словаря мы могли бы иметь операции следующих типов:

- Проверка n, α, β . "Если следующая литера аргумента кодируется числом $k \leq n$, перейти в ячейку $\alpha + k$ за следующей инструкцией; в противном случае перейти в ячейку β ".
- Сравнение n, α, β . "Сравнить оставшиеся литеры аргумента с n словами, расположенными в ячейках $\alpha, \alpha + 1, \dots, \alpha + n - 1$. Если подходящее слово имеет адрес $\alpha + k$, поиск кончается удачно и "значение" лежит в ячейке $\beta + k$, но если подходящего слова нет, поиск кончается неудачно".
- Расщепление α, β . "Слово, обработанное до данного места, возможно, является приставкой или основой. В продолжение поиска нужно направиться в ячейку α за следующей инструкцией. Если этот поиск неудачен, необходимо продолжить поиск, рассматривая оставшиеся литеры аргумента как новый аргумент. Если этот поиск удачен, скомбинировать найденное "значение" со "значением" β ".

Операция проверки реализует концепцию поиска по бору, а операция сравнения обозначает переход к последовательному поиску. Операция расщепления служит для обработки приставок и суффиксов. Можно, основываясь на более детальном изучении специфических черт английского языка, предложить некоторые другие операции. Дальнейшая экономия памяти достигается за счет исключения параметра β из каждой инструкции, так как память может быть организована так, чтобы β вычислялось с помощью α, n или адреса инструкции.

Более полная информация об организации словаря имеется в интересных статьях Сиднея Лэмба и Уильяма Якобсена, мл. [*Mechanical Translation*, 6 (1961), 76–107]; Ю. С. Шварца [*JACM*, 10 (1963), 413–439]; Галли и Ямады [*IBM Systems J.*, 6 (1967), 192–207].

Бинарный случай. Рассмотрим теперь частный случай $M = 2$, когда в каждый момент времени обрабатывается один бит аргумента поиска. Разработаны два интересных метода, особенно подходящие для данного случая.

Первый метод, который мы будем называть *цифровым поиском по дереву*, предложен Коффманом и Ивом [*SACM*, 13 (1970), 427–432, 436]. Так же как и в алгоритмах поиска по дереву из п. 6.2.2, в узлах дерева хранятся полные ключи, но для выбора левой или правой ветви используются не результат сравнения ключей, а биты аргумента. На рис. 32 изображено дерево, построенное этим методом; оно составлено из 31 наиболее употребительного английского слова, причем слова вставлялись в порядке уменьшения частот. Чтобы получить бинарные данные для этой иллюстрации, слова записывались в литерном коде машины MIX с последующим преобразованием в двоичные числа, содержащие по 5 битов в байте. Таким образом, слово WHICH превратится в "11010 01000 01001 00011 01000".

Чтобы найти WHICH на рис. 32, мы сначала сравниваем его со словом THE в корне дерева. Так как они не совпадают и первый бит слова WHICH равен 1, нужно идти вправо и произвести сравнение со словом OF. Так как совпадения не произошло и второй бит WHICH равен 1, мы идем вправо и сравниваем наш аргумент со словом WITH и т. д.

Дерево рис. 32 построено способом, похожим на способ построения дерева рис. 12 из п. 6.2.2, лишь для разветвления вместо сравнений использовались биты ключей, поэтому интересно отметить разницу между ними. Если рассмотреть заданные частоты, цифровой поиск по дереву рис. 32 требует в среднем 3.42 сравнения на удачный поиск; это несколько лучше, чем 4.04 сравнения, требующегося для рис. 12, хотя, разумеется, время самого сравнения будет различным в двух этих случаях.

Picture: Рис. 32. Дерево цифрового поиска для 31 наиболее употребительного английского слова, слова вставлены в порядке уменьшения частот.

Алгоритм D. (Цифровой поиск по дереву.) Этот алгоритм позволяет найти данный аргумент K в таблице записей, образующих описанным выше способом бинарное дерево. Если K нет в таблице, в дерево в подходящем месте вставляется новый узел, содержащий K .

Как и в алгоритме 6.2.2Т, здесь предполагается, что дерево непусто, а его узлы содержат поля KEY, LLINK и RLINK. Читатель увидит, что эти алгоритмы почти идентичны.

D1 [Начальная установка.] Установить $P \leftarrow \text{ROOT}$, $K1 \leftarrow K$.

D2 [Сравнение.] Если $K = \text{KEY}(P)$, поиск оканчивается удачно; в противном случае занести в b самый левый бит $K1$ и сдвинуть $K1$ на единицу влево (т. е. удалить этот бит и добавить справа 0). Если $b = 0$, перейти на D3; в противном случае перейти на D4.

D3 [Шаг влево.] Если $\text{LLINK}(P) \neq \Lambda$, установить $P \leftarrow \text{LLINK}(P)$ и вернуться на D2; в противном случае перейти на D5.

D4 [Шаг вправо.] Если $\text{RLINK}(P) \neq \Lambda$, установить $P \leftarrow \text{RLINK}(P)$ и вернуться на D2.

D5 [Вставка в дерево.] Выполнить $Q \leftarrow \text{AVAIL}$, $\text{KEY}(Q) \leftarrow K$, $\text{LLINK}(Q) \leftarrow \text{RLINK}(Q) \leftarrow \Lambda$. Если $b = 0$, установить $\text{LLINK}(P) \leftarrow Q$; в противном случае $\text{RLINK}(P) \leftarrow Q$. ■

Алгоритм 6.2.2Т по сути своей бинарный, однако приведенный алгоритм, как нетрудно заметить, можно легко распространить на M -арный цифровой поиск для любого $M \geq 2$ (см. упр. 13).

Picture: Рис. 33. Пример текста и дерева Патриции.

Дональд Моррисон [*JACM*, 15 (1968), 514–534] придумал очень привлекательный способ формирования N -узловых деревьев поиска, основанный на бинарном представлении ключей, без запоминания ключей в узлах. Его метод, названный "Патриция" (Практический алгоритм для выборки информации, закодированной буквами и цифрами)⁷, особенно подходит для работы с очень длинными ключами переменной длины, такими, как заголовки или фразы, хранящиеся в файле большого объема.

Основная идея Патриции состоит в том, чтобы строить бинарное дерево, избегая однопутевых разветвлений, для чего в каждый узел включается число, показывающее, сколько битов нужно пропустить, перед тем как делать следующую проверку. Существует несколько способов реализации этой идеи; возможно, простейший с точки зрения объяснения изображен на рис. 33. Имеется массив

⁷ В оригинале "Patricia"—Practical Algorithm To Retrieve Information Coded In Alphanumeric.—Прим. перев.

битов ТЕХТ (обычно довольно длинный); он может храниться во внешнем файле с прямым доступом, так как при каждом поиске обращение к нему происходит лишь раз. Каждый ключ, который должен храниться в нашей таблице, определяется местом начала в тексте; можно считать, что он идет от места начала до конца текста. (Патриция не ищет точного равенства между ключом и аргументом, а определяет, существует ли ключ, начинающийся с аргумента.)

В примере, изображенном на рис. 33, имеется семь ключей, начинающихся с каждого входящего в текст слова, а именно "THIS IS THE HOUSE THAT JACK BUILT.", "IS THE HOUSE THAT JACK BUILT.", ..., "BUILT.". Налагается важное ограничение, состоящее в том, что *один ключ не может служить началом другого*; это условие выполняется, если мы оканчиваем текст специальным кодом конца текста (в данном случае "."), который нигде больше не появляется. То же ограничение подразумевается в схеме бора алгоритма Т, где признаком конца служил "␣".

Дерево, используемое Патрицией для поиска, должно целиком помещаться в оперативной памяти, или его нужно организовать по страничной схеме, описанной в п. 6.2.4. Оно состоит из головы и $N - 1$ узлов, причем узлы содержат несколько полей:

KEY—указатель на текст. Это поле должно иметь длину не менее $\log_2 C$ битов, где C —число литер в тексте. Слова, изображенные на рис. 33 внутри узлов, на самом деле должны быть представлены указателями на текст; например, вместо "(JACK)" узел содержит число 24 (начальную точку ключа "JACK BUILT.").

LLINK и RLINK—ссылки внутри дерева. Они должны иметь длину не менее $\log_2 N$ битов.

LTAG и RTAG—однобитовые поля, по которым мы можем судить, являются ли LLINK и RLINK соответственно ссылками на сыновей или на предков. Пунктирные линии на рис. 33 соответствуют ссылкам, для которых TAG = 1.

SKIP—число битов, которые нужно пропустить при поиске, как объяснялось выше. Это поле должно быть достаточно большим для того, чтобы вместить наибольшее число k , для которого в двух различных ключах найдутся совпадающие подцепочки из k битов; обычно для практических целей достаточно считать, что k не слишком велико, и при превышении размеров поля SKIP выдавать сообщение об ошибке. На рис. 33 поля SKIP изображены внутри каждого узла числами.

Голова содержит лишь поля KEY, LLINK и LTAG.

Поиск в дереве Патриции выполняется следующим образом: предположим, мы ищем слово THAT (бинарное представление 10111 01000 00001 10111). По содержимому поля SKIP в корневом узле α мы заключаем, что сначала нужно проверить первый бит аргумента. Он равен 1, поэтому надо двигаться вправо. Поле SKIP следующего узла побуждает нас проверить $1 + 11 = 12$ -й бит аргумента. Он равен 0, и мы идем влево. Изучив поле SKIP следующего узла, видим, что нужно проверить $12 + 1 = 13$ -й бит, который равен 0. Так как LTAG = 1, мы идем на узел γ , который отсылает нас к массиву ТЕХТ. По тому же пути пойдет поиск для любого аргумента, имеющего бинарное представление $1xxxx xxxxx x00 \dots$, поэтому нужно сравнить аргумент с найденным ключом.

Предположим, с другой стороны, что мы ищем некоторый ключ, начинающийся с TH, или все такие ключи. Поиск начинается, как и раньше, но в конце концов мы обратимся к (несуществующему) 12-му биту десятибитового аргумента. Теперь нужно сравнить аргумент с фрагментом массива ТЕХТ, который специфицируется в текущем узле (в данном случае γ); если совпадения не произошло, то не существует ключа, началом которого служит аргумент поиска; если совпадение имело место, аргумент является началом любого ключа, на который указывают пунктирные линии, выходящие из узла γ и его потомков.

Более точно этот процесс описывается следующим образом.

Алгоритм Р. (Патриция.) Алгоритм, используя массив ТЕХТ и дерево с описанными выше полями KEY, LLINK, RLINK, LTAG, RTAG и SKIP в узлах, позволяет определить, существует ли в массиве ТЕХТ ключ, начинающийся с аргумента K . (Если имеется $r \geq 1$ таких ключей, можно за $O(r)$ шагов определить расположение каждого из них; см. упр. 14.) Предполагается, что есть по крайней мере один ключ.

P1 [Начальная установка.] Установить $P \leftarrow \text{HEAD}$ и $j \leftarrow 0$ (Переменная P является указателем, который будет двигаться вниз по дереву, а j —это счетчик, определяющий позиции битов аргумента.) Занести в n количество битов аргумента K .

P2 [Шаг влево.] Установить $Q \leftarrow P$ и $P \leftarrow \text{LLINK}(Q)$. Если $\text{LTAG}(Q) = 1$, перейти на **P6**.

P3 [Пропуск битов.] (В этот момент мы знаем, что, если первые j битов аргумента соответствуют некоторому ключу, они соответствуют ключу, начинающемуся в $\text{KEY}(P)$.) Установить $j \leftarrow j + \text{SKIP}(P)$. Если $j > n$, перейти на **P6**.

- Р4 [Проверка бита.] В этот момент мы знаем, что, если первые $(j-1)$ битов соответствуют некоторому ключу, они соответствуют ключу, начинающемуся в $KEY(P)$. Если j -й бит K равен 0, перейти на Р2; в противном случае перейти на Р5.
- Р5 [Шаг вправо.] Установить $Q \leftarrow P$ и $P \leftarrow RLINK(Q)$. Если $R\text{TAG}(Q) = 0$, перейти на Р3.
- Р6 [Сравнение.] (В этот момент мы знаем, что, если K соответствует некоторому ключу, он соответствует ключу, начинающемуся в $KEY(P)$.) Сравнить K с ключом, начинающимся в позиции $KEY(P)$ массива TEXT . Если произошло совпадение n первых битов, алгоритм заканчивается удачно; в случае несовпадения он заканчивается неудачно. ■

В упр. 15 прежде всего показано, как можно осуществить построение дерева Патриции. Имеется также возможность добавлять к тексту и вставлять новые ключи при условии, что новый текстовый материал оканчивается определенным разделителем (например, символом конца текста, за которым следует порядковый номер).

Характер Патриции слегка причудлив, и все ее достоинства заметны лишь внимательному наблюдателю.

Анализ алгоритмов. Завершим этот раздел математическим изучением боров, деревьев цифрового поиска и Патриции. Важнейшие выводы из этого анализа приведены в самом конце.

Picture: Рис. 34. Пример случайного бинарного бора.

Рассмотрим сначала бинарные бору, т. е. бору с $M = 2$. На рис. 34 изображен бинарный бор, образующийся, если 16 ключей из примеров сортировки гл. 5 рассматривать как десятибитовые двоичные числа. (Ключи приведены в восьмеричной записи, так что, например, 1144 представляет десятибитовое число $(1001100100)_2$.) Как и в алгоритме Т, мы используем бор для хранения информации о левых битах ключей до тех пор, пока впервые достигается точка, где ключ определяется однозначно; там он записывается полностью.

Если сравнить рис. 34 с табл. 5.2.2-3, обнаружится удивительная взаимосвязь между боровой памятью и обменной поразрядной сортировкой. (Возможно, эта взаимосвязь является очевидной.) 22 узла рис. 34 в точности соответствуют 22 стадиям расчленения в табл. 5.2.2-3 (узлы следует нумеровать в прямом порядке). Число проверяемых битов в стадии расчленения равно числу ключей в соответствующем узле-и его подборах; таким образом, можно сформулировать следующий результат.

Теорема Т. Если N различных двоичных чисел помещены в бинарный бор, как описано выше, то (i) число узлов бора равно числу стадий расчленения, нужных при обменной поразрядной сортировке этих чисел; (ii) среднее число проверок битов, требуемых для выборки ключа с помощью алгоритма Т, равно $1/N$, умноженному на число проверок битов при обменной поразрядной сортировке. ■

Благодаря теореме мы можем использовать весь математический аппарат, развитый в п. 5.2.2 для обменной поразрядной сортировки. Предположим, например, что ключами являются случайные, равномерно распределенные между 0 и 1 вещественные числа, представленные с бесконечной точностью. Тогда количество проверок битов, необходимых для выборки информации, будет равно $\log_2 N + \gamma/(\ln 2) + 1/2 + f(N) + O(N^{-1})$, а число узлов бора составит $N/(\ln 2) + Ng(N) + O(1)$. Здесь $f(N)$ и $g(N)$ — сложные функции, которыми можно пренебречь, так как их значения всегда меньше 10^{-6} (см. упр. 5.2.2-38, 48).

Разумеется, перед нами стоит более трудная задача: обобщить полученные результаты на случай M -арных боров. Мы опишем лишь отправную точку исследований, оставляя поучительные детали в качестве упражнений.

Пусть A_N — среднее число узлов в случайном M -арном бору поиска, содержащем N ключей. Тогда $A_0 = A_1 = 0$, а для $N \geq 2$ мы имеем

$$A_N = 1 + \sum_{k_1 + \dots + k_M = N} \left(\frac{N!}{k_1! \dots k_M!} M^{-N} \right) (A_{k_1} + \dots + A_{k_M}), \quad (3)$$

так как $N!M^{-N}/k_1! \dots k_M!$ есть вероятность того, что k_1 ключей содержится в первом подборе, \dots , k_M — в M -м. Воспользовавшись соображениями симметрии и проведя суммирование по k_2, \dots, k_M , перепишем это соотношение так:

$$\begin{aligned} A_N &= 1 + M^{1-N} \sum_{k_1 + \dots + k_M = N} \left(\frac{N!}{k_1! \dots k_M!} \right) A_{k_1} = \\ &= 1 + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} A_k, \quad N \geq 2. \end{aligned} \quad (4)$$

Аналогично, если через C_N обозначить среднее суммарное количество проверок битов, нужных для поиска в бору всех N ключей, то $C_0 = C_1 = 0$, а

$$C_N = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} C_k, \quad N \geq 2. \quad (5)$$

В упр. 17 показано, как работать с рекуррентными соотношениями такого вида, а в упр. 18–25 разрабатывается соответствующая теория случайных боров. [С другой точки зрения к анализу A_N впервые подошли Л. Р. Джонсон и М. Мак-Эндрю [IBM J. Res. and Devel., 8 (1964), 189–193] в связи с эквивалентным аппаратно-ориентированным алгоритмом сортировки.]

Переходя теперь к изучению деревьев цифрового поиска, мы обнаруживаем, что, с одной стороны, формулы похожи, а с другой—достаточно различны, и сразу не ясно, как исследовать асимптотическое поведение. Например, если \bar{C}_N обозначает среднее суммарное количество проверок битов, производимых при поиске всех N ключей в бинарном дереве цифрового поиска, нетрудно вывести, как и раньше, что $\bar{C}_0 = \bar{C}_1 = 0$ и

$$\bar{C}_{N+1} = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} \bar{C}_k, \quad N \geq 0. \quad (6)$$

Это почти идентично выражению (5), но появления $N + 1$ вместо N в левой, части уравнения достаточно для того, чтобы изменить общий характер рекуррентности, так что методы, используемые для изучения (5), в данном случае непригодны.

Рассмотрим сначала бинарный случай. На рис. 35 изображено дерево цифрового поиска, соответствующее 16 ключам рис. 34, вставленным в порядке, использованном в примерах из гл. 5. Среднее число проверок битов при случайном удачном поиске найти легко—оно равно длине внутреннего пути дерева, деленной на N , так как нужно l проверок, чтобы найти узел, расположенный на уровне l . Заметим, однако, что среднее число проверок битов при случайном неудачном поиске не так просто связано с длиной внешнего пути дерева, ибо неудачный поиск с большей вероятностью оканчивается вблизи корня; например, вероятность достижения левой ветви узла 0075 (рис. 35) равна $1/8$ (рассматриваются бесконечно точные ключи), а в левую ветвь узла 0232 мы попадаем с вероятностью, равной лишь $1/32$. (По этой причине при равномерном распределении ключей дерева цифрового поиска, как правило, лучше сбалансированы, чем бинарные деревья поиска, использовавшиеся в алгоритме 6.2.2Т.)

Для описания соответствующих характеристик деревьев цифрового поиска можно использовать производящую функцию. Пусть на уровне l расположено a_l узлов; рассмотрим производящую функцию $a(z) = \sum a_l z^l$. Например, рис. 35 соответствует

Picture: Рис. 35 Случайное дерево цифрового поиска, построенное с помощью алгоритма D.

функция $a(z) = 1 + 2z + 4z^2 + 5z^3 + 4z^4$. Если b_l узлов уровня l являются внешними и $b(z) = \sum b_l z^l$, то в силу упр. 6.2.1-25 имеем

$$b(z) = 1 + (2z - 1)a(z). \quad (7)$$

Например, $1 + (2z - 1)(1 + 2z + 4z^2 + 5z^3 + 4z^4) = 3z^3 + 6z^4 + 8z^5$. Среднее число проверок битов при случайном удачном поиске равно $a'(1)/a(1)$, так как $a'(1)$ есть длина внутреннего пути дерева, а $a(1)$ — количество внутренних узлов. Среднее число проверок битов для случайного неудачного поиска равно $\sum l b_l 2^{-l} = \frac{1}{2} b'(\frac{1}{2}) = a(\frac{1}{2})$, так как мы достигаем данного внешнего узла уровня l с вероятностью 2^{-l} . При неудачном поиске число сравнений совпадает с числом проверок битов, а при "удачном"—на единицу больше этого числа. Для дерева рис. 35 удачный поиск в среднем требует $2\frac{9}{16}$ проверок битов и $3\frac{9}{16}$ сравнений; в процессе неудачного поиска производится $3\frac{7}{8}$ сравнений и проверок (в среднем).

Введем теперь "усредненную" по всем деревьям с N узлами функцию $a(z)$ и обозначим ее через $g_N(z)$. Иными словами, $g_N(z) = \sum p_T a_T(z)$, где сумма берется по всем бинарным деревьям цифрового поиска T с N внутренними узлами; $a_T(z)$ есть производящая функция для внутренних узлов T , а p_T есть вероятность того, что при вставке N случайных чисел с помощью алгоритма D получается дерево T . Таким образом, для удачного поиска среднее число проверок битов равно $g'_N(1)/N$, а для неудачного— $g_N(\frac{1}{2})$.

Имитируя процесс построения дерева, $g_N(z)$ можно вычислить следующим образом. Если $a(z)$ есть производящая функция для дерева с N узлами, можно образовать $N + 1$ деревьев, делая вставку в позицию любого внешнего узла. Эта вставка производится в данный внешний узел уровня l с вероятностью 2^{-l} ; следовательно, сумма производящих функций для $N + 1$ новых деревьев, умноженных

на вероятность их появления, равна $a(z) + b\left(\frac{1}{2}z\right) = a(z) + 1 + (z-1)a\left(\frac{1}{2}z\right)$. Усредняя по всем деревьям с N узлами, получаем

$$g_{N+1}(z) = g_N(z) + 1 + (z-1)g_N\left(\frac{1}{2}z\right); \quad g_0(z) = 0. \quad (8)$$

С соответствующей производящей функцией для внешних узлов $h_N(z) = 1 + (2z-1)g_N(z)$ работать несколько легче, так как (8) эквивалентно формуле

$$h_{N+1}(z) = h_N(z) + (2z-1)h_N\left(\frac{1}{2}z\right); \quad h_0(z) = 1. \quad (9)$$

Множественно применяя это правило, находим, что

$$\begin{aligned} h_{N+1}(z) &= \\ &= h_{N-1}(z) + 2(2z-1)h_{N-1}\left(\frac{1}{2}\right) + (2z-1)(z-1)h_{N-1}\left(\frac{1}{4}z\right) = \\ &= h_{N-2}(z) + 3(2z-1)h_{N-2}\left(\frac{1}{2}\right) + 3(2z-1)(z-1)h_{N-2}\left(\frac{1}{4}z\right) + \\ &\quad + (2z-1)(z-1)\left(\frac{1}{2}-1\right)h_{N-2}\left(\frac{1}{8}z\right) \end{aligned}$$

и т. д.; окончательно имеем

$$h_N(z) = \sum_k \binom{N}{k} \prod_{0 \leq j < k} (2^{1-j}z - 1); \quad (10)$$

$$g_N(z) = \sum_{k \geq 0} \binom{N}{k+1} \prod_{0 \leq j < k} (2^{-j}z - 1). \quad (11)$$

Например, $g_4(z) = 4 + 6(z-1) + 4(z-1)\left(\frac{1}{2}z-1\right) + (z-1) \times \left(\frac{1}{2}z-1\right)\left(\frac{1}{4}z-1\right)$. Эти формулы позволяют выразить искомые величины в виде сумм произведений:

$$\bar{C}_N = g'_N(1) = \sum_{k \geq 0} \binom{N}{k+2} \prod_{1 \leq j \leq k} (2^{-j} - 1); \quad (12)$$

$$g_N\left(\frac{1}{2}\right) = \sum_{k \geq 0} \binom{N}{k+1} \prod_{1 \leq j \leq k} (2^{-j} - 1) = \bar{C}_{N+1} - \bar{C}_N. \quad (13)$$

Совсем не очевидно, что эти формулы для \bar{C}_N удовлетворяют (6)!

К сожалению, найденные выражения непригодны для вычислений или получения асимптотических оценок, так как $2^{-j} - 1 < 0$; получится большое количество членов, многие из которых сократятся. Более полезные формулы для \bar{C}_N можно вывести, применяя тождества упр. 5.1.1-16. Имеем

$$\begin{aligned} \bar{C}_N &= \left(\prod_{j \geq 1} (1 - 2^{-j}) \right) \sum_{k \geq 0} \binom{N}{k+2} (-1)^k \prod_{l \geq 0} (1 - 2^{-l-k-1})^{-1} = \\ &= \left(\prod_{j \geq 1} (1 - 2^{-j}) \right) \sum_{k \geq 0} \binom{N}{k+2} (-1)^k \sum_{m \geq 0} (2^{-k-1})^m \times \prod_{1 \leq r \leq m} (1 - 2^{-r})^{-1} = \\ &= \sum_{m \geq 0} 2^m \left(\sum_k \binom{N}{k} (-2^{-m})^k - 1 + 2^{-m}N \right) \prod_{j \geq 0} (1 - 2^{-j-m-1}) = \\ &= \sum_{m \geq 0} 2^m ((1 - 2^{-m})^N - 1 + 2^{-m}N) \sum_{n \geq 0} (-2^{-m-1})^n 2^{-n(n-1)/2} \times \prod_{1 \leq r \leq n} (1 - 2^{-r})^{-1}. \end{aligned} \quad (14)$$

Сразу не ясно, чем это лучше выражения (12), но выведенная формула имеет то важное преимущество, что она быстро сходится для любого фиксированного n . В точности аналогичная ситуация

возникает и в случае бора [формула (5.2.2-38)]; в самом деле, если рассмотреть в (14) лишь члены с $n = 0$, получается ровно $N - 1$ плюс число проверок в бинарном бору. Дальнейший вывод асимптотической оценки можно провести уже известным нам способом; см. упр. 27. Приведенный вывод в большой степени основан на подходе, предложенном Э. Г. Конхеймом и Д. Дж. Ньюманом [*Discrete Mathematics*, 4 (1973), 56–63].

И наконец, взглянем на Патрицию с точки зрения математика. Ее бинарное дерево похоже на соответствующий бинарный бор с теми же ключами, но сжатыми вместе (ибо поля SKIP устраняют однопутевые разветвления), так что имеется $N - 1$ внутренних и N внешних узлов. На рис. 36 изображено дерево Патриции, соответствующее шестнадцати ключам в бору рис. 34. Числа, заключенные во внутренних узлах, обозначают содержимое полей SKIP; ключи указаны вместе с внешними узлами, хотя последние не присутствуют явно (в действительности вместо внешних узлов существуют помеченные ссылки на внутренние узлы, которые в свою очередь ссылаются на массив TEXT). Однако для целей анализа можно считать, что внешние узлы существуют в том виде, как они изображены на рис. 36.

Так как удачный поиск с использованием Патриции оканчивается во внешних узлах, среднее число проверок битов, производимых при случайном удачном поиске, равно длине

Picture: Рис. 36. Патриция строит это дерево вместо рис. 34.

внешнего пути, деленной на N . Если, как и выше, ввести производящую функцию $b(z)$ для внешних узлов, это число будет равно $b'(1)/b(1)$. Неудачный поиск также кончается во внешнем узле, но с вероятностью 2^{-l} (l —номер уровня, на котором расположен узел), так что среднее число проверок битов составляет $\frac{1}{2}b'(\frac{1}{2})$. Например, для рис. 36 $b(z) = 3z^3 + 8z^4 - 3z^5 + 2z^6$; в среднем удачный поиск требует $4\frac{1}{4}$ проверок битов, а неудачный— $3\frac{25}{32}$.

Производящую функцию $b(z)$, усредненную по всем деревьям Патриции, построенным с N внешними узлами с использованием равномерно распределенных ключей, обозначим через $h_N(z)$. Рекуррентное соотношение

$$h_n(z) = 2^{1-N} \sum_k \binom{n}{k} h_k(z)(z + \delta_{kn}(1-z)), \quad h_0(z) = 0, \quad h_1(z) = 1, \quad (15)$$

по-видимому, не имеет простого решения. Но, к счастью, существует простое рекуррентное соотношение для средней длины внешнего пути $h'_n(1)$, поскольку

$$\begin{aligned} h'_n(1) &= 2^{1-n} \sum_k \binom{n}{k} h'_k(1) + 2^{1-n} \sum_k \binom{n}{k} (1 - \delta_{kn}) = \\ &= n - 2^{n-1}n + 2^{1-n} \sum_k \binom{n}{k} h'_k(1). \end{aligned} \quad (16)$$

Так как оно имеет вид (6), для нахождения $h'_n(1)$ можно использовать уже развитые методы; оказывается, что $h'_n(1)$ ровно на n меньше соответствующего числа проверок битов в случайном бинарном бору. Таким образом, поле SKIP позволяет сэкономить примерно одну проверку бита для удачного поиска при случайных данных. (См. упр. 31.) На самом же деле избыточность типичных данных приведет к большей экономии.

Если мы попытаемся найти среднее число проверок битов для случайного неудачного поиска, производимого Патрицией, то получим соотношение

$$a_n = 1 + \frac{1}{2^n - 2} \sum_{k < n} \binom{n}{k} a_k, \quad n \geq 2; \quad a_0 = a_1 = 0. \quad (17)$$

(Здесь $a_n = \frac{1}{2}h'_n(\frac{1}{2})$.) Оно не похоже ни на одно из изученных соотношений, и свести его к ним не просто. На самом деле оказывается, что решение содержит числа Бернулли:

$$\frac{na_{n-1}}{2} - n + 2 = \sum_{2 \leq k < n} \binom{n}{k} \frac{B_k}{2^{k-1} - 1}. \quad (18)$$

Эта формула является, вероятно, наиболее сложной асимптотической оценкой из всех, которые нам нужно было получить; решение в упр. 34 дает поучительный обзор многих ранее изученных вещей с некоторыми нюансами.

Выводы из анализа. Следующие результаты заслуживают наибольшего внимания:

- (a) Число узлов, необходимых для хранения N случайных ключей в M -арном бору, разветвления в котором происходят до тех пор, пока не останется подфайл из $\leq s$ ключей, примерно равно $N/(s \ln M)$. (Это приближение справедливо для больших N и малых s и M .) Так как узел бора содержит M полей ссылок, то, если выбрать $s = M$, всего потребуется лишь около $N/(\ln M)$ полей ссылок.
- (b) Для всех рассмотренных методов количество цифр или литер, проверяемых в процессе случайного поиска, примерно равно $\log_M N$. При $M = 2$ различными методами можно получить следующие более точные приближения для числа проверок битов:

	Удачный	Неудачный
Поиск по бору	$\log_2 N + 1.33275$	$\log_2 N - 0.10995$
Цифровой поиск по дереву	$\log_2 N - 1.71665$	$\log_2 N - 0.27395$
Патриция	$\log_2 N + 0.33275$	$\log_2 N - 0.31875$

[Все эти приближенные значения можно выразить через фундаментальные математические постоянные; например, число 0.31875 получилось из $(\ln \pi - \gamma)/(\ln 2) - 1/2$.]

- (c) "Случайность" данных здесь означает то, что M -ичные цифры являются равномерно распределенными, как если бы ключами были действительные числа между 0 и 1, записанные в M -ичной системе счисления. Методы цифрового поиска не чувствительны к порядку, в котором ключи помещены в файл (исключением является алгоритм D, лишь в малой степени чувствительный к порядку); но они очень чувствительны к распределению цифр. Например, если нулевые биты встречаются гораздо чаще единичных, деревья будут гораздо более асимметричными по сравнению с деревьями, полученными для случайных данных. (В упр. 5.2.2-53 разобран пример того, что происходит при таком воздействии на данные.)

Упражнения

- [00] Если дерево имеет листья, то что имеет бор?
- [20] Постройте алгоритм вставки нового ключа в M -арный бор, придерживаясь соглашений алгоритма T.
- [21] Постройте алгоритм удаления ключа из M -арного бора, придерживаясь соглашений алгоритма T.
- >4. [21] Большинство из 360 элементов табл. 1—пробелы (пустые ссылки). Однако можно сжать таблицу до 49 элементов, допуская перекрытие непустые и пустых элементов:

Picture: Рис. на стр. 594

[Узлы (1), (2), ..., (12) табл. 1 начинаются соответственно с позиций 20, 19, 3, 14, 1, 17, 1, 7, 3, 20, 18, 4 сжатой таблицы.]

Покажите, что, если заменить такой сжатой таблицей табл. 1, программа T будет работать и в этом случае, правда, не столь быстро.

- >5. [M26] (Й. Н. Пэтт.) В деревьях рис. 31 буквы упорядочены по алфавиту внутри каждого семейства. Это не является необходимым, и если мы, прежде чем строить бинарное дерево типа (2), изменим порядок узлов внутри каждого семейства, поиск может ускориться. Какое переупорядочение рис. 31 будет оптимальным с этой точки зрения? (Используйте частоты, данные на рис. 32, и найдите лес, который, будучи представлен в виде бинарного дерева, минимизирует время удачного поиска.)
- [15] Какое дерево цифрового поиска получится, если пятнадцать четырехбитовых бинарных ключей 0001, 0010, 0011, ..., 1111 вставить в возрастающем порядке с помощью алгоритма D? (Начните с ключа 0001 в корневом узле и затем произведите четырнадцать вставок.)
- >7. [M26] Если пятнадцать ключей из упр. 6 вставляются в другом порядке, может получиться другое дерево. Какова наихудшая среди всех $15!$ возможных перестановок этих ключей в том смысле, что она порождает дерево с наибольшей длиной внутреннего пути?
- [20] Рассмотрим следующие изменения алгоритма D, ведущие к исключению переменной $K1$: заменить " $K1$ " на " K " в шаге D2 и устранить операцию " $K1 \leftarrow K$ " из шага D1. Годится ли полученный алгоритм для поиска и вставки?
- [21] Напишите MIX-программу для алгоритма D и сравните ее с программой 6.2.2T. Вы можете использовать бинарные операции вроде SLB (бинарный сдвиг влево AX), JAE (переход, если A четно) и т. д., если сочтете нужным. Вы можете также использовать идею упр. 8.

10. [23] Имеется файл, все ключи которого суть n -битовые двоичные числа, и аргумент поиска $K = b_1 b_2 \dots b_n$. Предположим, что мы хотим найти максимальное число k , такое, что в файле существует ключ, начинающийся с $b_1 b_2 \dots b_k$. Как сделать это эффективно, если файл представлен в виде: (а) бинарного дерева поиска (ср. с алгоритмом 6.2.2Т); (б) бинарного бора (ср. с алгоритмом Т); (с) бинарного дерева цифрового поиска (ср. с алгоритмом D)?
11. [21] Можно ли использовать алгоритм 6.2.2D без изменений для удаления узла из дерева цифрового поиска?
12. [25] Будет ли случайным дерево, полученное в результате удаления случайного элемента из случайного дерева цифрового поиска, построенного с помощью алгоритма D? (Ср. с упр. 11 и с теоремой 6.2.2Н.)
13. [20] (M -арный цифровой поиск). Объясните, как из алгоритмов Т и D составить сообщенный алгоритм, сводящийся к алгоритму D при $M = 2$. Как следует изменить табл. 1, если использовать ваш алгоритм при $M = 30$?
- >14. [25] Постройте эффективный алгоритм, с помощью которого сразу после удачного завершения алгоритма Р можно было бы найти все места в массиве ТЕХТ, где встречается K .
15. [28] Придумайте, эффективный алгоритм, который можно использовать для построения дерева Патриции или для вставки новой ссылки на ТЕХТ в уже существующее дерево. Ваш алгоритм вставки должен обращаться к массиву ТЕХТ не более двух раз.
16. [22] Почему для Патриции хотелось бы иметь такое ограничение, что ни один ключ не должен служить началом другого?
17. [M25] Найдите способ выразить решение рекуррентного соотношения

$$x_0 = x_1 = 0; \quad x_n = a_n + m^{1-n} \sum_k \binom{n}{k} (m-1)^{n-k} x_k, \quad n \geq 2,$$

с помощью биномиальных преобразований, обобщая метод упр. 5.2.2-36.

18. [M21] Используя результат упр. 17, выразите решения уравнений (4) и (5) через функции U_n и V_n , аналогичные введенным в упр. 5.2.2-38.
19. [BM23] Найдите асимптотическое значение функции

$$K(n, s, m) = \sum_{k \geq 2} \binom{n}{k} \binom{k}{s} \frac{(-1)^k}{m^{k-1} - 1}$$

с точностью до $O(1)$ при $n \rightarrow \infty$ для фиксированных значений $s \geq 0$ и $m > 1$. [Случай $s = 0$ рассмотрен в упр. 5.2.2-50, а случай $s = l, m = 2$ — в упр. 5.2.2-48.]

- >20. [M30] Рассмотрим M -арную бортовую память, в которой, достигнув подфайла, содержащего не более s ключей, мы используем последовательный поиск. (Случаю $s = 1$ соответствует алгоритм Т.) Примените результаты предыдущих упражнений, чтобы проанализировать: (а) среднее число узлов бора; (б) среднее число проверяемых цифр или литер при удачном поиске; (с) среднее число сравнений, производимых при удачном поиске. Сформулируйте ответы в виде асимптотических формул ($N \rightarrow \infty$) для фиксированных M и s , ответ для (а) должен быть дан с точностью до $O(1)$, ответы для (б) и (с) — с точностью до $O(N^{-1})$. [Если $M = 2$, этот анализ применим также к модифицированной обменной поразрядной сортировке, когда подфайлы размера $< s$ сортируются посредством вставок].
21. [M25] Сколько узлов в случайном M -арном бору, содержащем N ключей, имеют пустую ссылку в качестве нулевой компоненты? (Например, 9 из 12 узлов табл. 1 в позиции "□" имеют пустую ссылку. "Случайный" в этом упражнении, как и обычно, означает, что литеры ключей равномерно распределены между 0 и $M - 1$.)
22. [M25] Сколько узлов расположено на 1-м уровне случайного M -арного бора, содержащего N ключей, при $l = 0, 1, 2, \dots$?
23. [M26] Сколько проверок цифр производится в среднем при неудачном поиске в M -арном бору, содержащем N случайных ключей?
24. [M30] Рассмотрим M -арный бор, представленный в виде леса (ср. с рис. 31). Найдите точные и асимптотические выражения для: (а) среднего числа узлов в лесу; (б) среднего количества выполнений присваивания $P \leftarrow \text{RLINK}(P)$ в процессе случайного удачного поиска.
- >25. [M24] Математический вывод асимптотических значений, сделанный в данном пункте, весьма труден; использовалась даже теория комплексного переменного. Дело в том, что мы не хотели ограничиваться главным членом асимптотики, а второй член действительно сложен. Цель этого упражнения состоит в том, чтобы элементарными методами получить некоторые результаты в

ослабленной форме, (а) Докажите по индукции, что решение уравнения (4) при $N \geq 1$ удовлетворяет неравенству $A_N \leq M(N-1)/(M-1)$. (б) Пусть $D_N = C_N - NH_{N-1}/(\ln M)$, где C_N находится из (5). Докажите, что $D_N = O(N)$, следовательно, $C_N = N \log_M N + O(N)$. [Указание: используйте (а) и теорему 1.2.7А.]

26. [23] Определите вручную значение бесконечного произведения

$$\left(1 - \frac{1}{2}\right) \left(1 - \frac{1}{4}\right) \left(1 - \frac{1}{8}\right) \left(1 - \frac{1}{16}\right) \dots$$

с точностью до 10^{-5} . [Указание: ср. с упр. 5.1.1-16.]

27. [BM31] С помощью соотношения (14) найдите асимптотическое значение \bar{C}_N с точностью до $O(1)$.

28. [BM26] Найдите асимптотическое поведение среднего числа проверок цифр при поиске в случайном дереве цифрового поиска, $M \geq 2$. Рассмотрите как удачный, так и неудачный поиск. Дайте ответ с точностью до $O(N^{-1})$.

29. [M46] Найдите асимптотику среднего числа узлов в M -арном дереве цифрового поиска, все M сылок которых пусты. (Исключив такие узлы, мы могли бы сэкономить память; ср. с упр. 13.)

30. [M24] Покажите, что производящую функцию Патриции $h_n(z)$, определенную в (15), можно выразить в таком достаточно "ужасном" виде:

$$n \sum_{m \geq 1} z^m \left(\sum_{\substack{a_1 + \dots + a_m = n-1 \\ a_1, \dots, a_m \geq 1}} \binom{n-1}{a_1, \dots, a_m} \times \frac{1}{(2^{a_1} - 1)(2^{a_1+a_2} - 1) \dots (2^{a_1+\dots+a_m} - 1)} \right).$$

[Таким образом, если бы существовала простая формула для $h_n(z)$, мы могли бы упростить это довольно громоздкое выражение.]

31. [M21] Найдите $h'_n(1)$ из соотношения (16).

32. [M21] Чему равно среднее значение суммы всех полей SKIP в случайном дереве Патриции с $N-1$ внутренними узлами?

33. [M30] Докажите, что [18] удовлетворяет рекуррентному соотношению (17). [Указание: рассмотрите производящую функцию $A(z) = \sum_{n \geq 0} a_n z^n / n!$.]

34. [BM40] Целью этого упражнения является нахождение асимптотического поведения функции (18).

(а) Докажите, что при $n \geq 2$

$$\frac{1}{n} \sum_{2 \leq k < n} \binom{n}{k} \frac{B_k}{2^{k-1} - 1} = \sum_{j \geq 1} \left(\frac{1^{n-1} + 2^{n-1} + \dots + (2^j - 1)^{n-1}}{2^{j(n-1)}} - \frac{2^j}{n} + \frac{1}{2} \right).$$

(б) Покажите, что слагаемые в правой части (а) можно приблизить функциями $1/(e^x - 1) - 1/x + 1/2$, где $x = n/2^j$; получающаяся при этом сумма отличается от первоначальной на $O(n^{-1})$.

(с) Покажите, что для действительного $x > 0$

$$\frac{1}{e^x - 1} - \frac{1}{x} + \frac{1}{2} = \frac{1}{2\pi i} \int_{-\frac{1}{2} - i\infty}^{-\frac{1}{2} + i\infty} \zeta(z) \Gamma(z) x^{-z} dx.$$

(д) Следовательно, рассматриваемая сумма равна

$$\frac{1}{2\pi i} \int_{-\frac{1}{2} - i\infty}^{-\frac{1}{2} + i\infty} \frac{\zeta(z) \Gamma(z) n^{-z}}{2^{-z} - 1} dx + O(n^{-1});$$

оцените этот интеграл.

>35. [M20] Какова вероятность того, что дерево Патриции, содержащее пять ключей, имеет вид

Picture: Рис. стр. 597

причем поля SKIP содержат a, b, c, d , как показано на рисунке? (Предполагается, что ключи имеют независимые случайные биты; ответ нужно представить в виде функции от a, b, c и d .)

36. [M25] Имеется пять бинарных деревьев с тремя внутренними узлами. Если мы рассмотрим, как часто каждое из них служит деревом, поиска в различных алгоритмах (данные предполагаются случайными), то получим следующие различные вероятности:

Picture: Рис. стр. 598 — в таблицу по частям:

Поиск по дереву (алгоритм 6.2.2.Т) $\frac{1}{7} \frac{1}{6} \frac{1}{3} \frac{1}{6} \frac{1}{6}$
 Цифровой поиск по дереву (алгоритм D) $\frac{1}{7} \frac{1}{6} \frac{1}{3} \frac{1}{6} \frac{1}{6}$
 Патриция (алгоритм Р) $\frac{1}{7} \frac{1}{6} \frac{1}{3} \frac{1}{6} \frac{1}{6}$

(Заметим, что дерево цифрового поиска имеет наибольшую тенденцию к сбалансированности.) В упр. 6.2.2-5 мы нашли формулу для вероятности в случае поиска по дереву: $\prod(1/s(x))$, где произведение берется по всем внутренним узлам x , а $s(x)$ —число внутренних узлов в поддереве, корнем которого служит x . Найдите аналогичные формулы для вероятности в случае: (а) алгоритма D; (б) алгоритма Р.

- >37. [M22] Рассмотрим бинарное дерево, на l -м уровне которого расположено b_l внешних узлов. В тексте указывалось, что время неудачного поиска в деревьях цифрового поиска не связано непосредственно с длиной внешнего пути $\sum lb_l$, а, по существу, пропорционально *модифицированной длине внешнего пути* $\sum lb_l 2^{-l}$. Докажите или опровергните следующее утверждение: среди всех деревьев с N внешними узлами наименьшую модифицированную длину внешнего пути имеет то дерево, все внешние узлы которого расположены не более чем на двух смежных уровнях (ср. с упр. 5.3.1-20).
38. [M40] Разработайте алгоритм для нахождения n -узлового дерева, имеющего минимальное значение величины $\alpha \times$ (длина внутреннего пути) $+$ $\beta \times$ (модифицированная длина внешнего пути), α и β —данные числа (ср. с упр. 37).
39. [M43] Придумайте алгоритм нахождения оптимальных деревьев цифрового поиска, аналогичных оптимальным бинарным деревьям поиска, рассмотренным в п. 6.2.2.
40. [25] Пусть $a_0 a_1 a_2 \dots$ —периодическая бинарная последовательность; $a_{N+k} = a_k$ при всех $k \geq 0$. Покажите, что любую фиксированную цепочку этого типа можно представить в $O(N)$ ячейках памяти таким образом, что следующая операция требует лишь $O(n)$ шагов: имея бинарную цепочку $b_0 b_1 \dots b_{n-1}$, нужно определить, сколько раз она встречается в периоде (т. е. найти количество значений p , $0 \leq p < N$, таких, что $b_k = a_{p+k}$ при $0 \leq k < n$). (Длина цепочки n , как и сама цепочка, является переменной. Предполагается, что каждая ячейка памяти достаточно велика, чтобы вместить произвольное целое число от 0 до N .)
41. [BM28] (Приложение к теории групп.) Пусть G —свободная группа над символами $\{a_1, \dots, a_n\}$, т. е. множество всех цепочек $\alpha = b_1 \dots b_r$, где b_i есть либо a_j , либо a_j^{-1} , и нет смежных пар $a_j a_j^{-1}$ или $a_j^{-1} a_j$. Обратным к α элементом является $b_r^{-1} \dots b_1^{-1}$; мы перемножаем две такие цепочки путем конкатенации и сокращения смежных взаимнообратных элементов. Пусть H —подгруппа группы G , порожденная цепочками $\{\beta_1, \dots, \beta_p\}$, т. е. множество всех элементов из G , которые можно записать в виде произведений β_i и β_j^{-1} . Можно показать (см. А. Г. Курош, Теория групп (М., "Наука", 1967), гл. 9], что всегда существует система образующих $\theta_1, \dots, \theta_m$ для H , $m \leq p$, удовлетворяющая "свойству Нильсена", которое гласит, что средний символ в θ_i (или не менее чем один из двух центральных символов, если θ_i имеет четную длину) никогда не сокращается в выражениях $\theta_i \theta_j^e$ или $\theta_j^e \theta_i$, $e = \pm 1$ (кроме очевидного исключения $i = j$, $e = -1$). Из этого свойства, вытекает, что существует простой алгоритм проверки принадлежности произвольного элемента G подгруппе H : используя $2n$ символов $a_1, \dots, a_n, a_1^{-1}, \dots, a_n^{-1}$, запишем $2m$ ключей $\theta_1, \dots, \theta_m, \theta_1^{-1}, \dots, \theta_m^{-1}$ в дерево, ориентированное на поиск по символам. Пусть $\alpha = b_1 \dots b_r$ —данный элемент из G ; если $r = 0$, то элемент α , разумеется, принадлежит H . В противном случае ищем α , находя самую длинную приставку $b_1 \dots b_k$, соответствующую ключу. Если более чем один ключ начинается с $b_1 \dots b_k$, то элемент α не принадлежит H ; в противном случае обозначим этот единственный ключ через $b_1 \dots b_k c_1 \dots c_l = \theta_l^e$ и заменим α на $\theta_l^{-e} \alpha = c_l^{-1} \dots c_1^{-1} b_{k+1} \dots b_r$. Если это новое значение α длиннее старого (т. е. если $l > k$), α не принадлежит H ; в противном случае повторяем процесс с новым значением α . Свойство Нильсена гарантирует конечность этого алгоритма. Если удалось свести α к пустой цепочке, мы можем восстановить исходное представление α в виде произведений θ_i .
 Например, пусть $\{\theta_1, \theta_2, \theta_3\} = \{bbb, b^{-1}a^{-1}b^{-1}, ba^{-1}b\}$ и $\alpha = bbabaab$. Лес

Picture: Рис. стр. 599

вместе с описанным алгоритмом позволяет получить $\alpha = \theta_1 \theta_3^{-1} \theta_1 \theta_3^{-1} \theta_2^{-1}$. Реализуйте этот алгоритм, используя в качестве входных данных для вашей программы θ_i .

42. [23] (Сжатие спереди и сзади.) Если набор бинарных ключей используется в качестве указателя для расчленения более крупного файла, то нет нужды хранить полные ключи. Например, шестнадцать ключей рис. 34 можно урезать справа так, чтобы оставалось достаточно цифр для их однозначного определения: 0000, 0001, 00100, 00101, 010, \dots , 1110001. Такие урезанные ключи можно использовать для расчленения файла на семнадцать частей; например, пятая часть

состоит из всех ключей, начинающихся с 0011 или 010, а последняя часть содержит все ключи, начинающиеся с 111001, 11101 или 1111. Урезанные ключи можно представить более компактно, если опустить левые цифры, общие с предыдущим ключом: 0000, $** * 1$, $** 100$, $*** * 1$, $* 10$, \dots , $***** 1$. За звездочками всегда следует единица, поэтому ее можно опустить. В большом файле будет много звездочек, и мы должны хранить лишь их число и значения следующих битов. (На этот способ сжатия автору указали Р. Э. Геллер и Р. Л. Йонсен.) Покажите, что суммарное число битов в сжатом файле, исключая звездочки и следующую за ними единицу, равно числу узлов в бинарном бору, содержащем эти ключи.

(Следовательно, среднее суммарное число таких битов во всем указателе примерно равно $N/(\ln 2)$, что составляет лишь около 1.44 бита на ключ. Возможно и еще большее сжатие, так как нам нужно представить лишь структуру бора, ср. с теоремой 2.3.1А).

6.4. ХЕШИРОВАНИЕ

До сих пор мы рассматривали методы поиска, основанные на сравнении данного аргумента K с имеющимися в таблице ключами или на использовании его цифр для управления процессом разветвления. Но есть и третий путь: не рыскать вокруг да около, а произвести над K некоторое арифметическое вычисление и получить функцию $f(K)$, указывающую адрес в таблице, где хранится K и ассоциированная с ним информация.

Например, рассмотрим вновь множество из 31 английского слова, которое мы подвергали воздействию различных стратегий поиска в п. 6.2.2 и § 6.3. Таблица 1 содержит короткую программу для MIX, преобразующую каждый из 31 ключа в уникальное число $f(K)$ между -10 и 30 . Если мы сравним этот метод с MIX-программами для уже изученных нами методов (например, с бинарным поиском, оптимальным поиском по дереву, боровой памятью, цифровым поиском по дереву), то увидим, что он лучше как с точки зрения пространства, так и с точки зрения скорости; только бинарный поиск использует несколько меньше пространства. В самом деле, при использовании программы из табл. 1 среднее время поиска составляет лишь около $17.8u$ (данные о частоте взяты из рис. 12), и для хранения 31 ключа требуется таблица всего для 41 элемента.

К сожалению, находить подобные функции $f(K)$ довольно сложно. Существует $41^{31} \approx 10^{50}$ различных отображений множества из 31 элемента в множество из 41 элемента, и только $41 \cdot 40 \cdot \dots \cdot 11 = 41!/10! \approx 10^{43}$ из них дают различные значения для каждого аргумента; таким образом, лишь одна из каждых 10 миллионов функций оказывается подходящей.

Функции, дающие неповторяющиеся значения, неожиданно редки даже в случае довольно большой таблицы. Например, знаменитый парадокс дней рождения утверждает, что, если в комнате присутствует не менее 23 человек, имеется хороший шанс на то, что у двух из них совпадет день рождения! Иными словами, если мы выбираем случайную функцию, отображающую 23 ключа в 365-элементную таблицу, то с вероятностью 0.4927 (менее половины) все ключи попадут в разные места. Скептики, сомневающиеся в этом, могут попытаться найти совпадение дней рождения на ближайшей большой вечеринке. [Парадокс дней рождения впервые упомянут в работах фон Мизеса (*Istanbul Üniversitesi Fen Fakültesi Mecmuası*, 4 (1939), 145–163) и У. Феллера (Введение в теорию вероятностей и ее приложения, М., "Мир", 1964, гл. 2, § 3).]

Сдругой стороны, подход, использованный в табл. 1, довольно гибкий [ср. с М. Греневски и В. Турски, *SACM*, 6 (1963), 322–323], и для таблицы среднего размера подходящую функцию можно найти примерно за день. В самом деле, очень занятно решать подобные головоломки.

Разумеется, такой метод имеет существенный недостаток, ибо содержимое таблицы должно быть известно заранее; добавление хотя бы одного ключа может все испортить, и нам придется начинать фактически на пустом месте. Можно получить гораздо более гибкий метод, если отбросить идею однозначности, допуская совпадения значений $f(K)$ для различных аргументов, и использовать особый метод разрешения неопределенности после вычисления $f(K)$.

Наши рассуждения приводят к широко известному классу методов, обычно называемых *хешированием* или *рассеянной памятью*. Английский глагол "to hash" имеет смысл нарезать, раскрошить что-либо или сделать из этого месиво; идея хеширования состоит в том, чтобы взять некоторые характеристики ключа и использовать полученную частичную информацию в качестве основы поиска. Мы вычисляем *хеш-функцию* $h(K)$ и берем это значение в качестве адреса начала поиска.

Парадокс дней рождения служит для нас предостережением, что, вероятно, найдутся различные ключи $K_i \neq K_j$, для которых $h(K_i) = h(K_j)$. Подобное событие называется *коллизией*; для разрешения коллизий были разработаны интересные подходы. Чтобы использовать рассеянную таблицу, программист должен принять два почти независимых решения: он должен выбрать хеш-функцию $h(K)$ и метод разрешения коллизий. Эти два аспекта задачи поиска мы и рассмотрим по очереди.

Хеш-функции. Для определенности на протяжении этого пункта будет предполагаться, что хеш-функция h имеет не более M различных значений и что эти значения удовлетворяют условию

$$0 \leq h(K) < M \quad (1)$$

для всех ключей K . В реальном файле много почти одинаковых ключей, поэтому желательно выбрать хеш-функцию, рассеивающую их по таблице. Это важно для уменьшения числа коллизий.

Теоретически невозможно так определить хеш-функцию, чтобы она создавала случайные данные из неслучайных реальных файлов. Но на практике нетрудно сделать достаточно хорошую имитацию случайности, используя простые арифметические действия, обсуждавшиеся в гл. 3. На самом деле мы можем поступить даже лучше, выявляя неслучайные свойства реальных данных и строя на их основе хеш-функцию, дающую меньше коллизий; чем когда имеются истинно случайные ключи.

Рассмотрим, например, случай десятизначных ключей на десятичном компьютере. Сам собой напрашивается следующий способ выбора хеш-функции: положить M равным, скажем, 1000, а в качестве $h(K)$ взять три цифры, выбранные примерно из середины 20-значного произведения $K \times K$. Казалось бы, это должно давать довольно равномерное распределение значений между 000 и 999 с низкой вероятностью коллизий. В самом деле, эксперименты с реальными данными показали, что такой метод "середины квадрата" неплох при условии, что ключи не содержат много левых или правых нулей подряд. Выяснилось, однако, что существуют более надежные и простые способы, подобно тому как в гл. 3 метод середины квадрата оказался не слишком хорошим датчиком случайных чисел.

Многочисленные проверки реальных файлов выявили очень хорошую работу двух основных типов хеш-функций. Один из них основан на делении, а другой на умножении.

Метод деления особенно прост: используется остаток от деления на M

$$h(K) = K \bmod M. \quad (2)$$

В этом случае, очевидно, некоторые значения M много лучше других. Например, если M —четное число, то значение $h(K)$ будет четным при четном K и нечетным в противном случае; часто это приводит к значительным смещениям данных. Совсем плохо брать M равным степени основания системы счисления ЭВМ, так как тогда $h(K)$ дает нам правые значащие цифры K ($K \bmod M$ не зависит от других цифр). Аналогично, M не должно быть кратно 3, ибо буквенные ключи, отличающиеся друг от друга лишь порядком букв, могли бы дать значения функции, разность между которыми кратна 3. (Причина кроется в том, что $10^n \bmod 3 = 4^n \bmod 3 = 1$.) Вообще мы хотели бы избежать значений M , делящих $r^k \pm a$, где k и a —небольшие числа, а r —"основание системы счисления" для множества используемых литер (обычно $r = 64, 256$ и 100), так как остаток от деления на такие значения M обычно оказываются простой суперпозицией цифр ключа. Наши рассуждения подсказывают, что лучше всего *взять в качестве M такое простое число, чтобы $r^k \not\equiv \pm a \pmod{M}$ при небольших k и a .* Практически во всех случаях, этот выбор оказывается вполне удовлетворительным.

Например, на машине MIX можно положить $M = 1009$, вычисляя $h(K)$ следующим образом:

$$\begin{array}{lll} \text{LDX} & K & rX \leftarrow K. \\ \text{ENTA} & 0 & rA \leftarrow 0. \\ \text{DIV} & =1009= & rA \leftarrow K \bmod 1009. \end{array} \quad (3)$$

Мультипликативную схему хеширования также легко реализовать, но несколько труднее описать, так как нужно представить, что мы работаем с дробями, а не с целыми числами. Пусть w есть размер машинного слова (для MIX это значение обычно равно 10^{10} или 2^{30}); целое число A можно рассматривать как дробь A/w , если мысленно поставить десятичную (или двоичную) точку слева от машинного слова, в котором записано A . Метод состоит в том, чтобы выбрать A взаимно простым с w и положить

$$h(K) = \left\lfloor M \left(\left(\frac{A}{w} K \right) \bmod 1 \right) \right\rfloor. \quad (4)$$

На двоичной ЭВМ M обычно берут равным степени двойки, так что $h(K)$ состоит из старших битов правой значащей половины произведения AK .

На двоичной версии машины MIX при $M = 2^m$ мультипликативная хеш-функция вычисляется так:

$$\begin{array}{lll} \text{LDA} & K & rA \leftarrow K. \\ \text{MUL} & A & rAX \leftarrow AK. \\ \text{ENTA} & 0 & rAX \leftarrow AK \bmod w. \\ \text{SLB} & m & \text{Сдвиг } rAX \text{ на } m \text{ битов влево.} \end{array} \quad (5)$$

Результат получается в регистре A. MIX имеет довольно медленные команды умножения и сдвига, поэтому (5) и (3) затрачивают одинаковое время; однако на многих машинах умножение значительно быстрее деления.

В сущности, этот метод можно рассматривать как обобщение (3), поскольку мы могли бы, например, взять в качестве A приближение к $w/1009$; умножение на обратную величину часто оказывается быстрее деления. Заметим, что (5) почти совпадает с методом середины квадрата, однако имеется одно важное отличие: в дальнейшем мы увидим, что умножение на подходящую константу имеет много полезных свойств.

Одна из привлекательных черт мультипликативной схемы состоит в том, что в (5) не происходит потери информации; мы могли бы вновь найти K , зная лишь содержимое rAX после выполнения инструкций (5). Дело в том, что A взаимно просто с w , и при помощи алгоритма Евклида можно найти константу A' : $AA' \bmod w = 1$; отсюда следует, что $K = (A'(AK \bmod w)) \bmod w$. Иными словами, если обозначить через $f(K)$ содержимое регистра X перед выполнением инструкции SLB в (5), то

$$K_1 \neq K_2 \text{ влечет } f(K_1) \neq f(K_2). \quad (6)$$

Конечно, $f(K)$ принимает значения в диапазоне от 0 до $w-1$ и не является сколько-нибудь подходящей хеш-функцией, но она может быть очень полезной в качестве *рассеивающей функции*, а именно функции, удовлетворяющей (6) и обычно приводящей к рандомизации ключей. Например, ее очень хорошо использовать в сочетании с алгоритмами поиска по дереву из п. 6.2.2, если порядок ключей нам безразличен, так как она устраняет опасность вырождения дерева при поступлении ключей в возрастающем порядке. Рассеивающая функция полезна также в связи с алгоритмами цифрового поиска по дереву из § 6.3, если биты истинных ключей имеют смещение.

Другая черта мультипликативного метода хеширования состоит в том, что он хорошо использует неслучайные свойства, которые обнаруживаются во многих файлах. Часто множества истинных ключей содержат арифметические прогрессии $\{K, K+d, K+2d, \dots, K+td\}$, например имена $\{\text{PART1, PART2, PART3}\}$ или $\{\text{TYPEA, TYPEB, TYPEC}\}$. Мультипликативный метод хеширования преобразует арифметические прогрессии в приближенные

Picture: Рис. 37. Фибоначчиево хеширование.

арифметические прогрессии $h(K), h(K+d), h(K+2d), \dots$ несовпадающих величин, уменьшая тем самым число коллизий по сравнению со случайной ситуацией. Метод деления обладает таким же свойством.

Рисунок 37 иллюстрирует этот аспект мультипликативного хеширования в особенно интересном случае. Предположим, что A/w приближенно равно золотому сечению $\phi^{-1} = (\sqrt{5}-1)/2 = 0.6180339887$; поведение последовательных значений $h(K), h(K+1), h(K+2), \dots$ можно изучать, рассматривая поведение величин $h(0), h(1), h(2), \dots$. Это наводит нас на мысль провести такой эксперимент: берется отрезок $[0, 1]$, на котором последовательно отмечаются точки $\{\phi^{-1}\}, \{2\phi^{-1}\}, \{3\phi^{-1}\}, \dots$, где $\{x\}$ обозначает дробную часть числа ($\{x\} = x - \lfloor x \rfloor = x \bmod 1$). Как показано на рис. 37, эти точки расположены не слишком близко друг к другу; каждая вновь добавляемая точка попадает в один из наибольших оставшихся интервалов и делит его золотым сечением! [Это явление было впервые замечено Я. Одерфельдом и доказано С. Сверчковски, *Fundamenta Math.*, **46** (1958), 187–189. Важную роль в доказательстве играют числа Фибоначчи.]

Это замечательное свойство золотого сечения в действительности является частным случаем очень общей теоремы, предугаданной Гуго Штейнгаузом и доказанной Верой Туран Шош [*Acta Math. Acad. Sci. Hung.*, **8** (1957), 461–471; *Ann. Univ. Sci. Budapest. Eötvös Sect. Math.*, **1** (1958), 127–134]:

Теорема S. Пусть θ — любое иррациональное число. Если поместить на отрезок $[0, 1]$ точки $\{\theta\}, \{2\theta\}, \dots, \{n\theta\}$, то длины получившихся отрезков будут выражаться не более чем тремя различными числами. Кроме того, следующая точка $\{(n+1)\theta\}$ попадет в один из отрезков наибольшей длины. ■

Таким образом, точки $\{\theta\}, \{2\theta\}, \dots, \{n\theta\}$ располагаются между 0 и 1 очень равномерно. Теорема верна и для рациональных $\{\theta\}$, если дать подходящую трактовку отрезков нулевой длины, образующихся при n , большем или равном знаменателю $\{\theta\}$. Доказательство теоремы S вместе с подробным анализом возникающей ситуации приводится в упр. 8. Оказывается, что отрезки данной длины создаются и разрушаются в порядке "первым включается — первым исключается". Конечно, некоторые значения $\{\theta\}$ предпочтительнее других; например, если взять $\{\theta\}$ близким к 0 или 1, то сначала образуется много маленьких отрезков и один большой. В упр. 9 показано, что два числа ϕ^{-1} и $\phi^{-2} = 1 - \phi^{-1}$ приводят к "наиболее равномерно распределенным" последовательностям среди всех $\{\theta\}$ от 0 до 1.

Приведенные рассуждения подводят нас к *фибоначчиеву хешированию*, когда в качестве A берется ближайшее к $\phi^{-1}w$ целое число, взаимно простое с w . Например, если бы MIX была десятичной ЭВМ, можно было взять

Picture: Рис. стр. 607

Этот множитель очень хорошо рассеет ключи вроде LIST1, LIST2, LIST3. Посмотрим, однако, что произойдет, если возрастание последовательности происходит в четвертой позиции, как в ключах SUM1□, SUM2□, SUM3□: картина будет такой, как если бы теорема S использовалась с $\theta = \{100A/w\} = 0.80339887$ вместо $\theta = 0.6180339887 = A/w$. Результирующее поведение, однако, будет достаточно хорошим, несмотря на то что это значение θ хуже ϕ^{-1} . Если же возрастание происходит во второй позиции, как в ключах A1□□□, A2□□□, A3□□□, то истинное значение $\theta = 0.9887$; пожалуй, это слишком близко к 1. Поэтому, может быть, лучше вместо (7) взять в качестве множителя

Picture: Рис. стр. 608

Подобный множитель разделит последовательные ключи, различающиеся в *любой* позиции. К сожалению, этот выбор страдает другим пороком, аналогичным делимости $r^k \pm 1$: ключи вроде XY и YX попадут в одно и то же место! Один из способов преодоления возникающей трудности состоит в более пристальном изучении ситуации, лежащей в основе теоремы S. Для коротких последовательностей ключей имеют значение лишь несколько первых неполных частных разложения θ в непрерывную дробь; малость неполных частных соответствует хорошим свойствам распределения. Поэтому наилучшие значения θ выбираются из интервалов

$$\frac{1}{4} < \theta < \frac{3}{10}, \quad \frac{1}{3} < \theta < \frac{3}{7}, \quad \frac{4}{7} < \theta < \frac{2}{3}, \quad \frac{7}{10} < \theta < \frac{3}{4}.$$

В качестве A можно взять такое значение, чтобы каждый из его байтов лежал в хорошем диапазоне и не был слишком близок к значениям других байтов или их дополнениям, например

Picture: Рис. стр. 608

Такой множитель можно рекомендовать. (Изложенные идеи в основном принадлежат Р. Флойду.)

Хорошая хеш-функция должна удовлетворять двум требованиям:

- a) ее вычисление должно быть очень быстрым;
- b) она должна минимизировать число коллизий.

Свойство (a) отчасти зависит от особенностей машины, а свойство (b)—от характера данных. Если бы ключи были действительно случайными, можно было бы просто выделить несколько битов и использовать их для хеш-функции, но на практике, чтобы удовлетворить (b), почти всегда нужна функция, зависящая от всех битов.

До сих пор мы рассматривали хеширование ключей, состоящих из одного слова. С ключами, состоящими из нескольких слов или имеющими переменную длину, можно работать как с представленными с многократной точностью числами и применить к ним рассмотренные методы. Однако обычно оказывается достаточно более быстрая процедура, когда отдельные слова сначала комбинируются в одно, а затем производится единственное умножение или деление. Для комбинирования можно использовать сложение по модулю w или операцию "исключающее или" (на двоичных ЭВМ). Достоинством обеих операций является их обратимость, т. е. их результат зависит от всех битов аргументов, причем "исключающее или" иногда предпочтительнее, так как не может привести к арифметическому переполнению. Заметим, что обе операции коммутативны, поэтому ключи (X, Y) и (Y, X) будут "брошены" по одному адресу. Чтобы избежать этого, Г. Д. Кнотт предложил предварительно делать циклический сдвиг.

Было придумано много других методов хеширования, но ни один из них не оказался предпочтительнее описанных выше простых схем деления и умножения. Обзор некоторых методов и их подробные статистические характеристики при работе с реальными файлами можно найти в статье [V. Y. Lum, P. S. T. Yuen, M. Dodd, *CACM*, 14 (1971), 228–239].

Из других испытанных методов хеширования, пожалуй, наиболее интересным является способ, основанный на алгебраической теории кодирования. Идея аналогична методу деления, только вместо деления на целое число используется деление на многочлен по модулю 2. (Как было замечено в § 4.6, эта операция представляет собой аналог деления, так же как сложение—аналог

”исключающего или”). Для предлагаемого метода M должно быть степенью 2: $M = 2^m$; кроме того, используется многочлен m -й степени $P(x) = x^m + p_{m-1}x^{m-1} + \dots + p_0$. Двоичный ключ $K = (k_{n-1} \dots k_1 k_0)_2$ можно рассматривать как многочлен $K(x) = k_{n-1}x^{n-1} + \dots + k_1x + k_0$, и вычислить остаток $K(x) \bmod P(x) = h_{m-1}x^{m-1} + \dots + h_1x + h_0$, используя полиномиальную арифметику по модулю 2: $h(K) = (h_{m-1} \dots h_1 h_0)_2$. При правильном выборе $P(x)$ такая хеш-функция позволяет избежать коллизий, между почти равными ключами. Например, если $n = 15$, $m = 10$ и

$$P(x) = x^{10} + x^8 + x^5 = x^4 + x^2 + x + 1, \quad (10)$$

то можно показать, что $h(K_1) \neq h(K_2)$, когда $K_1 \neq K_2$ и K_1 отличается от K_2 менее чем семью битами. (В упр. 7 вы найдете дополнительную информацию об этой схеме; разумеется, для нее больше подходит аппаратная или микропрограммная, а не программная реализация.)

Как показал опыт, при отладке программ удобно использовать постоянную хеш-функцию $h(K) = 0$; так как все ключи будут храниться вместе; эффективная $h(K)$ может быть введена позднее.

Разрешение коллизий методом цепочек. Мы уже говорили, что некоторые адреса могут порождаться несколькими ключами. Пожалуй, наиболее очевидный способ решения проблемы состоит в том, чтобы поддерживать M связанных списков, по одному на каждый возможный хеш-адрес. Все записи должны содержать поля LINK; кроме того, нужно иметь M головных узлов списков HEAD[i], где i меняется от 1 до M . После хеширования

Picture: Рис. 38. Раздельные цепочки.

ключа мы просто выполняем последовательный поиск в списке с номером $h(K) + 1$. (Ср. упр. 6.1-2. Ситуация аналогична сортировке вставками в несколько списков—программа 5.2.1M.) Рисунок 38 иллюстрирует этот простой метод цепочек при $M = 9$ для последовательности семи ключей

$$K = \text{EN, TO, TRE, FIRE, FEM, SEKS, SYV} \quad (11)$$

(так называются числа от 1 до 7 по-норвежски), имеющих соответственные хеш-коды

$$h(K) + 1 = 3, 1, 4, 1, 5, 9, 2. \quad (12)$$

Первый список содержит два элемента, три списка пусты.

Метод цепочек является весьма быстрым, поскольку списки коротки. Если в одной комнате собрать 365 человек, то найдется много пар, имеющих один и тот же день рождения, но данный день рождения в среднем имеет лишь один человек! Вообще, если имеется N ключей и M списков, средний размер списка равен N/M ; таким образом, хеширование уменьшает количество работы, требуемое на последовательный поиск, примерно в M раз. (Точная формула выведена в упр. 34.)

Этот метод представляет собой очевидную комбинацию обсуждавшихся ранее методов, поэтому нет нужды подробно описывать алгоритм для рассеянных таблиц с цепочками. Часто полезно содержать отдельные списки упорядоченными по ключам; это убыстряет вставки и неудачный поиск. Так, если делать списки возрастающими, на рис. 38 следовало бы поменять местами узлы TO и FIRE, а все пустые ссылки заменить на указатель на вспомогательную запись с ∞ в качестве ключа (см. алгоритм 6.1T). Другим возможным подходом является использование понятия ”самоорганизующегося файла”, которое обсуждалось в § 6.1; вместо упорядочения по ключам происходит упорядочение по времени последнего обращения к записи.

В целях экономии времени желательны большие M , но в этом случае многие ссылки будут пустыми, так что большая часть пространства, отводимого под M головных узлов, потратится зря. Для небольших по размеру записей напрашивается другой подход: можно наложить пространство для записей на пространство для головных узлов, отводя в таблице место под M записей и M ссылок, а не под N записей и $M + N$ ссылок. Иногда можно совершить один проход по данным и выяснить, какие головные узлы будут использоваться, вставляя на следующем проходе ”переполняющие” записи в свободные щели. Часто, однако, это нежелательно или невозможно; нам хотелось бы иметь метод, при котором каждая запись обрабатывается лишь один раз, при первом поступлении в систему. Следующий алгоритм, принадлежащий Ф. Уильямсу [CASM, 2, 6 (June 1959), 21–24], является общепринятым способом решения этой задачи.

Алгоритм С. (Поиск с вставкой по рассеянной таблице с цепочками.) Предлагаемый алгоритм позволяет отыскать в таблице из M элементов данный ключ K . Если K нет в таблице и она не полна, K вставляется в нее.

Элементы таблицы обозначаются через $TABLE[i]$, $0 \leq i \leq M$, и могут быть двух различных типов: *свободный* и *занятый*. Занятый узел содержит ключевое поле $KEY[i]$, поле ссылки $LINK[i]$ и, возможно, другие поля.

Алгоритм использует хеш-функцию $h(K)$. Для облегчения поиска свободного пространства используется вспомогательная переменная R ; если таблица пуста, $R = M + 1$; по мере проведения вставок будет оставаться в силе утверждение, что узлы $TABLE[j]$ заняты для всех j в диапазоне $R \leq j \leq M$. Условимся, что узел $TABLE[0]$ всегда будет свободен.

C1 [Хеширование.] Установить $i \leftarrow h(K) + 1$. (Теперь $1 \leq i \leq M$.)

C2 [Список?] Если узел $TABLE[i]$ свободен, то перейти на **C6**. (В противном случае этот узел занят, и мы последуем на имеющийся здесь список занятых узлов).

C3 [Сравнение.] Если $K = KEY[i]$, поиск завершен удачно.

C4 [Переход к следующему.] Если $LINK[i] \neq 0$, установить $i \leftarrow LINK[i]$ и вернуться на **C3**.

C5 [Найти свободный узел.] (Поиск был неудачным, и мы хотим найти в таблице свободное место.) Уменьшать R до тех пор, пока не будет получено такое значение, что узел $TABLE[R]$ свободен. Если $R = 0$, алгоритм заканчивается по переполнению (свободных узлов больше нет); в противном случае установить $LINK[i] \leftarrow R$, $i \leftarrow R$.

C6 [Вставить новый ключ.] Пометить $TABLE[i]$ как занятый узел с $KEY[i] \leftarrow K$ и $LINK[i] \leftarrow 0$. ■

В алгоритме допускается срастание нескольких списков, так что после вставки в таблицу записи перемещать не нужно.

Picture: Рис. 39. Поиск с вставкой по рассеянной таблице с цепочками.

[См., например, рис. 40, где $SEKS$ попадает в список, содержащий TO и $FIRE$, так как последний ключ уже был вставлен в позицию 9.]

Picture: Рис. 40. Сросшиеся списки.

Чтобы сравнить данный и другие алгоритмы этой главы, напишем следующую программу для MIX (при составлении программы учитывалось, что списки, как правило, будут короткими).

Программа С. (Поиск с вставкой по рассеянной таблице с цепочками.) Для удобства будем считать, что ключи состоят из трех байтов, а узлы имеют следующий формат:

Picture: Рис. стр. 613

В качестве размера таблицы M берется простое число; $TABLE[i]$ хранится по адресу $TABLE + i$; $rI1 \equiv i$, $rA \equiv K$.

KEY	EQU	3:5		
LINK	EQU	0:2		
START	LDX	K	1	C1. Хеширование.
	ENTA	0	1	
	DIV	=M=	1	
	STX	*+1 (0:2)	1	
	ENT1	*	1	$i \leftarrow h(K) + 1$.
	INC1	1	1	
	LDA	K	1	
	LD2	TABLE, 1 (LINK)	1	C2. Список?
	J2N	6F	1	На C6 , если в $TABLE[i]$ свободно.
	CMPA	TABLE, 1 (KEY)	A	C3. Сравнение.
	JE	SUCCESS	A	Выход, если $K = KEY[i]$.
	J2Z	5F	A - S1	На C5 , если $LINK[i] = 0$.
4H	ENT1	0, 2	C - 1	C4. Переход к следующему.
	CMPA	TABLE, 1 (KEY)	C - 1	C3. Сравнение.
	JE	SUCCESS	C - 1	Выход, если $K = KEY[i]$.
	LD2	TABLE, 1 (LINK)	C - 1 - S2	
	J2NZ	4B	C - 1 - S2	Продвигаться, если $LINK[i] \neq 0$.
5H	LD2	R	A - S	C5. Найти свободный узел.
	DEC2	1	T	$R \leftarrow R - 1$.
	LDX	TABLE, 2	T	

	JXNN	*-2	T	Повторять, пока занят TABLE[R].
	J2Z	OVERFLOW	$A - S$	Выход, если нет свободн. узлов
	ST2	TABLE, 1 (LINK)	$A - S$	LINK[i] ← R.
	ENT1	0, 2	$A - S$	$i \leftarrow R$.
	ST2	R	$A - S$	Изменить R в памяти.
6H	STZ	TABLE, 1 (LINK)	$1 - S$	С6. Вставить новый ключ.
	STA	TABLE, 1 (KEY)	$1 - S$	KEY[i] ← K.

■

Время работы этой программы определяют следующие параметры:

C = число элементов таблицы, исследуемых во время поиска;

$A = 1$, если первый испробованный узел оказался занятым;

$S = 1$ при удачном поиске, 0 при неудачном;

T = число элементов таблицы, просматриваемых в процессе поиска свободного пространства.

Здесь $S = S1 + S2$, где $S1 = 1$, если первая проба была удачной. Суммарное время работы фазы поиска программы С равно $(7C + 4A + 17 - 3S + 2S1)u$, вставка нового ключа при $S = 0$ требует дополнительного времени $(8A + 4T + 4)u$.

Предположим, что в начале работы программы таблица содержит N ключей, и введем

$$\alpha = N/M = \text{коэффициент заполнения (загрузки) таблицы.} \quad (14)$$

Тогда среднее значение A при неудачном поиске, очевидно, равно α (при случайной функции хеширования); в упр. 39 доказывалось, что среднее значение C при неудачном поиске составляет

$$C'_N = 1 + \frac{1}{4} \left(\left(1 + \frac{2}{M} \right)^N - 1 - \frac{2N}{M} \right) \approx 1 + \frac{1}{4} (e^{2\alpha} - 1 - 2\alpha). \quad (15)$$

Таким образом, если таблица заполнена наполовину, среднее число проб, производимых во время неудачного поиска, приблизительно равно $\frac{1}{4}(e + 2) \approx 1.18$; даже если таблица заполняется полностью, среднее число проб при вставке последнего элемента доставляет лишь около $\frac{1}{4}(e^2 + 1) \approx 2.10$. Как доказывалось в упр. 40, среднеквадратичное отклонение также мало. Эти выкладки показывают, что при случайной функции хеширования *списки остаются короткими, несмотря на возможность срастаний*. Разумеется, C может стать равным N , если хеш-функция плоха или если нам здорово не повезло.

Для удачного поиска A всегда равно 1. Среднее число проб можно вычислить, просуммировав величины $C + A$ по первым N неудачным поискам и поделив на N . Предполагается, что все ключи равновероятны. Имеем

$$\begin{aligned} C_N &= \frac{1}{N} \sum_{0 \leq k < N} \left(C'_k + \frac{k}{M} \right) = 1 + \frac{1}{8} \frac{M}{N} \left(\left(1 + \frac{2}{M} \right)^N - 1 - \frac{2N}{M} \right) + \frac{1}{4} \frac{N-1}{M} \approx \\ &\approx 1 + \frac{1}{8\alpha} (e^{2\alpha} - 1 - 2\alpha) + \frac{1}{4}\alpha. \end{aligned} \quad (16)$$

Даже в случае заполненной таблицы в среднем для нахождения элемента понадобится лишь около 1.80 пробы! Аналогично (см. упр. 42) среднее значение $S1$ оказывается равным

$$S1_N = 1 - \frac{1}{2}((N-1)/M) \approx 1 - \frac{1}{2}\alpha. \quad (17)$$

На первый взгляд шаг С5 может показаться неэффективным, так как в нем поиск свободной позиции производится последовательно. Но в действительности в процессе заполнения таблицы суммарное число проб в шаге С5 не превышает количества элементов в таблице; значит, в среднем на каждую вставку тратится не более одной такой пробы! В упр. 41 доказывалось, что для случайного неудачного поиска $T \approx \alpha e^\alpha$.

Можно было бы избежать срастания списков, соответствующим образом модифицировав алгоритм С, но тогда пришлось бы прибегнуть к перемещению записей. Рассмотрим, например, что происходит при попытке вставить SEKS в позицию 9 (см. рис. 40). Чтобы списки оставались отдельными, нужно переместить FIRE; при этом необходимо найти узел, содержащий ссылку на FIRE. Для решения этой задачи можно, как предложил Аллен Ньюэлл в 1962 г., вместо двухсвязевых использовать

кольцевые списки, так как в каждом списке немного элементов. Однако это, вероятно, замедлит главный цикл поиска, поскольку шаг С4 становится сложнее. В упр. 34 показано, что в случае непесекающихся списков среднее число проб уменьшается до

$$C'_N = \left(1 - \frac{1}{M}\right)^N + \frac{N}{M} \approx e^{-\alpha} + \alpha \quad (\text{неудачный поиск}); \quad (18)$$

$$C_N = 1 + \frac{N-1}{2M} \approx 1 + \frac{1}{2}\alpha \quad (\text{удачный поиск}). \quad (19)$$

Вряд ли стоит ради такого улучшения изменять алгоритм.

С другой стороны, Батлер Лэмпсон заметил, что большая часть пространства, в действительности занятого ссылками, может быть сэкономлена в методе цепочек, если избавиться от срастания списков. Это приводит к интересному алгоритму, обсуждающемуся в упр. 13.

Заметим, что метод цепочек можно использовать и при $N > M$, так что переполнение не представляет серьезной проблемы. Если используются отдельные списки, формулы (18) и (19) справедливы для $\alpha > 1$. Когда же списки срастаются, как в алгоритме С, можно помещать переполняющие элементы во вспомогательный пул памяти. Как доказал Л. Гюйба, в этом случае среднее число проб при вставке $(M + L + 1)$ -го элемента составляет $(L/2M + \frac{1}{4})((1 + 2/M)^M - 1) + \frac{1}{2}$.

Разрешение коллизий "открытой адресацией". Другой способ решения проблемы коллизий состоит в том, чтобы полностью отказаться от ссылок и просто просматривать один за другим различные элементы таблицы, пока не будут найдены ключ K или свободная позиция. Не плохо было бы иметь правило, согласно которому каждый ключ K определяет последовательность проб, т. е. последовательность позиций в таблице, которые нужно просматривать всякий раз при вставке или поиске K . Если мы, используя определяемую K последовательность проб, натолкнемся на свободную позицию, то можно сделать вывод, что K нет в таблице, так как та же последовательность проб выполняется каждый раз при обработке данного ключа. Этот общий класс методов У. Петерсон назвал *открытой адресацией* [IBM J. Research & Development, 1 (1957), 130–146].

Простейшая схема открытой адресации, известная как *линейное опробование*, использует циклическую последовательность

$$h(K), h(K) - 1, \dots, 0, M - 1, M - 2, \dots, h(K) + 1 \quad (20)$$

и описывается следующим образом.

Алгоритм L. (Поиск с вставкой по открытой рассеянной таблице.) Алгоритм позволяет разыскать данный ключ K в таблице из M узлов. Если K нет в таблице и она не полна, ключ K вставляется.

Узлы таблицы обозначаются через TABLE[i], $0 \leq i < M$, и принадлежат двум различным типам узлов—*свободных* и *занятых*. Занятый узел содержит ключ KEY[i] и, возможно, другие поля. Значение вспомогательной переменной N равно числу занятых узлов; эта переменная рассматривается как часть таблицы, и при вставке нового ключа ее значение увеличивается на 1.

Данный алгоритм использует хеш-функцию $h(K)$ и линейную последовательность проб (20) для адресации. Модификации этой последовательности обсуждаются ниже.

- L1 [Хеширование.] Установить $i \leftarrow h(K)$. (Теперь $0 \leq i < M$.)
- L2 [Сравнить.] Если узел TABLE[i] свободен, то перейти на L4. В противном случае, если KEY[i] = K , алгоритм заканчивается удачно.
- L3 [Перейти к следующему.] Установить $i \leftarrow i - 1$; если теперь $i < 0$, установить $i \leftarrow i + M$. Вернуться на L2.
- L4 [Вставить.] (Поиск был неудачным.) Если $N = M - 1$, алгоритм заканчивается по переполнению. (В данном алгоритме считается, что таблица полна при $N = M - 1$, а не при $N = M$; см. упр. 15.) В противном случае установить $N \leftarrow N + 1$, отметить, что узел TABLE[i] занят и установить KEY[i] $\leftarrow K$.

На рис. 41 показано, что происходит при вставке с помощью алгоритма L семи "норвежских" ключей (11), имеющих коды хеширования 2, 7, 1, 8, 2, 8, 1 соответственно. Последние три ключа—FEM, SEKS и SYV—смещены по сравнению со своими начальными адресами $h(K)$.

Picture: Рис. 41. Линейная открытая адресация.

Программа L. (Поиск с вставкой по открытой рассеянной таблице.) Программа работает с ключами, занимающими полное слово; однако нулевой ключ использовать нельзя, так как нуль обозначает

свободную позицию в таблице. (Другая возможность состоит в том, чтобы наложить условие неотрицательности ключей, а в свободные позиции записать -1 .) Предполагается, что размер таблицы M — простое число и что узел $\text{TABLE}[i]$ имеет адрес $\text{TABLE} + i$, $0 \leq i < M$. Для ускорения внутреннего цикла в ячейку $\text{TABLE} - 1$ помещен 0 . Ячейка VACANCIES содержит значение $M - 1 - N$; $rA \equiv K$, $rI1 \equiv i$.

Также для ускорения внутреннего цикла программы из него вынесена проверка " $i < 0$ ", так что в нем остались лишь существенные части шагов **L2** и **L3**. Поисковая фаза длится $(7C + 9E + 21 - 4S)u$, а вставка после неудачного поиска требует дополнительно $9u$.

START	LDX	K	1	L1. Хеширование.
	ENTA	0	1	
	DIV	=M=	1	
	STX	*+1(0:2)	1	
	ENT1	*	1	$i \leftarrow h(K)$.
	LDA	K	1	
	JMP	2F	1	
8H	INC1	M+1	E	L3. Перейти к следующему.
3H	DEC1	1	$C + E - 1$	$i \leftarrow i - 1$.
2H	CMPA	TABLE, 1	$C + E$	L2. Сравнить.
	JE	SUCCESS	$C + E$	Выход, если $K = \text{KEY}[i]$.
	LDX	TABLE, 1	$C + E - S$	
	JXNZ	3B	$C + E - S$	На L3 , если в $\text{TABLE}[i]$ занято.
	J1N	8B	$E + 1 - S$	На L3 с $i \leftarrow M$, если $i = -1$.
4H	LDX	VACANCIES	$1 - S$	L4. Вставить,
	JXZ	OVERFLOW	$1 - S$	Выход по переполн., если $N = M - 1$.
	DECX	1	$1 - S$	
	STX	VACANCIES	$1 - S$	Увеличить N на 1.
	STA	TABLE, 1	$1 - S$	$\text{TABLE}[i] \leftarrow K$.

■

Как и в программе **C**, по переменной C мы судим о количестве проб, по переменной S — был ли поиск удачным. Переменной E можно пренебречь, так как она равна 1, лишь если производилась проба фиктивного узла $\text{TABLE}[-1]$; ее среднее значение равно $(C - 1)/M$.

Эксперименты с линейным опробованием показывают, что этот метод работает прекрасно, пока таблица не слишком заполнена, но в конце концов процесс замедляется, длинные серии проб становятся все более частыми. Причину такого поведения можно понять, рассмотрев следующую гипотетическую рассеянную таблицу ($M = 19$, $N = 9$):

Picture: Рис. стр. 618

Заштрихованные квадраты обозначают занятые позиции. Ключ K , который должен быть вставлен в таблицу следующим, попадет в одну из десяти свободных позиций, но не с равными вероятностями. В самом деле, K будет вставлен в позицию 11, если $11 \leq h(K) \leq 15$, а в позицию 8 он попадет лишь при $h(K) = 8$. Следовательно, вероятность попасть в позицию 11 в пять раз больше, чем в позицию 8; длинные списки стремятся стать ещё длиннее.

Но это явление само по себе не объясняет относительно плохого поведения линейного опробования, так как аналогичный факт справедлив и для алгоритма **C**. (Рост списка из четырех элементов вчетверо вероятнее роста списка из одного элемента.) Настоящие трудности возникают при вставке в ячейку вроде 4 или 16 (см. (21)); два различных списка объединяются. Заметим, что в алгоритме **C** списки удлиняются не более чем на один элемент. Следовательно, характеристики линейного опробования резко ухудшаются, когда N приближается к M .

Далее в этом параграфе мы докажем, что среднее число проб, производимых алгоритмом **L**, приближенно равно

$$C'_N \approx \frac{1}{2} \left(1 + \left(\frac{1}{1 - \alpha} \right)^2 \right) \quad (\text{неудачный поиск}); \quad (22)$$

$$C_N \approx \frac{1}{2} \left(1 + \frac{1}{1 - \alpha} \right) \quad (\text{удачный поиск}), \quad (23)$$

где $\alpha = N/M$ есть коэффициент заполнения таблицы. Так что, если таблица заполнена меньше чем на 75%, программа **L** работает почти так же быстро, как и программа **C**, хотя последняя и работает

с неправдоподобно короткими ключами. С другой стороны, когда α приближается к 1, лучшее, что можно сказать о программе L, это то, что она работает медленно, но верно.

В самом деле, если $N = M - 1$, в таблице имеется лишь одно вакантное место, поэтому среднее число проб для неудачного поиска равно $(M + 1)/2$; мы докажем, что среднее число проб для удачного поиска в заполненной таблице приближенно равно $\sqrt{\pi M/8}$.

Явление сгущения, которое делает линейное опробование дорогостоящим в случае почти заполненных таблиц, усугубляется при использовании хеширования делением, если вполне вероятно появление последовательных значений ключей $\{K, K + 1, K + 2, \dots\}$, так как эти ключи будут иметь последовательные хеш-коды. Хеширование умножением рассеет такие группы достаточно хорошо.

Другой способ разрешения проблемы последовательных кодов хеширования состоит в том, чтобы в шаге L3 вместо $i \leftarrow i - 1$ установить $i \leftarrow i - c$. Годится любая положительная величина c , взаимно простая с M , так как мы проверим все позиции таблицы. Это изменение сделает программу L несколько медленнее и не разрешит проблемы сгущения, так как по-прежнему будут образовываться группы записей, удаленных друг от друга на c ; формулы (22) и (23) останутся в силе. Правда, теперь появление последовательных ключей $\{K, K + 1, K + 2, \dots\}$ из помехи превратится в помощь.

Хотя фиксированное значение c не устраняет сгущения, можно существенно улучшить ситуацию, если сделать c зависящим от K . Эта идея приводит к важному изменению алгоритма L, которое впервые открыл Гюи де Бальбин [докторская диссертация, Calif. Inst. of Technology (1968), 149–150].

Алгоритм D. (Открытая адресация с двойным хешированием.) Этот алгоритм почти совпадает с алгоритмом L, но использует несколько иную последовательность проб, вычисляя две хеш-функции $h_1(K)$ и $h_2(K)$. Как обычно, $h_1(K)$ порождает величины от 0 до $M - 1$ включительно; но значения $h_2(K)$ должны лежать от 1 до $M - 1$ и быть взаимно просты с M . (Например, если M — простое число, то $h_2(K)$ может быть любой величиной от 1 до $M - 1$ включительно, или, если $M = 2^m$, то $h_2(K)$ может быть любым нечетным числом между 1 и $2^m - 1$.)

D1 [Первое хеширование.] Установить $i \leftarrow h_1(K)$.

D2 [Первая проба.] Если узел TABLE[i] свободен, то перейти на D6. В противном случае, если KEY[i] = K , алгоритм заканчивается удачно.

D3 [Второе хеширование.] Установить $c \leftarrow h_2(K)$.

D4 [Перейти к следующему.] Установить $i \leftarrow i - c$; если теперь $i < 0$, установить $i \leftarrow i + M$.

D5 [Сравнение.] Если узел TABLE[i] свободен, то перейти на D6. В противном случае, если KEY[i] = K , алгоритм заканчивается удачно; в противном случае вернуться на D4.

D6 [Вставка.] Если $N = M - 1$, алгоритм заканчивается по переполнению. В противном случае установить $N \leftarrow N + 1$, пометить узел TABLE[i] как занятый и установить KEY[i] $\leftarrow K$. ■

Для вычисления $h_2(K)$ было предложено несколько способов: Если M — простое число и $h_1(K) = K \bmod M$, можно положить $h_2(K) = 1 + (K \bmod (M - 1))$; но так как $M - 1$ четно, было бы лучше положить $h_2(K) = 1 + (K \bmod (M - 2))$. Это наводит на мысль о таком выборе M , чтобы M и $M - 2$ были простыми числами-близнецами, например 1021 и 1019. Можно взять $h_2(K) = 1 + (\lfloor K/M \rfloor \bmod (M - 2))$, ибо частное $\lfloor K/M \rfloor$ можно получить в регистре как побочный продукт вычисления $h_1(K)$.

Если $M = 2^m$ и используется хеширование умножением, $h_2(K)$ можно получить, производя сдвиг еще на m битов влево и выполняя операцию "или" с 1, так что к последовательности команд (5) нужно добавить

$$\begin{array}{lll} \text{ENTA} & 0 & \text{Очистить rA.} \\ \text{SLB} & m & \text{Сдвиг rAX на } m \text{ битов влево.} \\ \text{ORR} & =1= & \text{rA} \leftarrow \text{rA} \vee 1. \end{array} \quad (24)$$

Это быстрее метода деления.

В каждом из предложенных выше методов $h_1(K)$ и $h_2(K)$ являются "независимыми" в том смысле, что на различных ключах значения обеих функций h_1 и h_2 совпадают с вероятностью $O(1/M^2)$, а не с вероятностью $O(1/M)$. Эмпирические проверки показывают, что число проб в алгоритме D при независимых хеш-функциях, в сущности, не отличимо от числа проб, которое потребовалось, если бы ключи вставлялись в таблицу случайно; "окучивания", имеющего место в алгоритме L, здесь практически нет.

Можно также допустить зависимость $h_2(K)$ от $h_1(K)$ (как предложил в 1968 г. Г. Кнотт); например, если M — простое число, положим

$$h_2(K) = \begin{cases} 1, & \text{если } h_1(K) = 0; \\ M - h_1(K), & \text{если } h_1(K) > 0. \end{cases} \quad (25)$$

Этот метод быстрее повторного деления, но он все же приводит к вторичному окучиванию, требуя несколько больше проб из-за увеличения вероятности того, что два или более ключей последуют одним

и тем же путем. Формулы, выведенные ниже, можно использовать, чтобы определить, перевешивает ли выигрыш во времени хеширования потери на пробах.

Алгоритмы L и D очень похожи, но имеют и достаточно различий, поэтому полезно сравнить времена работы соответствующих программ для MIX.

Программа D. (Открытая адресация с двойным хешированием.) Так как эта программа весьма напоминает программу L, она приводится без комментариев. Здесь $rI2 \equiv c - 1$.

```

START LDX K          1
      ENTA 0          1
      DIV  =M=        1
      STX  *+1 (0:2)  1
      ENT1 *          1
      LDX  TABLE,1  1
      CMPX K          1
      JE   SUCCESS   1
      JXZ  4F         1 - S1
      SRAX 5         A - S1
      DIV  =M-2=     A - S1
      STX  *+1 (0:2) A - S1
      ENT2 *         A - S1
      LDA  K         A - S1
3H    DEC1 1,2       C - 1
      J1NN *+2       C - 1
      INC1 M         B
      CMPA TABLE,1 C - 1
      JE   SUCCESS   C - 1
      LDX  TABLE,1 C - 1 - S2
      JXNZ 3B        C - 1 - S2
4H    LDX  VACANCIES 1 - S
      JXZ  OVERFLOW  1 - S
      DECX 1         1 - S
      STX  VACANCIES 1 - S
      LDA  K         1 - S
      STA  TABLE,1  1 - S

```

Счетчики частот A , C , $S1$, $S2$ имеют тот же смысл, что и в программе C. Среднее значение переменной B примерно равно $(C - 1)/2$. (Если сузить диапазон $h_2(K)$, скажем, до $1 \leq h_2(K) \leq \frac{1}{2}M$, значение B составило бы лишь $(C - 1)/4$, вероятно, увеличение числа проб не сведет на нет это увеличение скорости.) Если в таблице $N = \alpha M$ ключей, среднее значение A , разумеется, равно α для неудачного поиска и 1 для удачного. Как и в алгоритме C, среднее значение $S1$ для неудачного поиска составляет $1 - \frac{1}{2}((N - 1)/M) \approx 1 - \frac{1}{2}\alpha$. Трудно точно определить среднее число проб, однако эмпирические проверки показывают хорошее согласие с выведенными ниже формулами для "равномерного опробования":

$$C'_N = \frac{M + 1}{M + 1 - N} \approx (1 - \alpha)^{-1} \quad (\text{неудачный поиск}); \quad (26)$$

$$C_N = \frac{M + 1}{N} (H_{M+1} - H_{M+1-N}) \approx -\alpha^{-1} \ln(1 - \alpha) \quad (\text{удачный поиск}), \quad (27)$$

если $h_1(K)$ и $h_2(K)$ независимы. Когда же $h_2(K)$ зависит от $h_1(K)$, как в (25), вторичное окучивание вызывает увеличение средних значений до

$$C'_N = \frac{M + 1}{M + 1 - N} - \frac{N}{M + 1} + H_{M+1} - H_{M+1-N} + O(M)^{-1} \approx (1 - \alpha)^{-1} - \alpha - \ln(1 - \alpha); \quad (28)$$

$$C_N = 1 + H_{M+1} - H_{M+1-N} - \frac{N}{2(M + 1)} - (H_{M+1} - H_{M+1-N})/N + O(N^{-1}) \approx 1 - \ln(1 - \alpha) - \frac{1}{2}\alpha. \quad (29)$$

(См. упр. 44.) Заметим, что, когда таблица заполняется, т. е. когда N стремится к M , значение C_N приближается к $H_{M+1} - 1$, а C'_N — к $H_{M+1} - \frac{1}{2}$; это много лучше, чем в случае алгоритма L, но не столь хорошо, как в методах цепочек.

Поскольку в алгоритме L пробы занимают несколько меньше времени, двойное хеширование предпочтительно лишь при заполненной таблице. На рис. 42 сравниваются средние времена работы программ L, D и модифицированной программы D (причем эта модификация влечет за собой вторичное скучивание): мы заменяем довольно медленное вычисление $h_2(K)$ в строках 10–13 на три команды

$$\begin{array}{lll} \text{ENN2} & -M-1, 1 & c \leftarrow M - i. \\ \text{J1NZ} & *+2 & \\ \text{ENT2} & 0 & \text{Если } i = 0, c \leftarrow 1. \end{array} \quad (30)$$

Picture: Рис. 42. Время удачного поиска для трех схем открытой адресации.

В данном случае вторичное скучивание лучше независимого повторного хеширования.

На двоичной ЭВМ мы могли бы иным способом ускорить вычисление $h_2(K)$, заменив строки 10–13, скажем, на

$$\begin{array}{lll} \text{AND} & =511= & rA \leftarrow rA \bmod 512. \\ \text{STA} & *+1 (0:2) & \\ \text{ENT2} & * & c \leftarrow rA + 1. \end{array} \quad (31)$$

если M — простое число, большее 512. Эта идея, предложенная Беллом и Камá [SACM, 13 (1970), 675–677], независимо придумавшими алгоритм D, позволяет избежать вторичного скучивания без затрат на повторное деление.

Было предложено много других последовательностей проб в качестве улучшений алгоритма L, но, по-видимому, ни одна из них не лучше алгоритма D. (Быть может, исключение составляет метод, описанный в упр. 20.)

Picture: Рис. 43. Сколько раз компилятор ищет имена переменных в типичном случае. Имена выписаны слева направо в порядке их первого появления.

Изменение, предложенное Brentом. Ричард Brent нашел такой способ изменения алгоритма D, чтобы среднее время удачного поиска оставалось ограниченным по мере заполнения таблицы. Его метод основан на том факте, что во многих приложениях удачный поиск производится гораздо чаще вставок; поэтому он предлагает делать больше работы при вставке элемента, перемещая записи, чтобы уменьшить ожидаемое время выборки. [SACM, 16 (1973), 105–109.]

Например, на рис. 43 показано, сколько раз тот или иной идентификатор встречался в типичной процедуре на PL/1. Судя по этим данным, компилятор с PL/1, использующий рассеянную таблицу для хранения имен переменных, будет искать многие имена пять или более раз, а вставлять их лишь однажды. Аналогично Белл и Камá обнаружили, что компилятор с Кобола использовал свой алгоритм таблиц символов 10988 раз во время компиляции программы и совершил лишь 735 вставок, т. е. на один неудачный поиск приходилось примерно 14 удачных. Иногда таблица создается только один раз (например, таблица символических кодов операций в автокоде) и используется после этого исключительно для выборки.

Brent предложил следующим образом изменить процесс вставки в алгоритме D. Предположим, что при неудачном поиске были опробованы ячейки

$$p_0, p_1, \dots, p_{t-1}, p_t,$$

где $p_j = (h_1(K) - jh_2(K)) \bmod M$ и узел $\text{TABLE}[p_t]$ пуст. Если $t \leq 1$, мы, как обычно, вставляем K в позицию p_t , но если $t \geq 2$, вычисляем $c_0 = h_2(K_0)$, где $K_0 = \text{KEY}[p_0]$, и смотрим, свободно ли в $\text{TABLE}[(p_0 - c_0) \bmod M]$. Если условие выполнено, помещаем в данный узел содержимое ячейки $\text{TABLE}[p_0]$ и вставляем K в позицию p_0 . Это увеличивает время выборки K_0 на один шаг, но уменьшает время выборки K на $t \geq 2$ шагов. Выигрыш налицо. Аналогично, если в $\text{TABLE}[(p_0 - c_0) \bmod M]$ занято и $t \geq 3$, мы пробуем ячейку $\text{TABLE}[(p_0 - 2c_0) \bmod M]$; если и там занято, вычисляем $c_1 = h_2(\text{KEY}[p_1])$ и пробуем $\text{TABLE}[(p_1 - c_1) \bmod M]$ и т. д. Вообще, пусть $c_j = h_2(\text{KEY}[p_j])$ и $p_{j,k} = (p_j - kc_j) \bmod M$; если позиции $\text{TABLE}[p_{j,k}]$ оказались занятыми при всех j, k , таких, что $j + k < r$, и если $t \geq r + 1$, просматриваем узлы $\text{TABLE}[p_{0,r}], \text{TABLE}[p_{1,r-1}], \dots, \text{TABLE}[p_{r-1,1}]$. Если первой свободной оказалась позиция $p_{j,r-j}$, устанавливаем $\text{TABLE}[p_{j,r-j}] \leftarrow \text{TABLE}[p_j]$ и вставляем K в позицию p_j .

Как явствует из анализа Brentа, среднее число проб в процессе удачного поиска уменьшилось (см. рис. 44) и его максимальное значение приближенно равно 2.49.

Число $t + 1$ проб при неудачном поиске в результате предложенного изменения не уменьшилось; оно осталось на уровне, указанном формулой (26), и достигает $(M + 1)/2$ для заполненной таблицы. При вставке функцию h_2 приходится в среднем вычислять $\alpha^2 + \alpha^5 + \frac{1}{3}\alpha^6 + \dots$ раз. Согласно анализу Brenta, при $\alpha \rightarrow 1$, эта величина имеет порядок \sqrt{M} . Наконец, когда мы решаем, как произвести вставку, совершается около $\alpha^2 + \alpha^4 + \frac{4}{3}\alpha^5 + \alpha^6 + \dots$ дополнительных проб.

Удаления. Многие программисты свято верят в алгоритмы и очень удивляются, обнаружив, что очевидный способ удаления записей из рассеянной таблицы не работает. Например, при попытке удалить ключ EN (см. рис. 41) нельзя просто пометить эту позицию в таблице как свободную, ибо другой ключ FEM внезапно окажется потерянным! (Вспомним, что EN и FEM имеют одинаковые хеш-коды. При поиске FEM мы наткнемся на свободный узел, что свидетельствует о том, что поиск неудачен.) Аналогичное соображение справедливо для алгоритма С, в котором имеет место срастание списков; представьте себе удаление ключей TO и FIRE (см. рис. 40).

Вообще говоря, можно производить удаления, помещая в соответствующую ячейку специальное значение, т. е. имея три типа позиций в таблице: свободные, занятые и удаленные. При поиске ключа удаленные позиции нужно трактовать как занятые. В случае неудачного поиска ключ можно вставить в первую встреченную удаленную или свободную позицию.

Однако эта идея применима только при очень редких удалениях, потому что однажды занятая позиция не может снова стать свободной. После длинной последовательности вставок и удалений все свободное пространство в конце концов исчезнет и каждый неудачный поиск будет требовать M проб! Кроме того, пробы потребуют больше времени, так как в шаге D4 необходимо проверять, не вернулось ли i к своему начальному значению, а число проб при удачном поиске возрастет с C_N до C'_N . Если используется линейное опробование (т. е. алгоритм L), можно действовать рациональнее, при условии что мы согласны потратить некоторые дополнительные усилия на удаления.

Алгоритм R. (Удаление при линейном опробовании.) Настоящий алгоритм удаляет запись из данной позиции TABLE[i] открытой рассеянной таблицы, построенной с помощью алгоритма L.

R1 [Освободить ячейку.] Отметить ячейку TABLE[i] как свободную и установить $j \leftarrow i$.

R2 [Уменьшить i .] Установить $i \leftarrow i - 1$ и, если значение i стало отрицательным, установить $i \leftarrow i + M$.

R3 [Проверить TABLE[i .] Если в TABLE[i] свободно, алгоритм завершается. В противном случае установить $r \leftarrow h(\text{KEY}[i])$ —первоначальный хеш-код ключа, хранящегося теперь в позиции i . Если $i \leq r < j$, или $r < j < i$, или $j < i \leq r$ (другими словами, если r лежит циклически между i и j), вернуться на R2.

R4 [Переместить запись.] Установить TABLE[j] \leftarrow TABLE[i] и вернуться на R1. ■

В упр. 22 показано, что этот алгоритм не вызывает ухудшения характеристик, т. е. среднее число проб, предсказанное формулами (22) и (23), остается прежним. (Более слабый результат для удаления из дерева был доказан в теореме 6.2.2Н.) Однако справедливость алгоритма R сильно зависит от того факта, что используется линейное опробование; в случае алгоритма D не существует аналогичной процедуры удаления.

Разумеется, если используется метод цепочек и для каждого хеш-кода имеется отдельный список, удаление не является проблемой, так как сводится просто к удалению из связанного линейного списка. Удаления для алгоритма С обсуждаются в упр. 23.

***Анализ алгоритмов.** При хешировании мы полагаемся на вероятностные законы, поэтому особенно важно знать поведение методов хеширования в среднем. Наихудший случай в этих алгоритмах почти немислимо плох, так что нам необходима уверенность в том, что поведение в среднем является очень хорошим.

Прежде чем приступить к анализу линейного опробования и т. п., рассмотрим весьма приближенную модель данной ситуации, которую можно назвать *равномерным хешированием* (ср. с W. W. Peterson, *IBM J. Research & Development*, 1 (1957), 135–136). В данной модели предполагается, что ключи попадают в случайные позиции таблицы, так что все $\binom{M}{N}$ возможных конфигураций из N занятых и $M - N$ свободных ячеек равновероятны. Эта модель совершенно не учитывает влияния первичного или вторичного окучивания; в сущности, предполагается, что занятость любой ячейки не зависит от занятости других. Для рассматриваемой модели вероятность того, что для вставки $(N + 1)$ -го элемента требуется ровно r проб, равна числу конфигураций, в которых занято $r - 1$ данных ячеек, а еще одна данная ячейка свободна, деленному на $\binom{M}{N}$, т. е.

$$P_r = \binom{M - r}{N - r + 1} / \binom{M}{N};$$

следовательно, среднее число проб для равномерного хеширования составляет

$$\begin{aligned}
 C'_N &= \sum_{1 \leq r \leq M} rP_r = M + 1 - \sum_{1 \leq r \leq M} (M + 1 - r)P_r = \\
 &= M + 1 - \sum_{1 \leq r \leq M} (M + 1 - r) \binom{M - r}{M - N - 1} / \binom{M}{N} = \\
 &= M + 1 - \sum_{1 \leq r \leq M} (M - N) \binom{M + 1 - r}{M - N} / \binom{M}{N} = \\
 &= M + 1 - (M - N) \binom{M + 1}{M - N + 1} / \binom{M}{N} = \\
 &= M + 1 - (M - N) \frac{M + 1}{M - N + 1} = \frac{M + 1}{M - N + 1}, \quad 1 \leq N \leq M. \tag{32}
 \end{aligned}$$

(Мы уже решали, в сущности, ту же задачу, возникающую в связи со случайной выборкой; упр. 3.4.2-5.) Полагая $\alpha = N/M$, точную формулу для C'_N можно приближенно заменить на

$$\frac{1}{1 - \alpha} = 1 + \alpha + \alpha^2 + \alpha^3 + \dots \tag{33}$$

Этому ряду можно дать грубую интуитивную интерпретацию: с вероятностью α нужна более чем одна проба, с вероятностью α^2 —более чем две и т. д. Соответствующее среднее число проб при удачном поиске равно

$$\begin{aligned}
 C_N &= \frac{1}{N} \sum_{0 \leq k < N} C'_k = \frac{M + 1}{N} \left(\frac{1}{M + 1} + \frac{1}{M} + \dots + \frac{1}{M - N + 2} \right) = \\
 &= \frac{M + 1}{N} (H_{M+1} - H_{M-N+1}) \approx \frac{1}{\alpha} \log \frac{1}{1 - \alpha}. \tag{34}
 \end{aligned}$$

Как отмечалось выше, многочисленные тесты показали, что алгоритм D с двумя независимыми хеш-функциями ведет себя, по существу, как равномерное хеширование, и для всех практических нужд можно использовать формулы (32)–(34). На самом деле Л. Гюйба и Э. Семереди удалось доказать трудную теорему о том, что двойное хеширование асимптотически (при $M \rightarrow \infty$) эквивалентно равномерному [будет опубликовано].

На этом анализ равномерного хеширования завершается. Чтобы изучать линейное опробование и другие способы разрешения коллизии, нужно строить теорию иным, более реалистичным способом. В вероятностной модели, которую мы будем использовать для этой цели, предполагается, что все M^N возможных хеш-последовательностей

$$a_1 a_2 \dots a_N, \quad 0 \leq a_j < M, \tag{35}$$

равновероятны. Здесь a_j обозначает первоначальный хеш-адрес ключа, вставленного в таблицу в позицию j . Как и выше, среднее число проб в случае удачного поиска при любом данном алгоритме обозначим через C_N . Это есть среднее число проб, нужных для нахождения k -го ключа, усредненное по $1 \leq k \leq N$ при равновероятных ключах и по всем хеш-последовательностям (35), которые также считаются равновероятными. Аналогично, среднее число проб, необходимых для вставки N -го ключа, при условии что все последовательности (35) равновероятны, обозначим через C'_{N-1} ; это есть среднее число проб в неудачном поиске, если в таблице $N - 1$ элементов. Если используется открытая адресация, то

$$C_N = \frac{1}{N} \sum_{0 \leq k < N} C'_k, \tag{36}$$

так что, зная одну величину, можно получить другую, как было сделано в (34).

Строго говоря, даже эта более правильная модель имеет два недостатка. Во-первых, не все хеш-последовательности равновероятны, потому что сами ключи являются различными. Это делает вероятность равенства $a_1 = a_2$ несколько меньшей, чем $1/M$, однако разностью обычно можно пренебречь, так как количество всех возможных ключей в типичном случае много больше M . (См. упр. 24.) Далее, хорошая хеш-функция будет использовать неслучайность обычных данных, делая равенство $a_1 = a_2$ еще менее вероятным; в результате наши оценки числа проб будут пессимистическими. Другая неточность данной модели указана на рис. 43: ключи, появившиеся ранее (за некоторыми исключениями),

разыскиваются чаще. Следовательно, наши оценки C_N будут вдвойне пессимистическими, а алгоритмы на практике должны работать несколько лучше, чем предсказывает наш анализ.

После этой преамбулы мы готовы выполнить "точный" анализ линейного опробования⁸. Через $f(M, N)$ обозначим число хеш-последовательностей (35), таких, что позиция 0 таблицы будет свободна после вставки ключей с помощью алгоритма L. Из круговой симметрии линейного опробования следует, что позиция 0 свободна с той же вероятностью, что и любая другая позиция, т. е. с вероятностью $1 - N/M$; иными словами,

$$f(M, N) = \left(1 - \frac{N}{M}\right) M^N. \quad (37)$$

По определению полагаем $f(0, 0) = 1$. Через $g(M, N, k)$ обозначим число хеш-последовательностей (35), таких, что алгоритм оставляет позицию 0 свободной, позиции с 1 по k занятыми, а позицию $k + 1$ свободной. Имеем

$$g(M, N, k) = \binom{N}{k} f(k + 1, k) f(M - k - 1, N - k), \quad (38)$$

ибо все такие последовательности состоят из двух подпоследовательностей. Одна (содержащая k элементов $a_i \leq k$) оставляет позицию 0 свободной и позиции с 1 по k занятыми, а другая (содержащая $N - k$ элементов $a_j \geq k + 1$) оставляет позицию $k + 1$ свободной. Существует $f(k + 1, k)$ последовательностей первого типа и $f(M - k - 1, N - k)$ второго и $\binom{N}{k}$ способов смешать две такие подпоследовательности. Наконец, положим P_k равным вероятности того, что при вставке $(N + 1)$ -го элемента понадобится ровно $(k + 1)$ проб; можно показать (см. упр. 25), что

$$P_k = M^{-N} (g(M, N, k) + g(M, N, k + 1) + \dots + g(M, N, N)). \quad (39)$$

Теперь $C'_N = \sum_{0 \leq k \leq N} (k + 1) P_k$; подстановка соотношений (36)–(39) и упрощения дают следующий результат.

Теорема К. Среднее число проб, нужных при использовании алгоритма L (если считать, что все M^N хеш-последовательностей (35) равновероятны), равно

$$C_N = \frac{1}{2} (1 + Q_0(M, N - 1)) \quad (\text{удачный поиск}); \quad (40)$$

$$C'_N = \frac{1}{2} (1 + Q_1(M, N)) \quad (\text{неудачный поиск}), \quad (41)$$

где

$$\begin{aligned} Q_r(M, N) &= \binom{r}{0} + \binom{r+1}{1} \frac{N}{M} + \binom{r+2}{2} \frac{N(N-1)}{M^2} + \dots = \\ &= \sum_{k \geq 0} \binom{r+k}{k} \frac{N}{M} \frac{N-1}{M} \dots \frac{N-k+1}{M}. \end{aligned} \quad (42)$$

Доказательство. Подробные вычисления проведены в упр. 27. ■

Появляющаяся в теореме функция $Q_r(M, N)$ выглядит не очень изящно, но в действительности работать с ней можно. Имеем

$$N^k - \binom{k}{2} N^{k-1} \leq N(N-1) \dots (N-k+1) \leq N^k;$$

следовательно, если $N/M = \alpha$, то

$$\begin{aligned} \sum_{k \geq 0} \binom{r+k}{r} \left(N^k - \binom{k}{2} N^{k-1} \right) / M^k &\leq Q_r(M, N) \leq \sum_{k \geq 0} \binom{r+k}{k} N^k / M^k, \\ \sum_{k \geq 0} \binom{r+k}{k} \alpha^k - \frac{\alpha}{M} \sum_{k \geq 0} \binom{r+k}{k} \binom{k}{2} \alpha^{k-2} &\leq Q_r(M, \alpha M) \leq \sum_{k \geq 0} \binom{r+k}{k} \alpha^k, \end{aligned}$$

⁸ Здесь автор не может удержаться от биографического замечания. Впервые предлагаемый вывод был сформулирован мною в 1962 г., вскоре после начала работы над "Искусством программирования для ЭВМ". Мой первый успешный анализ нетривиального алгоритма оказал сильное влияние на структуру этих книг. Мог ли я думать, что пройдет более десяти лет, прежде чем этот вывод попадет в печать!

т. е.

$$\frac{1}{(1-\alpha)^{r+1}} - \frac{1}{M} \binom{r+2}{2} \frac{\alpha}{(1-\alpha)^{r+3}} \leq Q_r(M, \alpha M) \leq \frac{1}{(1-\alpha)^{r+1}}. \quad (43)$$

Это соотношение дает нам хорошую оценку $a_r(M, N)$, когда M велико и α не слишком близко к 1. (Нижняя оценка ближе к истине, чем верхняя.) Если α приближается к 1, формула (43) становится бесполезной, но, к счастью, $a_0(M, M-1)$ есть функция $a(M)$, асимптотическое поведение которой мы детально изучили в п. 1.2.11.3; а $a_1(M, M-1)$ просто равно M (упр. 50).

Г. Шай мл. и В. Спрут избрали другой подход к анализу линейного опробования [САСМ, 5 (1962), 459–462]. Хотя их метод дает лишь приближение к точным формулам теоремы К, он проливает дополнительный свет на алгоритм, поэтому мы кратко изложим его. Сначала рассмотрим удивительное свойство линейного опробования, впервые отмеченное У. Петерсоном в 1957 г.

Теорема Р. Среднее число проб при удачном поиске с помощью алгоритма L не зависит от порядка, в котором ключи были вставлены, а зависит лишь от того, сколько ключей имеют тот или иной хеш-адрес.

Иными словами, любое переупорядочение хеш-последовательности $a_1 a_2 \dots a_N$ дает последовательность с тем же средним смещением ключей от их хеш-адресов. (Как указано выше, мы предполагаем, что все ключи в таблице имеют одинаковую важность. Если к некоторым из них происходят более частые обращения, методом, аналогичным использованному в теореме 6.1S, можно доказать, что расположение ключей оптимально, если они вставлялись в порядке убывания частот.)

Доказательство. Достаточно показать, что общее число проб, необходимых при вставке ключей для хеш-последовательности $a_1 a_2 \dots a_N$, равно числу проб для последовательности $a_1 \dots a_{i-1} a_{i+1} a_i a_{i+2} \dots a_N$, $1 \leq i < N$. Очевидно, нужно рассмотреть лишь случай, когда $(i+1)$ -й ключ во второй последовательности попал в позицию, занимаемую i -м ключом в первой последовательности. Но если i -й и $(i+1)$ -й элементы просто поменялись местами, то число проб для $(i+1)$ -го ключа уменьшилось на столько, на сколько увеличилось число проб для i -го. ■

Теорема Р гласит, что среднюю длину поиска для хеш-последовательности $a_1 a_2 \dots a_N$ можно найти, зная числа $b_0 b_1 \dots b_{M-1}$, где b_j есть количество элементов a_i , равных j . По этой последовательности можно определить "последовательность переносов" $c_0 c_1 \dots c_{M-1}$, где c_j —число ключей, при вставке которых опробовались обе позиции j и $j-1$. Эта последовательность определяется правилом

$$c_j = \begin{cases} 0, & \text{если } b_j = c_{(j+1) \bmod M} = 0; \\ b_j + c_{(j+1) \bmod M} - 1 & \text{в противном случае.} \end{cases} \quad (44)$$

Пусть, например, $M = 10$, $N = 8$ и $b_0 \dots b_9 = 032010002$; тогда $c_0 \dots c_9 = 2310000123$, так как один ключ нужно перенести из позиции 2 в позицию 1, три—из позиции 1 в позицию 0, два—из позиции 0 в позицию 9 и т. д. Имеем $b_0 + b_1 + \dots + b_{M-1} = N$, а среднее число проб, требующихся для выборки этих N ключей, равно

$$1 + (c_0 + c_1 + \dots + c_{M-1})/N. \quad (45)$$

Кажется, что правило (44) циклически определяет числа c через самих себя, но в действительности при любом $N < M$ система имеет единственное решение (см. упр. 32).

Г. Шай и В. Спрут использовали эту идею для определения вероятностей q_k (вероятности того, что $c_j = k$) через вероятности p_k (вероятности того, что $b_j = k$). (Эти вероятности не зависят от j .) Так,

$$\begin{aligned} q_0 &= p_0 q_0 + p_1 q_0 + p_0 q_1, \\ q_1 &= p_2 q_0 + p_1 q_1 + p_0 q_2, \\ q_2 &= p_3 q_0 + p_2 q_1 + p_1 q_2 + p_0 q_3 \end{aligned} \quad (46)$$

и т. д., поскольку, например, вероятность того, что $c_j = 2$, есть вероятность того, что $b_j + c_{(j+1) \bmod M} = 3$. Пусть $B(z) = \sum p_k z^k$ и $C(z) = \sum q_k z^k$ —производящие функции для рассматриваемых вероятностных распределений; уравнения (46) эквивалентны соотношению

$$B(z)C(z) = p_0 q_0 + (q_0 - p_0 q_0)z + q_1 z^2 + \dots = p_0 q_0(1-z) + zC(z).$$

Так как $B(1) = 1$, можно написать $B(z) = 1 + (z-1)D(z)$. Отсюда вытекает, что

$$C(z) = \frac{p_0 q_0}{1-D(z)} = \frac{1-D(1)}{1-D(z)}, \quad (47)$$

поскольку $C(1) = 1$. Следовательно, среднее число проб, необходимых для выборки, в соответствии с (45) составит

$$1 + \frac{M}{N} C'(1) = 1 + \frac{M}{N} \frac{D'(1)}{1 - D(1)} = 1 + \frac{M}{2N} \frac{B''(1)}{1 - B'(1)}. \tag{48}$$

В силу предположения о равновероятности всех хеш-последовательностей имеем

$$p_k = (\text{Вероятность того, что для фиксированного } j \text{ в точности } k \text{ чисел } a_i \text{ равно } j) = \binom{N}{k} \left(\frac{1}{M}\right)^k \left(1 - \frac{1}{M}\right)^{N-k}; \tag{49}$$

поэтому

$$B(z) = \left(1 + \frac{z-1}{M}\right)^N, \quad B'(1) = \frac{N}{M}, \quad B''(1) = \frac{N(N-1)}{M^2} \tag{50}$$

и среднее число проб, согласно (48), составит

$$C_N = \frac{1}{2} \left(1 + \frac{M-1}{M-N}\right). \tag{51}$$

Понятно ли читателю, почему этот ответ отличается от результата теоремы К? (Ср. с упр. 33.)

***Исследование оптимальности.** Мы изучили несколько последовательностей проб для открытой адресации, и естественно спросить, какая из них является "наилучшей из возможных" в некотором разумном смысле. Интересную постановку этой задачи предложил Дж. Ульман [JACM, 19 (1972), 569–575]: вместо вычисления "хеш-адреса" $h(K)$ можно отобразить каждый ключ K в целую перестановку множества $\{0, 1, \dots, M-1\}$, представляющую собой последовательность проб при использовании K . Каждой из $M!$ перестановок приписывается вероятность, и предполагается, что обобщенная хеш-функция выбирает каждую перестановку с этой вероятностью. Вопрос ставится так: "Как нужно приписать перестановкам вероятности, чтобы получить наилучшие характеристики, т. е. чтобы соответствующее среднее число проб C_N или C'_N было минимальным?"

Если, например, приписать каждой перестановке вероятность $1/M!$, то, как легко видеть, получится в точности поведение *равномерного хеширования* (см. (32), (34)). Однако Ульман построил пример с $M = 4$ и $N = 2$, в котором C'_N меньше $\frac{5}{3}$ (это значение получается при равномерном хешировании). Он приписал ненулевые вероятности лишь следующим шести перестановкам:

Перестановка	Вероятность	Перестановка	Вероятность
0 1 2 3	$(1 + 2\epsilon)/6$	1 0 3 2	$(1 + 2\epsilon)/6$
2 0 1 3	$(1 - \epsilon)/6$	2 1 0 3	$(1 - \epsilon)/6$
3 0 1 2	$(1 - \epsilon)/6$	3 1 0 2	$(1 - \epsilon)/6$

(52)

Грубо говоря, на первом шаге мы предпочитаем 2 и 3, а на втором—0 и 1. Среднее число проб, необходимых для вставки третьего элемента, оказывается равным $\frac{5}{3} - \frac{1}{9}\epsilon + O(\epsilon^2)$, что меньше $\frac{5}{3}$ при малом положительном ϵ .

Однако при таком распределении вероятностей $C'_1 = \frac{23}{18} + O(\epsilon)$, а это больше $\frac{5}{4}$ (значение C'_1 для равномерного хеширования). Ульман доказал, что при любом способе приписывания вероятностей, таком, что $C'_N < (M+1)/(M+1-N)$ при некотором N , всегда существует $n < N$, такое, что $C'_n > (M+1)/(M+1-n)$; нельзя все время побеждать равномерное хеширование.

В действительности в качестве меры лучше подходит не C'_N , а число проб при *удачном* поиске C_N . Перестановки (52) не дают улучшения C_N ни при каком N , и кажется весьма разумным предположить, что никакое приписывание перестановкам вероятностей не может сделать C_N меньше "равномерного" значения $((M+1)/N)(H_{M+1} - H_{M+1-N})$.

Как оказывается, это утверждение очень трудно доказать главным образом потому, что получить эффект равномерного хеширования можно многими способами; мы не обязаны приписывать каждой перестановке вероятность $1/M!$. Например, следующее распределение при $M = 4$ эквивалентно равномерному хешированию:

Перестановка	Вероятность	Перестановка	Вероятность
0 1 2 3	$1/6$	0 2 1 3	$1/12$
1 2 3 0	$1/6$	1 3 2 0	$1/12$
2 3 0 1	$1/6$	2 0 3 1	$1/12$

(53)

(остальным 16 перестановкам приписывается нулевая вероятность).

Следующая теорема характеризует *все* распределения вероятностей, которые дают поведение равномерного хеширования:

Теорема У. Приписывание перестановкам вероятностей делает все $\binom{M}{N}$ конфигураций занятых и свободных ячеек после N вставок равновероятными для $0 < N < M$ тогда и только тогда, когда сумма вероятностей, приписанных всем перестановкам, первые N элементов которых принадлежат данному N -элементному множеству, равна $1/\binom{M}{N}$ для всех N и всех N -элементных множеств.

Например, сумма вероятностей, приписанных каждой из $3!(M-3)!$ перестановок, начинающихся с чисел $\{0, 1, 2\}$, стоящих в некотором порядке, должна составлять $1/\binom{M}{3} = 3!(M-3)!/M!$. Заметьте, что (53) удовлетворяет условию теоремы.

Доказательство. Пусть $A \subseteq \{0, 1, \dots, M-1\}$, и пусть $\Pi(A)$ есть множество всех перестановок, первые $\|A\|$ элементов которых принадлежат A . Сумму вероятностей, приписанных этим перестановкам, обозначим через $S(A)$. Символом $P_k(A)$ обозначим вероятность того, что первые $\|A\|$ ключей, вставленных процедурой открытой адресации, займут ячейки, определенные множеством A , и что последняя вставка потребует ровно k проб; положим $P(A) = P_1(A) + P_2(A) + \dots$. Доказательство проводится индукцией по $N \geq 1$; предполагается, что

$$P(A) = S(A) = 1/\binom{M}{n}$$

для всех множеств A с $\|A\| = n < N$. Пусть B есть N -элементное множество. Тогда

$$P_k(B) = \sum_{\substack{A \subseteq B \\ \|A\|=k}} \sum_{\pi \in \Pi(A)} \text{Pr}(\pi) P(B \setminus \{\pi_k\}),$$

где $\text{Pr}(\pi)$ есть вероятность, приписанная перестановке π , а π_k — ее k -й элемент. В силу индуктивного предположения

$$P_k(B) = \sum_{\substack{A \subseteq B \\ \|A\|=k}} \frac{1}{\binom{M}{N-1}} \sum_{\pi \in \Pi(A)} \text{Pr}(\pi),$$

что равняется

$$\binom{N}{k} / \binom{M}{N-1} \binom{M}{k}, \quad \text{если } k < N;$$

следовательно,

$$P(B) = \frac{1}{\binom{M}{N-1}} \left(S(B) + \sum_{1 \leq k < N} \binom{N}{k} \binom{M}{k} \right),$$

а это равняется $1/\binom{M}{N}$ тогда и только тогда, когда $S(B)$ имеет требуемое значение. ■

Внешний поиск. Методы хеширования вполне подходят для внешнего поиска на запоминающих устройствах с прямым доступом типа дисков или барабанов. Для таких приложений, как и в п. 6.2.4, мы хотим минимизировать число обращений к файлу, и это двояко влияет на выбор алгоритмов:

- 1) Разумно потратить больше времени на вычисление хеш-функции, потому что расплата за плохое хеширование много больше стоимости дополнительного времени, нужного для аккуратной работы.
- 2) Записи обычно сгруппированы в блоки, чтобы за один раз извлекать из внешней памяти несколько записей.

Обычно файл делится на M блоков по b записей в каждом. Коллизии не вызывают затруднений, пока не появятся более b ключей с данным хеш-адресом. По-видимому, наилучшими будут следующие три подхода к разрешению коллизий.

А) Раздельные цепочки. Если в один и тот же блок попадает больше b записей, в конце первого блока можно вставить ссылку на "переполняющую" запись. Переполняющие записи хранятся в специальной области. Обычно нет смысла иметь блоки в области переполнения, так как возникает сравнительно мало переполнений; таким образом, добавляющиеся записи обычно связываются друг с другом, и на $(b+k)$ -ю запись списка тратится $1+k$ обращений к внешнему устройству. Часто полезно оставить некоторое пространство на каждом "цилиндре" дискового файла, чтобы большинство обращений происходило к одному и тому же цилиндру.

Хотя такой метод обработки переполнений кажется неэффективным, число переполнений статистически достаточно мало, и среднее время поиска оказывается очень хорошим. В табл. 2 и 3 приводится среднее число обращений как функция от коэффициента заполнения

$$\alpha = N/Mb, \quad (54)$$

где α фиксировано и $M, N \rightarrow \infty$. Как ни странно, но при $\alpha = 1$ асимптотическое число обращений в случае неудачного поиска растет с ростом b .

В) *Срастающиеся цепочки*. Вместо отведения отдельной области переполнения, можно приспособить алгоритм С для работы с внешними файлами. Еще не заполненные блоки можно связать в двухсвязевый список свободного пространства. Согласно

Таблица 2

Среднее число обращений при неудачном поиске с помощью отдельных цепочек										
Размер блока b	Коэффициент заполнения, α									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.197	1.249	1.307	1.3
2	1.0012	1.0088	1.0269	1.0581	1.1036	1.1638	1.238	1.327	1.428	1.5
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.252	1.369	1.509	1.6
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.253	1.394	1.571	1.7
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.249	1.410	1.620	1.7
10	1.0000	1.0000	1.0004	1.0041	1.0222	1.0773	1.201	1.426	1.773	2.0
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.113	1.367	1.898	2.3
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.018	1.182	1.920	2.7

этой схеме, каждый блок содержит счетчик числа свободных позиций; блок удаляется из списка, лишь если этот счетчик обращается в 0. Можно использовать "блуждающий указатель" для распределения переполнений (ср. с упр. 2.5-6), чтобы в различных списках по возможности фигурировали различные блоки переполнения. Вероятно, полезно иметь отдельные списки свободного пространства для блоков на каждом цилиндре дискового файла.

Этот метод еще не проанализирован, но он должен оказаться весьма полезным.

Таблица 3

Среднее число обращения при удачном поиске с помощью отдельных цепочек										
Размер блока b	Коэффициент заполнения, α									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.350	1.400	1.450	1.5
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.238	1.299	1.364	1.4
3	1.0010	1.0071	1.0216	1.0458	1.0806	1.1259	1.181	1.246	1.319	1.4
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.145	1.211	1.290	1.3
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.119	1.186	1.286	1.3
10	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.056	1.115	1.206	1.3
20	1.0000	1.0000	1.0000	1.0000	1.0005	1.0038	1.018	1.059	1.150	1.2
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.015	1.083	1.2

С) *Открытая адресация*. Можно обойтись и без ссылок, используя "открытый" метод. Для внешнего поиска линейное опробование, вероятно, лучше случайного, ибо величину s можно выбрать так, чтобы минимизировать скрытые задержки между последовательными обращениями. Построенная выше приближенная теоретическая модель линейного опробования может быть обобщена для учета влияния блоков; она показывает, что

Таблица 4

Среднее число обращений при удачном поиске с помощью линейного опробования										
Размер блока b	Коэффициент заполнения, α									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0556	1.1250	1.2143	1.3333	1.5000	1.7500	2.167	3.000	5.500	10.5
2	1.0062	1.0242	1.0553	1.1033	1.1767	1.2930	1.494	1.903	3.147	5.6
3	1.0009	1.0066	1.0201	1.0450	1.0872	1.1584	1.286	1.554	2.378	4.0
4	1.0001	1.0021	1.0085	1.0227	1.0497	1.0984	1.190	1.386	2.000	3.2
5	1.0000	1.0007	1.0039	1.0124	1.0307	1.0661	1.136	1.289	1.777	2.7
10	1.0000	1.0000	1.0001	1.0011	1.0047	1.0154	1.042	1.110	1.345	1.8
20	1.0000	1.0000	1.0000	1.0000	1.0003	1.0020	1.010	1.036	1.144	1.4
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.005	1.040	1.1

линейное опробование действительно удовлетворительно, пока таблица не слишком полна. Из табл. 4 видно, что при $\alpha = 90\%$ и $b = 50$ среднее число обращений для удачного поиска составляет лишь 1.04. Это даже *лучше*, чем 1.08 обращения, которое требуется в методе цепочек (А) с тем же размером блока!

Анализ методов (А) и (С) весьма интересен и с математической точки зрения; мы приведем здесь лишь результаты, поскольку детальному исследованию посвящены упр. 49 и 55. Формулы содержат две функции, тесно связанные с Q -функциями теоремы К:

$$R(\alpha, n) = \frac{n}{n+1} + \frac{n^2\alpha}{(n+1)(n+2)} + \frac{n^3\alpha^2}{(n+1)(n+2)(n+3)} + \dots \quad (55)$$

и

$$\begin{aligned} t_n(\alpha) &= e^{-n\alpha} \left(\frac{(\alpha n)^n}{(n+1)!} + 2 \frac{(\alpha n)^{n+1}}{(n+2)!} + 3 \frac{(\alpha n)^{n+2}}{(n+3)!} + \dots \right) = \\ &= \frac{e^{-n\alpha} n^n \alpha^n}{n!} (1 - (1-\alpha)R(\alpha, n)). \end{aligned} \quad (56)$$

С помощью этих функций можно выразить среднее число обращений, необходимых при неудачном поиске с помощью метода (А)

$$C'_N = 1 + \alpha b t_b(\alpha) + O\left(\frac{1}{M}\right) \quad (57)$$

($M, N \rightarrow \infty$), и соответствующее число при удачном поиске

$$C_N = 1 + ((e^{-b\alpha} b^b \alpha^b) / 2b!) (2 + (\alpha - 1)b + (\alpha^2 + (\alpha - 1)^2(b - 1)) \times R(\alpha, b)) + O(1/M). \quad (58)$$

Это согласуется с данными табл. 2 и 3.

Поскольку метод цепочек (А) требует отдельной области переполнения, следует оценить возможное количество переполнений. Их среднее число составляет $M(C'_N - 1) = N t_b(\alpha)$, так как $C'_N - 1$ есть число переполнений в любом данном списке. Поэтому табл. 2 можно пользоваться для нахождения размера этой области. При фиксированном α и $M \rightarrow \infty$ среднеквадратичное отклонение общего числа переполнений будет приблизительно пропорционально \sqrt{M} .

Асимптотические значения C'_N и C_N приведены в упр. 53, но эти приближения не очень хороши при малых b или больших α ; к счастью, ряд для $R(\alpha, n)$ сходится очень быстро даже при больших α , так что можно без большого труда оперировать точными формулами. Максимум C_N и C'_N достигается при $\alpha = 1$ и выражается при $b \rightarrow \infty$ как

$$\max C'_N = 1 + \frac{e^{-b} b^{b+1}}{b!} = \sqrt{\frac{b}{2\pi}} + 1 + O(b^{-1/2}), \quad (59)$$

$$\max C_N = 1 + \frac{e^{-b} b^b}{2b!} (R(b) + 1) = \frac{5}{4} + \sqrt{\frac{2}{9\pi b}} + O(b^{-1}) \quad (60)$$

согласно приближению Стирлинга и анализу функции $R(n) = R(1, n) - 1$, проведенному в п. 1.2.11.3.

Среднее число обращений при удачном поиске с помощью *линейного* опробования имеет удивительно простой вид:

$$C_N \approx 1 + t_b(\alpha) + t_{2b}(\alpha) + t_{3b}(\alpha) + \dots \quad (61)$$

Это соотношение можно пояснить следующим образом: среднее число обращений при поиске всех N ключей равно NC_N , а это есть $N + T_1 + T_2 + \dots$, где T_k обозначает среднее число ключей, поиск которых требует более k обращений. Теорема Р гласит, что, не изменяя C_N , ключи можно вставлять в любом порядке; отсюда следует, что T_k равно среднему числу переполняющих записей, которые имелись бы при использовании метода цепочек с M/k блоками размера kb , т. е. $T_k = N t_{kb}(\alpha)$, что и утверждалось. Дальнейшее уточнение формулы (61) проведено в упр. 55.

Отличное обсуждение практических соображений по построению внешних рассеянных таблиц дал Чарльз Олсон [Proc. ACM Nat'1 Conf., 24 (1969), 539–549]. Работа содержит несколько полезных примеров; в ней указывается, что количество переполняющих записей значительно увеличивается, если файл служит объектом частых вставок и удалений без перераспределения записей. Представлен анализ этой ситуации, выполненный совместно Олсоном и де Пейстером.

Сравнение методов. Итак, мы изучили много методов поиска; чем же нам руководствоваться при выборе наилучшего из них; для конкретного приложения? Трудно в нескольких словах описать все,

что нам хотелось бы учесть при выборе метода поиска, однако следующие соображения, пожалуй, наиболее

Picture: Рис. 44. Сравнение методов разрешения коллизий: предельные значения среднего числа проб при $M \rightarrow \infty$. (a)—неудачный поиск (коэффициент заполнения $\alpha = N/M$); (b)—удачный поиск. (L —линейное опробование=алгоритм L ; $U2$ —случайное опробование со вторичным окучиванием; U — равномерное хеширование≈алгоритм D ; B —алгоритм D с изменением Брента; C —метод цепочек со срастанием=алгоритм C ; S —раздельные цепочки; SO —раздельные цепочки с упорядоченными списками.)

важны, если мы заинтересованы в сокращении времени поиска и объема занимаемой памяти.

На рис. 44 показаны результаты проведенного анализа. Видно, что различные способы разрешения коллизий приводят к различному числу проб. Но это еще не все, так как с изменением метода меняется время пробы, что заметно отражается на времени работы (см. рис. 42). При линейном опробовании чаще, чем в других методах, происходит обращение к таблице, зато этот метод прост. Более того, нельзя сказать, что даже линейное опробование совсем уж никуда не годится: если таблица заполнена на 90%, алгоритм L требует в среднем менее 5.5 пробы для обнаружения случайного элемента. (Однако среднее число проб при вставке *нового* элемента с помощью алгоритма L равно 50.5, если таблица полна на 90%.)

Из рис. 44 видно, что методы цепочек весьма экономны с точки зрения числа проб, но потребность в дополнительном пространстве памяти для полей ссылок иногда (при небольшом размере записей) делает более привлекательной открытую адресацию. Например, если нужно сделать выбор между таблицей с цепочками на 500 элементов и таблицей с открытой адресацией на 1000 элементов, то последняя, очевидно, предпочтительнее, ибо она обеспечивает эффективный поиск среди 500 записей и способна вместить в два раза больше данных. С другой стороны, порой в силу размера записей или их формата пространство под поля ссылок достается фактически бесплатно (ср. с упр. 65).

Как соотносятся методы хеширования с другими стратегиями поиска, изученными в этой главе? Сравнивая их по скорости, можно утверждать, что методы хеширования лучше, если число записей велико, поскольку среднее время поиска для методов хеширования остается ограниченным при $N \rightarrow \infty$ в случае, когда таблица не становится слишком заполненной. Например, программе L нужно лишь 55 единиц времени для удачного поиска в таблице, заполненной на 90%; это быстрее самой быстрой из известных нам MIX-программ бинарного поиска (см. упр. 6.2.1-24), если N больше 600 или около того, и платить за это приходится лишь 11% пространства памяти. Более того, бинарный поиск годится лишь для фиксированных таблиц, в то время как рассеянные таблицы допускают эффективные процедуры вставки.

Мы можем также сравнить программу L с методами поиска по дереву, допускающими динамические вставки. Программа L при $\alpha = 0.9$ быстрее программы 6.2.2Г, если N превышает приблизительно 90, и быстрее программы 6.3D (упр. 6.3-9), если N превышает примерно 75.

Только один метод этой главы в случае удачного поиска весьма эффективен и фактически не дает перерасхода памяти, а именно алгоритм D с изменением Брента. Он позволяет поместить N записей в таблицу размера $M = N + 1$ и находить любую запись в среднем за $2\frac{1}{2}$ пробы. Не нужно дополнительного пространства для полей ссылок и т. п.; однако неудачный поиск будет крайне медленным—он потребует около $\frac{1}{2}N$ проб.

Таким образом, хеширование имеет свои преимущества. С другой стороны, поиск в рассеянных таблицах все же уступает изученным ранее методам по трем важным пунктам.

а) После неудачного поиска в рассеянной таблице мы знаем лишь то, что нужного ключа там нет. Методы поиска, основанные на сравнениях, всегда дают больше информации; они позволяют найти наибольший ключ $\leq K$ или наименьший ключ $\geq K$, что важно во многих приложениях (например, для интерполяции значений функции по хранящейся таблице). Эти же методы можно использовать и для нахождения всех ключей, лежащих *между* двумя заданными величинами K и K' . Далее, алгоритмы поиска по дереву § 6.2 позволяют легко распечатать содержимое таблицы в возрастающем порядке без специальной сортировки, а это иногда бывает нужно.

б) Часто довольно трудно распределить память для рассеянных таблиц; под хеш-таблицу нужно отвести определенную область памяти, а размер ее не всегда ясен. Если отвести слишком много памяти, то такая расточительность отразится на других списках или на других пользователях ЭВМ, но если отвести мало места, таблица переполнится! При переполнении рассеянной таблицы, вероятно, лучше всего "рехешировать" ее, т. е. отвести больше пространства и изменить хеш-функцию, а затем вставить записи в большую таблицу. Ф. Хопгуд [*Comp. Bulletin*, 11 (1968), 297–300] предложил

рехешировать таблицу, если коэффициент заполнения достигнет α , заменяя M на d_0M ; с помощью приведенного выше анализа и характеристик данных можно определить критическую точку, когда рехеширование становится дешевле дальнейшей работы с той же таблицей, и тем самым выбрать значения параметров α_0 и d_0 . (Заметим, что метод цепочек свободен от мучительных переполнений и не нуждается в рехешировании, однако при фиксированном M и растущем N время поиска становится пропорциональным N .) Напротив, алгоритмы поиска со вставкой по дереву не избыточны тягостными рехешированиями; деревья растут не больше, чем это необходимо. При работе с виртуальной памятью нужно, по всей вероятности, использовать поиск по дереву или цифровой поиск по дереву вместо создания больших рассеянных таблиц, вызывающих подкачку новой страницы почти при каждом хешировании ключа.

с) Наконец, при использовании методов хеширования нужно свято верить в теорию вероятностей, ибо они эффективны лишь в среднем, а наихудший случай просто ужасен! Как и в ситуации с датчиками случайных чисел, мы не можем быть полностью уверенными в том, что при применении к новому множеству данных хеш-функция будет работать удовлетворительно. Поэтому рассеянная память не всегда подходит для работы в реальном масштабе времени, например для управления движением транспорта, поскольку на карту поставлены человеческие жизни. Алгоритмы сбалансированного дерева (п. 6.2.3 и 6.2.4) гораздо безопаснее, ведь они имеют гарантированную верхнюю границу времени поиска.

История. По-видимому, впервые идею хеширования высказал Г. Лан. В январе 1953 г., составляя один из внутренних документов фирмы ИВМ, он предложил использовать метод цепочек; это явилось также одной из первых публикаций по связанным линейным спискам. Он указал на желательность использования для внешнего поиска блоков, содержащих более одного элемента. Вскоре после этого Э. Лин продвинул анализ Лана немного дальше и предложил метод обработки переполнений с помощью "вырождающихся адресов"; например, при переполнении первичного блока 2748 записи попадают во вторичный блок 274; если и он переполнился, используется третичный блок 27 и т. д. Предполагается, что есть 10000 первичных блоков, 1000 вторичных, 100 третичных и т. д. Первоначально предложенные Ланом хеш-функции были литерно-цифровыми; например, складывались по модулю 10 пары соседних цифр ключа, так что 31415926 сжималось до 4548.

Примерно в это же время идея хеширования появилась у другой группы сотрудников фирмы ИВМ: Джин Эмдел, Элейн Боэм, Н. Рочестера и Артура Сэмюэля, строивших автокодную программу для ИВМ 701. Для разрешения проблемы коллизий Эмдел высказала идею открытой адресации с линейным опробованием.

В открытой печати хеширование впервые было описано Арнольдом Думи [*Computers and Automation*, 5, 12 (December 1956), 6–9]. Именно он впервые упомянул идею деления на простое число и использования остатка в качестве хеш-адреса. В статье Думи упоминается метод цепочек, однако открытой адресации в ней нет. А. П. Ершов в 1957 г. независимо открыл линейную открытую адресацию [*ДАН СССР*, 118 (1958), 427–430]; он опубликовал эмпирические результаты о числе проб, справедливо предположив, что при $N/M < 2/3$ среднее число проб для удачного поиска < 2 .

Классическая статья У. Петерсона [*IBM J. Research & Development*, 1 (1957), 130–146] была первой важной работой по вопросу поиска в больших файлах. Петерсон определил открытую адресацию в общем случае, проанализировал характеристики равномерного хеширования и привел многочисленные статистические данные о поведении линейной открытой адресации при различных размерах блоков, подметив ухудшение характеристик, возникающее при удалении элементов. Другой всесторонний обзор предмета шестью годами позже опубликовал Вернер Буххольц [*IBM Systems J.*, 2 (1963), 86–111]; он наиболее основательно изучил хеш-функции.

К этому времени линейное опробование было единственным типом схемы открытой адресации, описанным в литературе, хотя несколькими авторами независимо была разработана другая схема, основанная на неоднократном случайном опробовании с помощью независимых хеш-функций (см. упр. 48). В течение нескольких следующих лет хеширование стало использоваться очень широко, но о нем едва ли было опубликовано что-нибудь новое. Затем Роберт Моррис написал оказавший большое влияние обзор предмета [*SACM*, 11 (1968), 38–44], в котором он ввел идею случайного опробования (со вторичным скучиванием). Статья Морриса завершила усилия, приведшие к созданию алгоритма D и его усовершенствований.

Интересно отметить, что слово "хеширование" ("hashing") в его нынешнем значении, вероятно, не появлялось в печати до конца 60-х годов, хотя к тому времени в разных частях света оно уже стало общепринятым жаргоном. По-видимому, впервые это слово было использовано в книге Хеллермэна [*Digital Computers System Principles* (New York, 1967), 152]. Из примерно 60 документов, относящихся к 1953–1967 гг., изученных автором при написании данного параграфа, слово "хеширование" встречается лишь однажды—в неопубликованном документе фирмы, составленном У. Петерсоном

(1961 г.). Как по волшебству, глагол "хешировать" (to hash) в середине 60-х годов стал стандартным термином для преобразования ключа, хотя использовать такое недостойное слово в печати никто не осмеливался до 1967 г.!

Упражнения

- [20] Предположим, что байты 1, 2, 3 ключа K содержат коды литер, меньшие 30. В каких пределах заключено содержимое $r11$, когда достигается инструкция $9H$ в табл. 1?
- [20] Найдите довольно известное английское слово, которое можно добавить в табл. 1 без изменения программы.
- [23] Объясните, почему программу в табл. 1 нельзя заменить на более простую, начинающуюся со следующих пяти инструкций:

```
LD1 K(1:1) или LD1N K(1:1)
LD2 K(2:2) или LD2N K(2:2)
INC1 a, 2
LD2 K(3:3)
J2Z 9F
```

ни при какой константе a , учитывая, что разные ключи не могут иметь одинаковые хеш-адреса.

- [M30] Сколько человек нужно пригласить на вечеринку, чтобы с вероятностью $> \frac{1}{2}$ нашлись *трое*, имеющие один и тот же день рождения?
- [15] Мистер Тупица писал компилятор с ФОРТРАНа, используя десятичную машину MIX, и перед ним встала задача создания таблицы символов для хранения имен переменных компилируемой программы. Длина этих имен ограничена десятью литерами. Он решил использовать рассеянную таблицу с $M = 100$ и быструю хеш-функцию $h(K) =$ крайний левый байт K . Хорошо ли это?
- [15] Разумно ли заменить первые две инструкции в (3) на LDA K; ENTX 0?
- [BM30] (Полиномиальное хеширование.) Цель настоящего упражнения — рассмотреть построение многочленов $P(x)$ типа (10), которые преобразуют n -битовые ключи в m -битовые адреса и при этом разные ключи, отличающиеся не более чем t битами, получают разные адреса. По заданным значениям $n, t \leq n$, и целому числу k , такому, что n делит $2^k - 1$, мы построим многочлен, степень от которого является функцией n, t и k . (Обычно, если это необходимо, n увеличивается, поэтому k можно выбрать достаточно малым.)

Пусть S — минимальное множество целых чисел, такое, что $\{1, 2, \dots, t\} \subseteq S$ и $(2j) \bmod n \in S$ для всех $j \in S$. Например, если $n = 15, k = 4, t = 6$, имеем $S = \{1, 2, 3, 4, 5, 6, 8, 10, 12, 9\}$. Определим теперь многочлен $P(x) = \prod_{j \in S} (x - \alpha^j)$, где α есть элемент порядка n конечного поля $GF(2^k)$, а коэффициенты $P(x)$ вычисляются в этом поле. Степень m многочлена $P(x)$ равна числу элементов в S . Если α^j — корень $P(x)$, то и α^{2j} — корень; отсюда следует, что коэффициенты $P(x)$ удовлетворяют соотношению $p_i^2 = p_i$, т. е. они равны 0 или 1.

Докажите, что если $R(x) = r_{n-1}x^{n-1} + \dots + r_1x + r_0$ — ненулевой многочлен по модулю 2 с не более чем t ненулевыми коэффициентами, то $R(x)$ не кратен $P(x)$ по модулю 2. [Отсюда следует, что соответствующая хеш-функция ведет себя обещанным образом.]

- [M34] (Теорема о трех длинах.) Пусть θ — иррациональное число, лежащее между 0 и 1, представление которого в виде правильной непрерывной дроби (обозначения п. 4.5.3) есть $\theta = /a_1, a_2, a_3, \dots/$. Пусть $q_0 = 0, p_0 = 1, q_1 = 1, p_1 = 0$ и $q_{k+1} = a_k q_k + q_{k-1}, p_{k+1} = a_k p_k + p_{k-1}$ при $k \geq 1$. Обозначим $x \bmod 1 = x - [x]$ через $\{x\}$, а $x - [x] + 1$ через $\{x\}^+$. Будем нумеровать отрезки в том порядке, как они получаются при последовательных вставках в интервал $[0, 1]$ точек $\{\theta\}, \{3\theta\}, \dots$, так что первый отрезок данной длины имеет номер 0, второй — номер 1 и т. д. Докажите, что все следующие утверждения верны. Левым концом интервала номер s длины $\{t\theta\}$, где $t = rq_k + q_{k-1}, 0 \leq r < a_k, k$ четно, а $0 \leq s < q_k$, служит точка $\{s\theta\}$, а правым — точка $\{(s+t)\theta\}^+$. Левым концом интервала номер s длины $1 - \{t\theta\}$, где $t = rq_k + q_{k-1}, 0 \leq r < a_k, k$ нечетно и $0 \leq s < q_k$, служит точка $\{(s+t)\theta\}$, а правым — точка $\{s\theta\}^+$. Любое натуральное число можно единственным образом представить в виде $n = rq_k + q_{k-1} + s$ при $k \geq 1, 1 \leq r \leq a_k, 0 \leq s < q_k$. В этих обозначениях перед вставкой точки $\{n\theta\}$ имеется n интервалов: первые s интервалов (с номерами $0, \dots, s-1$) длины $\{(-1)^k(rq_k + q_{k-1})\theta\}$; первые $n - q_k$ интервалов (с номерами $0, \dots, n - q_k - 1$) длины $\{(-1)^k q_k \theta\}$; последние $q_k - s$ интервалов (с номерами $s, \dots, q_k - 1$) длины $\{(-1)^k((r-1)q_k + q_{k-1})\theta\}$.

Операция вставки $\{n\theta\}$ устраняет интервал номер s последнего типа и преобразует его в интервал номер s первого типа и интервал номер $n - q_k$ второго типа.

- [M30] Теорема S утверждает, что при последовательных вставках точек $\{\theta\}, \{2\theta\}, \dots$ в интервал $[0, 1]$ каждая новая точка разбивает один из наибольших имеющихся интервалов. Если интервал $[a, c]$ разбит таким образом на две части $[a, b], [b, c]$, мы назовем это *плохим разбиением*, когда одна из частей более чем вдвое длиннее другой, т. е. когда $b - a > 2(c - b)$ или $c - b > 2(b - a)$.

Докажите, что если $\theta \neq \phi^{-1}$ или $\theta \neq \phi^{-2}$, то при некотором n точка $\{n\theta\}$ дает плохое разбиение; если же $\theta = \phi^{-1}$ или $\theta = \phi^{-2}$, то плохих разбиений не получается.

10. [M43] (Р. Грэхем.) Докажите или опровергните следующее предположение о $3d$ длинах: если $\theta, \alpha_1, \dots, \alpha_d$ — вещественные числа, причем $\alpha_1 = 0$, если n_1, \dots, n_d — натуральные числа и если точки $\{n\theta + \alpha_i\}$ вставляются в интервал $[0, 1]$ при $0 \leq n < n_i, 1 \leq i \leq d$, то $n_1 + \dots + n_d$ получающихся интервалов (возможно, пустых) имеют не более $3d$ различных длин.
11. [16] Обычно удачные поиски производятся чаще неудачных. Разумно ли строки 12–13 программы С поменять местами со строками 10–11?
- >12. [21] Покажите, что можно так переписать программу С, что во внутреннем цикле останется лишь одна инструкция условного перехода. Сравните времена работы модифицированной и первоначальной программ.
13. [24] (Сокращенные ключи.) Пусть $h(K)$ — хеш-функция, а $g(K)$ — такая функция от K , что, зная $h(K)$ и $g(K)$, можно найти K . Например, при хешировании делением можно положить $h(K) = K \bmod M$ и $g(K) = \lfloor K/M \rfloor$; в мультипликативном хешировании в качестве $h(K)$ можно взять старшие разряды $(AK/w) \bmod 1$, а в качестве $g(K)$ — остальные разряды.
Покажите, что при использовании метода цепочек без наложения в каждой записи вместо K достаточно хранить $g(K)$. (Это иногда экономит пространство, необходимое для полей ссылок.) Измените алгоритм С, чтобы он мог работать с такими сокращенными ключами, устранив наложение списков, но не используя вспомогательных ячеек памяти для "переполняющих" записей.
14. [24] (Э. Элькок.) Покажите, что большая рассеянная таблица может использовать общую с другими связанными списками память. Пусть каждое слово списочной памяти содержит двухбитовое поле TAG, имеющее следующий смысл:

TAG(P) = 0 обозначает слово в списке свободного пространства; LINK(P) указывает на следующее свободное слово.

TAG(P) = 1 обозначает используемое слово, не являющееся частью рассеянной таблицы; формат остальных полей этого слова произволен.

TAG(P) = 2 обозначает слово в рассеянной таблице; LINK(P) указывает на другое слово. Всякий раз, когда при обработке списка, не являющегося частью рассеянной таблицы, мы достигаем слова с TAG(P) = 2, нужно установить $P \leftarrow \text{LINK}(P)$ до получения слова с TAG(P) ≤ 1 . (Для эффективности можно изменить одну из предшествующих ссылок; тогда не придется снова и снова перескакивать через одни и те же элементы рассеянной таблицы.)

Покажите, как определить подходящие алгоритмы для вставки и выборки ключей из такой комбинированной таблицы, предполагая, что в слове с TAG(P) = 2 имеется еще одно поле ссылки AUX(P).

15. [16] Почему в алгоритмах L и D разумно фиксировать переполнение при $N = M - 1$, а не при $N = M$?
16. [10] В программе L говорится, что K не должно равняться 0. А вдруг на самом деле она работает даже при $K = 0$?
17. [15] Почему не положить просто $h_2(K) = h_1(K)$ в (25), если $h_1(K) \neq 0$?
- >18. [21] Действительно ли (31) предпочтительнее (30) в качестве замены строк 10–13 программы D? Дайте ответ, исходя из средних значений A, S_1 и C .
19. [40] Проверьте эмпирически воздействие ограничения области значений $h_2(K)$ в алгоритме D так, что: (а) $1 \leq h_2(K) \leq r$ при $r = 1, 2, 3, \dots, 10$; (б) $1 \leq h_2(K) \leq \rho M$ при $\rho = \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \dots, \frac{9}{10}$.
20. [M25] (Р. Крутар.) Измените алгоритм D следующим образом, устраняя хеш-функцию $h_2(K)$: в шаге D3 установить $c \leftarrow 0$, а в начале шага D4 установить $c \leftarrow c + 1$. Докажите, что, если $M = 2^m$, соответствующая последовательность проб $h_1(K), (h_1(K) - 1) \bmod M, \dots, \left(h_1(K) - \binom{M}{2}\right) \bmod M$ будет перестановкой множества $\{0, 1, \dots, M - 1\}$. Как этот метод, будучи запрограммированным для машины MIX, выглядит по сравнению с тремя программами, рассматриваемыми на рис. 42, если поведение аналогично случайному опробованию со вторичным окучиванием?
- >21. [20] Предположим, что мы хотели бы удалить запись из таблицы, построенной с помощью алгоритма D, помечая соответствующую позицию как удаленную. Следует ли при этом уменьшать значение переменной N , управляющей алгоритмом D?
22. [27] Докажите, что алгоритм R оставляет таблицу точно в таком же виде, как если бы KEY[i] не был сперва вставлен.
- >23. [23] Придумайте алгоритм, аналогичный алгоритму R, для удаления элементов из рассеянной таблицы цепочек, построенной с помощью алгоритма С.
24. [M20] Предположим, что множество всех возможных ключей состоит из MP элементов, причем ровно P ключей имеют данный хеш-адрес. (В реальных случаях P очень велико; например,

если ключами являются произвольные десятизначные числа, а $M = 10^3$, то $P = 10^7$.) Предположим, что $M \geq 7$ и $N = 7$. Пусть из множества всех возможных ключей случайным образом выбираются семь различных элементов. Выразите через M и P точную вероятность получения хеш-последовательности 1 2 6 2 1 6 1 (т. е. $h(K_1) = 1, h(K_2) = 2, \dots, h(K_7) = 1$).

25. [M19] Объясните, почему верна формула (39).
 26. [M20] Сколько хеш-последовательностей $a_1 a_2 \dots a_9$ при использовании линейного опробования дадут картину занятых ячеек (21)?
 27. [M27] Завершите доказательство теоремы К. [Указание: пусть

$$s(n, x, y) = \sum_{k \geq 0} \binom{n}{k} (x+k)^{k+1} (y-k)^{n-k-1} (y-n);$$

используйте биномиальную теорему Абея [формула (1.2.6-16)] для доказательства того, что $s(n, x, y) = x(x+y)^n + ns(n-1, x+1, y-1)$

28. [M30] В те давние времена, когда ЭВМ были гораздо медленнее нынешних, по миганию лампочек можно было судить, с какой скоростью работает алгоритм L. Когда таблица становилась почти полной, одни элементы обрабатывались очень быстро, а другие весьма медленно.

Эти наблюдения наводят на мысль о том, что, если используется линейное опробование, среднеквадратичное отклонение величины C'_N довольно велико. Найдите формулу, выражающую дисперсию через функции Q_r , определенные в теореме К, и оцените ее при $N = \alpha M$ и $M \rightarrow \infty$.

29. [M21] (Задача о стоянке.) На некоей улице с односторонним движением имеется m мест для стоянки, расположенных в ряд и перенумерованных от 1 до m . Муж везет в автомобиле свою дремлющую жену. Внезапно жена просыпается и требует немедленно остановиться. Он послушно останавливается на первом свободном месте, но если нет свободных мест, к которым можно подъехать, не поворачивая обратно (т. е. если жена проснулась, когда машина достигла k -го места для стоянки, но все "позиции" $k, k+1, \dots, m$ заняты), муж приносит свои извинения и едет дальше

Предположим, что это происходит с n различными машинами, причем j -я жена просыпается в момент, когда машина находится перед местом a_j . Сколько имеется таких последовательностей $a_1 \dots a_n$, при которых все машины удачно припаркуются, предполагая, что первоначально улица пуста и никто со стоянки не уезжает? Например, если $m = n = 9$ и $a_1 \dots a_9 = 3 1 4 1 5 9 2 6 5$, автомобили расположатся следующим образом:

Picture: Рис. стр. 647

[Указание: воспользуйтесь анализом линейного опробования.]

30. [M28] (Джон Риордан.) Покажите, что в задаче о стоянке из упр. 29 при $n = m$ все машины припаркуются тогда и только тогда, когда существует перестановка $p_1 p_2 \dots p_n$ множества $\{1, 2, \dots, n\}$, такая, что $a_j \leq p_j$ для всех j .
 31. [M40] Если в задаче о стоянке (упр. 29) $n = m$, число решений оказывается равным $(n+1)^{n-1}$, а из упр. 2.3.4.4-22 мы знаем, что это есть число свободных деревьев над $n+1$ различными вершинами! Найдите интересную зависимость между последовательностями остановок и деревьями.
 32. [M26] Докажите, что система уравнений (44) имеет единственное решение $(C_0, C_1, \dots, C_{M-1})$ при любых целых неотрицательных b_0, b_1, \dots, b_{M-1} , сумма которых меньше M . Постройте алгоритм для нахождения этого решения.
 33. [M23] Объясните, почему (51) всего лишь приближение к истинному среднему числу проб, производимых алгоритмом L. [Какие нестрогости были допущены при выводе (51)?]
 34. [M22] Цель этого упражнения — исследовать среднее, число проб в рассеянной таблице цепочек, когда списки остаются отдельными, как на рис 38. (а) Чему равна вероятность P_{Nk} того, что данный список имеет длину k , если M^N хеш-последовательностей (35) равновероятны? (б) Найдите производящую функцию $P_N(z) = \sum_{k \geq 0} P_{Nk} z^k$. (с) Выразите через эту производящую функцию среднее число проб при удачном и неудачном поиске. (Предполагается, что неудачный поиск в списке длины k требует $k + \delta_{k0}$ проб.)
 35. [M21] (Продолжение упр. 34.) Найдите среднее число проб при неудачном поиске, если отдельные списки упорядочены по значениям ключей.
 36. [M22] Найдите дисперсию числа проб для метода отдельных цепочек, если поиск был неудачным. (См. (18).)
 >37. [M29] Найдите дисперсию числа проб для метода отдельных цепочек, если поиск был удачным. (См. (19).)
 38. [M32] (Использование деревьев при хешировании.) Искусный программист мог бы попытаться использовать в методе цепочек бинарные деревья поиска вместо линейных списков, комбинируя

таким образом алгоритм 6.2.2Т с хешированием. Проанализируйте среднее число проб, нужных этому комбинированному алгоритму и в случае удачного, и в случае неудачного поиска. [Указание: ср. с формулой (5.2.1-11).]

39. [М30] Цель этого упражнения—проанализировать среднее число проб в алгоритме С (срастающиеся цепочки). Обозначим через $c(k_1, k_2, k_3, \dots)$ количество хеш-последовательностей (35), заставляющих алгоритм С образовать ровно k_1 списков длины 1, k_2 списков длины 2 и т. д.; $k_1 + 2k_2 + 3k_3 + \dots = N$. Найдите рекуррентное соотношение, определяющее эти числа, и используйте его при выводе простой формулы для суммы

$$S_N = \sum_{\substack{j \geq 1 \\ k_1 + k_2 + \dots = N}} k_j c(k_1, k_2, \dots).$$

Как связана величина S_N с числом проб во время неудачного поиска с использованием алгоритма С?

40. [М33] Найдите дисперсию числа проб, производимых в алгоритме С, если поиск был неудачным (см. (15).)
41. [М40] Проанализируйте, сколько раз в среднем R уменьшается на 1 при вставке $(N+1)$ -го элемента с помощью алгоритма С. (Обозначим это число через T_N .)
42. [М20] Выведите (17).
43. [М42] Проанализируйте модификацию алгоритма С, использующую таблицу размера $M' \geq M$. Для хеширования используются лишь первые M позиций, так что первые $M' - M$ свободных узлов, найденных в шаге С5, расположатся в дополнительной части таблицы. Какой выбор M при фиксированном M' , $1 \leq M \leq M'$, дает наилучшие характеристики?
44. [М43] (Случайное опробование с вторичным сгущиванием.) Цель этого упражнения—определить среднее число проб в схеме открытой адресации с последовательностью опробований

$$h(K), (h(K) + p_1) \bmod M, (h(K) + p_2) \bmod M, \dots, (h(K) + p_{M-1}) \bmod M,$$

где $p_1 p_2 \dots p_{M-1}$ —случайно-выбранная перестановка множества $\{1, 2, \dots, M-1\}$, зависящая от $h(K)$. Иными словами, у всех ключей с одинаковым значением $h(K)$ одна и та же последовательность опробований, а $(M-1)!$ возможных выборов M последовательностей проб с этим свойством равновероятны.

Можно точно смоделировать ситуацию, если над первоначально пустым линейным массивом размера m выполнить следующую процедуру. Делаем n раз такую операцию:

С вероятностью p займем крайнюю левую свободную позицию. В противном случае (т. е. с вероятностью $q = 1 - p$) выберем любую позицию таблицы, кроме крайней левой, причем все $m - 1$ позиций равновероятны. Если выбранная позиция свободна, займем ее; в противном случае выберем *любую* свободную позицию (включая самую левую) и займем ее, считая, что все свободные позиции равновероятны.

Например, если $m = 5$ и $n = 3$, то после выполнения описанной процедуры конфигурация (занято, занято, свободно, занято, свободно) получается с вероятностью

$$\frac{7}{192}qqq + \frac{1}{6}pqq + \frac{1}{6}qrp + \frac{11}{64}qqr + \frac{1}{3}ppq + \frac{1}{4}pqr + \frac{1}{4}qrr.$$

(Эта процедура соответствует случайному опробованию с вторичным сгущиванием, когда $p = 1/m$, ибо можно перенумеровать элементы таблицы так, что некоторая последовательность проб совпадет с $0, 1, 2, \dots$, а все остальные будут случайными.)

Найдите формулу для среднего числа занятых позиций в левом конце массива (две в приведенном примере). Определите также асимптотическое значение этой величины при $p = 1/m$, $n = \alpha(m+1)$, $m \rightarrow \infty$.

45. [М43] Решите аналог упр. 44 с *третичным сгущиванием*, когда последовательность проб начинается с $h_1(K), (h_1(K) + h_2(K)) \bmod M$; выбор дальнейших проб случаен и зависит лишь от $h_1(K)$ и $h_2(K)$ (т. е. $(M-2)!$ возможных выборов последовательностей проб с этим свойством считаются равновероятными). Является ли предлагаемая процедура асимптотически эквивалентной равномерному опробованию?
46. [М42] Найдите C'_N и C_N для метода открытой адресации, использующего последовательность проб

$$h(K), 0, 1, \dots, h(K-1), h(K)+1, \dots, M-1.$$

46. [M25] Какое среднее число проб потребуется для открытой адресации с последовательностью проб

$$h(K), h(K) - 1, h(K) + 1, h(K) - 2, h(K) + 2, \dots ?$$

Эта последовательность была однажды предложена из-за того, что все расстояния между последовательными пробами при четном M различны [Указание: небольшая хитрость существенно облегчит задачу.]

- >48. [M21] Дана бесконечная последовательность взаимно независимых случайных хеш-функций $h_n(K)$. Проанализируйте метод открытой адресации, в котором пробуются позиции $h_1(K), h_2(K), h_3(K), \dots$. Заметьте, что возможно повторное опробование одной и той же позиции, например если $h_1(K) = h_2(K)$, но это весьма маловероятно.
49. [BM24] Обобщив упр. 34 на случай b записей в блоке, определите среднее число проб (т. е. обращений к файлу) C_N и C'_N при использовании отдельных цепочек, предполагая, что неудачный поиск в списке из k элементов требует $\min(1, k - b + 1)$ проб. Вместо точных вероятностей P_{Nk} из упр. 34 воспользуйтесь приближением Пуассона

$$\begin{aligned} \binom{N}{k} \left(\frac{1}{M}\right)^k \left(1 - \frac{1}{M}\right)^{N-k} &= \frac{N}{M} \frac{N-1}{M} \dots \frac{N-k+1}{M} \left(1 - \frac{1}{M}\right)^N \left(1 - \frac{1}{M}\right)^{-k} \frac{1}{k!} = \\ &= e^{-\rho} \rho^k / k! + O(M^{-1}), \end{aligned}$$

справедливым при $N = \rho M, M \rightarrow \infty$.

50. [M20] Покажите, что в обозначениях (42) $Q_1(M, N) = M - (M - N - 1)Q_0(M, N)$. [Указание: докажите сначала, что $Q_1(M, N) = (N + 1)Q_0(M, N) - NQ_0(M, N - 1)$.]
51. [BM16] Выразите функцию $R(\alpha, n)$, определенную в (55), через функцию Q_0 , определенную в (42).
52. [BM20] Докажите, что $Q_0(M, N) = \int_0^\infty e^{-t} (1 + t/M)^N dt$.
53. [BM20] Докажите, что функцию $R(\alpha, n)$ можно выразить через неполную гамма-функцию, и используйте результат упр. 1.2.11.3-9 для нахождения асимптотического значения $R(\alpha, n)$ с точностью до $O(n^{-2})$ при $n \rightarrow \infty$ и фиксированном $\alpha < 1$.
54. [40] Исследуйте поведение алгоритма С, если его приспособить к внешнему поиску так, как это описано в тексте.
55. [BM43] Обобщите модель Шая—Спрута, обсуждавшуюся после теоремы Р, на случай M блоков размера b . Докажите, что $C(z) = Q(z)/(B(z) - z^b)$, где $Q(z)$ —многочлен степени b и $Q(1) = 0$. Покажите, что среднее число проб равно

$$1 + \frac{M}{N} C'(1) = 1 + \frac{1}{b} \left(\frac{1}{1 - q_1} + \dots + \frac{1}{1 - q_{b-1}} - \frac{1}{2} \frac{B''(1) - b(b-1)}{B'(1) - b} \right),$$

где $q_1 \dots q_{b-1}$ суть корни многочлена $Q(z)/(z - 1)$. Заменяв биномиальное распределение вероятностей $B(z)$ приближением Пуассона $P(z) = e^{b\alpha(z-1)}$, где $\alpha = N/Mb$, и используя лагранжеву формулу обращения (ср. с формулой (2.3.4.4-9) и упр. 4.7-8), приведите ответ к виду (61).

56. [M48] Обобщите теорему К, получив точный анализ линейного опробования при использовании блоков размера b .
57. [M47] Дает ли приписывание последовательностям проб одинаковых вероятностей минимальное значение C_N среди всех методов открытой адресации?
58. [M21] (С. Джонсон.) Найдите десять перестановок множества $\{0, 1, 2, 3, 4\}$, эквивалентных равномерному хешированию в смысле теоремы U.
59. [M25] Докажите, что если приписывание вероятностей перестановкам эквивалентно равномерному хешированию (в смысле теоремы U), то число перестановок с ненулевыми вероятностями превзойдет M^a при любом фиксированном показателе a , если M достаточно велико.
60. [M48] Будем говорить, что схема открытой адресации включает *единственное хеширование*, если используется ровно M последовательностей проб, начинающихся со всех возможных значений $h(K)$ и встречающихся с вероятностью $1/M$.

Что асимптотически лучше (в смысле минимальности C_N): наилучшая схема с единственным хешированием или случайные схемы, анализировавшиеся в упр. 44?

61. [M46] Является ли метод, анализировавшийся в упр. 46, наихудшей возможной схемой с единственным хешированием? (Ср. с упр. 60.)
62. [M49] Насколько хороша может быть схема с единственным хешированием, если шаги $p_1 p_2 \dots p_{M-1}$ (обозначения упр. 44) фиксированы и не зависят от K ? (Примерами таких методов служат линейное опробование и последовательности, рассматривавшиеся в упр. 20 и 47.)

63. [M25] Если в рассеянной таблице производятся случайные вставки и удаления, то сколько нужно в среднем независимых вставок, чтобы все M позиций побывали в занятом состоянии?
64. [M46] Проанализируйте ожидаемое поведение алгоритма R. Сколько раз в среднем будет выполняться шаг R4?
- >65. [20] (*Ключи переменной длины.*) Во многих приложениях рассеянных таблиц ключи могут состоять из произвольного числа литер. В таком случае нельзя просто запомнить ключ в таблице, как это делалось в программах данного параграфа. Придумайте хороший способ использования рассеянных таблиц для хранения ключей переменной длины, если работа ведется на машине MIX.

6.5. ВЫБОРКА ПО ВТОРИЧНЫМ КЛЮЧАМ

Мы завершили изучение поиска по "первичным ключам", т. е. по ключам, однозначно определяющим запись в файле. Но иногда необходимо вести поиск, основываясь не на первичных ключах, а на значениях других полей записи, которые часто называются "вторичными ключами" или "атрибутами" записи. Например, в регистрационном файле, содержащем информацию о студентах университета, может понадобиться найти всех студентов-второкурсников из Огайо, не специализирующихся по математике или статистике, или может понадобиться найти всех незамужних аспиранток, говорящих по-французски, и т. д.

Вообще предположим, что каждая запись имеет несколько атрибутов и мы хотим найти все записи с определенными значениями определенных атрибутов. Спецификация искомым записей называется *запросом*. Обычно допускается не более трех следующих типов запросов.

- Простой запрос*, когда определенному атрибуту задается конкретное значение, например СПЕЦИАЛЬНОСТЬ = МАТЕМАТИКА или МЕСТОЖИТЕЛЬСТВО . ШТАТ = ОГАЙО.
- Запрос по области значений*, когда для определенного атрибута задается конкретная область значений, например ЦЕНА < 18.00\$; или $21 \leq \text{ВОЗРАСТ} \leq 23$.
- Булев запрос* состоит из запросов двух первых типов, соединенных операциями И, ИЛИ, НЕ; например,

$$(\text{КУРС} = \text{ВТОРОКУРСНИК}) \text{ И } (\text{МЕСТОЖИТЕЛЬСТВО} . \text{ШТАТ} = \text{ОГАЙО}) \\ \text{И НЕ}((\text{СПЕЦИАЛЬНОСТЬ} = \text{МАТЕМАТИКА}) \text{ ИЛИ } (\text{СПЕЦИАЛЬНОСТЬ} = \text{СТАТИСТИКА})).$$

Задача разработки эффективных методов поиска достаточно трудна уже для этих трех типов запросов, поэтому запросы более сложных типов обычно не рассматриваются. Например, железнодорожная компания могла бы иметь файл, описывающий текущее состояние всех принадлежащих ей товарных вагонов. Запрос типа "найди все свободные вагоны-холодильники в пределах 500 миль от Сиэтла" в явном виде был бы недопустим, если бы "расстояние от Сиэтла" было не атрибутом каждой записи, а сложной функцией нескольких атрибутов. Использование же логических кванторов в дополнение к И, ИЛИ и НЕ ведет к дальнейшим усложнениям, степень которых ограничена лишь фантазией автора запроса. Например, имея файл бейсбольной статистики, мы могли бы запросить данные о самой длинной серии удачных ударов в вечерних играх. Эти запросы сложны, но на них все же можно ответить, выполнив один проход по должным образом организованному файлу. Есть и еще более трудные запросы; например, найти все пары записей, имеющих одинаковые значения пяти или более атрибутов (без определения того, какие именно атрибуты должны совпадать). Подобные запросы можно рассматривать как общие задачи программирования, выходящие за рамки данной работы, хотя часто их можно разбить на подзадачи рассматриваемых нами типов.

Прежде чем переходить к изучению различных методов выборки по вторичным ключам, уместно рассмотреть экономическую сторону вопроса. Хотя довольно обширная область приложений попадает в жесткие рамки трех типов запросов, сложные методы, которые мы будем изучать, далеко не всегда являются удовлетворительными; иную работу можно быстрее сделать вручную! ЭВМ увеличили скорость научных вычислений приблизительно в 10^7 – 10^8 раз; повышение же эффективности в деле управления информацией неизмеримо меньше. При операциях с большим количеством данных современные ЭВМ работают все еще с механическими (а не электронными) скоростями, поэтому, заменяя ручную систему на ЭВМ, мы не получаем резкого улучшения характеристик на единицу затрат. Не следует ожидать от ЭВМ слишком многого только потому, что они здорово решают некоторые задачи...

Люди покорили Эверест "потому, что он существует", и потому, что было создано оборудование, сделавшее восхождение возможным; точно так же, встретившись с горой информации, люди попытались использовать ЭВМ для ответов на самые трудные мыслимые и немислимые вопросы в оперативном режиме и реальном масштабе времени, не учитывая должным образом стоимости работы. Требуемые вычисления возможны, но слишком дороги для каждодневного использования.

Рассмотрим, например, следующий простой способ выборки по вторичным ключам: после "буферизации" ряда запросов можно произвести последовательный поиск во всем файле, выбирая все нужные записи. ("Буферизация" означает, что мы накапливаем запросы, прежде чем начать их обработку.) Этот метод вполне удовлетворителен, если файл не слишком велик, а на запросы не надо отвечать немедленно. Он годится даже для файлов на лентах и лишь время от времени требует внимания ЭВМ, что делает его очень экономичным в смысле стоимости оборудования. Более того, предлагаемый подход применим даже для обработки вычислительных запросов типа "расстояния от Сиэтла".

Другой способ облегчить выборку по вторичным ключам — поручить *человеку* часть работы, обеспечив его должным образом оформленными указателями информации. Часто такой подход оказывается наиболее разумным и экономичным (разумеется, при том условии, что после выхода в свет нового указателя старая бумага перерабатывается).

Однако описанные простые схемы не являются удовлетворительными, если важны быстрые ответы на запросы, а файлы очень велики. Такая ситуация имеет место, например, если файл представляет собой объект непрерывных запросов от ряда одновременных пользователей или если запросы порождаются не человеком, а машиной. Цель настоящего параграфа — понять, насколько хорошо можно производить выборку по вторичным ключам, используя обычные ЭВМ, при различных предположениях о структуре файла.

Было выдвинуто немало хороших идей для решения этой задачи, но (как читатель наверняка уже догадался) эти алгоритмы оказались неизмеримо хуже имеющихся алгоритмов выборки по первичным ключам. Вследствие большого разнообразия файлов и приложений мы не сможем дать исчерпывающего обсуждения всех имеющихся возможностей, как и не сможем проанализировать поведение каждого алгоритма в типичных условиях. В настоящем параграфе содержатся лишь основные из предлагавшихся подходов; дело читателя решать, какая комбинация методов больше всего подходит в каждом конкретном случае.

Инвертированные файлы. Первый важный класс методов выборки по вторичным ключам основан на идее "инвертированного файла". Этот термин не означает, что файл перевернут вверх дном, он означает, что записи и атрибуты поменялись ролями. Вместо перечисления атрибутов данной записи перечисляются записи, имеющие данное значение атрибута.

В повседневной жизни мы довольно часто (правда, под другими именами) сталкиваемся с инвертированными файлами. Например, инвертированным файлом, соответствующим русско-английскому словарю, является англо-русский словарь. Инвертированным файлом, соответствующим этой книге, являются указатели, помещенные на последних страницах. Бухгалтеры традиционно используют "двойную бухгалтерию", когда все деловые соглашения регистрируются как в счете кассы, так и в счете клиента, что позволяет легко контролировать наличную сумму денег и долг клиента.

Вообще инвертированный файл обычно не существует сам по себе, его нужно использовать вместе с первоначальным, неинвертированным файлом для ускорения поиска с помощью содержащейся в нем избыточной информации. Компоненты инвертированного файла, т. е. списки всех записей, имеющих данное значение некоторого атрибута, называются *инвертированными списками*.

Как и все списки вообще, инвертированные списки можно представлять в памяти ЭВМ различными способами, и для разных приложений подходят разные представления. Некоторые поля вторичных ключей могут иметь лишь два значения (например, атрибут "пол"), а соответствующие инвертированные списки очень длинные, но другие поля обычно имеют очень много значений с редкими повторениями (например, атрибут "номер телефона").

Представим себе, что мы хотим хранить информацию в телефонной книге таким образом, чтобы все элементы можно было выбирать либо по имени, либо по номеру телефона, либо по месту жительства. Одно из решений состоит в том, чтобы просто иметь три отдельных файла, ориентированных на каждый тип ключей. Другая идея заключается в объединении этих файлов, например, с помощью построения трех рассеянных таблиц, служащих в методе цепочек головными узлами списков. В этой последней схеме каждая запись должна быть элементом трех списков и должна поэтому содержать три поля ссылки; такой метод называется *Многосписочным* и обсуждается ниже. Еще одна возможность — объединить три файла в один суперфайл по аналогии с библиотечным каталогом, в котором карточки с фамилиями авторов, карточки с названиями книг и карточки с темами книг все вместе упорядочены по алфавиту.

Изучение формата, использованного в указателях к данной книге, наталкивает на другие идеи о представлении инвертированных списков. Если определенному значению атрибута соответствует около пяти элементов, можно организовать обычный последовательный список адресов записей (аналогично номерам страниц в указателе к книге), имеющих данное значение вторичного ключа. Если соответствующие записи располагаются последовательно, может быть полезным указание диапазона

(например, со страницы 200 по 214). Если известно, что записи файла довольно часто перемещаются, вместо адресов записей в инвертированном файле, пожалуй, лучше использовать первичные ключи; тогда при смене адресов записей не нужно будет производить никаких изменений. Например, при ссылках на Библию всегда указывается глава и стих, а указатели в некоторых книгах основаны не на нумерации страниц, а на номерах параграфов.

Однако предложенные идеи не очень подходят для случая атрибута с двумя возможными значениями, типа атрибута "ПОЛ". В такой ситуации нужен лишь один инвертированный список, ибо не мужчина есть женщина и наоборот. Если каждое значение соответствует примерно половине элементов файла, инвертированные списки будут ужасно длинными, однако на двоичной ЭВМ эту трудность можно разрешить весьма изящно, используя представление в виде цепочки битов, где каждый бит определяет значение атрибута определенной записи. Так, цепочка битов 01001011101... могла бы означать, что первая запись файла описывает мужчину, вторая — женщину, следующие две — мужчин и т. д. (Ср. также с обсуждением представления простых чисел в конце § 6.1.)

Приведенные методы удовлетворительны для обработки запросов по конкретным значениям атрибутов. Некоторое обобщение этих методов позволит обрабатывать запросы по области значений. При этом вместо хеширования нужно использовать схему поиска, основанную на сравнениях (§ 6.2).

Для булевых запросов типа "(СПЕЦИАЛЬНОСТЬ = МАТЕМАТИКА) И (МЕСТОЖИТЕЛЬСТВО.ШТАТ = ОГАЙО)" нужно пересечь два инвертированных списка. Это можно сделать несколькими способами; например, если оба списка упорядочены, один проход по каждому из них позволяет выявить все общие элементы. Или же можно взять *кратчайший* список и просматривать его элементы, проверяя значения других атрибутов; однако такой метод применим к операции И и не применим к операции ИЛИ. Кроме того, он не годится для внешних файлов, ибо ведет к большому числу обращений к записям, не удовлетворяющим запросу.

Аналогичные рассуждения показывают, что упомянутая выше Многосписочная организация неэффективна для булевых запросов, если файл внешний, поскольку она требует много ненужных обращений. Представим себе, например, что произойдет, если указатель к этой книге организовать Многосписочным образом. Каждый элемент указателя будет ссылаться лишь на последнюю из тех страниц, где располагается описываемый объект; на каждой странице для каждого объекта должна иметься ссылка на место его предыдущего появления. В таком случае придется перелистать много страниц, чтобы найти весь материал, относящийся к теме "[Анализ алгоритмов] и [(Внешняя сортировка) или (Внутренняя сортировка)]". С другой стороны, тот же запрос можно обработать, взглянув всего на две страницы обычного указателя и произведя несложные операции над инвертированными списками для нахождения небольшого подмножества страниц, удовлетворяющих запросу.

Если инвертированный список представлен в виде цепочки битов, не составляет большого труда выполнить булевы комбинации запросов, поскольку ЭВМ могут манипулировать цепочками битов со сравнительно высокой скоростью. В случае смешанных запросов, когда некоторые атрибуты представлены посредством последовательных списков номеров записей, а другие посредством цепочек битов, нетрудно преобразовать последовательные списки в цепочки битов и произвести над ними булевы операции.

В этом месте полезно привести количественный пример гипотетического приложения. Предположим, имеется 1000000 записей по 40 литер в каждой, и пусть файл хранится на дисках MIX-ТЕС (см. п. 5.4.9). Сам файл занимает два дисковых пакета; инвертированные списки займут, вероятно, несколько больше места. Каждая дорожка содержит 5000 литер = 30 000 битов, поэтому инвертированный список для некоторого атрибута займет не более 34 дорожек. (Максимальное число дорожек соответствует случаю, когда представление в виде цепочек битов наиболее короткое.) Предположим, имеется довольно сложный запрос, состоящий из булевой комбинации 10 инвертированных списков; в худшем случае нам придется пересечь 340 дорожек информации из инвертированного файла. Полное время чтения составит $340 \times 25 \text{ мс} = 8.5 \text{ с}$. Среднее время задержки равно приблизительно половине этого количества, но если искусно составить программу, задержку можно исключить. Если хранить первые дорожки цепочек на одном цилиндре, вторые на следующем и т. п., то время поиска можно оценить величиной $34 \times 26 \text{ мс} \approx 0.9 \text{ с}$ (или 1.8 с, если чтение производится с двух дисков). Наконец, если запросу удовлетворяет k записей, то потребуются $k \times (60 \text{ мс (поиск)} + 12.5 \text{ мс (задержка)} + 0.2 \text{ мс (чтение)})$ дополнительного времени, чтобы достать их для последующей обработки. Таким образом, оптимистическая оценка полного времени обработки этого довольно сложного запроса дает число $(10 + 0.073kt) \text{ с}$. Полученный результат можно сопоставить с 210 с, нужными на чтение всего файла с максимальной скоростью.

Рассмотренный пример показывает, что для дисковой памяти оптимизация по пространству тесно связана с оптимизацией по времени; время обработки инвертированных списков приблизительно равно времени, затрачиваемому на их поиск и чтение.

До сих пор в большей или меньшей степени предполагалось, что в процессе обработки запроса файл не растет и не сокращается; но что делать, если необходимы частые изменения? Во многих приложениях достаточно буферизовать ряд требуемых изменений и позаботиться о них в свободную минуту, когда не нужно отвечать на запросы. Если же изменения файла имеют высокий приоритет, напрашивается использование *B*-деревьев (п. 6.2.4). Все множество инвертированных списков можно было бы поместить в одно гигантское *B*-дерево, причем нетерминальные узлы должны содержать значения ключей, а листья—и ключи, и списки указателей на записи.

В предыдущем обсуждении мы умолчали еще об одном трудном вопросе—о задаче булевых комбинаций запросов по области значений. Предположим, например, что записи файла описывают города Северной Америки и что запрашиваются названия всех городов с координатами

$$(21.49^\circ < \text{ШИРОТА} \leq 37.41^\circ) \text{ и } (70.34^\circ \leq \text{ДОЛГОТА} \leq 75.72^\circ).$$

Скорее всего, ни одна структура данных по-настоящему не годится для подобных ”запросов по прямоугольной области значений”. (Взглянув на карту, мы видим, что многие города удовлетворяют ограничению на широту, многие—на долготу, но едва ли найдется город, удовлетворяющий обоим ограничениям.) Пожалуй, наилучший подход состоит в довольно грубом расчленении множества всех возможных значений ШИРОТЫ и ДОЛГОТЫ, чтобы на каждый атрибут приходилось лишь несколько классов (например, можно округлять с недостатком до числа, кратного 5°), и в последующем построении инвертированного списка для каждого комбинированного (ШИРОТА, ДОЛГОТА) класса. Это все равно, что иметь карты, где для каждого небольшого района отведена отдельная страница. При использовании интервалов в 5° к запросу имеют отношение восемь страниц, а именно $(20^\circ, 70^\circ)$, $(25^\circ, 70^\circ)$, \dots , $(35^\circ, 75^\circ)$. Запрос необходимо обработать для каждой из них, либо производя более тонкое расчленение внутри страницы, либо непосредственно обращаясь к записям в зависимости от числа записей, соответствующих этой странице. По существу, получается структура дерева с двумерными разветвлениями во внутренних узлах.

Другой подход к запросам по прямоугольной области, также основанный на структуре дерева, предложил Брюс Мак-Натт. Пусть, например, нужно обработать запрос типа ”Какой город ближе всего к точке x ?”, если дано значение x . Каждый узел предлагаемого Мак-Наттом бинарного дерева поиска соответствует городу y и ”контрольному радиусу” r ; левое поддерево этого узла соответствует всем городам z , позднее вставленным в дерево, причем расстояние от y до z должно быть $\leq r + \delta$; аналогично правое поддерево предназначено для расстояний $\geq r - \delta$. Здесь δ —данный допуск; города, отстоящие от y меньше, чем на $r + \delta$, и больше, чем на $r - \delta$, должны входить в оба поддерева. Поиск в таком ”почтовом дереве” позволяет выявить все города, отстоящие от данной точки менее, чем на δ . (См. рис. 45.)

В 1972 г. Мак-Натт и Эдвард Принг, основываясь на этой идее, провели несколько экспериментов. В качестве базы данных они использовали 231 наиболее населенный город континентальных Соединенных Штатов, взятый в случайном порядке. Контрольный радиус r уменьшался регулярным образом: при шаге влево r заменяли на $0.67r$, а при шаге вправо—на $0.57r$. Если же выбиралась вторая из двух последовательных правых ветвей, радиус оставался неизменным. В результате получилось, что при $\delta = 20$ миль в дереве было 610 узлов, а при $\delta = 35$ миль—1600 узлов. На рис. 45 изображены верхние уровни меньшего дерева. (В оставшихся уровнях Орlando (штат Флорида) появляется и ниже Джексонвила, и ниже Майами. Некоторые города встречаются довольно часто; так, 17 узлов предназначены для Броктона (штат Массачусетс)!)

Picture: Рис 45. Верхние уровни ”почтового дерева”. Чтобы найти все города, расположенные вблизи данной точки x , начнем с корня: если x не далее 1800 миль от Лас-Вегаса, идем влево, в противном случае— вправо; затем повторяем этот процесс, пока не достигнем конечного узла. Метод построения дерева гарантирует, что в процессе поиска мы достигнем всех городов, отстоящих от x не более чем на 20 миль.

Составные атрибуты. Можно скомбинировать два или более атрибута в один суператрибут. Например, атрибут ”(КУРС, СПЕЦИАЛЬНОСТЬ)” в университетском регистрационном файле можно создать, комбинируя поля КУРС и СПЕЦИАЛЬНОСТЬ. При таком подходе на запрос часто можно ответить с помощью объединения нескольких коротких списков, а не с помощью пересечения более длинных списков.

Идею комбинации атрибутов развил В. Лум [АСМ, 13 (1970), 660–665]. Он предложил упорядочивать инвертированные списки, соответствующие составным атрибутам, лексикографически

слева направо и изготовлять несколько копий, в которых отдельные атрибуты переставлены надлежащим образом. Предположим, например, что имеются три атрибута A, B, C; можно образовать три комбинированных атрибута

$$(A, B, C), (B, C, A), (C, A, B) \quad (1)$$

и для каждого из них построить упорядоченный инвертированный список. (Так, в первом списке записи упорядочены по значению поля A; все записи с одним и тем же значением A упорядочены по B, а затем и по C.) Подобная организация позволяет отвечать на запросы, состоящие из любой комбинации этих трех атрибутов; например, все записи, имеющие определенные значения A и C, расположены в третьем списке друг за другом.

Аналогично из четырех атрибутов A, B, C, D можно образовать шесть комбинированных атрибутов

$$\begin{aligned} (A, B, C, D), & (B, C, D, A), & (B, D, A, C), \\ (C, A, D, B), & (C, D, A, B), & (D, A, B, C), \end{aligned} \quad (2)$$

что позволяет отвечать на все комбинации простых запросов, в которых фиксированы, значения одного, двух, трех или четырех атрибутов. Существует общая процедура построения $\binom{n}{k}$ комбинированных атрибутов из n атрибутов, где $k \leq \frac{1}{2}n$; при этом все записи, имеющие определенные комбинации $\leq k$ или $\geq n - k$ значений атрибутов, расположатся последовательно в одном из инвертированных списков (см. упр. 1). Или же можно обойтись меньшим числом комбинаций, если некоторые атрибуты имеют ограниченное количество значений. Например, если атрибут D может принимать лишь два значения, то три комбинации

$$(D, A, B, C), (D, B, C, A), (D, C, A, B), \quad (3)$$

полученные из (1) внесением D в скобки, будут почти столь же хороши, как и (2), ибо для ответов на запросы, не зависящие от D, один из списков достаточно просмотреть всего в двух местах. Избыточность же информации уменьшилась вдвое.

Бинарные атрибуты. Полезно рассмотреть частный случай, когда все атрибуты могут принимать лишь два значения. По существу, мы имеем *полную противоположность* комбинированию атрибутов, так как можем представить любое значение в виде двоичного числа и рассматривать биты этого числа как отдельные атрибуты. В табл. 1 (см. McCall, Cook Book (New York: Random House (1963), Ch. 9) приведен типичный файл, содержащий атрибуты "да-нет"; в данном случае записи описывают избранные рецепты печенья, а атрибуты определяют, какие ингредиенты используются. Например, для приготовления миндальных вафель с ромом нужны масло, мука, молоко, орехи и сахарный песок. Если трактовать табл. 1 как матрицу из нулей и единиц, то транспонированная матрица дает инвертированный файл в виде цепочек битов.

Picture: Таблица 1 Файл с бинарными атрибутами

В правом столбце табл. 1 указаны особые, редко употребляемые продукты Их можно было бы закодировать более эффективно, не отводя каждому по столбцу; то же справедливо и для столбца "маисовый крахмал". Можно было бы эффективнее закодировать столбец "мука", ибо мука входит во все блюда, кроме меренги. Сейчас, однако, давайте оставим в стороне эти соображения и просто проигнорируем столбец "особые ингредиенты".

Определим *базовый запрос* в файле с бинарными атрибутами, как запрос на все записи, имеющие нули в определенных столбцах, единицы—в некоторых других и произвольные значения—в оставшихся столбцах. Используя "*" для обозначения произвольного значения, любой базовый запрос можно представить в виде последовательности нулей, единиц и звездочек. Представим себе человека, которому вдруг очень захотелось печенья с кокосовыми орехами, тогда как шоколад вызывает у него аллергию, анис он не терпит, а ванилина в доме нет. Он может сформулировать такой запрос:

$$0 * * 0 * * * 1 * * * * * * * * * * * * * * * * * 0 * * \quad (4)$$

Таблица 1 покажет, что на самом деле ему хочется палочек ароматных с черносливом.

Прежде чем рассмотреть общую задачу организации файла для обработки базовых запросов, важно изучить частный случай, когда запрос состоит только из единиц и звездочек. Такой запрос можно назвать *включающим*, ибо он запрашивает все записи, включающие в себя некоторое множество атрибутов (естественно считать, что единица обозначает имеющийся атрибут, а нуль—отсутствующий). Например, из рецептов табл. 1 только глазированные имбирные пряники и старинное сахарное печенье содержат и пекарный порошок, и пищевую соду.

Таблица 2

Коды отдельных специй		Пример кодирования наложением	
Абрикосы	1000010000	Лимонный сок	1000100000
Апельсины	0100000100	Мед	0000000011
Арахисовое масло	0000000101	Меласса	1001000000
Бананы	0000100010	Мускатный орех	0000010010
Ванилин	0000001001	Мускатный "цвет"	0000010100
Душистый перец	0000100001	Орехи	0000100100
Засахаренная вишня	0000101000	Перец	0010000100
Зерна аниса	0000011000	Смородиновое желе	0010000001
Изюм	0101000000	Финики	1000000100
Имбирь	0000110000	Цитрон	0100000010
Кардамон	1000000001	Чернослив	0010000010
Кокосовые орехи	0001010000	Чеснок	0001100000
Корица	1000000010	Шоколад	0010001000
Кофе	0001000100	Экстракт миндаля	0100000001
Лимонная цедра	0011000000	Яблочный соус	0010010000
Наложенные коды			
Бананово-овсяное печенье			1000111111
Ванильно-ореховое мороженое			0000101101
Воздушное печенье			0000001001
Глазированные имбирные пряники			1001110010
Драгоценное печенье			0010101101
Драже в шоколаде			0010101100
Ириски			0010111101
Малайский крендель			1011100101
Медовые пряники			1011110111
Меренги			1000101100
Миндальное печенье с кокосами			0001111101
Миндальные вафли с ромом			0000100100
Моравское печенье со специями			1001110011
Овсяные палочки с финиками			1000100100
Палочки ароматные с черносливом			0111110110
Печенье с арахисовым маслом			0010001101
Печенье с орехами			1101010110
Печенье с перцем			1111111111
Печенье с яблочным соусом			1111111111
Печенье со сливочным сыром			0010001001
Полукруглый пирог с начинкой			1011101101
Путаница			1000001011
Райские палочки			0001111101
Рождественское анисовое печенье			0011011000
Старинное сахарное печенье			0000011011
Фигурные песочные коржики			0000000000
Финский какор			0100100101
Шведский крендель			0000000000
Швейцарское рассыпное печенье с корицей			1000000010
Шоколадный хворост			0010101101
Шотландские овсяные коржики			0000001001
Юбочки			0000001001

Для некоторых приложений достаточно обеспечить лишь ответы на включающие запросы. Это справедливо, например, для ручных файловых систем "на картах с краевой перфорацией" или "картотек признаков". Система на картах с краевой перфорацией, соответствующая табл. 1, содержала бы по карте на каждый рецепт, где каждому ингредиенту соответствует вырез. (См. рис. 4б.) Для обработки включающего запроса карты складывают в аккуратную стопку и вводят спицы в позиции, соответствующие требуемым атрибутам. Затем спицы поднимают и карты, имеющие нужные

атрибуты, выпадают.

Picture: Рис. 46. Карта с краевой перфорацией.

”Картотека признаков” аналогично работает с инвертированным файлом. В этом случае имеется по карте на каждый атрибут. Для каждой записи на картах отводится определенная позиция, и если запись обладает некоторым атрибутом, то в соответствующем месте делается пробивка. Следовательно, обычная 80-колонная перфокарта может содержать информацию о $12 \times 80 = 960$ записях. Для обработки включающего запроса отбирают карты нужных атрибутов, кладут их вместе и просвечивают; лучи света пройдут через все позиции, соответствующие искомым записям. Эта операция аналогична обработке булевых запросов путем пересечения инвертированных цепочек битов.

Кодирование наложением. Причина нашего особого интереса к ручным картотекам кроется в том, что было придумано много остроумных способов экономии места на картах с краевой перфорацией; тот же подход применим к представлению файлов в памяти ЭВМ. Кодирование наложением напоминает хеширование; в действительности его изобрели за несколько лет до открытия хеширования. Идея состоит в том, чтобы отобразить атрибуты в случайные k -битовые коды в n -битовых полях и наложить коды имеющихся в записи атрибутов. Включающий запрос о некотором множестве атрибутов можно преобразовать во включающий запрос о соответствующих наложенных двоичных кодах. Этому запросу могут удовлетворять несколько лишних записей, но количество таких ”ложных выпадений” можно статистически учесть. [Ср. с Calvin N. Mooers, *Amer. Chem. Soc. Meeting*, **112** (September, 1947), 14E–15E; *American Documentation*, **2** (1951), 20–32.] В качестве примера кодирования наложением вновь рассмотрим табл. 1, но не всю, а только часть, касающуюся специй. В табл. 2 показано, что получится, если присвоить атрибутам специй случайные двухбитовые коды в десятибитовом поле и наложить их. Например, элемент ”шоколадный хворост” получается наложением кодов для шоколада, орехов и ванилина:

$$0010001000 \vee 0000100100 \vee 0000001001 = 0010101101.$$

Наложение кодов привносит также несколько лишних атрибутов, в данном случае это душистый перец, засахаренная вишня, смородиновое желе, кокосовое масло и перец, что приведет к ”ложным выпадениям” при ответах на некоторые запросы (кроме того, у нас появился новый рецепт—ложного печенья!) На самом деле кодирование наложением не очень хорошо работает для табл. 2, являющейся маленьким примером со многими атрибутами. Действительно, печенье с яблочным соусом будет выпадать при *каждом* запросе, ибо эта запись получена наложением семи кодов, покрывающих все десять позиций; еще хуже обстоит дело с печеньем с перцем, полученным наложением двенадцати кодов! С другой стороны, в некоторых случаях табл. 2 работает удивительно хорошо; например, в ответ на запрос о ванилине зря выпадет лишь печенье с перцем.

Более подходящим является пример, когда у нас есть, скажем, 32-битовое поле и множество из $\binom{32}{3} = 4960$ различных атрибутов. Каждая запись может иметь до шести атрибутов, и каждый атрибут кодируется спецификацией 3-х из 32-х битов. Если мы предположим, что каждая запись имеет шесть случайно выбранных атрибутов, то вероятности ложного выпадения при включающем запросе таковы:

по одному атрибуту:	0.07948358;	
по двум атрибутам:	0.00708659;	
по трем атрибутам:	0.00067094;	
по четырем атрибутам:	0.00006786;	(5)
по пяти атрибутам:	0.00000728;	
по шести атрибутам:	0.00000082.	

Таким образом, если есть M записей, не удовлетворяющих запросу по двум атрибутам, то приблизительно $0.007M$ имеют наложенные коды, покрывающие все биты этих двух атрибутов. (Вероятности подсчитаны в упр. 4.) Для представления инвертированного файла потребуется $32 \times$ (количество записей) битов, что составляет менее половины от числа битов, необходимых для описания собственно атрибутов в исходном файле.

Малькольм Ч. Харрисон [*SACM*, **14** (1971), 777–779] заметил, что кодирование наложением можно использовать для ускорения поиска текста. Предположим, что нам нужно определить все вхождения некоторой цепочки литер в большой текст без построения громоздкого дерева Патриции. Предположим, кроме того, что текст поделен на строки $c_1 c_2 \dots c_{50}$ по 50 литер в каждой. Харрисон предлагает кодировать каждую из 49 пар $c_1 c_2, c_2 c_3, \dots, c_{49} c_{50}$ путем отображения ее в число от 0 до, скажем, 127. Затем подсчитать ”сигнатуру” строки $c_1 c_2 \dots c_{50}$ —цепочку из 128 битов $b_0 b_1 \dots b_{127}$, где $b_i = 1$ тогда и только тогда, когда $h(c_j, c_{j+1}) = i$ при некотором j .

Если нам нужно найти все вхождения слова ИГОЛКА в большой текстовый файл СТОГСЕНА, мы просто отыскиваем все строки, сигнатуры которых содержат 1 в позициях $h(\text{ИГ})$, $h(\text{ГО})$, $h(\text{ОЛ})$, $h(\text{ЛК})$, $h(\text{КА})$. При случайной хеш-функции вероятность того, что некоторая случайная строка содержит в сигнатуре все эти единицы, равна всего лишь 0.00341 (ср. с упр. 4); следовательно, пересечение пяти инвертированных списков цепочек битов позволит быстро найти все строки, содержащие слово ИГОЛКА (хотя, вероятно, будет и несколько ложных попаданий).

В данном приложении гипотеза о случайности на самом деле несправедлива, поскольку обычные тексты несут в себе большое количество избыточной информации; распределение двухбуквенных комбинаций в словах весьма неравномерно. Например, полезно отбросить все пары $c_j c_{j+1}$, содержащие литеру "пробел", так как обычно пробелы встречаются гораздо чаще других литер.

Другое интересное применение кодирования наложением к задачам поиска нашел Бартон Блум [САСМ, 13 (1970), 422–426]; на самом деле его метод предназначен для выборки по *первичным* ключам, но нам удобно обсудить его в этом параграфе. Представим себе, что производится поиск в большой совокупности данных, причем, если он оказался неудачным, никаких действий выполнять не нужно. Например, мы хотим проверить чей-либо номер паспорта или сумму выплаченных налогов и т. п.; и если в файле нет соответствующей этому лицу записи, то дальнейшего исследования не требуется. Аналогично при применении ЭВМ для типографского набора текстов можно придумать простой алгоритм, который позволит правильно делать переносы для большинства слов, но будет неприменим к 50000 словам-исключениям. Если какое-либо слово не удастся найти в файле исключений, можно использовать этот простой алгоритм.

В подобной ситуации можно хранить во внутренней памяти таблицу битов, так что в большинстве случаев отсутствие ключа будет опознано без обращений к внешней памяти. Вот как это делается. Обозначим внутреннюю таблицу битов через $b_0 b_1 \dots b_{M-1}$, где M весьма велико. Для каждого ключа K_j файла вычислим k независимых хеш-функций $h_1(K_j), \dots, h_k(K_j)$ и установим соответствующие k битов равными 1. (Эти k значений не обязаны быть различными.) Таким образом, $b_i = 1$ тогда и только тогда, когда $h_l(K_j) = i$ при некоторых j и l . Теперь для того, чтобы определить, содержится ли аргумент поиска K во внешнем файле, нужно сначала проверить, выполняется ли соотношение $b_{h_l(K)} = 1$ при $1 \leq l \leq k$: если нет, то незачем обращаться к внешней памяти, если же да, то при подходящем выборе K и M обычными методами поиска нам скорее всего удастся найти K . Вероятность ложного выпадения для файла из N записей приближенно равна $(1 - e^{-kN/M})^k$. По существу, в методе Блума весь файл рассматривается как одна запись, первичные ключи трактуются как имеющиеся атрибуты, а кодирование наложением производится в огромном M -битовом поле.

Еще один вариант кодирования наложением в своей докторской диссертации разработал Ричард Густафсон (Univ. South Carolina, 1969). Предположим, имеется N записей и каждая из них содержит 6 из 10000 возможных атрибутов. Например, записи могли бы описывать технические статьи, а атрибуты—имеющиеся в них ключевые слова. Пусть h —хеш-функция, отображающая каждый атрибут в число между 0 и 15. Если запись обладает атрибутами a_1, a_2, \dots, a_6 , то по методу Густафсона она отображается в 16-битовое число $b_0 b_1 \dots b_{15}$, где $b_i = 1$ тогда и только тогда, когда $h(a_j) = i$ при некотором j ; далее, если лишь k битов стали единичными, $k < 6$, то другие $6 - k$ единиц добавляются неким случайным образом (не обязательно зависящим от самих записей). Имеется $\binom{16}{9} = 8008$ 16-битовых кодов, содержащих ровно шесть единиц; при известной доле везения приблизительно $N/8008$ записей отобразятся в каждое значение. Можно хранить 8008 списков записей, с помощью подходящей функции вычисляя адрес, соответствующий коду $b_0 b_1 \dots b_{15}$. В самом деле, если единицы расположены в позициях $0 \leq p_1 < p_2 < \dots < p_6$, функция

$$\binom{p_1}{1} + \binom{p_2}{2} + \dots + \binom{p_6}{6}$$

преобразует каждую цепочку $b_0 b_1 \dots b_{15}$ в число между 0 и 8007, причем разные цепочки порождают разные адреса. (См. упр. 1.2.6-56 и 2.2.6-7.)

Если теперь мы хотим найти все записи, имеющие три определенных атрибута A_1, A_2, A_3 , то следует вычислить $h(A_1)$, $h(A_2)$ и $h(A_3)$; если эти значения различны, нам придется просмотреть записи, хранящиеся в $\binom{13}{3} = 286$ списках, коды которых $b_0 b_1 \dots b_{15}$ содержат единицы в позициях $h(A_1)$, $h(A_2)$ и $h(A_3)$. Иными словами, исследованию подвергнутся лишь $286 \times 100/8008 \approx 3.5\%$ записей.

Комбинаторное хеширование. Основная идея только что описанного метода Густафсона состоит в том, чтобы найти такой способ отображения записей в адреса памяти, чтобы лишь небольшое количество адресов было связано с определенным запросом. Однако этот метод применим лишь к включающим запросам, когда отдельные записи обладают малым числом атрибутов. Другой тип отображений, предназначенных для обработки произвольных базовых запросов типа (4), состоящих из нулей, единиц и звездочек, открыл в 1971 г. Рональд Райвест.

Предположим сначала, что имеется 1000000 записей и каждая из них содержит 10 вторичных ключей, причем каждый вторичный ключ может принимать довольно много различных значений. Запись со вторичными ключами $(K_1, K_2, \dots, K_{10})$ можно отобразить в 20-битовое число

$$h(K_1)h(K_2) \dots h(K_{10}), \quad (6)$$

где h —хеш-функция, преобразующая вторичный ключ в 2-битовое число, а (6) образовано последовательным выписыванием этих десяти пар битов. При такой схеме 1000000 записей отображаются в множество из $2^{20} = 1048576$ значений; все отображение можно рассматривать как хеш-функцию с $M = 2^{20}$. Для разрешения коллизий воспользуемся методом цепочек. Если мы хотим выбрать все записи, имеющие определенные значения пяти вторичных ключей, нам придется просмотреть лишь 2^{10} списков [соответствующих пяти неспецифицированным парам в (6)], т. е. в среднем нужно проверить около $1000 = \sqrt{N}$ записей. [Аналогичный подход предложил Арисава, *J. Inf. Proc. Soc. Japan*, 12 (1971), 163–167.]

Райвест развил эту идею дальше, так что во многих случаях мы имеем следующую ситуацию. Предположим, существует $N \approx 2^n$ записей и каждая из них содержит m вторичных ключей. Записи отображаются в n -битовые хеш-адреса таким образом, что запрос, не специфицирующий значения k ключей, затрагивает примерно $N^{k/m}$ адресов. Все остальные обсуждавшиеся в этом параграфе методы (кроме метода Густафсона) требуют для выборки информации порядка N шагов (правда, коэффициент пропорциональности невелик); при достаточно большом N метод Райвеста окажется более быстрым, к тому же он не требует инвертирования файлов.

Но, прежде чем применить такой подход, нужно определить подходящее отображение. Рассмотрим пример с небольшими значениями параметров, когда $m = 4$ и $n = 3$, а все вторичные ключи могут принимать лишь два значения; отобразим 4-битовые записи в восемь адресов следующим образом:

$$\begin{array}{rcll} * & 0 & 0 & 0 \rightarrow 1 & 0 & 1 & * & 0 \rightarrow 5 \\ * & 1 & 1 & 1 \rightarrow 2 & 1 & 0 & * & 1 \rightarrow 6 \\ 0 & * & 0 & 1 \rightarrow 3 & 0 & 0 & 1 & * \rightarrow 7 \\ 1 & * & 1 & 0 \rightarrow 4 & 1 & 1 & 0 & * \rightarrow 8 \end{array} \quad (7)$$

Исследование этой таблицы показывает, что все записи, соответствующие запросу 0^{**} , отображаются в пять адресов 1, 2, 3, 5 и 7; и вообще *любой* базовый запрос с тремя звездочками затрагивает ровно пять адресов. Базовый запрос с двумя звездочками затрагивает три адреса, и, наконец, базовый запрос с одной звездочкой затрагивает либо один, либо два, а в среднем $(8 \times 1 + 24 \times 2)/32 = 1.75$ адреса. Таким образом, имеем

Количество неспецифицированных битов в запросе	Количество затрагиваемых адресов
4	$8 = 8^{4/4}$
3	$5 \approx 8^{3/4}$
2	$3 \approx 8^{2/4}$
1	$1.75 \approx 8^{1/4}$
0	$1 = 8^{0/4}$

(8)

Разумеется, это лишь маленький пример; таблицы подобного размера проще обрабатывать без всяких ухищрений. Но он полезен и для нетривиальных приложений, когда $m = 4r$ и $n = 3r$, ведь $4r$ -битовые записи можно отобразить в $2^{3r} \approx N$ адресов, разделив вторичные ключи на r групп по четыре бита в каждой и применив (7) к каждой группе. Получающееся отображение обладает требуемым свойством: *запрос, оставляющий k из m битов неспецифицированными, затронет примерно $N^{k/m}$ адресов.* (См. упр. 6.)

Некоторые другие отображения, найденные Райвестом, приводятся в упр. 7; их можно использовать вместе с (7) для получения функций комбинаторного хеширования в случае $1 \leq m/n \leq 2$. На самом деле можно было бы использовать блоки размера b и положить $N \approx 2^{nb}$; мы взяли $b = 1$ для простоты изложения.

***Сбалансированные файловые системы.** Другой комбинаторный подход к задаче выборки информации, основанный на "системах сбалансированных неполных групп", явился предметом многочисленных исследований. Хотя он очень интересен с математической точки зрения, своей практической ценности этот метод, к сожалению, еще не проявил. Чтобы дать читателю возможность почувствовать изящество результатов, мы все же изложим здесь краткое введение в теорию, не теряя надежды, что кто-нибудь найдет ей практическое применение.

Штейнеровской системой троек называется такое распределение v объектов в неупорядоченные тройки, когда каждая пара объектов встречается ровно в одной тройке. Например, если $v = 7$, имеется, по существу, одна штейнеровская система троек, а именно

Тройка	Содержащиеся в ней пары
{1, 2, 4}	{1, 2}, {1, 4}, {2, 4}
{2, 3, 5}	{2, 3}, {2, 5}, {3, 6}
{3, 4, 6}	{3, 4}, {3, 6}, {4, 6}
{4, 5, 0}	{0, 4}, {0, 5}, {4, 5}
{5, 6, 1}	{1, 5}, {1, 6}, {5, 6}
{6, 0, 2}	{0, 2}, {0, 6}, {2, 6}
{0, 1, 3}	{0, 1}, {0, 3}, {1, 8}

Поскольку существует $\frac{1}{2}v(v-1)$ пар объектов, а в тройке — три пары, общее число троек равно $\frac{1}{6}v(v-1)$; поскольку каждый объект может составлять пару ровно с $v-1$ другими объектами, он должен присутствовать ровно в $\frac{1}{2}(v-1)$ тройках. Из этих соображений следует, что система Штейнера существует лишь тогда, когда числа $\frac{1}{6}v(v-1)$ и $\frac{1}{2}(v-1)$ целые. Значит, v должно быть нечетным и при делении на 3 не давать в остатке 2, т. е.

$$v \bmod 6 = 1 \text{ или } 3. \quad (10)$$

Обратно, в 1847 г. Киркман доказал, что, если $v \geq 1$ и выполнено условие (10), штейнеровская система троек действительно существует. Его интересная конструкция приведена в упр. 10.

Штейнеровские системы троек можно использовать для уменьшения избыточности в комбинированных инвертированных файлах. Рассмотрим вновь файл рецептов печений (табл. 1) и преобразуем правый столбец в 31-й атрибут, который равен 1, если требуется специальный ингредиент, и 0 в противном случае. Предположим, мы хотим отвечать на все включающие запросы о парах атрибутов, например на запрос "В каких рецептах используются и кокосовые орехи, и изюм?" Можно было бы построить инвертированный список для каждого из $\binom{31}{2} = 465$ возможных запросов. Однако эти списки займут много места, поскольку, например, печенье с перцем будет содержаться в $\binom{17}{2} = 136$ из них, а запись, обладающая всеми атрибутами, войдет в каждый список! Использование системы троек Штейнера несколько улучшает ситуацию. Для 31 объекта существует система Штейнера из 155 троек, причем каждая пара объектов входит ровно в одну тройку. Каждой тройке $\{a, b, c\}$ можно сопоставить четыре списка: один список — для записей, обладающих атрибутами a, b, \bar{c} (т. е. не c); другой — для a, \bar{b}, c ; третий — для \bar{a}, b, c ; четвертый — для a, b, c . Такая конструкция гарантирует, что никакая запись не попадет более чем в 155 инвертированных списков, и экономит пространство, если запись имеет три атрибута, соответствующих тройке системы.

Система троек является частным случаем систем групп, состоящих из трех или более объектов. Например, 31 объект можно так распределить по неупорядоченным шестеркам, что каждая пара объектов попадет ровно в одну шестерку:

{1, 5, 17, 22, 23, 25}	{7, 11, 23, 28, 29, 0}	{13, 17, 26, 3, 4, 6}	{20, 24, 5, 10, 11, 13}	{26, 30, 11, 16, 17, 19}
{2, 6, 18, 23, 24, 26}	{8, 12, 24, 29, 30, 1}	{14, 18, 30, 4, 5, 7}	{21, 25, 6, 11, 12, 14}	{27, 0, 12, 17, 18, 20}
{3, 7, 19, 24, 25, 27}	{9, 13, 25, 30, 0, 2}	{15, 19, 0, 5, 6, 8}	{22, 26, 7, 12, 13, 15}	{28, 1, 13, 18, 19, 21}
{4, 8, 20, 25, 26, 28}	{10, 14, 26, 0, 1, 3}	{16, 20, 1, 6, 7, 9}	{23, 27, 8, 13, 14, 16}	{29, 2, 14, 19, 20, 22}
{5, 9, 21, 26, 27, 29}	{11, 15, 27, 1, 2, 4}	{17, 21, 2, 7, 8, 10}	{24, 28, 9, 14, 15, 17}	{30, 3, 15, 20, 21, 23}
{6, 10, 22, 27, 28, 30}	{12, 16, 28, 2, 3, 5}	{18, 22, 3, 8, 9, 11}	{25, 29, 10, 15, 16, 18}	{0, 4, 16, 21, 22, 24}
		{19, 23, 4, 9, 10, 12}		

[Эта система получается из первой группы путем сложения по модулю 31. Для проверки того, что она обладает требуемым свойством, заметим, что 30 значений $(a_i - a_j) \bmod 31, i \neq j$, различны, если $(a_1, a_2, \dots, a_6) = (1, 5, 17, 22, 23, 25)$. Чтобы найти шестерку, содержащую данную пару (x, y) , выберем i и j так, что $a_i - a_j \equiv x - y \pmod{31}$; если $k = (x - a_i) \bmod 31$, то $(a_i + k) \bmod 31 = x$ и $(a_j + k) \bmod 31 = y$.]

Систему (11) можно использовать для хранения инвертированных списков, причем ни одна запись не появится более 31 раза. Каждой шестерке $\{a, b, c, d, e, f\}$ сопоставлено 57 списков, предназначенных для записей, имеющих два или более атрибута a, b, c, d, e или f , т. е. имеющих атрибуты $(a, b, \bar{c}, \bar{d}, \bar{e}, \bar{f}), (a, \bar{b}, c, \bar{d}, \bar{e}, f), \dots, (a, b, c, d, e, f)$; ответом на любой включающий запрос по двум атрибутам является объединение без пересечений 16 подходящих списков подходящей шестерки. Печенье с перцем войдет в 29 из 31 группы этой системы, поскольку названное блюдо имеет два из шести атрибутов во всех шестерках, кроме $\{19, 23, 4, 9, 10, 12\}$ и $\{13, 17, 29, 3, 4, 6\}$ (если мы занумеруем столбцы числами от 0 до 30).

Аналогичная идея применима для уменьшения избыточности в составных инвертированных списках, когда мы хотим отвечать на базовые, а не на включающие запросы. Предположим, например, что записи содержат пять вторичных ключей K_1, K_2, K_3, K_4, K_5 , каждый из которых может принимать одно из четырех значений 0, 1, 2 или 3. Чтобы ответить на запрос о записях, для которых $K_i = a$ и $K_j = b$ при данных a, b и $i \neq j$, мы могли бы образовать инвертированные списки для всех 16 подобных запросов; в таком случае каждая запись присутствовала бы в десяти инвертированных списках. Альтернативой является использование следующей системы упорядоченных пятерок, основанной на комбинаторной схеме, которая называется "взаимно ортогональные латинские квадраты":

(0, 0, 0, 0, 0)	(1, 0, 1, 2, 3)	(2, 0, 2, 3, 1)	(3, 0, 3, 1, 2)
(0, 1, 1, 1, 1)	(1, 1, 0, 3, 2)	(2, 1, 3, 2, 0)	(3, 1, 2, 0, 3)
(0, 2, 2, 2, 2)	(1, 2, 3, 0, 1)	(2, 2, 0, 1, 3)	(3, 2, 1, 3, 0)
(0, 3, 3, 3, 3)	(1, 3, 2, 1, 0)	(2, 3, 1, 0, 2)	(3, 3, 0, 2, 1)

Если мы теперь посмотрим на любые две из пяти компонент, то увидим, что все 16 возможных упорядоченных пар значений встречаются в этих позициях ровно один раз. Группе (a, b, c, d, e) этой системы можно сопоставить записи, удовлетворяющие по крайней мере двум из условий $K_1 = a, K_2 = b, \dots, K_5 = e$. Таким образом, каждой из 16 групп будут сопоставлены 376 из 1024 возможных записей, поэтому средняя избыточность уменьшилась с 0 до $16 \times 376/1024 = 5\frac{7}{8}$.

Теория систем таких групп и латинских квадратов детально разработана в "книге Marshall Hall, Jr., *Combinatorial Theory* (Waltham; Mass.: Blaisdell, 1967). Несмотря на всю красоту этого круга идей, в задачах выборки информации их удалось использовать лишь для уменьшения избыточности при построении составных инвертированных списков; Дэвид Чжоу [*Information and Control*, 15 (1969), 377–396] заметил, что того же результата можно достичь и без комбинаторных систем.

Краткая история и библиография. Первая опубликованная статья о методах выборки по вторичным ключам написана Л. Джонсоном [*SACM*, 4 (1961), 218–222]. Многосписочную систему примерно в одно и то же время независимо разработали Ной Прайвз, Х. Грэй, У. Ландауэр, Д. Лефкович и С. Литвин; ср. с [*IEEE Trans. on Communication and Electronics*, 68 (1963), 488–492]. Другая довольно ранняя публикация, оказавшая влияние на последующие работы, принадлежит Д. Р. Дэвису и Э. Д. Лину [*SACM*, 8 (1965), 243–246].

В обширной литературе, появившейся с тех пор по этому вопросу, основное внимание уделялось взаимодействию с пользователями и языкам программирования, но этих тем мы в нашей книге не касаемся. В дополнение к уже перечисленным статьям можно порекомендовать следующие опубликованные работы, оказавшие автору наибольшую помощь при написании этого параграфа: Jack Minker, Jerome Sable, *Ann. Rev. of Information Science and Technology*, 2 (1967), 123–160; Robert E. Bleier, *Proc. ACM Nat. Conf.*, 22 (1967), 41–49; Jerome A. Feldman, Paul D. Rovner, *SACM*, 12 (1969), 439–449; Burton H. Bloom, *Proc. ACM Nat. Conf.*, 24 (1969), 83–95; H. S. Heaps, L. H. Thiel, *Information Storage and Retrieval*, 6 (1970), 137–153; Vincent Y. Lum, Huei Ling, *Proc. ACM Nat. Conf.*, 26 (1971), 349–356. Хороший обзор ручных карточек сделан в гл. 5 книги Бурна *Methods of Information Handling* (New York: Wiley, 1963). "Сбалансированные файловые системы" впервые разработали Абрахам, С. Гхош и Д. Рой-Чоудхури в 1965 г.; пожалуй, лучшее резюме этой работы и последующего развития метода дали Р. Бозе и Гэри Коч [*SIAM, J. Appl. Math.*, 17 (1969), 1203–1214].

В настоящем параграфе мы обсудили довольно много интересных идей, полезных в очень разных ситуациях. Поскольку многие из этих методов появились незадолго до написания данного параграфа, то новые успехи, вероятно, не заставят себя ждать.

Интересно отметить, что человеческий мозг выполняет выборку по вторичным ключам гораздо лучше, чем ЭВМ; в самом деле, мы довольно легко распознаем лица или мелодии и т. п. по отрывочной информации, в то время как машины, пожалуй, вообще не в состоянии сделать это. И нет ничего невозможного в том, что однажды будет открыт принципиально новый подход к конструированию машин, который раз и навсегда решит задачу выборки по вторичным ключам, сделав тем самым весь этот параграф ненужным.

Упражнения

- >1. [M27] Пусть $0 \leq k \leq \frac{1}{2}n$. Докажите, что следующее построение дает $\binom{n}{k}$ перестановок чисел $\{1, 2, \dots, n\}$, таких, что каждое t -элементное подмножество множества $\{1, 2, \dots, n\}$ при $t \leq k$ или $t \geq n - k$ встречается по крайней мере один раз в качестве первых t элементов перестановки. Рассмотрим путь на плоскости из точки $(0, 0)$ в (n, r) , где $r \geq n - 2k$, в котором i -й шаг производится из точки $(i - 1, j)$ в $(i, j + 1)$ или в $(i, j - 1)$; последняя возможность имеется лишь для $j \geq 1$, поэтому путь никогда не опускается ниже оси x . Существует ровно $\binom{n}{k}$ таких путей.

Перестановка, соответствующая подобному пути, строится с использованием трех первоначально пустых списков следующим образом: если i -й шаг делается вверх, $i = 1, 2, \dots, n$, то число i помещается в список B ; если же вниз, то i помещается в список A , а максимальный на данный момент элемент списка B изымается из него и переносится в список C . Затем нужно выписать друг за другом элементы списков A , B (в возрастающем порядке) и C (также в возрастающем порядке). Построение перестановки завершено.

Например, при $n = 4$ и $k = 2$ получается шесть путей и соответствующих им перестановок

Picture: Рис. стр. 673

(Вертикальные линии разделяют списки A , B и C . Эти шесть перестановок соответствуют составным атрибутам в (2).)

[Указание. Представьте каждое t -элементное подмножество S с помощью пути из $(0, 0)$ в $(n, n - 2t)$, i -й шаг которого выполняется из $(i - 1, j)$ в $(i, j + 1)$, если $i \notin S$, и в $(i, j - 1)$, если $i \in S$. Преобразуйте каждый такой путь в путь описанного выше вида.]

2. [M25] (Сакти Гхош.) Найдите минимально возможную длину l списка $r_1 r_2 \dots r_l$ ссылок на записи, обладающего тем свойством, что все ответы на любой из включающих запросов $**1, *1*, 1**, *11, 1*1, 11*$ по трем ключам с двумя возможными значениями окажутся в последовательных элементах $r_i \dots r_j$.
3. [19] Рассмотрим табл. 2. Какие включающие запросы вызовут ложное выпадение: (а) старинного сахарного печенья; (б) овсяных палочек с финиками?
4. [M30] Найдите точные формулы для вероятностей (5), предполагая, что каждая запись имеет g различных атрибутов, случайно выбираемых из $\binom{n}{k}$ k -битовых кодов в n -битовом поле, и что запрос включает q различных, а в остальном случайных атрибутов. (Не пугайтесь, если формулы не будут упрощаться!)
5. [40] Поэкспериментируйте с различными способами уменьшения избыточности в текстах, если для поиска подцепочек используется метод Харрисона.
- >6. [M20] Общее число m -битовых запросов с k специфицированными битами равно $s = \binom{m}{k} 2^k$. Если комбинаторная хеш-функция, подобная (7), преобразует эти запросы в адреса l_1, l_2, \dots, l_s соответственно, то $L(k) = (l_1 + l_2 + \dots + l_s)/s$ дает среднее число адресов, приходящихся на запрос. [Например, в (7) имеем $L(3) = 1.75$.]

Рассмотрим теперь составную хеш-функцию над $(m_1 + m_2)$ -битовым полем, образованную отображением первых m_1 битов с помощью одной хеш-функции, а оставшихся m_2 битов — с помощью другой. Пусть $L_1(k)$ и $L_2(k)$ — соответствующие средние количества адресов, приходящихся на запрос. Найдите формулу, выражающую $L(k)$ (среднее для составной функции) через L_1 и L_2 .

7. [M24] (Р. Райвест.) Найдите значения функции $L(k)$, определенной в предыдущем упражнении, для следующих функций комбинаторного хеширования:

(a) $m = 3, n = 2$	(a) $m = 4, n = 2$
0 0 * \rightarrow 1	0 0 * * \rightarrow 1
1 * 0 \rightarrow 2	* 1 * 0 \rightarrow 2
* 1 1 \rightarrow 3	* 1 1 1 \rightarrow 3
1 0 1 \rightarrow 4	1 0 1 * \rightarrow 3
0 1 0 \rightarrow 4	* 1 0 1 \rightarrow 4
	1 0 0 * \rightarrow 4

8. [M47] Придумайте новые полезные классы функций комбинаторного хеширования, аналогичные (7).
9. [M20] Докажите, что при $v = 3^n$ множество всех троек вида

$$\{(a_1 \dots a_{k-1} 0 b_1 \dots b_{n-k})_3, (a_1 \dots a_{k-1} 1 c_1 \dots c_{n-k})_3, (a_1 \dots a_{k-1} 2 d_1 \dots d_{n-k})_3\}, 1 \leq k \leq n,$$

образует штейнеровскую систему троек. Предполагается, что a_i, b_i, c_i и d_i равны 0, 1 или 2 и $b_i + c_i + d_i \equiv 0 \pmod{3}$.

10. [M32] (Томас Киркман [Cambridge and Dublin Math Journal, 2 (1847), 191–204].) Назовем системой троек Киркмана порядка v такую организацию $v + 1$ объектов $\{x_0, x_1, \dots, x_v\}$ в тройки, что каждая пара $\{x_i, x_j\}$, $i \neq j$, встречается ровно в одной тройке; исключения составляют v пар $\{x_i, x_{(i+1) \bmod v}\}$, $0 \leq i < v$, не встречающихся ни в одной тройке. Например,

$$\{x_0, x_2, x_4\}, \{x_1, x_3, x_4\}$$

является системой троек Киркмана порядка 4.

- a) Докажите, что система троек Киркмана может существовать только при $v \bmod 6 = 0$ или 4.
- b) Дана штейнеровская система троек S над v объектами $\{x_1, \dots, x_v\}$. Докажите, что следующее построение приводит к штейнеровской системе троек S' над $2v + 1$ объектами и киркмановской системе троек K' порядка $2v - 2$: в S' входят все тройки из S плюс
- $\{x_i, y_j, y_k\}$, где $j + k \equiv i \pmod{v}$ и $j < k, 1 \leq i, j, k \leq v$;
 - $\{x_i, y_j, z\}$, где $2j \equiv i \pmod{v}, 1 \leq i, j \leq v$. В систему K' входят все тройки S' , не содержащие y_1 и/или y_v .
- c) Дана киркмановская система троек K над объектами $\{x_0, x_1, \dots, x_v\}$, где $v = 2u$. Докажите, что следующее построение приводит к штейнеровской системе троек S' над $2v + 1$ объектами и киркмановской системе троек K' порядка $2v - 2$: в S' входят все тройки из K плюс
- $\{x_i, x_{(i+1) \bmod v}, y_{i+1}\}, 0 \leq i < v$;
 - $\{x_i, y_j, y_k\}, j + k \equiv 2i + 1 \pmod{v - 1}, 1 \leq j < k - 1 \leq v - 2, 1 \leq i \leq v - 2$;
 - $\{x_i, y_j, y_v\}, 2j \equiv 2i + 1 \pmod{v - 1}, 1 \leq j \leq v - 1, 1 \leq i \leq v - 2$;
 - $\{x_0, y_{2j}, y_{2j+1}\}, \{x_{v-1}, y_{2j-1}, y_{2j}\}, \{x_v, y_j, y_{v-j}\}, 1 \leq j < u$;
 - $\{x_v, y_u, y_v\}$. В K' входят все тройки из S' , не содержащие y_1 и/или y_{v-1} .
- d) Используйте предыдущие результаты для доказательства того, что киркмановская система троек порядка v существует при любом $v \geq 0$, имеющем вид $6k$ или $6k + 4$, а штейнеровская система троек над v объектами существует при любом $v \geq 1$, имеющем вид $6k + 1$ или $6k + 3$.
11. [M25] В тексте описано использование штейнеровской системы троек в связи с включающими запросами; чтобы использовать их и для базовых запросов, естественно ввести следующее понятие. *Пополненной системой троек* порядка v называется такая организация $2v$ объектов $\{x_1, \dots, x_v, \bar{x}_1, \dots, \bar{x}_v\}$ в тройки, что каждая пара объектов встречается ровно в одной тройке; исключения составляют пары дополнительных элементов $\{x_i, \bar{x}_i\}$, не входящие ни в одну тройку. Например,
- $$\{x_1, x_2, x_3\}, \quad \{x_1, \bar{x}_2, \bar{x}_3\}, \quad \{\bar{x}_1, x_2, \bar{x}_3\}, \quad \{\bar{x}_1, \bar{x}_2, x_3\}$$
- есть пополненная система троек порядка 3.
- Докажите, что пополненная система троек порядка v существует для любого $v \geq 0$, не имеющего вида $3k + 2$.
12. [M23] В продолжение упр. 11 постройте пополненную систему *четверок* порядка 7.
13. [M25] Постройте систему четверок, содержащую $v = 4^n$ элементов, аналогичную системе троек из упр. 9.
14. [25] Обсудите задачу удаления узлов из почтовых деревьев, подобных изображенному на рис. 45.