А. Л. БРУДНО, Л. И. КАПЛАН

МОСКОВСКИЕ ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ

Под редакцией
академика Б. Н. НАУМОВА

ИЗДАНИЕ ВТОРОЕ, ПЕРЕРАБОТАННОЕ
И ДОПОЛНЕННОЕ

МОСКВА «НАУКА
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1990

ББК 22.18
Б89
УДК 519.85(023)

Брудно А. Л., Каплан Л. И. **Московские олимпиады по программированию**/Под ред. акад. Б. Н. Наумова.— 2-е изд., доп. и перераб.— М.: Наука. Гл. ред. физ.-мат. лит., 1990.— 208 с.— ISBN 5-02-014394-4.

В первой части собраны все задачи Московских олимпиад по программированию для школьников, проводившихся в 1980—1988 гг. Изложены алгоритмы и программы решения этих задач на Бейсике, Паскале, Си и Фортране.

Во второй части приведены лекции для школьников по программированию и смежным вопросам.

1-е изд.— 1985 г.

Для школьников и учащихся ПТУ. Издается по просьбе книготорговых организаций.

Табл. 1. Ил. 17. Библиогр. 11 назв.

Рецензент
кандидат физико-математических наук *И. Г. Фараджев*

Научно-популярное издание

БРУДНО Александр Львович, КАПЛАН Лев Исаакович
МОСКОВСКИЕ ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ

Заведующий редакцией *Т. В. Шароватова*
Редактор *Л. Г. Полякова* Художественный редактор *Т. Н. Кольченко*
Технический редактор *С. Я. Шкляр* Корректор *О. М. Карпова*

ИБ № 41060

Сдано в набор 05.12.89. Подписано к печати 20.09.90. Формат 84×108/32.
Бумага тип. № 2. Гарнитура литературная. Печать офсетная.
Усл. печ. л. 10,92. Усл. кр.-отт. 11,13. Уч.-изд. л. 10,56.
Тираж 200 000 экз. Заказ № 4085. Цена 1 р. 30 к.

Издательско-производственное и книготорговое объединение «Наука»

Главная редакция физико-математической литературы
117071 Москва, В-71, Ленинский проспект, 15

Ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО «Первая Образцовая типография» Государственного комитета СССР по печати. 113054, Москва, Валовая, 28.

Отпечатано в типографии издательства «Коммуна», г. Воронеж, пр. Революции, 39

Б $\frac{1404000000-119}{053(02)-90}$ 174—90

ISBN 5-02-014394-4

ПРЕДИСЛОВИЕ АВТОРОВ

В книге собраны все задачи Московских олимпиад по программированию, начиная с первой олимпиады, прошедшей в 1980 г. Для всех задач приводятся алгоритмы решений и решения (программы). Задачи приведены в таком виде, как они предлагались участникам. Алгоритмы решений написаны так, чтобы облегчить чтение программ, а решение каждой задачи приводится на четырех языках программирования: Бейсике, Паскале, Фортране и Си — с тем, чтобы читатель мог его разбирать на том языке, которым лучше владеет.

Эти олимпиады проводятся ежегодно Учебно-производственным центром вычислительной техники (УПЦ ВТ) Октябрьского РУНО г. Москвы совместно с его базовыми предприятиями: Институтом электронных управляющих машин (ИНЭУМ) Минприбора СССР и Институтом проблем информатики Академии наук СССР (ИПИАН). В оргкомитет и жюри входят преподаватели Центра, сотрудники базовых институтов и победители предыдущих олимпиад.

Число участников колеблется от 100 до 300. Школьники получают задания и пишут решения (в течение четырех часов) на любых языках программирования. Через 2—3 недели проводится итоговая конференция с разбором задач, докладами учащихся и награждением победителей.

Олимпиады и предлагаемая книга служат трем целям: увлечь школьников программированием, дать материал для работы способным школьникам и дать задачи, на которых школьник сможет выяснить, стоит ли ему выбирать такую специальность. Кроме того, преподаватель сможет за счет олимпиадного материала сделать интереснее свои уроки, вести на нем школьные кружки, а также проверять и повышать свою предметную квалификацию.

Вторую часть книги составляют лекции, прочитанные школьникам в УПЦ ВТ во время занятий, в периоды подготовки к олимпиадам и в факультативных кружках. Здесь рассматривается программирование перебора вариантов, метод ретроспективного анализа, занимательные задачи кибернетики, задачи вычислительной математики, история вычислительной техники и т. п.

Мы получили возможность написать эту книгу благодаря сотрудникам Центра и базовых институтов, принимавшим участие в работе Центра — от его организации до проверок олимпиадных работ. Всем этим лицам мы, естественно, благодарны. Воспользуемся случаем перечислить тех, кто внес бо́льший вклад в наши олимпиады. Это — директор ИНЭУМ, а затем ИПИАН академик Б. Н. Наумов, обеспечивший Центр вычислительной техникой и квалифицированными преподавателями, и директор Центра В. Д. Горский, по инициативе которого были начаты эти олимпиады. С 1984 г. директором ИНЭУМ стал Н. Л. Прохоров, а директором Центра — Т. П. Кравчук; они активно продолжили руководство Центром и олимпиадами. Далее нам приятно назвать М. К. Антонову — сотрудницу ИНЭУМ, все эти годы преподававшую программирование. Высокая квалификация сделала ее ведущим преподавателем, на уроки которого ходят другие учителя, а педагогический талант вызывает энтузиазм учеников.

Первая часть книги написана нами совместно, вторую написал Брудно А. Л.

Авторы благодарят М. А. Макаревского, М. М. Коган, Д. А. Ленского за подготовку программ на Паскале и Си.

Второе издание отличается от первого добавлением олимпиад 1985—88 гг. В него включены решения на Бейсике, Паскале и Си, а решения на Алголе исключены. Дополнена вторая часть и исключен материал об организации учебного процесса в УПК.

Глава 1

ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ

§ 1.1. Олимпиадные задачи

На Московских олимпиадах по программированию школьникам предлагались наборы из некоторого количества задач. Число задач было в разные годы от 5 до 11. Но задачи всегда были различной ценности, шли в порядке убывания трудности, и в зачет принимались только три лучших решения. Участники обо всем этом предупреждались. Первые по порядку задачи могли быть довольно трудны, а последние носили явно «утешительный» характер. На олимпиаду отводилось 4 часа. Решения можно было писать на любых языках программирования. Перед решением надо было обязательно привести словесное описание алгоритма, облегчающее чтение программы. Ясность алгоритма, экономность числа действий и краткость программы повышали оценку, а образование излишних массивов и ошибки снижали ее.

В формулировках задач и алгоритмах решений у нас приняты следующие обозначения.

Через $A[1:n]$ обозначается массив $A_1, A_2, \ldots, A_n$, через $A[i]$ — его элемент A_i. То же относится к обозначениям $A[1:m, 1:n]$ и $A[i, j]$ для двумерных массивов и т. п.

Числа, задающие количество элементов массива и индексы элементов, предполагаются натуральными. Остальные — реальными (т. е. с плавающей точкой), если не оговорено иначе.

Если в условии сказано, что «задан массив $A[1:n]$», то программу школьник должен был писать так, чтобы она вводила число n, образовывала массив из n элементов и вводила значения этих элементов. Если в языке (например, в Фортране) динамических массивов нет, то следовало принять, что $n<100$, и образовать массив из

100 элементов. Но вводить значения надо лишь для заданного числа n элементов. То же самое относится к двумерным массивам и массивам большего числа измерений. Допускалось, впрочем, что написанная программа является частью охватывающей, которая уже образовала нужные массивы и ввела нужные числа и элементы. Но это нужно было оговорить.

Все ответы следовало выводить на экран или на печать.

Еще раз подчеркнем, что в качестве решения должна была представляться программа и никакие рассуждения не могли ее заменить.

ОЛИМПИАДА 80

Участникам было предложено одиннадцать задач, разбитых на три группы. Группы располагались в порядке убывания трудности, внутри одной группы задачи считались равноценными. Из каждой группы в зачет шло не более одной задачи.

80.1.1. Простые до M. Напечатать все простые числа, не превосходящие заданное число M.

80.1.2. Перестановки. Задан массив $A[1:m]$ попарно различных чисел. Напечатать все перестановки этих чисел.

80.1.3. Быстрая степень. Ввести вещественное число A и натуральное k. Вычислить и напечатать A^k с выполнением следующих условий: операцией возведения в степень пользоваться нельзя; k может оказаться настолько большим, что недопустимо выполнять k умножений.

80.1.4. Арифметические действия. В написанном выражении ((((1?2)?3)?4)?5)?6 вместо каждого знака ? вставить знак одного из четырех арифметических действий: $+$, $-$, $*$, $/$ так, чтобы результат вычислений равнялся 35 (при делении дробная часть в частном отбрасывается). Достаточно найти одно решение.

80.2.1. Поиск равных. Дан массив $A[1:2, 1:15]$. Известно, что среди его элементов два и только два равны между собой. Напечатать их индексы.

80.2.2. Сумма квадратов. Можно ли заданное натуральное число M представить в виде суммы двух квадратов натуральных чисел? Написать программу решения этой задачи.

80.2.3. Различные числа. Задан числовой массив $A|1:m|$. Сосчитать и напечатать, сколько различных чисел в этом массиве. Например, в массиве 5, 7, 5 различных чисел два (5 и 7).

80.3.1. Заданная сумма цифр. Составить программу вывода всех трехзначных десятичных чисел, сумма цифр которых равна данному натуральному числу.

80.3.2. $M+1$ в двоичной записи. Целое неотрицательное число M задано массивом своих двоичных цифр $a_0, a_1, ..., a_{n-1}$:

$$M=a_{n-1}\cdot 2^{n-1}+a_{n-2}\cdot 2^{n-2}+...+a_1\cdot 2+a_0,$$

где a_i равно 0 или 1 ($i=0, 1, ..., n-1$). Напечатать массив двоичных цифр числа $M+1$.

80.3.3. Максимум минимумов. В массиве $X|1:m, 1:n|$ все числа различны. В каждой строке находится минимальный элемент, затем среди этих чисел выбирается максимальное. Напечатать номер строки массива X, в которой расположено выбранное число.

80.3.4. Перестановка 0, 1, 2. В массиве $X|1:n|$ каждый элемент равен 0, 1 или 2. Переставить элементы массива так, чтобы сначала располагались все нули, затем все единицы и, наконец, все двойки (дополнительного массива не заводить).

ОЛИМПИАДА 81

Участникам было предложено пять задач, расположенных в порядке убывания трудности. В оценку шли только три задачи.

81.1. Функция. Функция $f(n)$ для целых неотрицательных n определена так:

$$f(0)=0,\ f(1)=1,\ f(2n)=f(n),\ f(2n+1)=f(n)+f(n+1).$$

Для данного N найти и напечатать $f(N)$.

Обязательное условие: N столь велико, что недопустимо заводить массив из N чисел (равно как и массив, длина которого растет с ростом числа N).

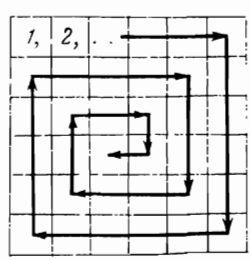


Рис. 1.1

81.2. Пара четверок. Найти минимальное число, которое представляется суммой четы-

рех квадратов натуральных чисел не единственным образом.

81.3. Спираль. Ввести число n и заполнить двумерный массив размером $n \times n$ числами 1, 2, ... по спирали (рис. 1.1).

81.4. Числа из разных цифр. Напечатать все четырехзначные натуральные числа, в десятичной записи которых нет двух одинаковых цифр.

81.5. Серия нулей. Задан числовой массив $A[1:n]$. Найти длину самой длинной последовательности подряд идущих элементов массива, равных нулю.

ОЛИМПИАДА 82

Участникам было предложено шесть задач, идущих в порядке убывания трудности. В оценку шли только три задачи.

82.1. Прямоугольники. На квадратном клетчатом листе бумаги размером 100×100 клеток нарисовано несколько прямоугольников. Каждый прямоугольник состоит из целых клеток, различные прямоугольники не накладываются друг на друга и не соприкасаются (см. пример на рис. 1.2).

Задан массив размером 100×100, в котором элемент $A[i, j] = 1$, если клетка $[i, j]$ принадлежит какому-либо прямоугольнику, и $A[i, j] = 0$ в противном случае. Написать программу, которая сосчитает и напечатает число прямоугольников.

82.2. Упорядоченные дроби. Напечатать в порядке возрастания все простые несократимые дроби, заключенные между 0 и 1, знаменатели которых не превышают 7.

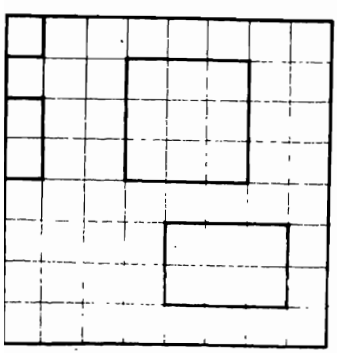


Рис. 1.2

82.3. Сумма по подмножеству. Даны целочисленный массив $A[1:n]$ и число M. Найти такое множество элементов $A[i_1]$, $A[i_2]$,..., $A[i_k]$ $(1 \leqslant i_1 < ... < i_k \leqslant n)$, что $A[i_1] + A[i_2] + ... + A[i_k] = M$. Предполагается, что такое множество заведомо существует.

82.4. Нули — в конец. Дан одномерный массив. Все его элементы, не

равные нулю, переписать (сохраняя их порядок) в начало массива, а нулевые элементы — в конец массива (новый массив не заводить).

82.5. Седловая точка. Задан числовой массив $A[1:m, 1:n]$. Некоторый элемент этого массива назовем седловой точкой, если он является одновременно наименьшим в своей строке и наибольшим в своем столбце. Напечатать номера строки и столбца какой-нибудь седловой точки и напечатать число 0, если такой точки нет.

82.6. Вхождение слова в текст. Даны два целочисленных массива $X[1:n]$ и $Y[1:k]$. Можно ли в первом из них выбрать такие k идущих подряд элементов $X[i+1]$, $X[i+2]$, ..., $X[i+k]$, чтобы $X[i+1] = Y[1]$, $X[i+2] = Y[2]$, ..., $X[i+k] = Y[k]$? Написать программу, которая решает эту задачу и печатает ответ ДА или НЕТ.

ОЛИМПИАДА 83

Участникам было предложено пять задач, следующих в порядке убывания трудности. В оценку шли только три задачи.

83.1. Бит-реверс. Целое положительное число m записывается в двоичной системе счисления и разряды (в этой записи) переставляются в обратном порядке. Получившееся число принимается за значение функции $B(m)$. Напечатать значения $B(m)$ для $m = 512, 513, 514, \ldots, 1023$.

Вот, для ясности, начало этой распечатки: 1, 513, 257, ...

83.2. Треугольник и точка. Заданы прямоугольные координаты $x1$, $y1$; $x2$, $y2$; $x3$, $y3$ вершин треугольника и координаты x, y точки. Определить и напечатать, находится ли точка в треугольнике. Погрешностями вычислений пренебречь.

83.3. Лабиринт. Может ли путник выйти из лабиринта? Если может, то напечатать путь от выхода до начального положения путника.

Лабиринт задан массивом A размером 40×40, в котором:

$A[k, m] = 0$, если клетка $[k, m]$ «проходима»;
$A[k, m] = 1$, если клетка $[k, m]$ «непроходима».

Начальное положение путника задается в проходимой клетке $[i, j]$. Путник может перемещаться из одной

проходимой клетки в другую, если они имеют общую сторону. Путник выходит из лабиринта, когда попадает в граничную клетку (то есть клетку $[k, m]$, где k или m равны 1 или 40).

83.4. Пила. Задан массив $X[1:m]$. Найти длину k самой длинной «пилообразной (зубьями вверх)» последовательности идущих подряд чисел:

$$X[p+1]<X[p+2]>X[p+3]<\ldots>X[p+k].$$

83.5. Сократить дробь. Даны натуральные числа m и n. Найти такие натуральные числа m_1 и n_1, не имеющие общих делителей, что $m_1/n_1=m/n$.

ОЛИМПИАДА 84

Участникам было предложено семь задач, следующих в порядке убывания трудности. В оценку шли только три задачи.

84.1. Инверсия. Пусть $P=(p_1,\ldots,p_n)$ является перестановкой чисел 1, 2, ..., n. Таблицей инверсий перестановки P называют последовательность $T=(t_1,\ldots,t_n)$, в которой t_i равно числу элементов перестановки P, стоящих (в P) левее числа i и больших i. Например, для перестановки $P=(5, 9, 1, 8, 2, 6, 4, 7, 3)$ чисел 1, 2, ..., 9 таблица инверсий $T=(2, 3, 6, 4, 0, 2, 2, 1, 0)$.

Написать программу, которая по заданной таблице инверсий восстанавливает перестановку.

84.2. Дорога. Даны натуральные числа $n\geqslant 2$ и m и вещественный массив $A[1:m, 1:m, 1:n-1]$. Найти минимальное значение суммы

$$R=A[i_1, i_2, 1]+A[i_2, i_3, 2]+\ldots+A[i_{n-1}, i_n, n-1]$$

для возможных наборов целых чисел $1\leqslant i_1, i_2,\ldots, i_n\leqslant m$.

Пояснение. Числа m, n — величины порядка нескольких десятков. Поэтому неприемлемо решение с числом действий порядка m^n.

84.3. Совершенные числа. Натуральное число называется совершенным, если оно равно сумме всех своих собственных делителей, включая 1. Напечатать все совершенные числа, меньшие, чем заданное M.

84.4. Период дроби. Ввести натуральные числа m и n и напечатать период десятичной дроби m/n.

Например, для дроби 1/7 периодом будет (142857), а если дробь конечная, то ее период состоит из одной цифры 0.

84.5. Слияние массивов. Даны два числа m и n и два упорядоченных массива $A[1] \leqslant A[2] \leqslant ... \leqslant A[m]$ и $B|1| \leqslant B|2| \leqslant ... \leqslant B|n|$. Образовать из этих элементов упорядоченный массив $C[1] \leqslant C[2] \leqslant ... \leqslant C[m+n]$.

Указание. Обратить внимание на число действий программы при больших m и n.

84.6. Календарь. Заданы три числа A, B, C, которые обозначают число, месяц и год. Найти номер N этого дня с начала года.

Указание: високосные годы — это те, у которых номер делится на 400, и те, у которых номер делится на 4, но не делится на 100.

84.7. Квадратики. Дан массив $A|1:m,\ 1:m|$, каждый элемент которого равен 0, 1, 5 или 11. Подсчитать в нем количество четверок $A[i,\ j]$, $A[i+1,\ j]$, $A[i,\ j+1]$, $A[i+1,\ j+1]$, в каждой из которых все элементы различны.

ОЛИМПИАДА 85

Участникам было предложено семь задач, идущих в порядке убывания трудности. В оценку шли только три задачи.

85.1. Разложение на слагаемые. Напечатать все представления натурального числа N суммой натуральных чисел. Перестановка слагаемых нового способа не дает.

85.2. Равные элементы. Задан целочисленный массив $A|1:m,\ 1:n|$. Каждая строка массива упорядочена по $\leqslant$, т. е. $A[i,\ 1] \leqslant A[i,\ 2] \leqslant ...$ при всех $i = 1,...,m$. Найти и напечатать число, встречающееся во всех строках, и напечатать надпись НЕТ, если такого числа не окажется.

85.3. Несоставляемое число. Задан массив натуральных чисел $P|1:n|$. Найти минимальное натуральное число, не представимое суммой никаких элементов массива P. Сумма может состоять и из одного слагаемого, но каждый элемент массива может входить в нее только один раз.

85.4. Тетраэдры. На гранях двух равных правильных тетраэдров M и N написаны числа $M1$, $M2$, $M3$, $M4$ и $N1$, $N2$, $N3$, $N4$ в порядке, указанном на рис. 1.3.

Можно ли совместить тетраэдры так, чтобы на совпавших гранях оказались написаны одинаковые числа? Напечатать ДА или НЕТ.

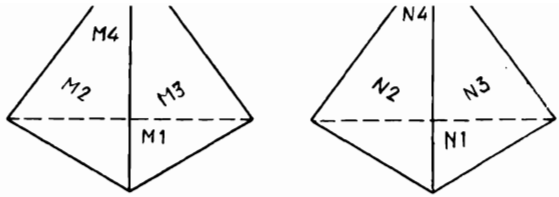


Рис. 1.3

85.5. Мода. В целочисленном массиве $A[1:n]$ найти число, повторяющееся максимальное количество раз. Если таких чисел несколько, то одно из них.

85.6. Системы счисления. В массиве $M[1:9]$ записаны разряды (цифры) некоторого натурального числа в I-ричной системе счисления ($M[1]$ — разряд единиц и т. д.). Отпечатать разряды этого числа в J-ричной системе счисления, начиная с разряда единиц. Числа I, J не превосходят 10.

85.7. Побочная диагональ. Найти сумму элементов $A[i, j]$ массива $A[1:m, 1:n]$, имеющих заданную разность индексов $i-j=k$.

Напоминание. Число k может быть и отрицательным.

ОЛИМПИАДА 86

Участникам было предложено пять задач, идущих в порядке убывания трудности. В оценку шли только три задачи.

86.1. Без тройных повторений. Найти последовательность из 50 нулей и единиц, в которой никакой отрезок не повторяется три раза подряд. Напечатать НЕТ, если такой последовательности не существует.

Например, в искомой последовательности нигде не должны встречаться такие отрезки, как 000, или 101010, или 101101101.

86.2. Покер. Задан массив из пяти чисел. Среди них:
если одинаковы 5, то напечатать число 1, иначе
если одинаковы 4, то напечатать число 2, иначе
если одинаковы 3 и 2, то напечатать число 3, иначе
если одинаковы 3, то напечатать число 4, иначе
если одинаковы 2 и 2, то напечатать число 5, иначе

если одинаковы 2, то напечатать число 6, иначе напечатать число 7.

86.3. Барабан. По окружности написаны 12 чисел a_1, a_2,..., a_{12}. Если их списать, начиная с номера k, то получится вектор x_k:

$$x_k=(a_k,\ a_{k+1},\ldots,\ a_{k+11}),$$

где под a_{13} понимается a_1, под a_{14} понимается a_2 и т. д. Вектор x_k считается меньше вектора x_p, если в первой же неравной паре будет $a_{k+i}<a_{p+i}$ $(i=0,\ 1,\ldots)$. Найти такое k, чтобы вектор x_k был минимален.

86.4. Дважды монотонный. Массив чисел $A[1:m,\ 1:n]$ упорядочен по $\leqslant$ по строкам и по столбцам, то есть

$$A[i,\ 1]\leqslant A[i,\ 2]\leqslant\ldots \text{ при всех } i=1,\ldots,m,$$
$$A[1,\ j]\leqslant A[2,\ j]\leqslant\ldots \text{ при всех } j=1,\ldots,n.$$

Найти элемент массива, равный заданному числу x, и напечатать его индексы. Напечатать НЕТ, если такого элемента не окажется.

Обязательное условие. Число действий в решении должно быть порядка $m+n$ (а не порядка $m\times n$).

86.5. Центральное селение. Имеется k селений. Если в селении i расположить пункт скорой помощи, то поездка по вызову в селение j займет время

$$A[i,\ i]+A[i,\ j] \qquad (1\leqslant i,\ j\leqslant k,\ i\neq j).$$

Найти номер селения i, от которого поездка в самое удаленное (по времени) селение занимала бы минимальное время. Массив $A[1:k,\ 1:k]$ задан. В нем все $A[i,\ j]>0$ и элемент $A[i,\ j]$ может быть не равен элементу $A[j,\ i]$.

ОЛИМПИАДА 87

Участникам было предложено пять задач, следующих в порядке убывания трудности. В оценку шли только три задачи.

87.1. Рюкзак. Из заданных n предметов выбрать такие, чтобы их суммарный вес был менее 30 кг, а стоимость — наибольшей. Напечатать суммарную стоимость выбранных предметов.

Точнее — заданы два массива положительных чисел $A[1:n]$ и $B[1:n]$. Выбрать такие попарно различные числа i_1, i_2, ..., i_k, чтобы сумма

$$A[i_1]+A[i_2]+\ldots+A[i_k]<30,$$

а сумма

$$B[i_1]+B[i_2]+\ldots+B[i_k]=\max$$

была максимальной. Напечатать только величину max.

Замечание. Можно предполагать, что предметы уже расположены в порядке возрастания или убывания веса $A[i]$, стоимости $B[i]$, цены $B[i]/A[i]$ или какого-либо иного признака.

87.2. Полукратные. Множество чисел A задано условиями:

а) $1\in A$,

б) если $k\in A$, то $2*k+1\in A$ и $3*k+1\in A$,

и других чисел множество A не содержит.

Напечатать первые $n<1000$ чисел множества A в порядке возрастания.

Вот начало этой распечатки: 1, 3, 4, 7, 9, 10, 13, 15, 19,...

87.3. Сумма кубов. Сколькими способами заданное натуральное число N можно представить в виде суммы двух кубов натуральных чисел:

$$N=i^3+j^3?$$

Перестановка слагаемых нового способа не дает. Операцией возведения в степень 1/3 пользоваться нельзя.

87.4. Переверты ши. Задан числовой массив $A[1:n]$. Найти отрезок массива максимальной длины, в котором первое число равно последнему, второе — предпоследнему и т. д. Напечатать длину этого отрезка.

87.5. Индексы порядка. Задан числовой массив $A[1:n]$. Найти и отпечатать такую перестановку i_1, i_2,..., i_n чисел 1, 2,..., n, чтобы

$$A[i_1]\leqslant A[i_2]\leqslant\ldots\leqslant A[i_n].$$

ОЛИМПИАДА 88

Участникам было предложено пять задач, следующих в порядке убывания трудности. В оценку шли только три задачи.

88.1. Правый бóльший. Задан массив положительных чисел $A[1:n]$. Для каждого $A[i]$ среди элементов массива, следующих (по порядку) за $A[i]$ и больших чем $A[i]$, выберем элемент с наименьшим номером j и заменим значение $A[i]$ на $A[j]$. Если такого элемента

$A[j]$ не найдется, то заменим значение $A[i]$ нулем. Распечатать получившийся массив.

Обязательное условие. Число действий в решении должно быть порядка n (а не $n*n$). Можно завести вспомогательные массивы.

Пояснение. Например, массив 2, 9, 8, 5, 9, 3, 4, 5, 2 после замены станет таким: 9, 0, 9, 9, 0, 4, 5, 0, 0.

88.2. Многочлен. Вычислить коэффициенты $a[0]$, $a[1]$,..., $a[n-1]$ многочлена

$$P(x)=a[0]+a[1]*x+a[2]*x\wedge 2+\ldots$$
$$\ldots+a[n-1]*x\wedge(n-1)+x\wedge n$$

с заданными действительными корнями $x[1]$, $x[2]$,..., $x[n]$.

Напоминание. По теореме Безу

$$P(x)=(x-x[1])*(x-x[2])*\ldots*(x-x[n]).$$

88.3. Простые делители. Задано натуральное число N. Найти и напечатать все его простые делители.

88.4. Оптовая покупка. Пара носков стоит 1.05 руб., связка (12 пар) стоит 10.25 руб., а коробка (12 связок) стоит 114 руб. Составьте программу, которая по числу n пар носков, которые хочет купить покупатель, вычисляет числа $n1$, $n2$, $n3$ коробок, связок и пар носков, которые ему следует купить.

Пояснение. Вместо 11 пар носков следует, например, покупать связку — это обойдется дешевле.

88.5. Обнуление. В заданном двумерном массиве $A[1:m, 1:n]$ заменить нулями элементы, стоящие в строках или столбцах, где имеются нули.

Условие. Можно завести вспомогательный одномерный массив, но нельзя заводить вспомогательный двумерный массив.

§ 1.2. Алгоритмы решений

В этом параграфе приводятся алгоритмы решений. Иногда это будет законченное рассуждение, которое остается переписать на языке программирования, иногда лишь идея решения, а иногда комментарий, поясняющий программу. Но во всех случаях мы стремились облегчить чтение программы, изложить прием программирования, которым стоит овладеть, и готовы были ограничиться этим. Число действий мы, как правило, указываем с точностью до постоянного множителя.

80.1.1. Простые до M. Чтобы ускорить вычисления, полезно завести таблицу для уже найденных простых чисел и проверять делимость очередного кандидата только на числа из этой таблицы. Четные числа, естественно, не рассматривать.

Таблица понадобится менее чем на $\sqrt{M}/2$ чисел. Поэтому таблицы на 1000 чисел хватит для печати простых чисел до 4 000 000.

80.1.2. Перестановки. Специфика этой задачи в том, что нужно получить все перестановки. Это отличает ее от общей задачи перебора и значительно облегчает решение. И все же его описание длиннее программы.

Представим себе все перестановки чисел 1, 2,...,m. Упорядочим их (мысленно) в словарном порядке и научимся (в действительности) по заданной перестановке строить непосредственно следующую. И тогда, отправляясь от первой перестановки (1, 2,...,m), мы последовательно построим все перестановки этих чисел. Попутно для каждой перестановки $P=(p_1, p_2,..., p_m)$ чисел 1, 2,..., m напечатаем такую же перестановку $(A(p_1),..., A(p_m))$ заданных чисел A.

Чтобы по данной перестановке $P=(p_1, p_2,..., p_m)$ чисел 1, 2,..., m построить непосредственно следующую, мы будем просматривать числа p_1, p_2,..., p_m с конца. И остановимся, когда впервые попадется член p_i, меньший стоящего правее него ($p_i<p_{i+1}$). Если такого члена нет, то перестановка P имеет вид (m, $m-1$,..., 1), то есть является последней. Ясно, что члены $p_{i+1}>p_{i+2}>...>p_m$ образуют убывающую последовательность. Найдем среди них первый (если их рассматривать с конца) член p_j, уже больший чем p_i, и поменяем их местами.

Остается переставить члены p_{i+1}, p_{i+2},..., p_m в порядке возрастания, и искомая перестановка (назовем ее $Q=(q_1,..., q_m)$) будет получена (рис. 1.4).

Действительно, $P<Q$, ибо первые $i-1$ членов у них совпадают, а $p_i<q_i$ (так как по самому выбору члена $q_i=p_j>p_i$). Далее, при своих заданных первых i членах P является максимальной, а Q — минимальной перестановками, ибо в P остальные члены идут в убывающем порядке, а в Q — в возрастающем. Наконец, если перестановка R лежит между P и Q, то первые $i-1$ членов у нее совпадают с первыми членами P и Q, а член r_i равен p_i или q_i, так как при уже занятых первых $i-1$ членах нет чисел, расположенных между p_i и q_i. Если $p_i=r_i$ и $P\leqslant R$, то $P=R$ (так как P максимальна при заданных $p_1, p_2, ..., p_i$), а если $r_i=q_i$, то аналогично $R=Q$.

В программе это реализовано так. Мы заводим массив P для текущей перестановки, заполняем его первой перестановкой $P=(1, 2,..., m)$ и печатаем тождественную перестановку $(a_1, a_2,..., a_m)$ членов заданного массива A. Пусть очередная перестановка P получена и отвечающая ей перестановка A отпечатана. В соответствующем разделе программы отыскивается элемент $p_i<p_{i+1}$ с максимальным номером i. Если его нет, то перестановка P была последней. Иначе в следующем разделе программы ищется наибольший номер $j>i$, для которого $p_i<p_j$. Далее элементы p_i и p_j переставляются, после чего в последовательности $p_{i+1}, p_{i+2},..., p_m$ порядок меняется на обратный: для этого меняют местами p_{i+1} и p_m, затем p_{i+2} и p_{m-1} и т. д.

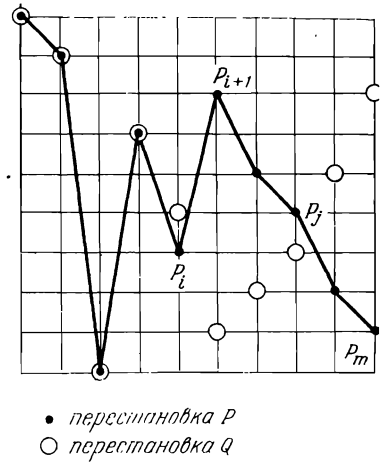


Рис. 1.4

На этом заканчивается получение следующей перестановки P, и печатается отвечающая ей перестановка массива A.

Задача о перестановках становится трудной, если потребовать, чтобы очередная перестановка получалась из предыдущей перестановкой двух соседних элементов. Такая задача решается программой 80.1.2—2. Программа короткая, но сложная. Читателю предоставляется разобрать ее самостоятельно, руководствуясь списком перестановок, которые она печатает (для $m=4$ элементов):

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

80.1.3. Быстрая степень. При обычном вычислении a^k заводят переменную b, вначале равную единице, и многократно выполняют операторы:

$$k=k-1 \ : \ b=b*a$$

Когда переменная k станет равной нулю (это потребует k циклов), b станет равно искомой величине a^k.

Идея сокращения вычислений в следующем. Вначале положим $b=1$. Если k нечетно, по-прежнему выполняем операторы

$$k=k-1 \ : \ b=b*a$$

Но если k четно, то, воспользовавшись тождеством

$$a^k=(a^2)^{k/2},$$

сделаем замены:

$$k=k/2 \ : \ a=a*a$$

Теперь, когда переменная k, наконец, станет равной нулю, переменная b станет равна искомой величине.

Для доказательства обозначим искомую величину a^k через w и положим $b=1$. Тогда

$$a^k*b=w \qquad (*)$$

Если k нечетно, то, выполнив замены

$$k=k-1 \ : \ b=b*a$$

мы не нарушим равенства (*). Если же k четно, то замены

$$k=k/2 \ : \ a=a*a$$

тоже его не нарушают. Когда k станет равным нулю, то равенство (*) перейдет в равенство

$a^0*b=w$, то есть $b=w$

и, значит, b будет равно искомой величине.

80.1.4. Арифметические действия. Эта задача — тоже на образование всех перестановок, только с повторениями. Запишем заданное выражение в виде

$$w=((((1a_2 2)a_3 3)a_4 4)a_5 5)a_6 6.$$

Здесь элементы массива $a[2:6]$ изображают знаки арифметических действий. Условимся, что знаки $+$, $-$, $*$, $/$ изображаются значениями 1, 2, 3, 4.

Поскольку вариантов может оказаться много ($4^5>1000$), постараемся организовать вычисления экономно. Для этого, вычисляя w, будем запоминать промежу-

точные результаты в элементах массива $B[1:6]$, полагая

$$B_1 = 1 : B_2 = B_1 a_2 2 : \ldots : B_6 = B_5 a_6 6$$

и $w = B_6$. Далее, перебирая варианты, будем сначала менять $a_6 = 1, 2, 3, 4$. При каждом таком изменении пересчитывать понадобится лишь B_6. Потом увеличим $a_5 = a_5 + 1$, пересчитаем B_5 и снова рассмотрим $a_6 = 1, 2, 3, 4$ и т. д.

Этот алгоритм и реализован в программе 80.1.4. Дадим некоторые пояснения к ее варианту на Бейсике. Переменными m и n обозначены числа 35 и 6. Первые строки программы (до строки с номером 60) сначала лучше пропустить. Они подготавливают такое «доначальное положение», чтобы операторы, начинающиеся с номера 60 и рассчитанные на выработку «очередного» варианта, сначала образовали первый вариант.

Рассмотрим программу, начиная с номера 60. Здесь отыскивается первый член a_i в последовательности a_n, a_{n-1}, ..., a_2, не равный 4. Он увеличивается на 1, а все предыдущие полагаются равными 1. Теперь вычисляется B_i по B_{i-1}, а затем все остальные члены: B_{i+1}, B_{i+2},..., B_n. Последнее несложно, так как соответствующие a_j равны 1, то есть обозначают сложение. Если $B_n \neq m$, то значение w оказалось не заданным, и программа переходит на строку с номером 60 для рассмотрения следующего варианта. Если же $B_n = m$, то программа также переходит на 60, но предварительно печатает найденный вариант.

В других программах 80.1.4 обозначения несколько отличаются: на Паскале и Си роль номера 60 строки выполняет метка R, на Фортране — метка 2.

80.2.1. Поиск равных. Для удовлетворительного решения этой задачи надо не брать для сравнения одну и ту же пару элементов $A[i, j]$, $A[p, q]$ дважды и не запутаться в случаях, когда $i = p$ и $j = q$.

В программе на Фортране эти трудности обходятся переходом к одномерному массиву с помощью оператора EQUIVALENCE.

80.2.2. Сумма квадратов. Программа читается просто.

Эта задача допускает решение и без применения функции «квадратный корень». Такое решение основывается на идее, указанной в программе 87.3. Читатель,

разобравший эту программу, при желании может вернуться к задаче 80.2.2 и без труда восстановить детали.

80.2.3. Различные числа. Программа 80.2.3 читается просто. Но ее можно ускорить, сравнивая испытуемый элемент не с предшествующими, а с последующими, до первого повторения.

От этого число сравнений упадет с $m*m$ до $m*k$, где m — число всех чисел, а k — число различных чисел в заданном массиве. Это сделано в программе 80.2.3—2.

80.3.1. Заданная сумма цифр. Можно, разумеется, написать тройной цикл по цифрам искомого числа:

по i от 0 до 9, по j от 0 до 9, по k от 1 до 9.

В нем подсчитать сумму $i+j+k$ и, если она окажется равной заданному числу, напечатать трехзначное число:

$$M=i+10j+100k.$$

Но это плохое решение. В нем 900 циклов вместо возможных 100.

Приемлемое решение содержит двойной цикл по i и по j, а k вычисляется по заданной сумме n:

$$k=n-i-j.$$

Добавив проверку $1\leqslant k\leqslant 9$, мы придем к программе 80.3.1.

Читатель может усовершенствовать программу, начав ее с проверки условия $n\leqslant 27$ и введя другие ускорения счета в случае $n<18$.

80.3.2. $M+1$ в двоичной записи. Будем просматривать числа $a_0, a_1, \ldots$, заменяя единицы на нули, до первого нуля — его заменим единицей и на этом прекратим замену чисел. Надо только учесть, что ответ может содержать $n+1$ чисел, а не n, как в условии.

80.3.3. Максимум минимумов. Приведенная программа составлена так, чтобы избежать отдельного рассмотрения первой строки и каждого первого элемента строки. Кроме того, обработка очередной строки прекращается, как только становится ясно, что она не сможет содержать искомого элемента.

80.3.4. Перестановка 0, 1, 2. При решении этой задачи ничего не нужно переставлять. Нужно сосчитать, сколько в массиве нулей, единиц и двоек, и заполнить массив требуемым образом.

81.1. Функция. Задача может показаться неразрешимой, поскольку формулу для $f(n)$ угадать трудно, а на каждом шаге уменьшения аргумента n число функций, подлежащих рассмотрению, как будто увеличивается. Действительно, отправляясь от одной функции $f(2n+1)$, мы получим две функции:

$$f(2n+1)=f(n)+f(n+1).$$

Из двух аргументов n и $n+1$ один нечетный. На следующем шаге он породит две функции и т. д. Однако выполнив этот второй шаг, мы замечаем, что функций остается по-прежнему две (а не три) и всегда будет оставаться две.

Для доказательства этого обстоятельства — попутно дающего программу — наряду с заданной функцией $f(n)$ одного аргумента введем функцию

$$g(n, i, j)=if(n)+jf(n+1)$$

трех аргументов n, i, j. Для нее легко проверяются рекуррентные формулы:

$$g(2n, i, j)=g(n, i+j, j), \quad g(2n+1, i, j)=g(n, i, i+j).$$

Теперь искомое значение $f(n)$ можно записать в виде

$$f(n)=g(n, 1, 0),$$

а многократным применением рекуррентных формул сделать первый аргумент функции g равным нулю и получить

$$f(n)=g(n, 1, 0)=\ldots=g(0, i, j)=j.$$

Остается выполнить формальности, требуемые языком, на котором пишется программа.

81.2. Пара четверок. Эта задача содержит небольшую ловушку, так как вместо минимального числа с нужными свойствами можно получить первое число, которое встретится при определенной организации перебора слагаемых. Работу программ, приведенных на Бейсике, Паскале и Си, читатель может ускорить, заметив, что у искомого числа два представления (в виде суммы квадратов) должны отличаться наибольшим слагаемым. Это учтено в программе на Фортране.

81.3. Спираль. Разделим виток спирали на четыре прямолинейных участка: горизонтальный слева на-

право, вертикальный сверху вниз, горизонтальный справа налево и вертикальный снизу вверх. В программе каждый из них заполняется по отдельности и используется «скатывание» одного участка на другой. Окончание в ней проверяется по занесению последнего числа $k=n*n$.

81.4. Числа из разных цифр. Программа 81.4 достаточно ясна. Программа на Фортране 81.4—2 экономит на арифметических вычислениях, но загружена из-за отсутствия удобных операторов печати (читатель может найти в литературе сколько угодно противоположных утверждений) без перехода на новую строку.

81.5. Серия нулей. Программа 81.5 проста, а программа на Фортране 81.5—2 заканчивает вычисления, когда не имеет шансов набрать последовательность нулей более длинную, чем уже встретившиеся. Выгоды от этого немного.

ОЛИМПИАДА 82

82.1. Прямоугольники. Эта задача решается в «одно соображение»: прямоугольников столько, сколько их северо-западных углов (иначе говоря — верхних левых). Остается только не запутаться в случае, когда угол стоит у границы. По-разному это затруднение преодолевается в программах 82.1 и 82.1—2. Предостережем только читателя, пожелавшего сократить программу 82.1, от использования выражений типа $i>1$ AND $A[i-1, j]=0$. Такое выражение окажется синтаксической ошибкой при $i=1$, если индексация начинается с единицы.

82.2. Упорядоченные дроби. В программе можно обойтись без вспомогательных массивов и без проверок на сократимость дробей. Для этого заведем дробь m/n и положим сначала $m=0$, $n=1$. Отпечатаем m/n. Теперь среди всех дробей a/b, больших чем m/n, и таких, что $b\leqslant P$ (в условии $P=7$), выберем минимальную. Пусть это будет i/j. Если окажется, что $i/j<1$, то заменим дробь m/n дробью i/j и продолжим процесс.

Укажем на некоторые особенности программы.

Для заданного знаменателя $b=2, \ldots, P$ числитель a не подбирается, а вычисляется по формуле

$$a=m*b\backslash n+1$$

Каждая найденная дробь m/n автоматически будет несократимой. Это вытекает из того, что среди всех

дробей, равных ей, она имеет наименьший знаменатель.

Отметим еще, что при сравнении дробей лучше сравнивать не частные (которые представляются в машине приближенно), а произведения: не $a/b<i/j$, но $a*j<b*i$.

82.3. Сумма по подмножеству. Это — задача на перечисление всех подмножеств. Внешне она похожа на перебор вариантов, но гораздо проще него. Пусть b — натуральное число, и b_i — его двоичные разряды ($b_i=0$ или $b_i=1$):

$$b=b_1+2b_2+...+2^{n-1}b_n.$$

Когда b пробегает последовательность значений $b=1$, 2, ..., 2^n-1, то наборы индексов i элементов b_i, равных единице, пробегают все (не пустые) подмножества множества $\{1, 2, ..., n\}$. Поэтому в программе заведен массив B, с элементами которого мы обращаемся как с двоичными разрядами числа b.

82.4. Нули — в конец. Задача легко решается с использованием двух циклов: первый переписывает ненулевые элементы в начало массива, а второй заполняет остаток нулями. Так это и сделано в программе на Фортране. Но можно эти циклы и совместить, как это сделано в других вариантах программы.

82.5. Седловая точка. Если mi — минимум элементов a_{ij} в строке i, а Mj — максимум элементов a_{ij} в столбце j, то

$$mi\leqslant a_{ij}\leqslant Mj.$$

Это позволяет сделать несколько замечаний:

а) минимум в любой строке не более максимума в любом столбце, то есть всегда

$$mi\leqslant Mj;$$

б) если для каких-нибудь i и j будет выполнено равенство $mi=Mj$, то максимальный минимум совпадает с минимальным максимумом, а на пересечении строки i со столбцом j будет стоять седловая точка:

$$mi=a_{ij}=Mj; \qquad (*)$$

в) если точка a_{ij} седловая, то для нее выполняется (*), и, следовательно, максимальный минимум равен минимальному максимуму.

Остается реализовать это в программе. В каждой строке мы ищем минимум mi, среди этих минимумов

выбираем максимум Ma и запоминаем строку $i0$, где он находится. Поиск минимума в строке следует прекращать, если ясно, что он заведомо окажется меньше текущего значения Ma.

Затем ищется столбец, в котором максимальный элемент равен найденному значению Ma; если такого столбца нет, то нет и седловой точки. Фактически в программе мы пользуемся тем, что если в столбце нет элементов, больших Ma, то столбец будет искомым в силу пункта а).

Следует отметить, что программа, не использующая изложенных здесь соображений, работала бы не намного медленнее.

82.6. Вхождение слова в текст. Программа читается просто.

ОЛИМПИАДА 83

83.1. Бит-реверс. Выпишем для ясности (см. таблицу) несколько значений m и $B(m)$ в десятичной и двоичной системах счисления.

Заметим, что старший двоичный разряд числа m изображает величину 512 и ему соответствует младший разряд числа $B(m)$, изображающий величину 1. Для следующих разрядов это будут величины 256 и 2 и т. д. Единица старшего разряда у числа m всегда есть, так как $m \geqslant 512$. Удалим ее, заменив m на $m-512$, и занесем единицу в число $B(m)$. В следующем разряде единица у числа m будет, если новое значение $m \geqslant 256$. Если она есть, удалим ее и добавим к $B(m)$ число 2. Проверим наличие следующей единицы и т. д. Это приведет к программе 83.1, нужно только прекратить работу при обнулении числа m.

Предыдущая программа тратит на свои вычисления 512*10 циклов. Быстрее будет работать программа,

Таблица

Система	10	2	2	10
Значения m и $B(m)$	512	1000000000	0000000001	1
	513	1000000001	1000000001	513
	514	1000000010	0100000001	257
	515	1000000011	1100000001	769
	516	1000000100	0010000001	129
	....	.....	.....	....
	1023	1111111111	1111111111	1023

которая по числу $B(m)$ строит число $B(m+1)$. Глядя на двоичную запись значений $B(m)$, можно сообразить, что при построении $B(m+1)$ нужно двигаться по двоичной записи числа $B(m)$ слева направо, заменяя единицы нулями (то есть вычитая из $B(m)$ числа 512, 256...), до первого нуля — его нужно заменить единицей, и число $B(m+1)$ будет получено. Эта программа истратит на вычисление половины чисел по одному циклу, на вычисление четверти — по два цикла, одной восьмой — по три и т. д. Всего уйдет менее 512*2 циклов. Так и построена программа 83.1—2.

83.2. Треугольник и точка. Заметим следующее. Пусть точка $M(x, y)$ лежит на прямой $L12$, проходящей через точки $M1(x1, y1)$ и $M(x2, y2)$. Тогда из подобия треугольников получим

$$\frac{y-y1}{x-x1}=\frac{y2-y1}{x2-x1}.$$

Обходя ловушку, связанную с обращением в нуль знаменателей, перепишем это уравнение в виде

$$F_{12}(x, y)\equiv(x-x1)*(y2-y1)-(x2-x1)*(y-y1)=0.$$

Ясно, что для точек $M(x, y)$, не лежащих на прямой $L12$, функция $F_{12}(x, y)$ не обращается в 0. Более того, как легко проверить, $F_{12}(x, y)<0$ для точек M, лежащих слева от прямой $L12$ (если смотреть из $M1$ в направлении $M2$), и $F_{12}(x, y)>0$ для точек M справа от $L12$. Но это значит, что точка M лежит внутри треугольника $(M1, M2, M3)$ тогда и только тогда, когда имеют одинаковые знаки все три числа:

$$F_{12}(x, y),\ F_{23}(x, y),\ F_{31}(x, y),$$

полученные для прямых $L12=(M1-\ -M2)$, $L23=$ $=(M2-\ -M3)$ и $L31=(M3-\ -M1)$.

Теперь для решения задачи удобно ввести два массива $X[1:3]$, $Y[1:3]$ для координат точек $M1$, $M2$, $M3$ и определить функцию

$$\Phi(i, j)=\operatorname{sign}[(x-X[i])*(Y[j]-Y[i])-$$
$$-(X[j]-X[i])*(y-Y[i])].$$

Обратившись к ней три раза, мы получим числа

$$t1=\Phi(1, 2),\ t2=\Phi(2, 3),\ t3=\Phi(3, 1).$$

Если они равны, то точка M лежит внутри треугольника, если нет, то — вне его (или на границе).

Если язык программирования не позволяет одной буквой обозначать и массив и переменную, то можно обозначить координаты точки M через $X[0]$, $Y[0]$, или через xt, yt и т. п.

Другое решение этой задачи основывается на сравнении площадей. Обозначим через S площадь данного треугольника ($M1$, $M2$, $M3$), а через $S1$, $S2$ и $S3$ — площади треугольников, которые получаются, если соединить точку M с двумя из вершин $M1$, $M2$, $M3$. Если $S=S1+S2+S3$, то точка находится внутри треугольника, иначе — вне его. Это решение приведено в программах на Фортране и Паскале. Площадь треугольника вычисляется по формуле Герона. Точка считается лежащей вне треугольника, если $S1+S2+S3> > 1{,}000001*S$. Множитель 1.000001 должен учесть погрешность вычислений.

83.3. Лабиринт. Эта хорошо известная задача у нас слегка изменена. Решение распадается на две части: поиск пути для выхода и печать «обратного» пути — от выхода до начального положения путника.

Простейшее решение первой части представляется таким. Запишем в клетку $A[i, j]$, где вначале находится путник, число 2 и положим $k=2$. Просмотрим все клетки A лабиринта. Для каждой из них, если в ней записан 0, прочтем четырех ее соседей. Если хоть в одной из соседних клеток записано число k (сейчас еще равное 2), то в рассматриваемую клетку A впишем число $k+1$.

Теперь увеличим $k=k+1$ и снова рассмотрим все клетки A. Этот процесс закончится, когда либо число будет вписано в граничную клетку (выход найден), либо за весь просмотр клеток A оно не будет вписано ни в одну новую клетку (выхода нет). Просмотров всего массива будет столько, сколько клеток окажется в кратчайшей дорожке.

Предыдущий алгоритм легко улучшить. На каждом шаге будем по-прежнему рассматривать все клетки A лабиринта. Если в клетке записан 0, а в какой-либо соседней находится число $k\geqslant 2$, то впишем в клетку A число $k+1$. Ясно, что это тоже решение первой части задачи. Но если повезет, то здесь решение найдется быстрее.

Заведомо эффективнее будет программа, которая, начиная с исходной клетки (куда вначале записывается число 2), ищет первого же свободного (то есть с числом 0) соседа. Вписывает в него число 3 и ищет его свободного соседа, чтобы вписать в него 4, и т. д. Процесс

оборвется либо при достижении границы (выход найден), либо оттого, что свободных соседей не будет (тупик). Если тупик возникает в исходной клетке (где записано число 2), то выхода нет. Если же тупиковая клетка иная и в нее вписано число $k>2$, то следует вписать в нее число 1 (сделать ее не проходимой) и перейти к ее соседней клетке с числом $k-1$. Такая клетка есть и единственна. Это решение и приведено в нашей программе 83.3. В ней легко узнать общую схему перебора вариантов.

Программу для решения задачи о лабиринте можно сильно сократить, если записать ее рекурсивно. Мы это сделали в программе 83.3—2. Но такую программу легче написать, чем прочесть. Предупредим все же читателя, что она верна.

Приведенные алгоритмы (кроме первого) могут давать и не кратчайший путь. Для экономного поиска кратчайшего пути в лабиринте можно завести специальные массивы X и Y для списка координат (x, y) клеток, которые следует рассмотреть. Такой прием называется *поиском в ширину*.

Вначале в X, Y заносятся координаты исходной клетки путника. На каждом шаге из массивов X, Y извлекаются для рассмотрения координаты очередной клетки (с номером b), а ее свободные соседи в любом порядке дописываются в продолжение списка X, Y (с номера e). Таким образом, список обрабатывается с начала и удлиняется с конца.

Поиск заканчивается либо при достижении свободной клетки на границе лабиринта, либо при исчерпании списка X, Y (когда выхода нет). Этот алгоритм реализован в программе 83.3—3.

Поиск и печать обратного пути в нерекурсивных программах выполняется одинаково, а в рекурсивной происходит автоматически.

83.4. Пила. Не требует пояснений.

83.5. Сократить дробь. Можно запрограммировать алгоритм Евклида, найти наибольший общий делитель нод (m, n) чисел m, n и сократить на него. Это приведет к программе 83.5.

Опишем суть алгоритма Евклида. Пусть $m1 \geqslant m2$. Любой общий делитель пары $(m1, m2)$ является общим делителем пары $(m2, m1-m2)$, а значит, и пары $(m2, m3)$, где

$$m3 = m1 - (m1 \backslash m2) * m2$$

является остатком от деления $m1$ на $m2$, так, что заведомо $m3<m2$. Верно и обратное: всякий общий делитель пары ($m2$, $m3$) является общим делителем пары ($m1$, $m2$). Поэтому

$$\text{нод}\,(m1,\ m2)=\text{нод}\,(m2,\ m3).$$

Последовательно заменяя больший аргумент у функции нод ($m1$, $m2$) его остатком от деления на меньший, мы получим последовательность

$$\text{нод}\,(m1,\ m2)=\text{нод}\,(m2,\ m3)=\ldots=\text{нод}\,(mK,\ 0)=mK,$$

в которой $m1\geqslant m2>\ldots>mK>0$ и mK будет наибольшим общим делителем исходных чисел $m1$, $m2$.

Впрочем, если число n не велико, то можно прямо подобрать дробь $i/j=m/n$ с минимальным знаменателем j — она, разумеется, будет несократимой. Нужно только проверять равенство дробей не по равенству отношений, а по равенству произведений:

$$i*n=j*m$$

Это приведет к программе 83.5—2.

Заметим все же, что 83.5—2 может потребовать n действий, в то время как 83.5 всегда уложится в $\log_2 n$ действий.

ОЛИМПИАДА 84

84.1. Инверсия. Очистим массив P нулями. Возьмем очередное $i=1,\ 2,\ \ldots,\ n$ и число T_i. Пройдем по элементам $P_1,\ P_2,\ldots$ до тех пор, пока не встретится T_i+1 нулевых элементов, и впишем в последний из них число i.

84.2. Дорога. Все варианты выбора чисел $i_1,\ i_2,\ \ldots,\ i_n$ перебрать невозможно — их m^n. Но есть другой путь решения задачи — индукцией по $k=1,\ 2,\ \ldots,\ n-1$.

При фиксированных числах k и i_{k+1} положим

$$B[k,\ i_{k+1}]=\min\,(A[i_1,\ i_2,\ 1]+\ldots+A[i_k,\ i_{k+1},\ k]),$$

где min берется по всевозможным наборам $i_1,\ i_2,\ \ldots,\ i_k$. Тогда, как легко видеть,

$$B[1,\ i_2]=\min\,(A[i_1,\ i_2,\ 1]) \qquad \text{по всем } i_1,$$
$$B[2,\ i_3]=\min\,(B[1,\ i_2]+A[i_2,\ i_3,\ 2]) \qquad \text{по всем } i_2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$B[n-1,\ i_n]=\min\,(B[n-2,\ i_{n-1}]+A[i_{n-1},\ i_n,\ n-1]) \qquad \text{по всем } i_{n-1}.$$

И искомое

$R = \min B|n-1,\ i_n|$ по всем i_n.

Таким образом, для вычисления одного $B|k,\ i_{k+1}|$ будет рассматриваться лишь m вариантов (выбора i_k). Для фиксированного k и всех $B|k,\ i_{k+1}|$ будет рассмотрено m^2 вариантов, а вся задача потребует рассмотрения менее $m^2 n$ вариантов.

84.3. Совершенные числа. Чтобы решить, будет ли натуральное i совершенным, можно просмотреть все числа $j = 1, 2, \ldots, i-1$, определить, какие из них являются делителями числа i, и сложить эти делители. Таким образом, возникает программа 84.3.

Можно ускорить эту программу, просматривая «кандидатов в делители» не до $i-1$, а до $\sqrt{i}$. Для этого найдем $k = i \backslash j\ (k \geqslant j)$, и если j окажется делителем числа i, то учтем не только j, но и k. Нужно лишь позаботиться, чтобы ни один делитель не был взят дважды (при $j = k$). Этот алгоритм реализован в программе 84.3--2.

84.4. Период дроби. Решение сильно зависит от дополнительных требований, не упомянутых в условии.

При делении натурального числа M на натуральное N получается целая часть частного и остаток:

$$i = M \backslash N\ :\ k = M - i*N$$

Для решения задачи сначала удалим из дроби M/N целую часть, заменив числитель M остатком:

$$M = M - M \backslash N * N$$

Теперь мы можем последовательно получать цифры частного i и остатки M получившейся дроби:

$$i = 10*M \backslash N\ :\ M = 10*M - i*N \text{ и т. д.}$$

Каждый остаток не превосходит N. Следовательно, различных остатков будет не более чем N, и они начнут повторяться. А когда повторится остаток, то начнут повторяться частные и начнется период. Чтобы уловить повторение, проще всего завести массив $D|1:N|$, вписывать в него остатки в порядке получения и каждый новый остаток сравнивать со всеми предыдущими.

Описанное решение верное, но плохое. На обработку k-го по счету остатка уйдет k сравнений, а на все остатки может уйти $N^2/2$ сравнений. Между тем на современных машинах легко взять целые M и N порядка 10^8, но на выполнение 10^{16} действий уйдут годы. Поэтому, раз мы

уже решили завести массив D на N чисел, очистим его и будем отмечать появление очередного остатка M в элементе $D[M]$. Тогда проверка «новизны» k-го остатка займет одно сравнение. Описанный алгоритм реализован в программе 84.4.

Но и это решение обладает серьезным недостатком. Число N может быть столь велико, что N действий мы еще сможем сделать, но завести в оперативной памяти массив на N элементов уже не сумеем. Попробуем отыскать период, не заводя массива для остатков (!).

Сначала, как и раньше, выделим из M часть, кратную N. Затем пропустим N цифр частного. И теперь, когда период заведомо начался, запомним один единственный остаток и будем печатать цифры частного, пока он не повторится. Это реализовано в программе 84.4—2. Любопытно, что она короче предыдущей.

84.5. Слияние массивов. Эта важная задача принципиально должна быть выполнена за $m+n$ действий. Возьмем из A и B по первому элементу. Меньший из них занесем в C и заменим следующим из его же массива. Снова выберем меньший из двух, занесем в C и т. д. После каждого сравнения в C добавляется элемент — значит, сравнений будет меньше, чем $m+n$. Нужно только позаботиться о том, чтобы программа работала верно и при исчерпании одного из массивов.

84.6. Календарь. Удобно образовать таблицу $M[1:11]$ с числом дней в месяцах (не високосного) года. Удобно использовать и функцию $MOD(m, n)$, дающую остаток от деления m на n, или аналогичную ей.

84.7. Квадратики. Если все числа, стоящие в углах квадратика, различны, то их сумма $0+1+5+11=17$. Но заметим, что верно и обратное утверждение. Это упростит программу.

ОЛИМПИАДА 85

85.1. Разложение на слагаемые. Чтобы избежать повторения разложений при перестановках слагаемых, будем рассматривать только такие разложения числа n на натуральные слагаемые

$$n=m_1+m_2+\ldots,$$

в которых $m_1 \geqslant m_2 \geqslant \ldots$

Упорядочим разложения. Пусть

$$n = m'_1 + m'_2 + \ldots, \qquad (m')$$
$$n = m''_1 + m''_2 + \ldots \qquad (m'')$$

— два разложения. Будем считать, что разложение m' предшествует разложению m'', если в первой же неравной паре $m'_i \neq m''_i$ будет $m'_i > m''_i$, т. е.

$m'_j = m''_j$ при всех $j < i$ и $m'_i > m''_i$.

По этому правилу разложения числа $n = 5$ на слагаемые расположатся в таком порядке:

$5 = 5,$
$5 = 4 + 1,$
$5 = 3 + 2,$
$5 = 3 + 1 + 1,$
$5 = 2 + 2 + 1,$
$5 = 2 + 1 + 1 + 1,$
$5 = 1 + 1 + 1 + 1 + 1.$

Чтобы получить все разложения заданного числа n на слагаемые, начнем с разложения $n = n$. Если очередное разложение

$$n = m_1 + m_2 + \ldots + m_i \qquad (*)$$

уже получено, то отпечатаем его и перейдем к построению непосредственно следующего за ним. Для этого, двигаясь по массиву справа налево, будем просматривать m_i, m_{i-1},..., m_k до первого слагаемого m_k, большего единицы. Если такого нет, то полученное разложение (*) было последним и работа закончена. Пусть слагаемое $m_k > 1$ нашлось. Уменьшив m_k на единицу и отбросив следующие за ним слагаемые-единицы, мы уменьшим всю сумму на величину $s = 1 + i - k$. Теперь, для $j = k + 1$, $k + 2$,... определим последовательно новые значения слагаемых m_j. Если $s > m_k$, то мы полагаем $m_j = m_k$ и уменьшаем s на величину m_k. Если же $s \leqslant m_k$, то полагаем $m_j = s$ и на этом заканчиваем получение нового разложения числа n на слагаемые.

85.2. Равные элементы. Надо завести массив $j[1:m]$, чтобы для каждой строки i запоминать в $j[i]$ номер столбца, до которого мы уже дошли в этой строке, и положить вначале все $j[i] = 1$. «Кандидата» в равные элементы обозначим через KR, а очередной рассматриваемый элемент массива A — через II. Вначале положим

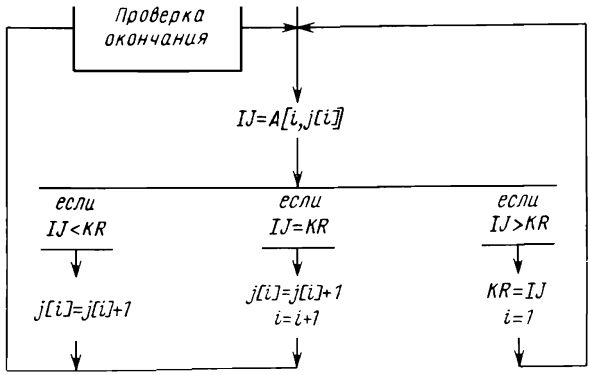


Рис. 1.5

еще $KR=A|1,\ 1|$ и $i=1$. Дальнейшее (редкий случай) стоит описать блок-схемой (рис. 1.5).

Перед присваиванием $IJ=A|i,\ j|i||$ нужно дописать проверку окончания. Если окажется, что $i>m$, то нужное значение найдено. Если же $j|i|>n$, то его нет. Этот алгоритм реализован в программе 85.2. В программе на Бейсике вместо элементов $j|i|$ использованы элементы $A(i,\ 0)$.

Но можно вместо вспомогательного столбца $j|1:m|$ завести вспомогательную строку $i0|1:n|$. Вначале в нее заносится первая строка массива A, а число «действующих» элементов $n0$ строки $i0$ полагается равным n. В дальнейшем очередная строка $i=2,\ 3,\ldots,\ m$ массива A сравнивается со строкой $i0$. В строке $i0$ оставляются (и сдвигаются в начало) лишь элементы, встретившиеся в строке i, а число $n0$ элементов, оставшихся в $i0$, принимает новое значение. Если $n0$ на каком-то шаге обнулится, то равных элементов нет, иначе искомым элементом будет $i0|1|$. Этот алгоритм реализован в программе 85.2—2. И здесь в Бейсике (и в Паскале) вместо элементов $i0|j|$ берутся элементы $A(0,\ j)$ (соответственно $A|0,\ j|$).

85.3. Несоставляемое число. Эта задача интереснее, чем может показаться на первый взгляд, ибо решается за $n*n$ (и даже за $n*\log n$) действий. Вот возможное решение. Положим $s=1$ и посмотрим, есть ли в массиве P какой-либо элемент $P|j|\leqslant s$. Если такого элемента нет, то s будет решением. Если такой элемент найдется, то добавим его значение к s, то есть положим

$s=s+P|j|$, удалим $P|j|$ из массива и снова поищем, найдется ли среди оставшихся элемент, не превосходящий нового значения s и т. д. Можно доказать по индукции (проделайте это!), что так будет найдено первое не составляемое число. Фактически на очередном $i=1, 2,...$ шаге надо просматривать элементы массива P, начиная с $P|i|$, а найденный элемент $P|j|$ заменить на элемент $P|i|$.

Для экономии числа действий (на случай наихудшего расположения чисел в массиве) можно предварительно упорядочить массив P. После этого сравнивать с s нужно будет лишь очередной элемент массива P. Описанные агоритмы реализованы в программах 85.3 и 85.3—2.

85.4. Тетраэдры. Решение разбивается на две части: на выбор грани N' тетраэдра N, для совмещения с $M1$, и на повороты тетраэдра N, оставляющие N' на $M1$. Первое делается четырьмя способами, второе — тремя. Нужно только после выбора N' выписать «собственными руками» остальные грани N в определенном порядке (скажем против часовой стрелки), начиная с какой-нибудь. Вторую часть решения надо оформить в виде функции (см. программу 85.4).

Не удовлетворительно решение, в котором непосредственно выписываются все 12 вариантов.

85.5. Мода. Можно упорядочить исходный массив, а затем, за один просмотр, подсчитать частоты значений и найти самое часто встречающееся значение. Массива для записи частот заводить не следует — достаточно помнить кандидата в самые частые и сравнивать его частоту с найденной частотой рассмотренного элемента (см. программу 85.5).

Если почему-либо желательно избежать упорядочивания, то следует поступить следующим образом. Очередной элемент $A[i]$ (вначале — это $A[1]$) сравнивается со следующими за ним. При этом, во-первых, подсчитывается частота появления значения $A[i]$, а во-вторых, элементы $A[j]$, оказавшиеся равными $A[i]$, последовательно заменяются элементами $A[i+1]$, $A[i+2]$,... После того как массив A досмотрен до конца и частота k элемента $A[i]$ определена, следующей определяется частота элемента $A[i+k]$ (а элементы между $A[i]$ и $A[i+k]$ пропускаются). Этот алгоритм выглядит эффектно и требует $m*n$ действий, если различных элементов в массиве m. Но если все элементы различны, то потребуется $n*n$ действий (см. программу 85.5—2.)

85.6. Системы счисления. Величину заданного числа следует вычислить по схеме Горнера:

$$N=((...(M|9|*i+M|8|)*i+...+M|2|)*i+M|1|.$$

Предполагается, что N не превышает максимальной величины, допустимой для целых чисел. Разряды числа N в j-ричной системе получаются как остатки от деления N на j нацело:

$$K=N\backslash j \ : \ M=N-K*j \ : \ N=K \qquad \text{и т. д.}$$

Остатки M, по условию задачи, можно не запоминать, а печатать.

85.7. Побочная диагональ. Определим два числа: $p=\max(1,\ 1-k)$ и $q=\min(n,\ m-k)$. Тогда искомая величина будет равна сумме элементов $A[k+j,\ j]$ по всем $j=p,\ p+1,...,\ q$ (она равна нулю, если таких значений j нет, т. е. если $p>q$). Не удовлетворительным следует считать решение, в котором рассматриваются все $m*n$ элементов массива A.

ОЛИМПИАДА 86

86.1. Без тройных повторений. Искомая последовательность

$$a_1,\ a_2,...,\ a_i,...$$

будет строиться по обычной схеме перебора вариантов. Нет нужды повторять здесь ее описание, приведенное в § 2.1. Сделаем нужные уточнения. Число i будет у нас номером хода, а значения $a_i=0$ и $a_i=1$ — (двумя) вариантами этого хода. Прописка и выписка здесь не нужны. Центральным пунктом программы становится проверка. Опишем ее.

После выбора (или смены) значения a_i мы будем проверять, что на полученном участке

$$a_1,\ a_2,...\ a_i \qquad (*)$$

никакой отрезок не повторяется три раза подряд. Если это условие (назовем его для краткости $NPT(i)$) не выполнено, то продолжать участок (*) бессмысленно и мы перейдем к изменению члена a_i (если уже $a_i=1$, то это приведет к изменению предыдущего члена a_{i-1}). Если же условие $NPT(i)$ выполнено, то мы перейдем к выбору следующего члена a_{i+1}.

Рис. 1.6

Но эта же проверка проводилась для участка $a_1,\ldots, a_{i-1}$. Значит, к выбору члена a_i мы пришли при условии $NPT(i-1)$, и тройной повтор на участке (*) может случиться только для отрезков, оканчивающихся на a_i. Эти отрезки имеют длину $j=1, 2,\ldots, i\backslash 3$ и каждый из них нужно сравнивать с двумя предшествующими (рис. 1.6). В результате на проверку условия $NPT(i)$ уйдет i^2 действий (с точностью до множителя).

Проверку очередного отрезка (длины j) следует прекращать на первом же повторившемся члене, а проверку всех отрезков — на первом же отрезке, повторившемся три раза подряд.

Так написана программа 86.1. Чтобы меньше загружать память машины, массив A можно взять булевским (а не числовым) и придавать его элементам значения true и false.

Если Вы и без программы можете найти искомую последовательность, то напишите программу, доставляющую первую из них (в словарном порядке).

86.2. Покер. Вот простое решение. Сосчитаем, сколько равных пар имеется в данном массиве A. Число их (оказывается!) однозначно характеризует комбинацию. Для подсчета числа равных пар мы обнуляем счетчик s, сравниваем каждый элемент $A[i]$ со всеми последующими $A[j]$ для $j=i+1, i+2,\ldots$ и добавляем к s по 1 в случае $A[i]=A[j]$.

Нужный для печати ответ p получается по значению счетчика s по таблице:

s	p	s	p
$4+3+2+1=10$	1	$1+1=2$	5
$3+2+1=6$	2	$1=1$	6
$2+1+1=4$	3	0	7
$2+1=3$	4		

Кроме первых двух случаев, $p=7-s$.

86.3. Барабан. Простейшим представляется такое решение. Пусть k — номер вектора «кандидата в ми-

нимальные», p — номер очередного вектора. Вначале $k=1$ и $p=2$.

Если в какой-то момент выясняется:

(1) что $X_k \leqslant X_p$, то p увеличивается на 1;

(2) что $X_k > X_p$, то k заменяется на p и p увеличивается на 1.

Если после этого $p \leqslant 12$, то надо перейти к сравнению новой пары векторов X_k и X_p. Если же $p > 12$, то число k является ответом. Этот алгоритм реализован в программе 86.3.

Возможно и другое решение. Оно может оказаться (в некоторых случаях) более быстрым. Заведем дополнительный массив $L[1:12]$ для списка кандидатов (точнее — для списка номеров, с которых начинаются кандидаты) в минимальные векторы. Вначале это будут все номера от 1 до 12. Затем мы оставим из них те, у которых первый элемент минимален, из них — те, у которых минимален второй, и т. д., пока не останется один кандидат или не будут рассмотрены все 12 элементов у кандидатов в минимальные векторы. Программа 86.3—2 много сложнее предыдущей; чтобы облегчить ее чтение, договоримся об обозначениях:

$i+1$ — номер элемента, рассматриваемого в векторе $(0 \leqslant i \leqslant 11)$;

1, j, jn — номера первого, очередного и последнего кандидатов (в списке L);

$L[j]$ — собственный номер кандидата;

q — номер последнего перенесенного кандидата (в списке L).

86.4. Дважды монотонный. Проще всего положить $i=1$, $j=n$ и сравнить (сначала этот, а затем очередной) элемент $A[i, j]$ с числом x:

если $A[i, j]=x$, то ответ найден,
если $A[i, j]<x$, то положить $i=i+1$,
если $A[i, j]>x$, то положить $j=j-1$.

В двух последних случаях надо проверить, что осталось $i \leqslant m$ и $j \geqslant 1$. Если это так, то вернуться к сравнениям, а если нет — то напечатать НЕТ.

На каждом шаге мы либо увеличиваем i, либо уменьшаем j. Следовательно, шагов будет не более $m+n$.

Нетрудно убедиться в верности этого алгоритма. Действительно. Если оказалось, что $A[i, j]<x$, то числа x нет в строке i массива $A[i:m, 1:j]$ и эту строку можно

отбросить. Если же $A|i, j| > x$, то по аналогичной причине можно отбросить столбец j.

86.5. Центральное селение. Сформулируем эту задачу следующим образом. В каждой строке i массива A выберем максимальное среди чисел $A|i, j|$ $(j \neq i)$ и сложим его с $A|i, i|$. Найти i, при котором соответствующая сумма будет минимальна. В такой формулировке задача допускает стандартное решение.

ОЛИМПИАДА 87

87.1. Рюкзак. После того как предметы, весящие более 30 кг, удалены, а остальные расположены в каком-либо порядке, определим дерево вариантов следующим образом.

На очередном ходе $i=1, 2, \ldots, n$ будем рассматривать предмет с номером i, а вариантов j хода i всегда будет два: $j=0$ означает брать предмет, а $j=1$ — не брать его (по поводу терминологии см. § 2.1). Можно заметить, что у нас получилось двоичное дерево, все ветви которого имеют длину n (но это понимать не обязательно).

Кроме заданных массивов $A[1:n]$ и $B[1:n]$ заведем еще массив $P[1:n]$ и несколько переменных:

i — номер очередного предмета,

S — вес предметов в рюкзаке,

Z — суммарная стоимость предметов в рюкзаке,

ZM — максимальная стоимость рассмотренных вариантов,

$P[k]=0$ или $P[k]=1$, если предмет $k \leqslant i$ взят или не взят в рюкзак.

Вначале i, S, Z, ZM обнуляются.

При рассмотрении вариантов важно прекращать перебор, как только станет ясно, что он (и все его продолжения) не представляют интереса!

При движении *вперед* мы пытаемся добавить предмет в рюкзак (если $S+A[i]<30$). В этом случае мы идем по левой ветке:

$$S=S+A[i] \ : \ Z=Z+B[i] \ : \ P[i]=0$$

Если же предмет добавить нельзя, то мы его не берем (т. е. движемся по правой ветке, отбрасывая все дерево вариантов, идущих влево) и отмечаем $P[i]=1$. В обоих случаях продолжаем двигаться вперед, пока не будет рассмотрен последний предмет.

Если все предметы рассмотрены, то вариант получен. Он сравнивается с ZM

IF $ZM<Z$ THEN $ZM=Z$

и начинается движение *назад*.

При движении назад пропускается вся группа идущих подряд взятых предметов (у них $P[i]=0$), поскольку изменения в одной этой группе могут лишь снизить суммарную цену предметов в рюкзаке. Просмотренные предметы попутно удаляются из рюкзака:

IF $P[i]=0$ THEN $S=S-A[i]$: $Z=Z-B[i]$

Далее пропускается вся группа не взятых ранее предметов (у них $P[i]=1$), ибо изменение в этой группе приводит к левой ветке, которая должна была быть оценена раньше.

Короче говоря, мы движемся назад до достижения такого номера i, что $P[i]=0$ и $P[i+1]=1$. При этом движении из рюкзака удаляются имеющиеся там предметы. После этого мы движемся вперед. Если же нужного i не окажется, то работа закончена.

Переборную программу почти всегда можно несколько улучшить. Можно, например, ввести переменную ZS для суммы цен предметов, взятых в рюкзак, и предметов, еще не рассмотренных (на данной ветке). Это позволит отбрасывать ветку, как только обнаружится, что $ZS\leqslant ZM$. Предоставим сделать это читателю самостоятельно.

87.2. Полукратные. Мы будем получать и запоминать в массиве $A[1 : n]$ «полукратные» в порядке возрастания. Пусть уже получены первые i полукратных:

$$A[1]<\ldots<A[i] \qquad (*)$$

и пусть известны $K2$ и $K3$ — минимальные номера чисел, входящих в (*), для которых

$A2=2*A[K2]+1$ и $A3=3*A[K3]+1$

уже не входят в (*).

Посмотрим, что произойдет при добавлении к (*) еще одного члена. Ясно, что

$A[i+1]=\mathrm{MIN}(A2, A3)$.

Если $A[i+1]=A2$, то $K2$ увеличивается на 1; если же $A[i+1]=A3$, то $K3$ увеличивается на 1 (если

$A[i+1]=A2=A3$, то на 1 увеличиваются оба числа $K2$ и $K3$).

Вначале $i=1$, $A[1]=1$, $K2=1$, $K3=1$. Следующие члены $A[2]$, $A[3]$, ... вычисляются описанным алгоритмом.

87.3. Сумма кубов. Возьмем $M=0$, $I-1$, а J — первое число, для которого $J\wedge 3+1\geqslant N$. Положим $K=I\wedge 3+J\wedge 3$ и сравним K с N. Возможны три случая:

(1) если $K<N$, то сделаем $I=I+1$;

(2) если $K>N$, то сделаем $J=J-1$;

(3) если $K=N$, то сделаем $M=M+1$: $J=J-1$: $I=I+1$

Эти вычисления повторяются, пока $I<=J$. Можно еще заметить, что в случае (3) вместо $I=I+1$ удобно сразу написать $I=I+2$.

Поскольку вначале $I\leqslant J\leqslant N\wedge(1/3)+1$ и на каждом шаге либо увеличивается I либо уменьшается J, то шагов будет не больше, чем $N\wedge(1/3)$.

Для доказательства верности алгоритма обозначим через

$$P(I, J)=\{(x, y)\}$$

множество таких пар натуральных чисел x, y, для которых

$$I<=x<=y<=J \text{ и } x\wedge 3+y\wedge 3=N.$$

Положим $K=I\wedge 3+J\wedge 3$. Если $K<N$, то для любой пары (x, y) из $P(I, J)$ будет

$$I\wedge 3+J\wedge 3<x\wedge 3+y\wedge 3,$$

но $y<=J$, следовательно, $I<x$ и, значит,

если $K<N$, то $P(I, J)=P(I+1, J)$.

Аналогично,

если $K>N$, то $P(I, J)=P(I, J-1)$.

Наконец, при $K=N$, после удаления пары (I, J) из $P(I, J)$, будем иметь

если $K=N$, то $P(I, J)\backslash(I, J)=P(I+1, J-1)$.

87.4. Переверты ши. Эта задача требует определенной сообразительности (чтобы уложиться в N^2 действий).

Исследуемый отрезок нужно задать его центром (это даст N вариантов) и в обе стороны от центра сравнивать симметричные элементы (это требует не больше, чем N групп действий). Таким образом, решение и будет найдено за N^2 действий (с точностью до постоянного множителя).

При оформлении решения вместо центра отрезка (который может прийтись и на элемент и между элементами) удобно взять номера двух элементов $LN<PN$, соседних с центром. Эти номера будут передвигаться понятным образом:

$$(LN,\ PN)=(1,\ 2),\ (1,\ 3),\ (2,\ 3),\ (2,\ 4),\ \ldots$$

Что касается номеров L и P сравниваемых элементов, то вначале берется

$$L=LN\ :\ P=PN$$

и затем на каждом шаге делается

$$L=L-1\ :\ P=P+1$$

пока сравниваемые элементы $A|L|$ и $A|P|$ станут неравными, или одна из точек L или P выйдет за границы $(1,\ N)$.

Тогда максимальный отрезок с «центром» $(LN,\ PN)$ будет длины

$$M=P-L-1,$$

и эту длину можно будет сравнивать с максимально достигнутой MX.

Передвижение «центра» $(LN,\ PN)$ к правому краю массива $A[1:N]$ можно прекратить, если уже выполнено неравенство

$$2*(N-PN+1)<=MX$$

и, значит, рассматриваемый отрезок уже не сможет увеличить значения величины MX.

87.5. Индексы порядка. Проще всего завести дополнительно массив $I[1:N]$ для искомых «индексов порядка», заполнить его вначале числами 1, 2, ..., N, а затем, упорядочивая заданный массив A каким-либо способом, переставлять в том же порядке элементы массива I. Когда A будет упорядочен, I станет искомым. Это сделано в программе 87.5, где использован простейший метод упорядочивания: в массиве A находится минимальный элемент и этот элемент меняется местами с первым. Потом та же процедура применяется к отрезку $A[2:N]$ и т. д.

Если пойти на усложнение программы, то можно не только обойтись без массива I, но и не изменять элементов массива A. Достаточно печатать и запоминать

индекс очередного минимального элемента (MI). Придется, правда, заранее найти индекс первого такого элемента и индекс первого максимального элемента (MA). Это сделано в программе 87.5—2.

ОЛИМПИАДА 88

88.1. Правый бóльший. Для решения используются два соображения. Во-первых, массив $A[1:n]$ просматривается с конца; во-вторых, заводится вспомогательный массив $B[1:n]$ из тех элементов массива A, которые могут понадобиться при дальнейшем просмотре массива A для замены его элементов. Точнее — к моменту рассмотрения очередного элемента $A[i]$ в массиве B будут находиться те элементы $A[j]$ ($j>i$), которые больше всех элементов $A[i+1 : j-1]$. Элементы в массиве B будут располагаться в том же порядке, в каком они были в A, но будут сдвинуты к правому краю, заполняя некоторый отрезок $B[k:n]$.

Массивы A и B обрабатываются одновременно. Вначале полагаем

$$k=n \ : \ B[k]=A[n] : A[n]=0$$

При рассмотрении очередного элемента $A[i]$ все элементы на участке $A[i+1 : n]$ уже заменены и массив $B[k:n]$ заполнен нужным образом. В массиве $B[k:n]$ ищется первый элемент $B[j]>A[i]$. Если такой номер j нашелся, то делается

$$k=j-1 : B[k]=A[i] : A[i]=B[j]$$

Если же такого номера j нет, то полагается

$$k=n : B[k]=A[i] : A[i]=0$$

В обоих случаях новый отрезок $B[k:n]$ сохраняет объявленные свойства.

Для оценки числа действий алгоритма заметим, что при каждом сравнении элемента $A[i]$ с элементом $B[j]$ либо элемент $A[i]$ заносится в B, либо элемент $B[j]$ удаляется из B. Таким образом, каждый элемент массива A порождает одно сравнение первого типа (когда он заносится в B) и не более одного — второго типа (когда он удаляется из B). Значит, сравнений не более чем $2n$, а действий порядка n.

88.2. Многочлен. Удобно положить $P_0(x)=1$ и обозначить через $P_m(x)$ произведение первых m скобок

$$P_m(x)=(x-x[1])*...*(x-x[m]),$$

определяющее многочлен

$$P_m(x)=a(m,\ 0)+a(m,\ 1)*x+... \\ ...+a(m,\ m-1)*x\wedge(m-1)+x\wedge m.$$

Очевидно,

$$P_m(x)=P_{m-1}(x)*(x-x[m])$$

и коэффициенты $a(m,\ i)$ выразятся через коэффициенты $a(m-1,\ i)$ следующим образом:

$$a(m,\ m-1)=a(m-1,\ m-2)-x[m]$$
$$a(m,\ i)=a(m-1,\ i-1)-a(m-1,\ i)*x[m]$$
$$a(m,\ 0)=-a(m-1,\ 0)*x[m]$$

Если располагать коэффициент $a(m,\ i)$ в элементе $A[i]$ массива A, то переход от коэффициентов многочлена P_{m-1} к коэффициентам многочлена P_m будет выполняться операторами (при $m>1$):

$A[m-1]=A[m-2]-x[m]$
$A[i]=A[i-1]-A[i]*x[m]$ для $i=m-2,\ ...,\ 1$
$A[0]=-A[0]*x[m]$

Это и реализовано в программе 88.2. Она содержит двойной цикл — внешний по m от 1 до n и внутренний по i от $m-1$ до 0, с поправками для начальных и конечных значений.

Можно было бы оформить программу иначе — с внешним циклом по i от $n-1$ до 0 и внутренним — по m от $n-i$ до n.

88.3. Простые делители. Будем менять число $i=2,\ 3,\ ...$ до тех пор, пока i не станет делителем числа N. Поделим N на i

$$N=N/i$$

и будем повторять деление, пока N делится на i. После этого, если $N>1$, перейдем к следующему значению i и т. д. Нетрудно видеть, что таким образом мы найдем все простые делители исходного числа N и только их.

Введем еще одну переменную j, равную вначале 0, а в дальнейшем — последнему найденному делителю. Новый делитель в первый раз будет отличен от j и вызовет печать.

Чтобы ускорить программу, можно отдельно рассмотреть случай $i=2$, а в дальнейшем двигаться по нечетным значениям i. Читатель это может сделать самостоятельно.

88.4. Оптовая покупка. Эту задачу часто решают неверно. Приведем верное решение и тестовые примеры для обнаружения ошибок в неверных решениях.

Оптимальная покупка без излишков находится очевидным образом:

$$n1=n\backslash 144 \ : \ m=n-n1*144$$
$$n2=m\backslash 12 \ : \ n3=m-n2*12$$

Удешевить ее можно лишь двумя способами — либо взять лишнюю связку и не брать пар, либо лишнюю коробку и не брать ни связок, ни пар. Если $n3*1.05>10.25$, то надо взять лишнюю связку:

$$n2=n2+1 \ : \ n3=0$$

Если для получившейся покупки (старой или изменившейся) будет $n2*10.25+n3*1.05>114.00$, то надо сделать

$$n1=n1+1 \ : \ n2=0 \ : \ n3=0$$

Это решение пригодно для любых цен (при которых коробка дешевле, чем 12 связок, а связка — чем 12 пар).

Тестовые примеры

n	$n1$	$n2$	$n3$
9	0	0	9
10	0	1	0
131	0	11	0
134	1	0	0

88.5. Обнуление. Проще всего завести два вспомогательных массива $B[1 : m]$ и $C[1 : n]$. Далее, просматривая элементы массива A, отметим в массивах B и C строки и столбцы, где встретились нули, то есть сделаем

if $A[i, j]=0$ then $B[i]=1 \ : \ C[j]=1$

Теперь при повторном проходе массива A заменим нулями те элементы, которые стоят в строках или столбцах, имеющих отметки в массивах B или C, т. е.

if $B[i]=1$ or $C[j]=1$ then $A[i, j]=0$

Можно предложить другое решение, использующее лишь один вспомогательный массив $C[1 : n]$. В этом случае массив A просматривается по строкам. Столбцы, где встретились нулевые элементы по-прежнему отмечаются в массиве C, а встретился ли нуль в просматриваемой строке отмечается в переменной z, т. е.

if $A[i, j]=0$ then $C[j]=1 : z=1$

После просмотра строки i она обнуляется, если z стало равно 1. В заключение просматривается массив C и обнуляются столбцы j, где $C[j]=1$. Эта программа приведена в 88.5.

Типичные ошибки решения этой задачи следующие.

(1) Просматривается массив A, и если элемент $A[i, j]=0$, то обнуляется или строка i, или столбец j, или и то и другое. При дальнейшем просмотре обнуленные элементы принимаются за исходно равные нулю и вызывают излишние обнуления.

(2) Просматривается массив A, и если $A[i, j]=0$, то в очередной элемент $C[k]$ заносится j. От этого один и тот же номер j может заноситься многократно и массив C может потребоваться длины $m*n$ вопреки условию.

§ 1.3. Решения на Бейсике

Бейсик определен не совсем строго, и его наречия немного различаются. Поэтому мы дадим небольшие пояснения к тому наречию, на котором написаны программы в этой книжке.

(1) В одной строке можно писать несколько операторов, разделяя их двоеточиями (:). (2) В условном операторе после then можно написать несколько операторов, разделив их двоеточиями. Все они будут выполняться (или не выполняться) в зависимости от выполнения условия. То же относится к операторам после else. (3) Число элементов массива и значения, определяющие цикл, могут быть выражениями. (4) Действия + — * над не слишком большими целыми величинами выполняются точно, даже если эти величины определены с плавающей точкой. (5) Деление нацело обозначается обратной косой чертой (\), причем у операндов предварительно отбрасываются дробные части. (6) Большие и малые буквы считаются одинаковыми (написанными разными почерками). Мы этим пользуемся, чтобы оттенить разный характер величин, но не пишем одну и ту же букву по-разному в одном имени. (7) Все переменные и (массивы) вначале автоматически обнуляются. Это мы используем (иногда). (8) Длинную строчку программы мы записываем в несколько строк, как бы с пробелами, но так, чтобы это было понятно.

ОЛИМПИАДА 80

```
 10 'Простые до M                                          OLI 80.1.1
 20 input "m=";m
 30 if m>1 then print 2;
 40 if m>2 then print 3;
 50 if m<5 goto 160
 60 n=sqr(m)\2+1:dim P(n)
 70 k=1:P(1)=3
 80 for i=5 to m step 2
 90   for j=1 to k:q=P(j)
100     if i mod q = 0 goto 150
110     if q*q>i goto 130
120   next j
130   print i;:if k>=n goto 150
140   k=k+1:P(k)=i
150 next i
160 end
```

```
 10 'Перестановки                                          OLI 80.1.2
 20 input "m=";m:dim A(m),P(m)
 30 for i=1 to m
 40   input A(i):P(i)=i
 50 next:if m>1 goto 170
 60 print A(1):goto 190
 70 for i=m-1 to 1 step -1
 80   if P(i)<P(i+1) goto 100
 90 next i:goto 190
100 for j=m to i step -1
110   if P(i)<P(j) goto 130
120 next j
```

```
130 swap P(i),P(j)
140 for k=1 to (m-i)\2
150   swap P(i+k),P(m+1-k)
160 next k
170 for i=1 to m:print A(P(i));:next
180 print:goto 70
190 end
```

```
 10 'Перестановки                                              OLI 8
 20 input "m=";m
 30 dim P(m),C(m),Z(m)
 40 for i=1 to m
 50   P(i)=i:C(i)=1:Z(i)=1
 60 next i:C(m)=-1
 70 goto 110
 80 k=C(i)+x
 90 swap P(k),P(k+1)
100 C(i)=C(i)+Z(i)
110 for j=1 to m:print P(j);:next
120 print
130 i=1:x=0:goto 170
140   Z(i)=-Z(i):C(i)=C(i)+Z(i)
150   if Z(i)>0 then x=x+1
160   i=i+1
170 if C(i)>m-i or C(i)=0 goto 140
180 if i<m goto 80
190 end
```

```
 10 'Быстрая степень                                           OLI
 20 input "a,k=";a,k
 30 print a;"^";k;
 40 b=1
 50   if k mod 2 = 0 goto 70
 60   b=b*a
 70   k=k\2:a=a*a
 80 if k>0 goto 50
 90 print "=";b
100 end
```

```
 10 'Арифметические действия                                   OLI
 20 m=35:n=6
 30 dim A(n),B(n)
 40 B(1)=1:k=0:A(2)=0
 50 for i=3 to n:A(i)=4:next
 60 for i=n to 2 step -1
 70   if A(i)=4 then A(i)=1:goto 300
 80   A(i)=A(i)+1:y=B(i-1)
 90   on A(i) goto 100,110,120,130
100   z=y+i:goto 140
110   z=y-i:goto 140
120   z=y*i:goto 140
130   z=y\i
140   B(i)=z:if i=n goto 160
150   for j=i+1 to n:B(j)=B(j-1)+j:next
160   if B(n)<>m goto 60
```

```
170   k=k+1
180   for j=2 to n-1:print "(";:next
190   print 1;
200   for j=2 to n
210     if j>2 then print ")";
220     on A(j) goto 230,240,250,260
230     print "+";:goto 270
240     print "-";:goto 270
250     print "*";:goto 270
260     print "\";
270     print j;
280   next j:print "=";m
290   goto 60
300 next i:print k
310 end
```

OLI

```
 10 'Поиск равных
 20 dim A(2,15)
 30 for i=1 to 2:for j=1 to 15
 40   input A(i,j)
 50 next j,i
 60 for i=1 to 2:for j=1 to 15
 70   for p=i to 2
 80     if i<p then q=1 else q=j+1
 90     goto 120
100       if A(i,j)=A(p,q) goto 150
110       q=q+1
120     if q<=15 goto 100
130   next p
140 next j,i
150 print i;j;p;q
160 end
```

OLI

```
 10 'Сумма квадратов
 20 defint a-z
 30 input m
 40 n=sqr(m/2)+0.1
 50 for i=1 to n
 60   j=sqr(m-i*i)+0.1
 70   if i*i+j*j=m goto 100
 80 next i
 90 print "НЕТ":goto 110
100 print i;j;m
110 end
```

OLI

```
 10 'Различные числа
 20 input "m=";m:dim A(m)
 30 k=1:if m=1 goto 100
 40 for i=1 to m:input A(i):next
 50 for i=2 to m
 60   for j=1 to i-1
 70     if A(i)=A(j) goto 90
 80   next j:k=k+1
 90 next i
100 print k
```

```
110 end
```

```
 10 'Заданная сумма цифр                                        OLI
 20 input "n=";n
 30 for i=0 to 9:for j=0 to 9
 40   k=n-i-j
 50   if 1<=k and k<=9
      then print i+10*j+100*k
 60 next j,i
 70 end
```

```
 10 'M+1 в двоичной записи                                      OLI
 20 input "n=";n
 30 p=1
 40 for i=0 to n-1
 50   input j
 60   if p=0 then print j
             else print 1-j
 70   if j=0 then p=0
 80 next i
 90 if p=1 then print 1
100 end
```

```
 10 'Максимум минимумов                                         OLI
 20 input "m,n=";m,n:dim X(m,n)
 30 for i=1 to m:for j=1 to n
 40   input X(i,j)
 50 next j,i
 60 for i=1 to m:for j=1 to n
 70   if i>1 and X(i,j)<=ma goto 100
 80   if j=1 or X(i,j)<mi then mi=X(i,j)
 90 next j:ma=mi:k=i
100 next i:print k
110 end
```

```
 10 'Перестановка 0,1,2                                         OLI
 20 input "n=";n:dim X(n)
 30 for i=1 to n:input X(i):next
 40 a1=0:a2=0
 50 for i=1 to n
 60   if X(i)=2 then a2=a2+1
                else a1=a1+X(i)
 70 next i
 80 a0=n-a1-a2:a1=a0+a1
 90 for i=1 to n
100   if i<=a0 then X(i)=0 else
      if i<=a1 then X(i)=1 else
                    X(i)=2
110 next i
120 for i=1 to n:print X(i);:next
130 end
```

```
 10 'Функция                                                    OLI
 20 input "n=";n
 30 i=1:j=0
 40   if n mod 2 then j=i+j else i=i+j
 50   n=n\2
 60 if n>0 goto 40
 70 print j
 80 end
```

```
 10 'Пара четверок                                              OLI
 20 n=4
 30 b=0:i=1:mi=1
 40 j=1:mj=mi+1
 50 k=1:mk=mj+1
 60 p=sqr(n-mk):p=int(p+0.1)
 70 mp=mk+p*p
 80 if p>k or mp<>n goto 110
 90 if b goto 180
100 b=1:il=i:jl=j:kl=k:pl=p
110 k=k+1:mk=mj+k*k
120 if k<=j and mk<n goto 60
130 j=j+1:mj=mi+j*j
140 if j<=i and mj<n-1 goto 50
150 i=i+1:mi=i*i
160 if mi<n-2 goto 40
170 n=n+1:goto 30
180 print n
190 print i;j;k;p;i*i+j*j+k*k+p*p
200 print il;jl;kl;pl;il*il+jl*jl+kl*kl+pl*pl
210 end
```

```
 10 'Спираль                                                    OLI
 20 input "n=";n
 30 dim A(n,n)
 40 m=n*n:k=1:i=1:j=1
 50 gosub 200:j=j+1
 60 if j<=n-i goto 50
 70 gosub 200:i=i+1
 80 if i<j goto 70
 90 gosub 200:j=j-1
100 if j>n-i+1 goto 90
110 gosub 200:i=i-1
120 if i>j+1 then 110 else 50
130 for i=1 to n:for j=1 to n
140   print using "###";A(i,j);
150 next j:print:next i
160 end

200 A(i,j)=k:k=k+1
210 if k<=m then return
220 return 130
```

```
 10 'Числа из разных цифр                                        OL
 20 for i=1 to 9
 30 for j=0 to 9: if j=i goto 90
 40 for k=0 to 9: if k=i or k=j goto 80
 50 for m=0 to 9: if m=i or m=j or m=k goto 70
 60 print ((i*10+j)*10+k)*10+m;
 70 next m
 80 next k
 90 next j,i
100 end
```

```
 10 'Серия нулей                                                 OL
 20 input "n="; n
 30 dim A(n)
 40 for i=1 to n
 50   print "A("; i; ")="; : input A(i)
 60 next i
 70 k=0: m=0
 80 for i=1 to n
 90   if A(i)<>0 then k=0 else
      k=k+1: if k>m then m=k
100 next i
110 print m
120 end
```

ОЛИМПИАДА 82

```
 10 'Прямоугольники                                              OLI
 20 input "m, n="; m, n
 30 dim A(m, n)
 40 for i=0 to m: for j=0 to n
 50   if i*j=0 then A(i, j)=0
      else print i; j; : input A(i, j)
 60 next j, i: k=0
 70 for i=1 to m: for j=1 to n
 80   if A(i, j) and A(i-1, j)+A(i, j-1)=0
      then k=k+1
 90 next j, i: print "k="; k
100 end
```

```
 10 'Упорядоченные дроби                                         OL
 20 p=7
 30 m=0: n=1: goto 50
 40   print m; "/"; n; "="; m/n
 50   i=1: j=1
 60   for b=2 to p
 70     a=(m*b)\n+1
 80     if a*j<b*i then i=a: j=b
 90   next b
100 if i<j then m=i: n=j: goto 40
110 end
```

```
 10 'Сумма по подмножеству                                       OL
 20 input "m, n="; m, n
```

```
 30 dim A(n),B( )
 40 for i=1 to n:input A(i):next
 50 for i=1 to n
 60   B(i)=1-B( )
 70   if B(i) goto 100
 80   s=s-A(i)
 90 next i
100 s=s+A(i):if s<>m goto 50
110 for i=1 to n
120   if B(i) then print i;
130 next i
140 end
```

```
 10 'Нули - в конец                                          OLI
 20 input "n=";n
 30 dim X(n):j=0
 40 for i=1 to n:input X(i):next
 50 for i=1 to n
 60   if X(i)=0 goto 90
 70   j=j+1
 80   if j<i then X(j)=X(i):X(i)=0
 90 next i
100 for i=1 to n:print X(i);:next
110 end
```

```
 10 'Седловая точка                                          OLI
 20 input "m,n=";m,n
 30 dim A(m,n)
 40 for i=1 to m:for j=1 to n
 50   input A(i,j)
 60 next j,i
 70 for i=1 to m:for j=1 to n
 80   if i>1 and A(i,j)<=Ma goto 110
 90   if j=1 or A(i,j)<mi then mi=A(i,j)
100 next j:Ma=mi:i0=i
110 next i
120 for j=1 to n:for i=1 to m
130   if A(i,j)>Ma goto 150
140 next i:print i0;j:goto 160
150 next j:print 0
160 end
```

```
 10 'Вхождение слова в текст                                 OLI
 20 input "n,k=";n,k
 30 dim X(n),Y(k)
 40 for i=1 to n:input X(i):next
 50 for j=1 to k:input Y(j):next
 60 if n<k goto 110
 70 for i=0 to n-k:for j=1 to k
 80   if X(i+j)<>Y(j) goto 100
 90 next j:print "ДА";i+1:goto 120
100 next i
110 print "НЕТ"
120 end
```

```
 10 'Бит-реверс
 20 for m=512 to 1023
 30   n=m: b=0
 40   k=512
 50     if n<k goto 70
 60     n=n-k: b=b+512/k
 70   k=k/2: if n goto 50
 80   print b;
 90 next m
100 end
```

```
 10 'Бит-реверс
 20 b=1: goto 60
 30     b=b-k: k=k/2
 40   if b>=k goto 30
 50   b=b+k
 60   print b,: k=512
 70 if b<1023 goto 40
 80 end
```

```
 10 'Треугольник и точка
 20 dim x(3), y(3)
 30 def fnf(i, j)=
      (x-x(i))*(y(j)-y(i))>(x(j)-x(i))*(y-y(i))
 40 for i=1 to 3
 50   input x(i), y(i)
 60 next i: input x, y
 70 t=fnf(1, 2)
 80 if t=fnf(2, 3) and t=fnf(3, 1)
      then print "внутри" else print "вне"
 90 end
```

```
 10 'Лабиринт
 20 input "m, n="; m, n
 30 dim A(m, n)
 40 input "i, j="; i, j
 50 for p=1 to m: for q=1 to n
 60   input A(p, q)
 70 next q, p: k=2: A(i, j)=2
 80 if i=1 or i=m or j=1 or j=n goto 190
 90 p=i: q=j
100 for i=p-1 to p+1: for j=q-1 to q+1
110   if i<>p and j<>q or A(i, j) goto 130
120   k=k+1: A(i, j)=k: goto 80
130 next j, i
140 if k=2 then print "нет выхода": goto 270
150 A(p, q)=1: k=k-1
160 for i=p-1 to p+1: for j=q-1 to q+1
170   if A(i, j)=k goto 90
180 next j, i
190 for k=k to 2 step -1
200   print k; i; j
210   p=i: q=j
```

```
220   for i=p-1 to p+1:for j=q-1 to q+1
230     if i<1 or i>m or j<1 or j>n goto 250
240     if A(i,j)=k-1 goto 260
250   next j,i
260 next k
270 end
```

```
 10 'Лабиринт                                                    OLI
 20 input "m,n=";m,n
 30 dim A(m,n),X(m*n),Y(m*n)
 40 input "i,j=";i,j
 50 for p=1 to m:for q=1 to n
 60   input A(p,q)
 70 next q,p
 80 A(i,j)=2:X(1)=i:Y(1)=j
 90 if i=1 or i=m or j=1 or j=n goto 200
100 e=1:b=1
110   p=X(b):q=Y(b)
120   for i=p-1 to p+1:for j=q-1 to q+1
130     if i<>p and j<>q or A(i,j) goto 170
140     A(i,j)=A(p,q)+1
150     if i=1 or i=m or j=1 or j=n goto 200
160     e=e+1:X(e)=i:Y(e)=j
170   next j,i
180 b=b+1:if b<=e goto 110
190 print "нет выхода":goto 250
200 print A(i,j);i;j:if A(i,j)=2 goto 250
210   p=i:q=j
220   for i=p-1 to p+1:for j=q-1 to q+1
230     if (i=p or j=q) and
            1<=i and i<=m and 1<=j and j<=n
        then if A(i,j)=A(p,q)-1 goto 200
240   next j,i
250 end
```

```
 10 'Пила                                                        OL
 20 input "m=";m
 30 dim X(m)
 40 for i=1 to m:input X(i):next
 50 i=1:j=1:k=1:if m<3 goto 120
 60   if X(i)<X(i+1) and X(i+1)>X(i+2) goto 90
 70   i=i+1:j=1
 80 if i+k<m then 60 else 120
 90   i=i+2:j=j+2
100   if k<j then k=j
110 if i+1<m then 60
120 print k
130 end
```

```
 10 'Сократить дробь                                             OL
 20 input "m,n=";m,n
 30 i=m:j=n
 40   k=j mod i:j=i:i=k
 50 if i>0 goto 40
 60 i=m/j:j=n/j
```

```
 70 print m;"/";n;"=";i;"/";j
 80 end
```

```
 10 'Сократить дробь                                          OLI
 20 input "m,n=";m,n
 30 for j=1 to n
 40   i=j*m\n
 50   if i*n=j*m goto 70
 60 next j
 70 print m;"/";n;"=";i;"/";j
 80 end
```

ОЛИМПИАДА 84

```
 10 'Инверсия                                                 OL
 20 input "n=";n
 30 dim P(n),T(n)
 40 for i=1 to n
 50   input T(i):P(i)=0
 60 next i
 70 for i=1 to n
 80   j=0:k=0
 90     k=k+1
100     if P(k)=0 then j=j+1
110   if j<=T(i) then 90 else P(k)=i
120 next i
130 for i=1 to n:print P(i);:next
140 end
```

```
 10 'Дорога                                                   OL
 20 input "m,n=";m,n
 30 dim A(m,m,n-1),B(m),C(m)
 40 for i=1 to m:for j=1 to m:for k=1 to n-1
 50   input A(i,j,k)
 60 next k,j:B(i)=0:next i
 70 for k=1 to n-1
 80   for j=1 to m
 90     r=B(1)+A(1,j,k)
100     for i=1 to m
110       x=B(i)+A(i,j,k)
120       if x<r then r=x
130     next i:C(j)=r
140   next j
150   for j=1 to m:B(j)=C(j):next j
160 next k:r=B(1)
170 for i=1 to m:if B(i)<r then r=B(i):next
180 print r
190 end
```

```
 10 'Совершенные числа                                        OL
 20 input "введите m>3";m
 30 for i=3 to m-1:s=1
 40   for j=2 to i-1
 50     if i mod j = 0 then s=s+j
```

```
 60   next j:if i=s then print i
 70 next i
 80 end
```

OLI 84.3-2

```
 10 'Совершенные числа
 20 input "введите m>3";m
 30 for i=3 to m-1
 40   s=1:j=1
 50   j=j+1:k=i\j
 60   if i<>j*k or j>k goto 80
 70   s=s+j:if j<k then s=s+k
 80   if j<k goto 50
 90   if i=s then print i
100 next i
110 end
```

OLI 84.4

```
 10 'Период дроби
 20 input "m,n=";m,n:dim D(n)
 30 m=m mod n:b=0
 40 for i=0 to n-1:D(i)=0:next
 50   D(m)=1:m=10*m
 60     i=m\n:m=m mod n
 70     if b=1 then print i;
 80   if D(m)=0 goto 50
 90 if b=0 then b=1:goto 40
100 end
```

OLI 84.4-2

```
 10 'Период дроби
 20 input "m,n=";m,n
 30 m=m mod n
 40 for k=1 to n:m=10*m mod n:next
 50 j=m
 60 m=10*m:print m\n;
 70 m=m mod n:if m<>j goto 60
 80 end
```

OLI 84.5

```
 10 'Слияние массивов
 20 input "m,n=";m,n
 30 dim A(m),B(n),C(m+n)
 40 for i=1 to m:input A(i):next
 50 for j=1 to n:input B(j):next
 60 i=1:j=1
 70 for k=1 to m+n
 80   if i>m goto 110
 90   if j>n goto 120
100   if A(i)<B(j) goto 120
110   C(k)=B(j):j=j+1:goto 130
120   C(k)=A(i):i=i+1
130   print C(k);
140 next k
150 end
```

OLI 84.6

```
 10 'Календарь
 20 def fnv(x)=((c mod x)=0)
 30 dim M(11)
```

```
 40 for i=1 to 10: M(i)=31: next
 50 M(0)=0: M(2)=28: M(4)=30:
    M(6)=30: M(9)=30: M(11)=30
 60 input "число, месяц, год"; a, b, c
 70 j=a
 80 for i=0 to b-1: j=j+M(i): next
 90 v=fnv(4) and (not fnv(100)) or fnv(400)
100 if b>2 and v then j=j+1
110 print a; b; c; j
120 end
```

```
 10 'Квадратики                                                    0
 20 input "m="; m
 30 dim A(m, m)
 40 for i=1 to m: for j=1 to m
 50   input A(i, j)
 60 next j, i: s=0
 70 for i=1 to m-1: for j=1 to m-1
 80   if A(i, j)+A(i, j+1)+A(i+1, j)+A(i+1, j+1)=17
      then s=s+1
 90 next j, i
100 print s
110 end
```

ОЛИМПИАДА 85

```
 10 'Разложение на слагаемые                                       0
 20 input "n="; n: dim M(n)
 30 M(1)=n: i=1: goto 80
 40 t=M(k)-1: s=t+i-k+1
 50 for i=k to n
 60   if s>t then M(i)=t: s=s-t
            else M(i)=s: goto 80
 70 next i
 80 for k=1 to i: print M(k);: next
 90 print
100 for k=i to 1 step -1
110   if M(k)>1 goto 40
120 next k
130 end
```

```
 10 'Равные элементы                                               0
 20 input "m, n="; m, n
 30 dim A(m, n)
 40 for i=1 to m: A(i, 0)=1
 50   for j=1 to n: input A(i, j): next j
 60 next i
 70 KR=A(1, 1)
 80 for i=1 to m
 90   if A(i, 0)>n goto 150
100   IJ=A(i, A(i, 0))
110   if IJ>KR then KR=IJ: goto 80
120   A(i, 0)=A(i, 0)+1
130   if IJ<KR goto 90
```

```
140 next i:print KR:goto 160
150 print "НЕТ"
160 end
```

OLI

```
 10 'Равные элементы
 20 input "m,n=";m,n
 30 dim A(m,n)
 40 for i=1 to m:for j=1 to n
 50   input A(i,j)
 60 next j,i
 70 for j=1 to n:A(0,j)=A(1,j):next
 80 n0=n:if m=1 goto 170
 90 for i=2 to m
100   j=1:j0=1:jn=1
110     if A(0,j0)<A(i,j) then j0=j0+1:goto 140
120     if A(0,j0)>A(i,j) then j=j+1:goto 140
130     A(0,jn)=A(0,j0):j=j+1:j0=j0+1:jn=jn+1
140   if j0<=n0 and j<=n goto 110
150   n0=jn-1:if n0=0 goto 180
160 next i
170 print A(0,1):goto 190
180 print "НЕТ"
190 end
```

OL

```
 10 'Несоставляемое число
 20 input "n=";n
 30 dim P(n)
 40 for i=1 to n:input P(i):next
 50 s=1
 60 for i=1 to n
 70   for j=i to n
 80     if P(j)<=s goto 100
 90   next j:goto 120
100   s=s+P(j):P(j)=P(i)
110 next i
120 print s
130 end
```

OLI

```
 10 'Несоставляемое число
 20 input "n=";n
 30 dim P(n)
 40 for i=1 to n:input P(i):next
 50 'массив P упорядочен по <=
 60 s=1
 70 for i=1 to n
 80   if P(i)<=s then s=s+P(i) else 100
 90 next i
100 print s
110 end
```

OL

```
 10 'Тетраэдры
 20 def fnD(i,j,k,l)=(m1=i) and
      ((m2=j) and (m3=k) and (m4=l) or
       (m2=k) and (m3=l) and (m4=j) or
       (m2=l) and (m3=j) and (m4=k))
```

```
 30 input m1,m2,m3,m4,
          n1,n2,n3,n4
 40 D=fnD(n1,n2,n3,n4) or fnD(n2,n1,n4,n3) or
      fnD(n3,n1,n2,n4) or fnD(n4,n1,n3,n2)
 50 if D then print "ДА" else print "НЕТ"
 60 end
```

```
 10 'Мода                                                    OLI
 20 input "n=";n:dim A(n)
 30 for i=1 to n:input A(i):next
 40 'массив A упорядочен
 50 i=1:m=0
 60 for j=1 to n
 70   if A(i)=A(j) goto 100
 80   if j-i>m then am=A(i):m=j-i
 90   i=j
100 next j
110 if j-i>m then am=A(i)
120 print am
130 end
```

```
 10 'Мода                                                    OLI 8
 20 input "n=";n:dim A(n)
 30 for i=1 to n:input A(i):next
 40 m=0:i=1
 50   j=i:k=0
 60     if A(i)=A(j) then A(j)=A(i+k):k=k+1
 70     j=j+1
 80   if j<=n goto 60
 90   if m<k then am=A(i):m=k
100   i=i+k
110 if i+m<=n goto 50
120 print am,m
130 end
```

```
 10 'Системы счисления                                       OLI
 20 dim M(9)
 30 input i,j
 40 for k=1 to 9:input M(k):next
 50 for k=1 to 9:print M(k);:next
 60 n=0
 70 for k=9 to 1 step -1
 80   n=n*i+M(k)
 90 next k:print n
100   print (n mod j);
110   n=n\j
120 if n>0 goto 100
130 end
```

```
 10 'Побочная диагональ                                      OLI
 20 input "m,n,k=";m,n,k:dim A(m,n)
 30 for i=1 to m:for j=1 to n
 40   input A(i,j)
 50 next j,i
 60 if 1-k<1 then p=1 else p=1-k
```

```
 70 if n<m-k then q=n else q=m-k
 80 s=0: goto 110
 90   s=s+A(p+k,p)
100   p=p+1
110 if p<=q goto 90
120 print s
130 end
```

ОЛИМПИАДА 86

```
 10 'Без тройных повторений                                    OL
 20 input "n="; n: dim A(n)
 30 i=0
 40 i=i+1: if i>n goto 140                          'Per
 50 A(i)=-1
 60 if A(i)>0 goto 130                              'Bok
 70 A(i)=A(i)+1: if i<3 goto 40
 80 for j=1 to i\3                                  'Prv
 90   for k=i to i-j+1 step -1
100     if A(k)<>A(k-j) or A(k)<>A(k-j-j)
        goto 120
110   next k: goto 60
120 next j: goto 40
130 i=i-1: if i>1 goto 60                           'Zad
140 if i<n then print "НЕТ": goto 160                'Vyv
150 for i=1 to n: print using "#"; A(i);: next
160 end
```

```
 10 'Покер                                                     OL
 20 dim A(5)
 30 for i=1 to 5: input A(i): next
 40 s=0
 50 for i=1 to 4: for j=i+1 to 5
 60   if A(i)=A(j) then s=s+1
 70 next j,i
 80 if s=6 then s=5
 90 if s=10 then s=6
100 print 7-s
110 end
```

```
 10 'Барабан                                                   OL
 20 input "n="; n: dim A(n)
 30 for i=1 to n: input A(i): next
 40 k=1
 50 for p=2 to n
 60   for i=0 to n-1
 70     j=k+i: if j>n then j=j-n
 80     q=p+i: if q>n then q=q-n
 90     if A(j)>A(q) then k=p
100     if A(j)<>A(q) goto 120
110   next i
120 next p
130 print "k="; k
140 end
```

```
 10 'Барабан                                                    OLI
 20 input "n="; n: dim A(n), L(n)
 30 for i=1 to n
 40   input A(i): L(i)=i
 50 next i
 60 jn=n: i=0
 70   q=1
 80   for j=2 to jn
 90     il=L(1)+i: if il>n then il=il-n
100     ij=L(j)+i: if ij>n then ij=ij-n
110     if A(ij)<A(il) then q=1: L(1)=L(j)
120     if A(ij)=A(il) then q=q+1: L(q)=L(j)
130   next j: jn=q
140 i=i+1: if i<n and 1<jn goto 70
150 print L(1)
160 end
```

```
 10 'Дважды монотонный                                          OL
 20 input "m, n="; m, n: dim A(m, n)
 30 for i=1 to m: for j=1 to n
 40   input A(i, j)
 50 next j, i: input "x="; x
 60 i=1: j=n
 70   if A(i, j)=x goto 110
 80   if A(i, j)<x then i=i+1 else j=j-1
 90 if i<=m and j>=1 goto 70
100 print "НЕТ": goto 120
110 print "("; i; ","; j; ")"
120 end
```

```
 10 'Центральное селение                                        OL
 20 input "k="; k
 30 for i=1 to k: for j=1 to k
 40   input A(i, j)
 50 next j, i
 60 for i=1 to k
 70   s=0
 80   for j=1 to k
 90     if j<>i and s<A(i, j) then s=A(i, j)
100   next j: s=s+A(i, i)
110   if i=1 or s<t then t=s: il=i
120 next i: print il; t
130 end
```

ОЛИМПИАДА 87

```
 10 'Рюкзак                                                     OL
 20 input "n="; n
 30 dim A(n), B(n), P(n)
 40 for i=1 to n
 50   print "A("; i; ")="; : input A(i)
 60   print "B("; i; ")="; : input B(i)
 70 next i
 80 s=0: z=0: zm=0: i=0: t=30
```

```
 90   for i=i+1 to n
100     if s+A(i)>=t then P(i)=1
        else s=s+A(i): z=z+B(i): P(i)=0
110   next i
120   if zm<z then zm=z
130   for i=n-1 to 1 step -1
140     if P(i+1)=0 then s=s-A(i+1): z=z-B(i+1)
150     if P(i)=0 and P(i+1)=1 goto 180
160   next i
170 print zm: goto 190
180 s=s-A(i): z=z-B(i): P(i)=1: goto 90
190 end
```

```
 10 'Полукратные                                                    OLI
 20 input "n="; n
 30 dim A(n)
 40 k2=1: k3=1: A(1)=1
 50 print 1;: if n=1 goto 120
 60 for i=2 to n
 70   A2=2*A(k2)+1: A3=3*A(k3)+1
 80   if A2<=A3 then A(i)=A2: k2=k2+1
 90   if A3<=A2 then A(i)=A3: k3=k3+1
100   print A(i);
110 next i
120 end
```

```
 10 'Сумма кубов                                                    OLI
 20 input "n="; n
 30 m=0: i=1: j=1
 40 if j^3+1<n then j=j+1: goto 40
 50   k=i*i*i+j*j*j
 60   if k=n then m=m+1
 70   if k<=n then i=i+1
 80   if k>=n then j=j-1
 90 if i<=j goto 50
100 print m
110 end
```

```
 10 'Перевертыши                                                    OLI
 20 input "n="; n
 30 dim A(n)
 40 for i=1 to n
 50   print "A("; i; ")=";: input A(i)
 60 next i
 70 ln=1: pn=2: mx=1: z=1
 80 goto 170
 90   l=ln: p=pn
100     if A(l)<>A(p) goto 130
110     l=l-1: p=p+1
120   if l<=1 and p<=n goto 100
130   m=p-l-1
140   if mx<m then mx=m
150   if z>0 then pn=pn+1 else ln=ln+1
160   z=-z
170 if 2*(n-pn+1)+1>mx goto 90
```

```
180 print mx
190 end
```

```
 10 'Индексы порядка                                                    OL
 20 input "n=";n
 30 dim A(n),I(n)
 40 for j=1 to n
 50   print "A(";j;")=";:input A(j)
 60   I(j)=j
 70 next j
 80 for k=1 to n
 90   m=k
100   for j=k to n
110     if A(m)>A(j) then m=j
120   next j
130   swap A(k),A(m)
140   swap I(k),I(m)
150 next k
160 for j=1 to n
170   print I(j);
180 next j
190 end
```

```
 10 'Индексы порядка                                                    OLI
 20 input "n=";n
 30 dim A(n)
 40 for k=1 to n
 50   print "A(";k;")=";:input A(k)
 60 next k
 70 mi=1:ma=1
 80 for k=1 to n
 90   if A(k)<A(mi) then mi=k
100   if A(k)>A(ma) then ma=k
110 next k
120 print mi;:if n=1 goto 230
130 for m=2 to n
140   i=ma
150   for k=1 to n.
160     if A(k)<A(mi) then 200
170     if A(k)=A(mi) and k<=mi then 200
180     if A(k)=A(mi) then i=k:goto 210
190     if A(k)<A(i) then i=k
200   next k
210   mi=i:print mi;
220 next m
230 end
```

ОЛИМПИАДА 88

```
 10 'Правый больший                                                     OL
 20 input "n=";n
 30 dim A(n),B(n)
 40 for i=1 to n:input A(i):next
 50 B(n)=A(n):A(n)=0:j=n+1
```

```
 60 for i=n-1 to 1 step -1
 70   for j=j-1 to n
 80     if A(i)<B(j) goto 100
 90   next j
100   B(j-1)=A(i)
110   if j<=n then A(i)=B(j) else A(i)=0
120 next i
130 for i=1 to n:print A(i);:next
140 end
```

```
 10 'Многочлен                                                    OLI
 20 input "n=";n
 30 dim A(n),X(n)
 40 for i=1 to n:input X(i):next
 50 A(0)=1
 60 for m=1 to n:if m=1 goto 100
 70   for i=m-1 to 1 step -1
 80     A(i)=A(i-1)-A(i)*X(m)
 90   next i
100   A(m)=1:A(0)=-A(0)*X(m)
110 next m
120 for i=0 to n-1:print A(i);:next i
130 end
```

```
 10 'Простые делители                                             OLI
 20 input "n=";n
 30 i=2:goto 70
 40   if n mod i then i=i+1:goto 40
 50   if i>j then print i:j=i
 60   n=n\i
 70 if n>1 goto 40
 80 end
```

```
 10 'Оптовая покупка                                              OLI
 20 input "n=";n
 30 n1=n\144
 40 m=n-n1*144
 50 n2=m\12
 60 n3=m-n2*12
 70 if n3*1.05>10.25 then n2=n2+1:n3=0
 80 if n2*10.25+n3*1.05>114 then n1=n1+1:n2=0:n3=0
 90 print n;n1;n2;n3
100 end
```

```
 10 'Обнуление                                                    OLI
 20 input "m,n=";m,n
 30 dim A(m,n),C(n)
 40 for i=1 to m:for j=1 to n
 50   input A(i,j)
 60 next j,i
 70 for i=1 to m:z=0
 80   for j=1 to n
 90     if A(i,j)=0 then z=1:C(j)=1
100   next j:if z=0 goto 120
110   for j=1 to n:A(i,j)=0:next j
```

```
120 next i
130 for j=1 to n
140   if C(j)=0 goto 160
150   for i=1 to m: A(i,j)=0: next i
160 next j
170 for i=1 to m
180   for j=1 to n: print A(i,j);: next j
190   print
210 next i
220 end
```

§ 1.4. Решения на Паскале

В приведенных ниже программах на Паскале нами были сделаны следующие отступления от стандарта языка: (1) Стандартные файлы ввода-вывода можно не указывать — по умолчанию это Input и Output соответственно. (2) Взаимное расположение описаний в программах не определено строго. Учитывая, что при изменении начальных параметров в тексте программ почти всегда изменяется только раздел описания констант (const), мы всюду вынесли его в начало, поставив перед разделом описания меток (label), идущим первым в стандарте. (3) Допускаются не только цифровые, но и любые буквенные метки. (4) При обработке условного оператора и операторов циклов логическое выражение вычисляется слева направо до тех пор, пока результат не станет ясен. Поэтому в правой части выражения могут находиться переменные, не имеющие еще значений (т. е. неопределенные). (5) Использован новый оператор exit, вызывающий прекращение работы программы (аналог STOP в Бейсике).

Все эти отступления от стандарта допускаются в версии Turbo Pascal v. 5. 0, использованной нами.

Кроме того, для удобства чтения, в программах принято несколько соглашений: (a) все имена констант, массивов, а так же логические операции и слова BEGIN и END, выделяющие основную часть программы, пишутся большими буквами; (b) все имена функций, определенных в программе, начинаются с большой буквы; (c) все остальные переменные и ключевые слова пишутся маленькими буквами.

ОЛИМПИАДА 80

```
{ Простые до M                                              OLI80.1.1
PROGRAM OLI8011;
const N=200;
label BR,NP;
var m,i,ii,j,k,q : integer;
    P            : array[1..N] of integer;
BEGIN
write('M:=   ');
readln(m);
writeln('Program get N=',N);
if (m>=2) then writeln(2);
if (m>=3) then writeln(3);
```

```
k:=1; P[k]:=3; i:=5;
while i<=m do
begin
   for j:=1 to k do
    begin
      q:=P[j];
       if (q*q>i) then goto BR;
       if (i mod q = 0) then goto NP
    end;
  if k=N then
     begin
      i:=m-1;
      write  ('The rest numbers ');
      writeln('can not be calculated correctly')
     end
   else
 BR: begin
      writeln(i);
      if (k<=N-1) then begin
                         k:=k+1;
                         P[k]:=i
                       end
     end;
 NP: i:=i+2
end
END.
```

```
{ Перестановки                                                    OLI8
PROGRAM OLI8012;
const MM=100;
var m,i,j,k,n : integer;
    A,P       : array[1..MM] of integer;
BEGIN
write('M:= ');
readln(m);
writeln('Program get M=',m);
for i:=1 to m do
  begin
    readln(A[i]);
    P[i]:=i
  end;
for i:=1 to m do write(A[i],' '); writeln;
for i:=m-1 downto 1 do  if (P[i]<P[i+1]) then
  begin
   n:=P[i];
   for j:=m downto i do
    if (n<P[j]) then
      begin
        P[i]:=P[j]; P[j]:=n;
         k:=1;
         while i+k<m-k+1 do
            begin
             n:=P[i+k];
             P[i+k]:=P[m+1-k];
             P[m+1-k]:=n;
```

```
            k:=k+1
          end;
        j:=i
      end;
   for i:=1 to m do write(A[P[i]],' '); writeln
  end
END.
```

```
{ Быстрая степень                                                    OLI8
PROGRAM OLI8013;
var a,b : real;
    k,n : integer;
BEGIN
writeln('A,K -');
readln(a,k); write(a,'^',k,'=');
  b:=1;
  while k>0 do
    begin
      n:=k div 2; if (n+n<k) then b:=b*a;
      k:=n; a:=a*a;
    end;
  writeln(b)
END.
```

```
{ Арифметические действия                                            OLI8
PROGRAM OLI8014;
const M=1;
      N=9;
label R;
var i,j,k,y : integer;
    A,B     : array[1..N] of integer;
BEGIN
B[1]:=1; k:=0; A[2]:=0;
for i:=3 to N do A[i]:=4;
R:
 for i:=N downto 2 do
   if (A[i]=4) then A[i]:=1
   else
   begin
     A[i]:=A[i]+1; y:=B[i-1];
     case (A[i]) of
       1: B[i]:=y+i;
       2: B[i]:=y-i;
       3: B[i]:=y*i;
       4: B[i]:=y div i
     end;
     for j:=i+1 to N do B[j]:=B[j-1]+j;
     if (B[N]<>M) then goto R; k:=k+1;
     for j:=2 to N-1 do write('('); write('1');
     for j:=2 to N do
     begin
       if (j>2) then write(')');
       case A[j] of
         1: write('+');
         2: write('-');
```

```
          3: write('*');
          4: write('%');
        end;
        write(j);
      end;
      writeln('=',M); goto R;
    end;
  writeln(k);
END.
```

```
{ Поиск равных                                                    OLI80.
PROGRAM OLI8021;
label F;
var i,j,p,q,b : integer;
    A         : array[1..2,1..15] of real;
    ff        : boolean;
BEGIN
  for j:=1 to 15 do
  begin
    write('A[1,',j,']:= ');
    readln(A[1,j]);
    write('A[2,',j,']:= ');
    readln(A[2,j])
  end;
  ff:=FALSE;
  for i:=1 to 2 do
    for j:=1 to 15 do
      for p:=i to 2 do
      begin
        if i<p then b:=1 else b:=j+1;
        for q:=b to 15 do  if (A[i,j]=A[p,q]) then
        begin
          ff:=TRUE;
          goto F
        end
      end;
F:if ff then writeln('A[',i,',',j,']=A[',p,',',q,']')
        else  writeln('----')
END.
```

```
{ Сумма квадратов                                                 OLI80.
PROGRAM OLI8022;
var m,i,j : integer;
    ff    : boolean;
BEGIN
write('M:= ');
readln(m);
writeln('Program get M = ',m);
 i:=1;
 ff:=TRUE;
 while (2*i*i<=m) AND ff do
 begin
   j:=round(sqrt(m-i*i));
   if (i*i+j*j=m) then ff:=FALSE
                  else i:=i+1
```

```
 end;
if ff then writeln('Net')
         else writeln(i,'*',i,' + ',j,'*',j,' = ',m)
END.
```

```
{ Различные числа                                                OLI80
PROGRAM OLI8023;
const N=100;
var m,i,j,s : integer;
    b       : boolean;
    A       : array[1..N] of integer;
BEGIN
  write('M:= ');
  readln(m);
  writeln('Program get M= ',m);
  for i:=1 to m do
  begin
    write('A[',i,']:= ');
    readln(A[i])
  end;
   s:=0;
   for i:=1 to m do
   begin
     b:=FALSE;
      j:=i+1;
      while (j<=m) AND NOT b do
      begin
        b:=b OR (A[i]=A[j]);
        j:=J+1
      end;
     if NOT b then s:=s+1;
   end;
  writeln(s);
END.
```

```
{ Заданная сумма цифр                                            OLI
PROGRAM OLI8031;
var n,i,j,k : integer;
BEGIN
write('N:= ');
readln(n);
if n in [1..27] then
  for i:=0 to 9 do
   for j:=0 to 9 do
   begin
     k:=n-i-j;
     if k in [1..9] then writeln(i+10*j+100*k);
   end
END.
```

```
{ M+1 в двоичной записи                                          OLI
PROGRAM OLI8032;
var n,i,j : integer;
    b     : boolean;
```

```
BEGIN
write('N:= ');
readln(n);
b:=TRUE;
for i:=1 to n do
begin
  readln(j);
  if b then writeln(' ',1-j) else writeln(' ',j);
  if j=0 then  b:=FALSE;
end;
  if b then writeln(' 1');
END.
```

```
{ Максимум минимумов                                          OLI80.
PROGRAM OLI8033;
const MM=10;
      NN=20;
label SI;
var i,j,k,m,n : integer;
    X         : array[1..MM,1..NN] of integer;
    min,max   : integer;
BEGIN
write('M:= ');
readln(m);
write('N:= ');
readln(n);
for i:=1 to m do
  for j:=1 to n do
  begin
    write('X[',i,',',j,']:= ');
    readln(X[i,j])
  end;
for i:=1  to m do
   begin
     for j:=1 to n do
     begin
       if (i>1) AND (X[i,j]<=max) then goto SI;
       if (j=1) OR (X[i,j]<=min) then min:=X[i,j];
     end;
     max:=min; k:=i;
SI:end;
writeln(k)
END.
```

```
{ Перестановка 0,1,2                                          OLI80.
PROGRAM OLI8034;
const NN=100;
type el=0..2;
var i,n : integer;
    A   : array[el] of 0..NN;
    X   : array[1..NN] of el;
BEGIN
write('N:= ');
readln(n);
writeln('Program get N= ',n);
```

```
for i:=1 to n do readln(X[i]);
writeln;
 A[0]:=0;  A[1]:=0;  A[2]:=0;
 for i:=1 to n do A[X[i]]:=A[X[i]]+1;
 for i:=1 to n do
   if i<=A[0] then X[i]:=0
              else if i<=n-A[2] then X[i]:=1 else X[i]:=2;
for i:=1 to n do writeln(X[i])
END.
```

ОЛИМПИАДА 81

```
{ Функция                                                    OL
PROGRAM OLI811;
var n,i,j : integer;
BEGIN
write('N:= ');
readln(n);
writeln('Program get N= ',n);
i:=1;  j:=0;
while n>0 do
begin
 if n mod 2 = 0  then i:=i+j else j:=j+i;
 n:=n div 2;
end;
writeln(j)
END.
```

```
{ Пара четверок                                              OL
PROGRAM OLI812;
label BRJ,VSJO;
var n,i,j,k,p   : integer;
    i1,j1,k1,p1 : integer;
    b           : boolean;
BEGIN
for n:=2 to maxint do
begin
  b:=FALSE;
  for i:=1 to trunc(sqrt(n)) do
  begin
    for j:=1 to i do
    begin
      if i*i+j*j>=n then goto BRJ;
      for k:=1 to j do
        if i*i+j*j+k*k<n then
        begin
          p:=trunc(sqrt(n-i*i-j*j-k*k));
          if (i*i+j*j+k*k+p*p=n) and (p<=k) then
          begin
            if (b) then goto VSJO;
            i1:=i;  j1:=j;  k1:=k;  p1:=p;
            b:=TRUE;
          end
        end
```

```
    end; BRJ:
  end;
end;
VSJO:
writeln(i*i+j*j+k*k+p*p,' = ');
write
 (i,'*',i,' + ',j,'*',j,' + ',k,'*',k,' + ',p,'*',p,' = ');
writeln
 (il,'*',il,' + ',jl,'*',jl,' + ',kl,'*',kl,
                                          ' + ',pl,'*',pl)
END.
```

```
{ Спираль                                                    OL
PROGRAM OLI813;
CONST NN=19;
var i,j,k,n : integer;
    A       : array[1..NN,1..NN] of integer;
function MOV:boolean;
begin
  MOV:=FALSE;
  if k<=n*n then begin A[i,j]:=k; k:=k+1; MOV:=TRUE end
end;
BEGIN
 write('N:= ');
 readln(n);
 k:=1; i:=1; j:=1;
 repeat
   while MOV and (i+j<n+1) do j:=j+1; k:=k-1;
   while MOV and (i<j)     do i:=i+1; k:=k-1;
   while MOV and (i+j>n+1) do j:=j-1; k:=k-1;
   while MOV and (i>j+1)   do i:=i-1; k:=k-1;
 until k=n*n;
 for i:=1 to n do
 begin
   for j:=1 to n do write(A[i,j]:4);
   writeln
 end
END.
```

```
{ Числа из разных цифр                                       OL
PROGRAM OLI814;
var i,j,k,m : integer;
BEGIN
for i:=1 to 9 do
  for j:=0 to 9 do if i<>j then
    for k:=0 to 9 do if (k<>i) and (k<>j) then
      for m:=0 to 9 do
        if (m<>i) and (m<>j) and (m<>k) then
              writeln((((i*10+j)*10+k)*10)+m)
END.
```

```
{ Серия нулей                                                OL
PROGRAM OLI815;
CONST NN=100;
var n,i,t,max : integer;
```

```
    A         : array [1..NN] of real;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do readln(A[i]);
t:=0;  max:=0;
 for i:=1 to n do
 begin
   if A[i]<>0 then
   begin
     if (max<t) then max:=t;
     t:=0
   end
   else t:=t+1;
 end;
if max<t then max:=t;
writeln;
writeln(max)
END.
```

ОЛИМПИАДА 82

```
{ Прямоугольники                                                    OLI
PROGRAM OLI821;
const MM = 100;
      NN = 100;
var m,n,j,i,s : integer;
    A         : array [0..MM,0..NN] of byte;
BEGIN
writeln('M,N:=');
readln(m,n);
for i:=0 to m do
  for j:=0 to n do
      if (i=0) OR (j=0) then A[i,j]:=0
                        else
                        begin
                          write('A[',i,',',j,']:= ');
                          readln(A[i,j])
                        end;
s:=0;
for i:=1 to m do
  for j:=1 to n do
    if (A[i,j]=1) AND (A[i-1,j]+A[i,j-1]=0) then s:=s+1;
writeln(s);
END.

{ Упорядоченные дроби
PROGRAM OLI822;
var p,n,m,i,j,a,b : integer;
BEGIN
write('P:= ');
readln(p);
m:=0;  n:=1;
repeat
```

```
  writeln(m:4,' / ',n:4,' = ',m/n);
  i:=1; j:=1;
  for b:=2 to p do
  begin
    a:=m* b div n +1;
    if a*j<b*i then begin i:=a; j:=b end
  end;
  m:=i; n:=j;
until i>=j
END.
```

```
{ Сумма по подмножеству                                          OLI
PROGRAM OLI823;
const  NN = 100;
label BR;
var n,m,s,i : integer;
    A       : array [1..NN] of integer;
    B       : array [1..NN] of boolean;
BEGIN
writeln('N,M: = ');
readln(n,m);
for i:=1 to n do begin
                   write('A[',i,']:= ');
                   readln(A[i]);
                   B[i]:=FALSE
                 end;
s:=0;
repeat
  for i:=1 to n do
    if (B[i]) then begin B[i]:=FALSE; s:=s-A[i] end
              else goto BR;
    BR: B[i]:=TRUE; s:=s+A[i];
until s=m;
for i:=1 to n do if (B[i]) then write(i:4); writeln
END.
```

```
{ Нули - в конец                                                 OLI
PROGRAM OLI824;
const NN = 100;
var n,i,j : integer;
    X     : array [1..NN] of integer;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do readln(X[i]);
j:=0;
for i:=1 to n do
  if X[i]<>0 then
  begin
    j:=j+1;
    if i<>j then begin X[j]:=X[i]; X[i]:=0 end
  end;
for i:=1 to n do write(X[i],' '); writeln
END.
```

```
{ Седловая точка                                                    OL
PROGRAM OLI825;
const  MM = 20;
       NN = 20;
label SI,BRI;
var m,n,i,j,i0,mi,ma : integer;
    A                : array [1..MM,1..NN] of integer;
BEGIN
writeln('M,N: = ');
readln(m,n);
for i:=1 to m do
  for j:=1 to n do
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j])
  end;
for i:=1 to m do
  begin
    for j:=1 to n do
    begin
      if (i>1) AND (A[i,j]<=ma) then goto SI;
      if (j=1) OR (A[i,j]<mi) then mi:=A[i,j]
    end;
    ma:=mi; i0:=i;
SI:end;
for j:=1 to n do
    begin
      for i:=1 to m do if A[i,j]>ma then goto BRI;
      writeln(i0:3,j:3); exit;
BRI:end;
writeln(0)
END.

{ Вхождение слова в текст                                           OL
PROGRAM OLI826;
const  NN = 100;
       KK =  50;
var n,k,i,j : integer;
    X       : array [1..NN] of integer;
    Y       : array [1..KK] of integer;
    ff      : boolean;
BEGIN
writeln('N,K:=');
readln(n,k);
writeln;
writeln('Array X:');
for i:=1 to n do readln(X[i]);
writeln;
writeln('Array Y:');
for j:=1 to k do readln(Y[j]);
writeln;
i:=0;
ff:=TRUE;
while (i<=n-k) AND ff do
begin
```

```
  j:=1;
  while (j<=k) AND ff do
  if X[i+j]<>Y[j] then ff:=FALSE
                  else j:=j+1;
  if ff then begin writeln('Da ',i+1); ff:=FALSE end
        else begin ff:=TRUE; i:=i+1 end
end;
if ff then writeln('Net')
END.
```

ОЛИМПИАДА 83

```
{ Бит-реверс                                                    OLI83
PROGRAM OLI831;
var
  m,b,k : integer;
BEGIN
  k:=512; m:=1;
  writeln(m);
  while m<1024 do
  begin
    while m>=k do begin m:=m-k; k:=k div 2 end;
    m:=m+k; k:=512;
    writeln(m)
  end
END.

{ Треугольник и точка                                           OLI83
PROGRAM OLI832;
type POINT = record
               x:real;
               y:real
             end;
var tr : array[1..3] of POINT;
    t  : POINT;
    i  : 1..3;

function Dis(p,q:POINT):real;
begin
Dis:=sqrt(sqr(p.x-q.x)+sqr(p.y-q.y))
end;

function Grn(a,b,c:POINT):real;
var da,db,dc,p:real;
begin
da:=Dis(c,b); db:=Dis(a,c); dc:=Dis(a,b);
p:=(da+db+dc)/2;
Grn:=sqrt(p*(p-da)*(p-db)*(p-dc))
end;

BEGIN
for i:=1 to 3 do
begin
  writeln('X',i,' , ','Y',i,':=');
```

```
  readln(tr[i].x,tr[i].y)
end;
writeln('X,Y:=');
readln(t.x,t.y);
if Grn(tr[1],tr[2],tr[3])*1.000001 <
   Grn(t,tr[1],tr[2])+Grn(t,tr[1],tr[3])+Grn(t,tr[2],tr[3])
        then writeln('net')
        else writeln('da')
END.
```

```
{ Лабиринт                                                    OLI
PROGRAM OLI833;
const MM=15;
      NN=15;
var m,n,i,j : integer;
    vyh     : boolean;
    A       : array [1..MM,1..NN] of byte;
procedure L(i,j:integer);
begin
if NOT vyh then
 if A[i,j]=0 then
 begin
   if (i=1) OR (i=m) OR (j=1) OR (j=n) then vyh:=TRUE;
   A[i,j]:=1; L(i,j-1); L(i,j+1); L(i-1,j); L(i+1,j);
   if vyh then writeln(i,' ',j)
 end
end;
BEGIN
writeln('M,N:=');
readln(m,n);
for i:=1 to m do
  for j:=1 to n do
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j])
  end;
writeln;
writeln('I,J:=');
readln(i,j);
writeln;
vyh:=FALSE;
L(i,j);
if NOT vyh then writeln('net vyhoda');
END.
```

```
{ Пила                                                        0
PROGRAM OLI834;
const MM=100;
var  i,j,k,m : integer;
     X       : array [1..MM] of real;
BEGIN
write('M:= ');
readln(m);
for i:=1 to m do readln(X[i]);
j:=1; k:=1; i:=1;
```

```
while i+1<m do
begin
  if (X[i]<X[i+1]) AND (X[i+1]>X[i+2]) then
  begin
    i:=i+2; j:=j+2;
    if k<j then k:=j
  end
  else
  begin
    i:=i+1; j:=1;
    if i+k>=m then i:=m-1;
  end
end;
writeln;
writeln(k)
END.
```

```
{ Сократить дробь                                          OLI83.
PROGRAM OLI835;
label BRJ;
var m,n,i,j : integer;
BEGIN
writeln('M,N:=');
readln(m,n);
writeln('Program get M= ',m,'    N= ',n);
   for j:=1 to n do
   begin
     i:=j*m div n;
     if i*n=j*m then goto BRJ
   end;
BRJ: writeln('M/N = ',i,' / ',j)
END.
```

ОЛИМПИАДА 84

```
{ Инверсия                                                 OLI8
PROGRAM OLI841;
const NN=100;
var
    n,i,j,k : integer;
    P,T     : array[1..NN] of integer;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do
begin
  write('T[',i,']:= ');
  readln(T[i]);
  P[i]:=0
end;
for i:=1 to n do
begin
  j:=0; k:=0;
  repeat
```

```
    k:=k+1;
    if P[k]=0 then j:=j+1
  until j>T[i];
  P[k]:=i
end;
for i:=1 to n do writeln(P[i])
END.
```

```
{ Дорога                                                    OL
PROGRAM OLI842;
const MM = 10;
      NN = 10;
var m,n,i,j,k : integer;
    x,r       : real;
    A         : array [1..MM,1..MM,1..NN] of real;
    B,C       : array [1..MM] of real;
BEGIN
writeln('M,N:= ');
readln(m,n);
for i:=1 to m do
begin
  B[i]:=0;
  for j:=1 to m do
    for k:=1 to n-1 do
    begin
      write('A[',i,',',j,',',k,']:= ');
      readln(A[i,j,k])
    end
end;
for k:=1 to n-1 do
begin
  for j:=1 to m do
  begin
    r:=B[1]+A[1,j,k];
    for i:=2 to m do
      begin
        x:=B[i]+A[i,j,k];
        if x<r then  r:=x
      end;
      C[j]:=r
  end;
  for j:=1 to m do B[j]:=C[j]
end;
r:=B[1];
for i:=2 to m do if B[i]<r then r:=B[i];
writeln(r)
END.
```

```
{ Совершенные числа                                         OLI
PROGRAM OLI433;
var  m,i,j,k,s : integer;
BEGIN
write('M:= ');
readln(m);
writeln('Program get M= ',m);
```

```
for i:=2 to m do
begin
  s:=1; j:=1;
  repeat
    j:=j+1;
    k:=i div j;
    if (i=k*j) AND (j<=k) then
    begin
      s:=s+j; if j<k then  s:=s+k
    end
  until j>=k;
  if s=i then writeln(i)
end
END.
```

```
{ Период дроби                                                    OLI84.
PROGRAM OLI844;
var  m,n,i,j,k : integer;
BEGIN
writeln('M,N:=');
readln(m,n);
writeln;
m:=m-(m div n)*n;
k:=1;
while (k<=n) OR (j<>m) do
begin
  if k=n then  j:=m;
  i:=10*m div n;
  m:=10*m-i*n;
  if k>=n then write(i);
  k:=k+1
end;
writeln
END.
```

```
{ Слияние массивов                                                OLI8
PROGRAM OLI845;
const MM    = 100;
      NN    = 100;
      MMPNN = 200;
var m,n,i,j,k : integer;
    A         : array [1..MM] of integer;
    B         : array [1..NN] of integer;
    C         : array [1..MMPNN] of integer;
BEGIN
writeln('M,N:=');
readln(m,n);
writeln('Array A:');
for i:=1 to m do readln(A[i]);
writeln('Array B:');
for j:=1 to n do readln(B[j]);
i:=1; j:=1;
for k:=1 to m+n do
begin
  if ((i>m) OR (A[i]>B[j])) AND NOT(j>n) then
```

```
  begin C[k]:=B[j]; j:=j+1 end
  else
  begin C[k]:=A[i]; i:=i+1 end
end;
writeln('Array C:');
for k:=1 to m+n do writeln(C[k])
END.
```

```
{ Календарь                                                    OL
PROGRAM OLI846;
var a,b,c,i,j : integer;
    M         : array [1..11] of integer;

function D(x:integer):boolean;
begin
D:=(c mod x)=0
end;

BEGIN
writeln('A,B,C:=');
readln(a,b,c);
writeln;
for i:=1 to 11 do
case i of
  1,3,5,7,8,10    : M[i]:=31;
  4,6,9,11        : M[i]:=30;
  2               : M[i]:=28;
end;
j:=a;
for i:=1 to b-1 do j:=j+M[i];
if (b>2) AND (D(4) AND NOT D(100) OR D(400)) then j:=j+1;
writeln(j)
END.
```

```
{ Квадратики                                                   OL
PROGRAM OLI847;
const MM = 50;
var i,j,m,s : integer;
    A       : array [1..MM,1..MM] of integer;
BEGIN
write('M:= ');
readln(m);
for i:=1 to m do
  for j:=1 to m do
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j]);
  end;
s:=0;
for i:=1 to m-1 do
  for j:=1 to m-1 do
    if (A[i,j]+A[i,j+1]+A[i+1,j]+A[i+1,j+1]=17) then s:=s+1;
writeln(s)
END.
```

```
{ Разложение на слагаемые                                          OLI
PROGRAM OLI851;
const NN = 100;
label RA,BR;
var
    n,i,k,t,s : integer;
    M         : array [1..NN] of integer;
BEGIN
   write('N: = ');
   readln(n);
   writeln(n);
   M[1]:=n; k:=1; i:=1;
RA: t:=M[k]-1; s:=t+i-k+1;
   for i:=k to n do
     if s>t then begin M[i]:=t; s:=s-t end
            else begin M[i]:=s; goto BR end;
BR: for k:=1 to i do write(M[k],' '); writeln;
   for k:=i downto 1 do if M[k]>1 then goto RA
END.

{ Равные элементы                                                  OLI85
PROGRAM OLI852;
const
      NN = 20;
      MM = 20;
label MJ,NET;
var m,n,i,j,j0,n0,jn : integer;
    A                : array [0..MM,1..NN] of integer;
BEGIN
writeln('M,N:=');
readln(m,n);
for i:=1 to m do
  for j:=1 to n do
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j])
  end;
for j:=1 to n do A[0,j]:=A[1,j];
n0:=n;
for i:=2 to m do
begin
    j:=1; j0:=1; jn:=1;
MJ: if A[0,j0]<A[i,j] then j0:=j0+1
    else
    if A[0,j0]>A[i,j] then j:=j+1
    else
    begin
      A[0,jn]:=A[0,j0];
      j:=j+1;
      jn:=jn+1;
      j0:=j0+1
    end;
    if (j0<=n0) AND (j<=n) then goto MJ;
```

```
    n0:=jn-1;
    if (j<=n) then goto MJ;
    if n0=0 then goto NET
end;
writeln(A[0,1]); exit;
NET: writeln('Net')
END.
```

```
{ Несоставляемое число                                          OL
PROGRAM OLI853;
const NN = 10;
label PI,BR;
var  n,i,j,s : integer;
     P       : array [1..NN] of integer;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do readln(P[i]);
writeln;
s:=1;
for i:=1 to n do
  begin
    for j:=i to n do if P[j]<=s then goto PI;
    goto BR;
PI:  s:=s+P[j]; P[j]:=P[i]
  end;
BR: writeln(s)
END.
```

```
{ Тетраэдры                                                     OL
PROGRAM OLI854;
var m1,m2,m3,m4,n1,n2,n3,n4 : integer;

function Da(i,j,k,l:integer):boolean;
begin
Da:=((m1=i) AND (((m2=j) AND (m3=k) AND (m4=l))
            OR   ((m2=l) AND (m3=j) AND (m4=k))
            OR   ((m2=k) AND (m3=l) AND (m4=j))))
end;

BEGIN
writeln('M1,M2,M3,M4:=');
readln(m1,m2,m3,m4);
writeln('N1,N2,N3,N4:=');
readln(n1,n2,n3,n4);
writeln;          ;
if Da(n1,n2,n3,n4) OR Da(n2,n1,n4,n3) OR
   Da(n3,n1,n2,n4) OR Da(n4,n1,n3,n2)  then writeln('Da')
                                       else writeln('Net')
END.
```

```
{ Мода                                                          OLI8
PROGRAM OLI855;
const NN = 100;
var n,m,am,i,j,k : integer;
```

```
    A            : array [1..NN] of integer;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do readln(A[i]);
writeln;
m:=0; i:=1;
while i+m<=n do
begin
  k:=1;
  for j:=i+1 to n do
    if A[j]=A[i] then
    begin
      A[j]:=A[i+k];
      k:=k+1
    end;
  if m<k then begin am:=A[i]; m:=k end;
  i:=i+k
end;
writeln(am)
END.
```

```
{ Системы счисления                                                    OL
PROGRAM OLI856;
var i,j,k : integer;
    n     : real;
    M     : array [1..9] of integer;
BEGIN
writeln('I,J:=');
readln(i,j);
for k:=9 downto 1 do
begin
  write('M[',k,']:= ');
  readln(M[k])
end;
writeln;
n:=0;
for k:=9 downto 1 do n:=n*i+M[k];
writeln(trunc(n));
repeat
  write(trunc(n) mod j,' ');
  n:=trunc(n) div j;
until n=0;
writeln
END.
```

```
{ Побочная диагональ                                                   OL
PROGRAM OLI857;
const  MM = 10;
       NN = 10;
var m,n,j,k,p,q,s : integer;
    A             : array [1..MM,1..NN] of integer;
BEGIN
writeln('M,N,K:=');
readln(m,n,k);
```

```
for p:=1 to m do
  for j:=1 to n do
  begin
    write('A[',p,',',j,']:= ');
    readln(A[p,j])
  end;
if k>0   then p:=1 else p:=1-k;
if k+n<m then q:=n else q:=m-k;
s:=0;
for j:=p to q do s:=s+A[k+j,j];
writeln(s)
END.
```

ОЛИМПИАДА 86

```
{ Без тройных повторений                                          OL
PROGRAM OLI861;
const N = 50;
label PER, BOK, PRV, JP1, ZAD, VYH;
var i,j,k : integer;
    A     : array [1..N] of integer;
BEGIN
     i:=0;
PER: i:=i+1; if i>N then goto VYH;
     A[i]:=-1;
BOK: if A[i]>0 then goto ZAD;
     A[i]:=A[i]+1;
PRV: for j:=1 to i div 3 do
     begin
       for k:=i downto i-j+1 do
         if (A[k]<>A[k-j]) OR (A[k]<>A[k-j-j]) then goto JP1;
       goto BOK;
JP1: end;
     goto PER;
ZAD: i:=i-1;
     if i>=1 then goto BOK;
VYH: if i<N then writeln('Net')
     else for i:=1 to N do writeln(A[i])
END.
```

```
{ Покер                                                           OL
PROGRAM OLI862;
var i,j,p : integer;
    A     : array [1..5] of integer;
BEGIN
for i:=1 to 5 do readln(A[i]);
writeln;
p:=0;
for i:=1 to 4 do
  for j:=i+1 to 5 do
    if A[i]=A[j] then p:=p+1;
if p=10 then writeln(1)
else
if p=6 then writeln(2)
```

```
else
writeln(7-p)
END.
```

```
{ Барабан                                                    OLI
PROGRAM OLI863;
const  N = 12;
label BR;
var i,j,l,k,p : integer;
    A         : array [1..N] of real;
BEGIN
for i:=1 to N do
begin
  write('A[',i,']:= ');
  readln(A[i])
end;
k:=1;
for p:=2 to N do
   begin
     for i:=0 to N-1 do
       begin
         j:=k+i; if j>N then j:=j-N;
         l:=p+i; if l>N then l:=l-N;
         if A[l]<A[j] then k:=p;
         if A[l]<>A[j] then goto BR
       end;
BR: end;
writeln;
writeln(k)
END.
```

```
{ Дважды монотонный                                          OLI
PROGRAM OLI864;
const MM = 20;
      NN = 20;
label VYV;
var  i,j,m,n,x : integer;
     A         : array [1..MM,1..NN] of real;
BEGIN
     writeln('M,N:=');
     readln(m,n);
     write('X:= ');
     readln(x);
     for i:=1 to m do
       for j:=1 to n do
       begin
         write('A[',i,',',j,']:= ');
         readln(A[i,j])
       end;
     i:=1; j:=n;
     while (i<=m) AND (j>=1) do
     begin
       if A[i,j]=x then  goto VYV;
       if A[i,j]<x then i:=i+1
                   else j:=j-1
```

```
     end;
     writeln('Net'); exit;
VYV: writeln(i,' ',j)
END.
```

```
{ Центральное селение                                                OL
PROGRAM OLI865;
const  KK = 20;
var i,j,k,il,s,t : integer;
    A            : array [1..KK,1..KK] of real;
BEGIN
write('K:= ');
readln(k);
for i:=1 to k do
  for j:=1 to k do
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j])
  end;
for i:=1 to k do
begin
   s:=0;
   for j:=1 to k do
     if (j<>i) AND (s<A[i,j]) then s:=A[i,j];
   s:=s+A[i,i];
   if (i=1) OR (s<t) then
   begin
     il:=i;
     t:=s
   end
end;
writeln(il)
END.
```

ОЛИМПИАДА 87

```
{ Рюкзак                                                             OL
PROGRAM OLI871;
const  NN = 100;
       T  = 30;
label V;
var i,s,z,zm,n : integer;
    A,B        : array [1..NN] of real;
    P          : array [1..NN] of boolean;
BEGIN
  write('N:= ');
  readln(n);
  for i:=1 to n do
  begin
    write('A[',i,']:= ');
    readln(A[i]);
    write('B[',i,']:= ');
    readln(B[i])
  end;
```

```
  s:=0; z:=0; zm:=0; i:=0;
V: for i:=i+1 to n do
    if s+A[i] >= T then
      P[i]:=FALSE
    else
    begin
      s:=s+A[i];
      z:=z+B[i];
      P[i]:=TRUE
    end;
  if zm<z  then zm:=z;
  for i:=n-1 downto 1 do
  begin
    if P[i+1] then
    begin
      s:=s-A[i+1];
      z:=z-B[i+1]
    end;
    if P[i] AND NOT P[i+1] then
    begin
      s:=s-A[i];
      z:=z-B[i];
      P[i]:=FALSE;
      goto V
    end;
  end;
writeln(zm)
END.
```

```
{ Полукратные                                                    OL
PROGRAM OLI872;
const  NN = 100;
var i,k2,k3,a2,a3,n : integer;
    A               : array [1..NN] of integer;
BEGIN
write('N:= ');
readln(n);
k2:=1; k3:=1; A[1]:=1;
writeln(1);
for i:=2 to n do
begin
  a2:=2*A[k2]+1;
  a3:=3*A[k3]+1;
  if a2<=a3 then
  begin
    A[i]:=a2;
    k2:=k2+1
  end;
  if a3<=a2 then
  begin
    A[i]:=a3;
    k3:=k3+1
  end;
  writeln(A[i])
end
```

```
END.
```

OLI37.

```
{ Сумма кубов
PROGRAM OLI873;
var  i, j, m, k, n : integer;
BEGIN
write('N: = ');
readln(n);
m: =0; i: =1; j: =1;
while j*j*j+1<n do j: =j+1;
repeat
  k: =i*i*i+j*j*j;
  if k=n  then m: =m+1;
  if k<=n then i: =i+1;
  if k>=n then j: =j-1;
until i>j;
writeln(m)
END.
```

OLI87.

```
{ Перевертыши
PROGRAM OLI874;
const  NN = 100;
var  i, ln, pn, l, p, m, max, n : integer;
     z                          : boolean;
     A                          : array [1..NN] of real;
BEGIN
write('N: = ');
readln(n);
for i: =1 to n do
begin
  write('A[', i, ']: = ');
  readln(A[i])
end;
max: =1; z: =TRUE; ln: =1; pn: =2;
while 2*(n-pn+1)+1>max do
begin
  l: =ln; p: =pn;
  while (l>=1) AND (p<=n) AND (A[l]=A[p]) do
  begin
    l: =l-1;
    p: =p+1
  end;
  m: =p-l-1;
  if (max<m) then max: =m;
  if z then pn: =pn+1 else ln: =ln+1;
  z: =NOT z
end;
writeln(max)
END.
```

OLI87.5-

```
{ Индексы порядка
PROGRAM OLI875;
const  NN  = 100;
label BIL, NOV;
var i, j, m, mi, ma, k, n : integer;
```

```
    A               : array [1..NN] of real;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do
begin
  write('A[',i,']:= ');
  readln(A[i])
end;
mi:=1; ma:=1;
for k:=1 to n do
begin
  if A[k]<A[mi] then  mi:=k;
  if A[k]>A[ma] then  ma:=k
end;
writeln(mi);
for m:=2 to n do
begin
  i:=ma;
  for k:=1 to n do
     begin
       if (A[k]<A[mi]) OR ((A[k]=A[mi]) AND (k<=mi)) then
         goto BIL;
       if A[k]=A[mi] then
       begin
         i:=k;
         goto NOV
       end;
       if A[k]<A[i] then
         i:=k;
BIL: end;
NOV: mi:=i;
writeln(mi)
end
END.
```

ОЛИМПИАДА 88

```
{ Правый больший                                                    OL
PROGRAM OLI881;
const NN = 100;
var i,j,n,k : integer;
    ff      : boolean;
    A,B     : array [1..NN] of real;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do
begin
  write('A[',i,']:= ');
  readln(A[i])
end;
B[n]:=A[n]; A[n]:=0; k:=n;
for i:=n-1 downto 1 do
```

```
begin
  j:=k; ff:=FALSE;
  while (j<=n) AND NOT ff do
  begin
    ff:=A[i]<B[j];
    j:=j+1
  end;
  if ff then
  begin
    k:=j-1-1;
    B[k]:=A[i];
    A[i]:=B[j-1]
  end
  else
  begin
    k:=n;
    B[k]:=A[i];
    A[i]:=0
  end;
end;
for i:=1 to n do writeln(A[i])
END.
```

```
{ Многочлен                                                    OL
PROGRAM OLI882;
const  NN = 100;
var i,m,n : integer;
    X     : array[1..NN]  of  real;
    A     : array[0..NN-1] of real;
BEGIN
write('N:= ');
readln(n);
for i:=1 to n do
begin
  write('X[',i,']:= ');
  readln(X[i])
end;
A[0]:=-X[1]; A[1]:=1;
for m:=2 to n do
begin
  A[m-1]:=A[m-2]-X[m];
  i:=m-2;
  while i>=1 do
  begin
    A[i]:=A[i-1]-A[i]*X[m];
    i:=i-1
  end;
  A[0]:=-A[0]*X[m]
end;
writeln;
for i:=0 to n-1 do writeln('A[',i,']:= ',A[i])
END.
```

```
{ Простые делители                                             OL
PROGRAM OLI833;
```

```
var i,j,n : integer;
BEGIN
write('N:= ');
readln(n);
writeln('Program get N= ',n);
j:=0; i:=2;
while  n>1 do
begin
  while (n mod i)<>0 do i:=i+1;
  if i<>j then
  begin
    writeln(i);
    j:=i
  end;
  n:=n div i
end
END.
```

```
{ Оптовая покупка                                                    OL
PROGRAM OLI884;
var n1,n2,n3,n,m : integer;
BEGIN
write('N:= ');
readln(n);
n1:=n div 144;
m:=n-n1*144;
n2:=m div 12;
n3:=m-n2*12;
if (n3*1.05)>10.25 then
begin
  n2:=n2+1;
  n3:=0
end;
if (n2*10.25+n3*1.05)>114 then
begin
  n1:=n1+1;
  n2:=0;
  n3:=0
end;
writeln(n1,' ',n2,' ',n3)
END.
```

```
{ Обнуление                                                          OL
PROGRAM OLI885;
const NN = 20;
      MM = 20;
var m,n,i,j : integer;
    z       : boolean;
    A       : array [1..MM,1..NN] of real;
    C       : array [1..NN] of boolean;
BEGIN
writeln('M,N:=');
readln(m,n);
for i:=1 to m do
  for j:=1 to n do
```

```
  begin
    write('A[',i,',',j,']:= ');
    readln(A[i,j])
  end;
for j:=1 to n do C[j]:=FALSE;
for i:=1 to m do
begin
  z:=FALSE;
  for j:=1 to n do
    if A[i,j]=0 then begin z:=TRUE; C[j]:=TRUE end;
  if z then  for j:=1 to n do A[i,j]:=0
end;
for j:=1 to n do
  if C[j] then  for i:=1 to m do A[i,j]:=0;
for i:=1 to m do
begin
  for j:=1 to n do write(A[i,j]:4);
  writeln
end
END.
```

§ 1.5. Решения на Си

Программы этого параграфа написаны в соответствии со стандартом языка Си (см. [9]). Независимо от указаний, данных в условии задач, все числовые переменные в наших программах мы описали как целые. Это не привело к изменению изложенных выше алгоритмов, но позволит школьнику исполнить приведенные программы на компиляторе Си, который не поддерживает выполнение арифметических действий над числами с плавающей точкой. Исключение составляют задачи 80.2.2 и 81.2, в решении которых используется функция «квадратный корень» (см., впрочем, п. 80.2.2 § 1.2).

ОЛИМПИАДА 80

```
/*  Простые до M                                          OLI80.1.1 *
#include <stdio.h>
#define N 150
main()
{ int m,i,j,k,q,P[N];
  scanf("%d",&m);
  if (m>=2) printf("2\n");
  if (m>=3) printf("3\n");
  P[k=0]=3;
  for (i=5; i<=m; i+=2)
    { for (j=0; j<=k; j++)
        { q=P[j]; if (q*q>i) break;
          if (!(i%q)) goto NP;
        }
      printf("%d\n",i);
      if (k<N-1) P[++k]=i;
```

```
NP:;)
}
```

/* Перестановки OLI80.

```
#include <stdio.h>
#define MM 100
#define PECH for (i=0; i<m; i++)  printf("%d ",A[P[i]]);\
             putchar('\n');
main()
{ int m,i,j,k,n,P[MM],A[MM];
B:printf("Введите число: 0<m<=%d \n",MM);
  scanf("%d",&m); if (m<=0 || m>MM) goto B;
  for (i=0; i<m; i++) {scanf("%d",&A[i]); P[i]=i;}
  PECH
  for (;;)
    { for (i=m-2; i>=0; --i)
        if (P[i]<P[i+1]) goto R;  exit(0);
R:    n=P[i]; j=m-1; while (n>=P[j] && j>i) j--;
      P[i]=P[j]; P[j]=n;
      for (k=1; i+k<m-k; k++)
        {n=P[i+k]; P[i+k]=P[m-k]; P[m-k]=n;}
      PECH
    }
}
```

/* Быстрая степень OLI80.

```
#include <stdio.h>
main()
{ int a,k,b=1;
  scanf("%d %d",&a,&k); printf("%d ^ %d = ",a,k);
  while (k)
    {if (k%2) b*=a; k/=2; a*=a;}
  printf("%d\n",b);
}
```

/* Арифметические действия OLI80.

```
#include <stdio.h>
#define M 35          /* значение результата */
#define N 6
char S[]="0+-*/";
main()
{ int i,j,k,y,A[N+1],B[N+1];
  B[1]=1; k=A[2]=0;
  for (i=3; i<=N; i++) A[i]=4;
R:for (i=N; i>1; --i)
    if (A[i]==4) A[i]=1;
    else
    { A[i]+=1; y=B[i-1];
      switch (A[i])
      { case 1: B[i]=y+i; break;
        case 2: B[i]=y-i; break;
        case 3: B[i]=y*i; break;
        case 4: B[i]=y/i;
      }
      for (j=i+1; j<=N; j++) B[j]=B[j-1]+j;
```

```
      if (B[N]!=M) goto R; k++;
      for (j=2; j<N; j++) printf("("); printf("1");
      for (j=2; j<=N; j++)
          { if (j>2) printf(")");
            printf("%c%d",S[A[j]],j);
          }
      printf("=%d\n",M);
      goto R;      /*поиск других вариантов*/
    }
  printf("Вариантов решения - %d\n",k);
}
```

```
/* Поиск равных                                                    OLI80
#include <stdio.h>
main()
{ int i,j,*p,A[2][15];
  p=&A[0][0];
  for (i=0; i<30; i++) scanf("%d",p+i);
  for (i=0; i<30; i++)
    printf("%3d%c",*(p+i),(i+1)%15? ' ':'\n');
  for (i=0; i<30; i++)
    for (j=i+1; j<30; j++)
      if (*(p+i)==*(p+j)) goto F;
F:printf("A[%d][%d]=A[%d][%d]=%d\n",
          i/15,i%15,j/15,j%15,*(p+i));
}
```

```
/* Сумма квадратов                                                 OLI80
#include <stdio.h>
#include <math.h>  /*Транслировать (cc) надо с ключом -lm */
main()
{ int m,i,j;
  scanf("%d",&m);
  for (i=1; 2*i*i<=m; i++)
    { j=sqrt((double)(m-i*i));
      if (i*i+j*j==m) break;
    }
  if (i*i+j*j==m) printf("%d^2+%d^2=%d\n",i,j,m);
  else printf("Нет\n");
}
```

```
/* Различные числа                                              OLI80.2
#include <stdio.h>
#define N 100
main()
{ int m,i,j,s,b,A[N];
  scanf("%d",&m);
  for (i=0; i<m; i++) scanf("%d",&A[i]);
  for (s=0,i=0; i<m; i++)
    { for (b=1,j=i+1; j<m; j++)
        if (A[i]==A[j]) {b=0; break;}
      s+=b;
    }
  printf("В массиве \n");
  for (i=0; i<m; i++)
```

```
    printf("%5d%c",A[i],i%10==9||i==m-1 ? '\n':' ');
  printf("разных чисел -  %d\n",s);
}
```

OLI80.3.1

```
/* Заданная сумма цифр */
#include <stdio.h>
main()
{ int N,i,j,k;
  scanf("%d",&N);
  for (i=0; i<=9; i++)
    for (j=0; j<=9; j++)
      { k=N-i-j;
      if (k>=1 && k<=9) printf("%d\n",i+10*j+100*k);
      }
}
```

OLI80.3.2

```
/* M+1 в двоичной записи */
#include <stdio.h>
#define M 20      /* цифр в числе не больше 18 */
main()
{ int i=0; char A[M];
  printf("Введите цифры двоичного числа\
         (младшие разряды слева)\n");
  scanf("%s",A);
  while (A[i]=='1') A[i++]='0';
  if (A[i]=='\0') A[i+1]='\0';  A[i]='1';
  printf("%s\n",A);
}
```

OLI80.3.3-2

```
/* Максимум минимумов */
#include <stdio.h>
#define MM 10
#define NN 20
main()
{ int i,j,k,m,n,X[MM][NN],min,max;
  scanf("%d %d",&m,&n);
  for (i=0; i<m; i++)
    for (j=0; j<n; j++) scanf("%d",&X[i][j]);
  for (i=0; i<m; i++)
    { for (j=0; j<n; j++)
        { if (i>0 && X[i][j]<max) goto SI;
          if (j==0 || X[i][j]<min) min=X[i][j];
        }
      max=min; k=i;
SI:;}
  printf("max(min)=%d в строке %d\n",max,k);
}
```

OLI80.3.4

```
/* Перестановка 0,1,2 */
#include <stdio.h>
#define NN 100
main()
{ int n,i,A[3],X[NN];
  scanf("%d",&n);
  for (i=0; i<n; i++) scanf("%d",&X[i]);
```

```
  A[0]=A[1]=A[2]=0;
  for (i=0; i<n; i++) A[X[i]]++;
  for (i=0; i<n; i++)
    { X[i]=i<A[0]? 0: i<n-A[2]? 1:2;
      printf("%d",X[i]);
    }
  printf("\n");
}
```

ОЛИМПИАДА 81

```
/* Функция                                                    OLI
#include <stdio.h>
main()
{ int n,i,j;
  scanf("%d",&n);
  i=1; j=0;
  while (n>0)
    {if (!(n%2)) i+=j; else j+=i; n/=2;}
  printf("%d\n",j);
}
```

```
/* Пара четверок                                              OLI
#include <stdio.h>
#include <math.h>     /* Трансляция (cc) с ключом -lm */
main()
{ int n,b,i,j,k,p,i1,j1,k1,p1,r,r1,r2;
  for (b=0,n=7; ; n++)
    { for (i=1; (r=i*i)<n; i++)
        for (j=1; j<=i && (r1=r+j*j)<n; j++)
          for (k=1; k<=j && (r2=r1+k*k)<n; k++)
            { p=sqrt((double)(n-r2));
              if (r2+p*p==n && p<=k)
                { if (b) goto END;
                  i1=i; j1=j; k1=k; p1=p;
                  b=1;
                }
            }
    }
END: printf("%d=%d^2+%d^2+%d^2+%d^2=%d^2+%d^2+%d^2+%d^2\n",
           n,i,j,k,p,i1,j1,k1,p1);
}
```

```
/* Спираль                                                    OLI
#include <stdio.h>
#define NN  19
#define AK  A[i][j]=++k; if (k==n*n) goto REZ;
main()
{ int i,j,k,n,A[NN][NN];
  scanf("%d",&n);
  k=i=j=0;
  for ( ; ; )
    { do {AK; j++;} while (i+j<n-1);
      do {AK; i++;} while (i<j);
```

```
      do {AK; j--;} while (i+j>n-1);
      do {AK; i--;} while (i>j+1);
    }
REZ:  for (i=0; i<n; i++)
       { for (j=0; j<n; j++) printf("%4d ",A[i][j]);
         printf("\n");
       }
}
```

```
/* Числа из разных цифр                                          OLI
#include <stdio.h>
main()
{ int i,j,k,m;
  for (i=1; i<=9; i++)
    for (j=0; j<=9; j++) if (i!=j)
      for (k=0; k<=9; k++) if (k!=i && k!=j)
        for (m=0; m<=9; m++)
          if (m!=i && m!=j && m!=k)
             printf("%d\n",(((i*10+j)*10+k)*10)+m);
}
```

```
/* Серия нулей                                                   OLI
#include <stdio.h>
#define NN 100
main()
{ int n,i,t,max,A[NN];
  scanf("%d",&n);
  for (i=0; i<n; i++) scanf("%d",&A[i]);
  for (t=max=i=0; i<n; i++)
    if (A[i])
      {if (max<t) max=t; t=0;}
    else t++;
  if (max<t) max=t;
  printf("%d\n",max);
}
```

ОЛИМПИАДА 82

```
/* Прямоугольники                                              OLI82
#include <stdio.h>
#define MM 20
#define NN 20
main()
{ int m,n,j,i,s,A[MM][NN];
  scanf("%d %d",&m,&n);
  for (i=0; i<=m; i++)
    for (j=0; j<=n; j++)
      if (A[i][j]=i*j) scanf("%d",&A[i][j]);
  for (s=0,i=1; i<=m; i++)
    for (j=1; j<=n; j++)
      if (A[i][j] && !(A[i-1][j]+A[i][j-1])) s++;
  printf("%d\n",s);
}
```

```
/* Упорядоченные дроби                                          OLI82.
#include <stdio.h>
main()
{ int P,n,m,i,j,a,b;
  scanf("%d",&P);
  m=0; n=1;
  do
    { printf("%d/%d\n",m,n);
      i=j=1;
      for (b=2; b<=P; b++)
        { a=m*b/n+1;
          if (a*j<b*i) {i=a; j=b;}
        }
      m=i; n=j;
    }
  while (i<j);
}

/* Сумма по подмножеству                                        OLI82.
#include <stdio.h>
#define NN 100
main()
{ int n,m,s,i,p,A[NN],B[NN];
  scanf("%d  %d",&n,&m);
  for (i=0; i<n; i++) {scanf("%d",&A[i]); B[i]=0;}
  s=0;
  do
    { for (i=0; i<n; i++)
        if (B[i]) {B[i]=0; s-=A[i];} else break;
      B[i]=1; s+=A[i];
    }
  while (s!=m);
  for (p=1,i=0; i<n; i++)
    if (B[i])  {printf("%c%d",p?' ':'+',A[i]); p=0;}
  printf("=%d\n",m);
}

/* Нули - в конец                                               OLI82.
#include <stdio.h>
#define NN 100
main()
{ int n,i,j,X[NN];
  scanf("%d",&n);
  for (i=0; i<n; i++) scanf("%d",&X[i]);
  for (j=-1,i=0; i<n; i++)
    if (X[i]) {j++; if(j<i) {X[j]=X[i];X[i]=0;}}
  for (i=0; i<n; i++) printf("%d\n",X[i]);
}

/* Седловая точка                                               OLI82.
#include <stdio.h>
#define MM 20
#define NN 20
main()
{ int m,n,i,j,i0,mi,ma,A[MM][NN];
```

```
  scanf("%d %d",&m,&n);
  for (i=0; i<m; i++)
    for (j=0; j<n; j++) scanf("%d",&A[i][j]);
  for (i=0; i<m; i++)
    { for (j=0; j<n; j++) printf("%3d ",A[i][j]);
      printf("\n");
    }
  for (i=0; i<m; i++)
    { for (j=0; j<n; j++)
        { if (i>0 && A[i][j]<=ma) goto SI;
          if (j==0 || A[i][j]<mi) mi=A[i][j];
        }
      ma=mi; i0=i;
SI:;}
  for (j=0; j<n; j++)
    { for (i=0; i<m; i++)
        if (A[i][j]>ma) goto KJ;
      printf("%d %d\n",i0,j); goto END;
 KJ:;}
  printf("Нет\n"); END:;
}

/* Вхождение слова в текст
#include <stdio.h>
#define NN 100
#define KK  50
main()
{ int n,k,i,j,X[NN],Y[KK];
  scanf("%d %d",&n,&k);
  for (i=0; i<n; i++) scanf("%d ",&X[i]);
  for (j=0; j<k; j++) scanf("%d ",&Y[j]);
  for (i=0; i<=n-k; i++)
    { for (j=0; j<k; j++)
        if (X[i+j]!=Y[j]) goto NI;
      printf("Да, с элемента X[%d]\n",i); goto END;
NI:;}
  printf("Нет\n"); END:;
}
```

ОЛИМПИАДА 83

```
/* Бит-реверс                                          0
#include <stdio.h>
main()
{ int m,n,b,k;
  k=512; m=1;
  while (m<1024)
    { while (m>=k) {m-=k; k/=2;}
      m+=k; k=512;
      printf("%4d(%3x)\n",m,m);
    }
}
```

```
/* Бит-реверс                                                    OLI83.
#include <stdio.h>
main()
{ int m,b,k;
  for (m=512; m<1024; m++)
    { for (b=k=0; k<10; k++)
          b |= ( (m >> k) & 01) << (9-k);
      printf("%4d(%3x)  %4d(%3x)\n",m,m,b,b);
    }
}
```

```
/* Треугольник и точка                                           OLI8
#include <stdio.h>
#define RZ(A,B) (xt-x[A])*(y[B]-y[A])>(x[B]-x[A])*(yt-y[A])
main()
{ int x[3],y[3],xt,yt,d;
  scanf("%d  %d  %d",x,x+1,x+2);
  scanf("%d  %d  %d",y,y+1,y+2);
  scanf("%d  %d",&xt,&yt);
  d =RZ(0,1);
  d=(d==RZ(1,2))&&(d==RZ(2,0));
  printf("%s\n",d? "да":"нет");
}
```

```
/* Лабиринт                                                      OLI83.
#include <stdio.h>
#define MM 10
#define NN 10
int pr,A[MM][NN],m,n;
L(i,j)
int i,j;
{ if (pr) ;
  else if ( A[i][j] == 0 )
         { if (i==0 || i==m-1 || j==0 || j==n-1) pr=1;
           A[i][j]=1;
           L(i,j-1); L(i,j+1); L(i-1,j); L(i+1,j);
           if (pr) printf("%d %d \n",i,j);
         }
}
main()
{ int i,j,p,q;
  scanf("%d %d",&m,&n);
  for (p=0; p<m; p++)
    for (q=0; q<n; q++)
      scanf("%d",&A[p][q]);
  for (p=0; p<m; p++)
    {for (q=0; q<n; q++)
       printf("%d",A[p][q]);
     printf("\n");
    }
  scanf("%d %d",&i,&j);
  pr=0;  L(i,j);
  if (!pr) printf("Нет выхода\n");
}
```

```
/* Пила                                                    OLI
#include <stdio.h>
#define MM 100
main()
{ int i,j,k,m,X[MM];
  scanf("%d",&m);
  for (i=0; i<m; i++) scanf("%d",&X[i]);
  for (j=k=1,i=0; i<m-2; )
    { if (X[i]<X[i+1] && X[i+1]>X[i+2])
      { i+=2; j+=2;
        if (k<j) k=j;
      }
      else
      { i++; j=1;
        if (i+k>=m-1) break;
      }
    }
 printf("%d\n",k);
}
```

```
/* Сократить дробь                                         OLI83
#include <stdio.h>
main()
{ int m,n,i,j;
  scanf("%d %d",&m,&n);
  for (j=1; j<=n; j++)
  { i=j*m/n;
    if (i*n==j*m) break;
  }
  printf("%d/%d=%d/%d\n",m,n,i,j);
}
```

ОЛИМПИАДА 84

```
/* Инверсия                                                OLI
#include <stdio.h>
#define NN 100
main()
{ int N,i,j,k,P[NN],T[NN];
  scanf("%d",&N);
  for (i=1; i<=N; i++) {scanf("%d",&T[i]); P[i]=0;}
  for (i=1; i<=N; i++)
    { j=k=0;
      do
        if (P[++k]==0) j++;
      while(j<=T[i]);
      P[k]=i;
    }
  for (i=1; i<=N; i++) printf("%3d  %3d\n",P[i],T[i]);
}
```

```
/* Дорога                                                  OLI
#include <stdio.h>
#define MM 10
```

```
#define NN 10
main()
{ int m,n,i,j,k;
  int x,r,A[MM][MM][NN-1],B[MM],C[MM];
  scanf("%d %d",&m,&n);
  for (k=0; k<n-1; k++)
    { for (i=0; i<m; i++)
        { B[i]=0;
          for (j=0; j<m; j++) scanf("%d",&A[i][j][k]);
        }
    }
  for (k=0; k<n-1; k++)
    { for (j=0; j<m; j++)
        { r=B[0]+A[0][j][k];
          for (i=1; i<m; i++)
            { x=B[i]+A[i][j][k];
              if (x<r) r=x;
            }
          C[j]=r;
        }
      for (j=0; j<m; j++) B[j]=C[j];
    }
  r=B[0];
  for (i=1; i<m; i++) if (B[i]<r) r=B[i];
  printf("%d\n",r);
}

/* Совершенные числа                                          OLI 84
#include <stdio.h>
main()
{ int m,i,j,k,s;
  scanf("%d",&m);
  for (i=2; i<=m; i++)
    { s=j=1;
      do
        { j++;
          k=i/j;
          if (i==k*j && j<=k)
            {s+=j; if (j<k) s+=k;}
        } while (j<k);
      if (s==i) printf("%d\n",i);
    }
}

/* Период дроби                                               OLI 84
#include <stdio.h>
main()
{ int M,N,i,j,k;
  scanf("%d %d",&M,&N);
  M%=N;
  for (k=1; (k<=N)||(j!=M); k++)
    { if (k==N) j=M;
      i=10*M/N;
      M=(10*M)%N;
```

```
      if (k>=N) printf("%d",i);
    }
  printf("\n");
}

/* Слияние массивов                                                    OLI
#include <stdio.h>
#define MM 100
#define NN 100
main()
{ int M,N,i,j,k,A[MM],B[NN],C[MM+NN];
  scanf("%d %d",&M,&N);
  for (i=0; i<M; i++) scanf("%d",&A[i]);
  for (j=0; j<N; j++) scanf("%d",&B[j]);
  for(k=i=j=0; k<M+N; k++)
    { if              (i>=M) ^   C[k]=B[j++];
      else if         (j>=N)     C[k]=A[i++];
           else if (A[i]>B[j])   C[k]=B[j++];
                else             C[k]=A[i++];
    }
  for (k=0; k<M+N; k++)
     printf("%4d%c",C[k], (k+1)%10==0||k==M+N-1?'\n':' ');
}

/* Календарь                                                           OLI
#include <stdio.h>
#define D(x) (c%x==0)
main()
{ int a,b,c,i,j,M[12];
  M[1]=M[3]=M[5]=M[7]=M[8]=M[10]=M[12]=31;
  M[4]=M[6]=M[9]=M[11]=30;
  M[2]=28;
  printf("Введите дату: число, месяц и год\n");
  scanf("%d %d %d",&a,&b,&c);
  j=a;
  for (i=1; i<b; i++) j+=M[i];
  if ((b>2) && (D(4) && !D(100) || D(400))) j++;
  printf("Этот день %d в году\n",j);
}

/* Квадратики                                                          OLI
#include <stdio.h>
#define MM 100
main()
{ int i,j,M,s,A[MM][MM];
  scanf("%d",&M);
  for (i=0; i<M; i++)
    for (j=0; j<M; j++)  scanf("%d",&A[i][j]);
  for (i=0; i<M; i++)
    { for (j=0; j<M; j++) printf("%2d ",A[i][j]);
      putchar('\n');
    }
  for (s=i=0; i<M-1; i++)
    for (j=0; j<M-1; j++)
      if (A[i][j]+A[i][j+1]+A[i+1][j]+A[i+1][j+1]==17) s++;
```

```
  printf("%d\n",s);
}
```

ОЛИМПИАДА 85

```
/* Разложение на слагаемые                                          OLI
#include <stdio.h>
#define NN 100
main()
{   int N,M[NN],i,k,t,s;
    scanf("%d",&N); printf("%d\n",N);
    M[0]=N; k=i=0;
RA: t=M[k]-1; s=t+i-k+1;
    for (i=k; i<N; i++)
      if (s>t) {M[i]=t; s-=t;}
      else     {M[i]=s; break;}
    for (k=0; k<=i; k++) printf("%d ",M[k]);
    printf("\n");
    for (k=i; k>=0; --k) if (M[k]>1) goto RA;
}
```

```
/* Равные элементы                                                  OLI
#include <stdio.h>
#define NN 20
#define MM 20
main()
{  int M,N,A[MM][NN],B[NN],i,j,k,l;
   scanf("%d %d",&M,&N);
   for (i=0; i<M; i++)
     {B[i]=0; for (j=0; j<N; j++) scanf("%d",&A[i][j]);}
   for (i=0; i<M; i++)
      { for (j=0; j<N; j++) printf("%3d ",A[i][j]);
        printf("\n");
      }
   k=0; l=0;
BI:for (i=1; i<M && k<N; i++)
BJ:{ l++; if (A[0][B[0]]==A[i][B[i]]) ;
          else if (A[0][B[0]]>A[i][B[i]])
                 {if ((k=++B[i])<N) goto BJ;}
               else {k=++B[0]; goto BI;}
   }
      if (k>=N) printf("Нет\n");
      else printf("%d\n",A[0][B[0]]);
    printf("сделано %d шагов\n",l);
}
```

```
/* Несоставляемое число                                             OLI
#include <stdio.h>
#define NN 100
main()                  /* без упорядочивания массива */
{ int N,P[NN],i,j,s;
  scanf("%d",&N);
  for (i=0; i<N; i++) scanf("%d",&P[i]);
  for (s=1,i=0; i<N; i++)
```

```
    { for (j=i; j<N; j++) if (P[j]<=s) goto PI;
      break;
PI:   s+=P[j]; P[j]=P[i];
    }
  printf("%d\n",s);
}

/* Тетраэдры                                                    OLI
#include <stdio.h>
int M1,M2,M3,M4,N1,N2,N3,N4;
DA(i,j,k,l)
   int i,j,k,l;
  { return( ((M1==i)&&(((M2==j)&&(M3==k)&&(M4==l))\
                      ||((M2==l)&&(M3==j)&&(M4==k))\
                      ||((M2==k)&&(M3==l)&&(M4==j)))) );
  }
main()
{ int k;
  scanf("%d %d %d %d\n",&M1,&M2,&M3,&M4);
  scanf("%d %d %d %d\n",&N1,&N2,&N3,&N4);
  k =  DA(N1,N2,N3,N4) || DA(N2,N1,N4,N3)\
    || DA(N3,N1,N4,N2) || DA(N4,N1,N3,N2);
  printf("%s\n", k ? "DA":"NET");
}

/* Мода                                                      OLI 8
#include <stdio.h>
#define NN 100
main()
{ int N,m,am,i,j,k,A[NN];
  scanf("%d",&N);
  for (i=0; i<N; i++) scanf("%d",A+i);
  for (m=0,i=0; i+m<N; i+=k)
    { for(k=1,j=i+1; j<N; j++)
        if (A[j]==A[i]) k++;
      if (m<k) {am=A[i]; m=k;}
    }
  printf("число %d встречается %d раз\n",am,m);
}

/* Системы счисления                                            OLI
#include <stdio.h>
main()
{ int i,j,k,n,p,M[9];
  scanf("%d %d",&i,&j);
  for (k=0; k<9; k++) scanf("%d",&M[k]);
  for (n=0,k=8; k>=0; --k) n=n*i+M[k];
  printf("%d в десятичной системе равно\n",n);
  p=8; while (!M[p]) p--;
  for (k=0; k<=p; k++) printf("%ld",M[k]);
  printf("(слева разряд единиц) в %d-ричной системе и\n",i);
  do
    {printf("%d",n%j); n/=j;}
  while (n>0);
```

```
  printf("(слева разряд единиц) в %d-ричной системе\n", j);
}
```

```
/* Побочная диагональ                                          OLI
#include <stdio.h>
#define MM 10
#define NN 10
main()
{ int M, N, j, k, p, q, s, A[MM][NN];
  scanf("%d %d %d", &M, &N, &k);
  for (p=0; p<M; p++)
    for (j=0; j<N; j++) scanf("%d", &A[p][j]);
  p = k>0 ? 0: -k;
  q = k+N<M ? N: M-k;
  for (s=0, j=p; j<q; j++) s+=A[k+j][j];
  printf("%d\n", s);
}
```

ОЛИМПИАДА 86

```
/* Без тройных повторений                                      OLI
#include <stdio.h>
#define N 51          /* Последовательность начнется с A[1] */
main()
{     int i, j, k, A[N];
      i=0;
PER:  i++; if (i>N) goto VYH; A[i]=-1;
BOK:  if (A[i]>0) goto ZAD; A[i]++; if (i<3) goto PER;
PRV:  for (j=1; j<=i/3; j++)
        { for (k=i; k>=i-j+1; --k)
            if ((A[k]!=A[k-j]) || (A[k]!=A[k-j-j])) goto JP1;
          goto BOK;
JP1:;   }
      goto PER;
ZAD:  i--; if (i>=1) goto BOK;
VYH:  if (i<N) printf("Нет");
      else for (i=1; i<N; i++) printf("%d", A[i]);
      printf("\n");
}
```

```
/* Покер                                                       OLI
#include <stdio.h>
main()
{ int i, j, p, A[5];
  for (i=0; i<5; i++) scanf("%d", &A[i]);
  for (p=0, i=0; i<4; i++)
    for (j=i+1; j<5; j++) p+=A[i]==A[j];
  if      (p==10) printf("1\n");
  else if (p==6)  printf("2\n");
       else       printf("%d\n", 7-p);
}
```

```
/* Барабан                                                     OLI
#include <stdio.h>
```

```
#define N 12
main()
{ int i,j,l,k,p,A[N+1];
  for (i=1; i<=N; i++) scanf("%d",&A[i]);
  for (k=1,p=2; p<=N; p++)
    { for (i=0; i<=N-1; i++)
        { j=k+i; if (j>N) j-=N;
          l=p+i; if (l>N) l-=N;
          if (A[l]<A[j]) k=p;
          if (A[l]!=A[j]) break;
        }
    }
  printf("%d\n",k);
}
```

```
/* Дважды монотонный                                          OL
#include <stdio.h>
#define MM 20
#define NN 20
main()
{ int i,j,M,N,x,A[MM][NN];
  scanf("%d %d %d",&M,&N,&x);
  for (i=0; i<M; i++)
    for (j=0; j<N; j++) scanf("%d",&A[i][j]);
  for (i=0,j=N-1; i<M && j>=0; )
    { if (A[i][j]==x)
        {printf("A[%d][%d]=%d\n",i,j,x); break;}
      if (A[i][j]<x) i++; else j--;
    }
  if (i>=M || j<0) printf("нет числа %d\n",x);
}
```

```
/* Центральное селение                                        OL
#include <stdio.h>
#define K 20
main()
{ int i,j,k,il,s,t,A[K][K];
  scanf("%d",&k);
  for (i=0; i<k; i++)
    for (j=0; j<k; j++) scanf("%d",&A[i][j]);
  for (i=0; i<k; i++)
    { for (s=0,j=0; j<k; j++)
        if (j!=i && s<A[i][j]) s=A[i][j];
      s+=A[i][i];
      if (i==0 || s<t) {il=i; t=s;}
    }
  printf("%d\n",il);
}
```

ОЛИМПИАДА 87

```
/* Рюкзак                                                     OL
#include <stdio.h>
#define NN 100
```

```
#define T 30
main()
{ int i,s,z,zm,N,A[NN],B[NN],P[NN];
  scanf("%d",&N);
  for (i=0; i<N; i++) scanf("%d %d",&A[i],&B[i]);
  s=z=zm=0; i=-1;
V: for (i=i+1; i<N; i++)
    if (s+A[i]>=T) P[i]=1;
    else {s+=A[i]; z+=B[i]; P[i]=0;}
  if (zm<z) zm=z;
  for (i=N-2; i>=0; i--)
    { if (P[i+1]==0) {s-=A[i+1]; z-=B[i+1];}
      else if (P[i]==0) {s-=A[i]; z-=B[i]; P[i]=1; goto V;}
    }
  printf("%d\n",zm);
}
```

```
/* Полукратные                                                    OLI
#include <stdio.h>
#define NN 100
main()
{ int i,k2,k3,a2,a3,N,A[NN];
  scanf("%d",&N);
  k2=k3=0; A[0]=1; printf("1\n");
  for (i=1; i<N; i++)
    { a2=2*A[k2]+1; a3=3*A[k3]+1;
      if (a2<=a3) {A[i]=a2; k2++;}
      if (a3<=a2) {A[i]=a3; k3++;}
      printf("%d\n",A[i]);
    }
}
```

```
/* Сумма кубов                                                    OLI
/* Для всех чисел до MM проверяется, какие числа                 */
/* представимы суммой двух кубов натуральных чисел               */
#include <stdio.h>
main()
{ int i,j,m,k,N,MM;
  scanf("%d",&MM);
  for (N=1; N<MM; N++)
    { m=0;
      for (i=j=1; j*j*j+1<N; j++);
      do
       { k=i*i*i+j*j*j;
         if (k==N)
            {m++; printf("%d^3+%d^3=%d m=%d\n",i,j,N,m);}
         if (k<=N) i++;
         if (k>=N) j--;
       } while (i<=j);
    }
}
```

```
/* Перевертыши                                                    OLI
#include <stdio.h>
#define NN 100
```

```
main()
{ int i,ln,pn,l,p,m,max,z,N,A[NN];
  scanf("%d",&N);
  for (i=0; i<N; i++) scanf("%d",&A[i]);
  max=z=1;
  for (ln=0,pn=1; 2*(N-pn+1)+1>max; (z=-z)>0?pn++:ln++)
    { for (l=ln,p=pn; l>=0 && p<N && A[l]==A[p]; l--,p++);
      m=p-l-1;
      if (max<m) max=m;
    }
  printf("%d\n",max);
}

/* Индексы порядка                                          0
#include <stdio.h>
#define NN          100
#define SWAP(a,i,j) r=a[i]; a[i]=a[j]; a[j]=r;
#define FORJN       for (j=0; j<N; j++)
main()
{ int j,mi,k,r,N,A[NN],I[NN];
  scanf("%d",&N);
  FORJN {scanf("%d",A+j); I[j]=j;}
  FORJN { mi=j; for (k=j+1; k<N; k++)
                  if (A[k]<A[mi]) mi=k;
          SWAP(A,j,mi)   SWAP(I,j,mi)
          printf("%d ",I[j]);
        }
}

/* Индексы порядка                                          OLI
#include <stdio.h>
#define NN 100
main()
{ int i,j,m,mi,ma,k,N,A[NN];
  scanf("%d",&N);
  for (i=0; i<N; i++) scanf("%d",&A[i]);
  for (mi=ma=k=0; k<N; k++)
    { if (A[k]<A[mi]) mi=k;
      if (A[k]>A[ma]) ma=k;
    }
  printf("%d\n",mi);
  for (m=1; m<N; m++)
    { i=ma;
      for (k=0; k<N; k++)
        { if (A[k]<A[mi]) continue;
          if ((A[k]==A[mi]) && (k<=mi)) continue;
          if (A[k]==A[mi]) {i=k; break;}
          if (A[k]<A[i]) i=k;
        }
      mi=i;  printf("%d\n",mi);
    }
}
```

```
/* Правый больший                                                    OLI
#include <stdio.h>
#define NN 100
main()
{ int i,j,k,N,A[NN],B[NN];
  printf("Введите размер массива\n"); scanf("%d",&N);
  printf("Введите элементы массива\n");
  for (i=0; i<N; i++) scanf("%d",&A[i]);
  k=N-1;  B[k]=A[N-1]; A[N-1]=0;
  for (i=N-2; i>=0; i--)
    { for (j=k; j<N; j++)
        if (A[i]<B[j]) break;
      k=j-1; B[k]=A[i];
      if (j<N) A[i]=B[j]; else A[i]=0;
    }
  for (i=0; i<N; i++) printf("%d  ",A[i]);
}
```

```
/* Многочлен                                                         OLI
#include <stdio.h>
#define NN 100
main()
{ int i,m,N,A[NN],X[NN];
  scanf("%d",&N);
  for (i=1; i<=N; i++) scanf("%d",&X[i]);
  A[0]=-X[1]; A[1]=1;
  for (m=2; m<=N; m++)
    { for (i=m-1; i>=1; i--) A[i]=A[i-1]-A[i]*X[m];
      A[m]=1; A[0]=-A[0]*X[m];
    }
  for (i=0; i<N; i++) printf("%d\n",A[i]);
}
```

```
/* Простые делители                                                  OLI
#include <stdio.h>
main()
{ int i,j,N;
  scanf("%d",&N);
  i=2; j=0;
    while( N>1 )
      if ( N%i ) i++;
      else
       {N/=i; if(i!=j) {printf("%d\n",i); j=i;}}
}
```

```
/* Оптовая покупка                                                   OLI
#include <stdio.h>
main()
{ int n1,n2,n3,N,m;
  scanf("%d",&N);
  n1=N/144;
  m=N%144;
  n2=m/12;
```

```
  n3=m%12;
  if (n3*105>1025) {n2++; n3=0;}
  if (n2*1025+n3*105>11400) {n1++; n2=n3=0;}
  printf("Для покупки %d пар надо купить\
        %d кор., %d св., %d пар.\n",N,n1,n2,n3);

}
```

```
/* Обнуление                                                    OLI 88.5
#include <stdio.h>
#define NN 10
#define MM 15
#define FORIM for(i=0; i<M; i++)
#define FORJN for(j=0; j<N; j++)
main()
{ int M,N,i,j,z,A[MM][NN],C[NN];
  printf("Введите размеры массива\n"); scanf("%d %d",&M,&N);
  FORIM FORJN scanf("%d",&A[i][j]);
  printf(" Исходный массив : \n");
  FORIM {FORJN printf("%3d ",A[i][j]); printf("\n");}
  FORJN C[j]=0;
  FORIM { z=0; FORJN if (A[i][j]==0) C[j]=z=1;
          if (z) FORJN A[i][j]=0;
        }
  FORJN if (C[j]) FORIM A[i][j]=0;
  printf(" Результат :\n");
  FORIM {FORJN printf("%3d ",A[i][j]); printf("\n");}
}
```

§ 1.6. Решения на Фортране

Приведенные здесь программы написаны на широко распространенном Фортране IV и могут быть выполнены почти на любом трансляторе Фортрана. Школьники, изучившие более поздние версии, например Фортран 77, смогут некоторые программы переписать короче

ОЛИМПИАДА 80

```
C     Простые до M                                               OLI 80.1.
      DIMENSION K(500)
      READ 100,M
C     Число M>=5
      PRINT 110,M
      N=1
      K(1)=3
      DO 2 I=5,M,2
      DO 1 J=1,N
      KJ=K(J)
      IF(I/KJ*KJ.EQ.I) GOTO 2
    1 CONTINUE
      PRINT 100,I
```

```
    IF(I*I.GT.M) GOTO 2
    N=N+1
    K(N)=I
  2 CONTINUE
    STOP
100 FORMAT(1X,I6)
110 FORMAT(' M=',I6/7H       2/7H       3)
    END
```

Перестановки OLI

```
    REAL A(9),B(9)
    INTEGER P(9)
    READ 100,M,(A(I),I=1,M)
    IF(M.EQ.1) GOTO 10
    M1=M-1
    DO 1 I=1,M
  1 P(I)=I
    GOTO 8
  2 DO 3 I=1,M1
    K=M-I
    IF(P(K).LT.P(K+1)) GOTO 4
  3 CONTINUE
    GOTO 11
  4 N=P(K)
    DO 5 J=1,I
    L=M-J+1
    IF(N.LT.P(L)) GOTO 6
  5 CONTINUE
  6 P(K)=P(L)
    P(L)=N
    L=(M-K)/2
    DO 7 I=1,L
    N=P(K+1)
    P(K+1)=P(M+1-I)
  7 P(M+1-I)=N
  8 DO 9 I=1,M
    N=P(I)
  9 B(I)=A(N)
    PRINT 110,(B(I),I=1,M)
    GOTO 2
 10 PRINT 110,A(1)
 11 STOP
100 FORMAT(I1,9(2X,F5.2))
110 FORMAT(' ',9(F5.2,2X))
    END
```

Быстрая степень OLI

```
    READ 100,A,K
    PRINT 100,A,K
    B=1
    GOTO 2
  1 A=A*A
  2 N=K/2
    IF(N+N.LT.K) B=B*A
    K=N
```

```
      IF(K.GT.0) GOTO 1
      PRINT 110,B
      STOP
100   FORMAT(1X,F5.2,2X,I2)
110   FORMAT(' ',E13.6)
      END
```

Арифметические действия OLI 80

```
      INTEGER A(8),B(9),C(9),
     #Z(4)/'+','-','*','/'/,E/'='/
      M=35
      N=6
      K=0
      B(1)=1
      C(N)=E
      A(1)=0
      N1=N-1
      DO 1 I=2,N1
    1 A(I)=4
    2 DO 12 I=1,N1
      J=N-I
      IF(A(J).EQ.4) GOTO 11
      A(J)=A(J)+1
      L=A(J)
      J=J+1
      GOTO(3,4,5,6), L
    3 B(J)=B(J-1)+J
      GOTO 7
    4 B(J)=B(J-1)-J
      GOTO 7
    5 B(J)=B(J-1)*J
      GOTO 7
    6 B(J)=B(J-1)/J
    7 J=J+1
      IF(J.GT.N) GOTO 9
      DO 8 L=J,N
    8 B(L)=B(L-1)+L
    9 IF(B(N).NE.M) GOTO 2
      K=K+1
      DO 10 L=1,N1
      J=A(L)
   10 C(L)=Z(J)
      PRINT 100,(L,C(L),L=1,N),M
      GOTO 2
   11 A(J)=1
   12 CONTINUE
      PRINT 110,K
      STOP
  100 FORMAT(9(I2,1X,A1),I2)
  110 FORMAT(' ',I5)
      END
```

Поиск равных OLI 80

```
      DIMENSION A(2,15),C(30)
      EQUIVALENCE (A(1,1),C(1))
```

```
      READ 100, A
      PRINT 100, ((A(I, J), J=1, 15), I=1, 2)
      DO 1 I=1, 29
      I1=I+1
      R=C(I)
      DO 1 J=I1, 30
      IF(C(J).EQ.R) GOTO 2
    1 CONTINUE
    2 L=(I+1)/2
      K=I/L
      L1=(J+1)/2
      K1=J/L1
      PRINT 110, K, L, K1, L1
      STOP
  100 FORMAT(15(1X, F4.1))
  110 FORMAT(1X, I1, 2X, I2)
      END
```

```
C     Сумма квадратов                                          OLI
      READ 100, M
      PRINT 100, M
      M1=SQRT(FLOAT(M/2))+0.1
      DO 1 I=1, M1
      J=SQRT(FLOAT(M-I*I))+0.1
      IF(I*I+J*J.EQ.M) GOTO 2
    1 CONTINUE
      PRINT 110
      STOP 1
    2 PRINT 120, I, J
      STOP 2
  100 FORMAT(1X, I4)
  110 FORMAT(' Нет')
  120 FORMAT(2(1X, I2))
      END
```

Добавка 0.1 введена на случай, когда корень из квадрата на ного числа окажется вычислен с недостатком, а в иных случаях эта ка не вредит.

```
C     Различные числа                                          OLI 8
      DIMENSION A(1000)
      READ 100, M, (A(I), I=1, M)
      PRINT 100, M, (A(I), I=1, M)
      K=1
      IF(M.EQ.1) GOTO 3
      M1=M-1
      DO 2 I=1, M1
      R=A(I)
      DO 1 J=I, M1
      IF(A(J+1).EQ.R) GOTO 2
    1 CONTINUE
      K=K+1
    2 CONTINUE
    3 PRINT 100, K
      STOP
  100 FORMAT(1X, I4/15(1X, F4.1))
```

```
      END

      Заданная сумма цифр                                    OLI 80
      READ 100,N
      PRINT 100,N
      DO 2 I=1,10
      DO 1 J=1,10
      K=N-(J-1)-(I-1)
      IF(K.LT.1) GOTO 2
      IF(K.GT.9) GOTO 1
      M=100*K+10*(J-1)+I-1
      PRINT 110,M
    1 CONTINUE
    2 CONTINUE
      STOP
  100 FORMAT(1X,I2)
  110 FORMAT(1X,I3)
      END

      M+1 в двоичной записи                                  OLI 80
      INTEGER*2 A(31)
      READ 100,N,(A(I),I=1,N)
      PRINT 100,N,(A(I),I=1,N)
      DO 1 I=1,N
      IF(A(I).EQ.1) GOTO 1
      A(I)=1
      GOTO 2
    1 A(I)=0
      N=N+1
      A(N)=I
    2 PRINT 100,N,(A(I),I=1,N)
      STOP
  100 FORMAT(1X,I2/31(1X,I1))
      END

      Максимум минимумов                                     OLI 80
      REAL MAX,MIN,X(50,50)
      READ 100,M,N,((X(I,J),J=1,N),I=1,M)
      DO 1 I=1,M
    1 PRINT 110,(X(I,J),J=1,N)
      DO 3 I=1,M
      DO 2 J=1,N
      IF(I.GT.1.AND.X(I,J).LT.MAX) GOTO 3
      IF(J.EQ.1.OR.X(I,J).LT.MIN) MIN=X(I,J)
    2 CONTINUE
      MAX=MIN
      K=I
    3 CONTINUE
      PRINT 120,K
      STOP
  100 FORMAT(I2,1X,I2/10(1X,F7.3))
  110 FORMAT(10(2X,F7.3))
  120 FORMAT(' ',I2)
      END
```

```
C     Перестановка 0,1,2                                        OLI
      INTEGER*2 X(1000)
      READ 100,N,(X(I),I=1,N)
      PRINT 110,(X(I),I=1,N)
      N0=0
      N1=0
      DO 1 I=1,N
      IF(X(I).LE.1) N1=N1+1
    1 N0=N0+X(I)
      N0=N1-N0+(N-N1)*2
      DO 4 I=1,N
      IF(I.LE.N0) GOTO 3
      IF(I.LE.N1) GOTO 2
      X(I)=2
      GOTO 4
    2 X(I)=1
      GOTO 4
    3 X(I)=0
    4 CONTINUE
      PRINT 110,(X(I),I=1,N)
      STOP
  100 FORMAT(I4/40(1X,I1))
  110 FORMAT(60(1X,I1))
      END
```

ОЛИМПИАДА 81

```
      Функция                                                   OL
      READ 100,N
      PRINT 100,N
      I=1
      J=0
    1 K=N/2
      IF(K+K.LT.N) GOTO 2
      I=I+J
      GOTO 3
    2 J=I+J
    3 N=K
      IF(N.GT.0) GOTO 1
      PRINT 110,J
      STOP
  100 FORMAT(1X,I3)
  110 FORMAT(' ',I4)
      END

      Пара четверок                                             OLI
      LOGICAL V
      N=3
    1 N=N+1
      V=.FALSE.
      N1=SQRT(FLOAT(N))
      DO 4 I=1,N1
      DO 3 J=1,I
      IF(I*I+J*J.GE.N) GOTO 4
```

```
      DO 2 K=1,J
      IF(I*I+J*J+K*K.GE.N) GOTO 3
      L=SQRT(FLOAT(N-I*I-J*J-K*K))+0.1
      IF(I*I+J*J+K*K+L*L.NE.N.OR.L.GT.I) GOTO 2
      IF(V) GOTO 5
      I1=I
      J1=J
      K1=K
      L1=L
      V=.TRUE.
      GOTO 4
    2 CONTINUE
    3 CONTINUE
    4 CONTINUE
      GOTO 1
    5 PRINT 100,N,I,J,K,L,I1,J1,K1,L1
      STOP
  100 FORMAT(' ',I3/8(1X,I2))
      END
```

См. примечание к программе 80.2.2.

```
      Спираль                                                    OLI
      COMMON L,K
      INTEGER A(30,30)
      READ 100,N
      PRINT 100,N
      L=N*N
      K=0
      I=1
      J=1
    1 CALL SP(A(I,J),&5)
      J=J+1
      IF(I+J.LE.N) GOTO 1
    2 CALL SP(A(I,J),&5)
      I=I+1
      IF(I.LT.J) GOTO 2
    3 CALL SP(A(I,J),&5)
      J=J-1
      IF(I+J.GT.N+1) GOTO 3
    4 CALL SP(A(I,J),&5)
      I=I-1
      IF(I.GT.J+1) GOTO 4
      GOTO 1
    5 DO 6 I=1,N
    6 PRINT 110,(A(I,J),J=1,N)
      STOP
  100 FORMAT(1X,I2)
  110 FORMAT(30(1X,I3))
      END

      SUBROUTINE SP(X,*)
      COMMON L,K
      INTEGER X
      K=K+1
      X=K
```

```
      IF(X.EQ.L) RETURN 1
      RETURN
      END
```

```
      Числа из разных цифр                                    OLI 81.
      INTEGER A(24)
      N=0
      DO 3 I=2,10
      M1=(I-1)*10
      DO 3 J=1,10
      IF(J.EQ.I) GOTO 3
      M2=(M1+(J-1))*10
      DO 2 K=1,10
      IF(K.EQ.I.OR.K.EQ.J) GOTO 2
      M3=(M2+(K-1))*10
      DO 1 L=1,10
      IF(L.EQ.I.OR.L.EQ.J.OR.L.EQ.K) GOTO 1
      N=N+1
      A(N)=M3+(L-1)
      IF(N.LT.24) GOTO 1
      PRINT 100,A
      N=0
    1 CONTINUE
    2 CONTINUE
    3 CONTINUE
      STOP
   00 FORMAT(24(1X,I4))
      END
```

```
      Серия нулей                                             OLI 81.
      DIMENSION A(1000)
      READ 100,N,(A(I),I=1,N)
      PRINT 100,N,(A(I),I=1,N)
      MAX=0
      K=0
      I=0
    1 I=I+1
      IF(A(I).EQ.0) GOTO 2
      K=0
      IF(I+MAX.LT.N) GOTO 1
      GOTO 3
    2 K=K+1
      IF(K.GT.MAX) MAX=K
      IF(I.LT.N) GOTO 1
    3 PRINT 100,MAX
      STOP
   00 FORMAT(1X,I4/10(1X,F7.3))
      END
```

ОЛИМПИАДА 82

```
      Прямоугольники                                          OLI 8
      INTEGER*2 A(100,100)
```

```
      READ 100, M, N, ((A(I, J), J=1, N), I=1, M)
      DO 1 I=1, M
    1 PRINT 110, (A(I, J), J=1, N)
      K=0
      DO 4 I=1, M
      DO 4 J=1, N
      IF(A(I, J).EQ.0) GOTO 4
      IF(J.EQ.1) GOTO 2
      IF(A(I, J-1).EQ.1) GOTO 4
    2 IF(I.EQ.1) GOTO 3
      IF(A(I-1, J).EQ.1) GOTO 4
    3 K=K+1
    4 CONTINUE
      PRINT 120, K
      STOP
  100 FORMAT(I3, 1X, I3/40(I1, 1X))
  110 FORMAT(' ', 100I1)
  120 FORMAT(' ', I4)
      END

C     Упорядоченные дроби
      INTEGER A, B, P
      READ 100, P
      PRINT 100, P
      PRINT 110
      M=0
      N=1
    1 I=1
      J=1
      DO 2 B=1, P
      A=M*B/N+1
      IF(A*J.GE.B*I) GOTO 2
      I=A
      J=B
    2 CONTINUE
      M=I
      N=J
      R=M/(N+0.)
      PRINT 120, M, N, R
      IF(N.GT.1) GOTO 1
      STOP
  100 FORMAT(1X, I3)
  110 FORMAT('    0/  1=0')
  120 FORMAT(1X, I3, '/', I3, '=', F8.6)
      END

C     Сумма по подмножеству
      INTEGER S, A(1000), B*2(1000)
      READ 100, M, N, (A(I), I=1, N)
      PRINT 100, M, N, (A(I), I=1. N)
      DO 1 I=1, N
    1 B(I)=0
      S=0
    2 I=1
      GOTO 4
```

```
  3 B(I)=0
    S=S-A(I)
    I=I+1
  4 IF(B(I).EQ.1) GOTO 3
    B(I)=1
    S=S+A(I)
    IF(S.NE.M) GOTO 2
    DO 5 I=1,N
    IF(B(I).EQ.1) PRINT 110,I,A(I)
  5 CONTINUE
    STOP
100 FORMAT(1X,I8,2X,I4/10(2X,I5))
110 FORMAT(1X,I4,2X,I5)
    END
```

Нули- в конец OL

```
    DIMENSION A(1000)
    READ 100,N,(A(I),I=1,N)
    PRINT 110,(A(I),I=1,N)
    K=1
    DO 1 I=1,N
    IF(A(I).EQ.0) GOTO 1
    A(K)=A(I)
    K=K+1
  1 CONTINUE
    IF(K.GT.N) GOTO 3
    DO 2 I=K,N
  2 A(I)=0
  3 PRINT 110,(A(I),I=1,N)
    STOP
100 FORMAT(I4/15(1X,F4.1))
110 FORMAT(25(1X,F4.1))
    END
```

Седловая точка OL

```
    REAL MA,MI,A(50,50)
    READ 100,M,N,((A(I,J),J=1,N),I=1,M)
    DO 2 I=1,M
    DO 1 J=1,N
    IF(I.GT.1.AND.A(I,J).LE.MA) GOTO 2
    IF(J.EQ.1.OR.A(I,J).LT.MI) MI=A(I,J)
  1 CONTINUE
    MA=MI
    I0=I
  2 CONTINUE
    DO 4 J=1,N
    DO 3 I=1,M
    IF(A(I,J).GT.MA) GOTO 4
  3 CONTINUE
    PRINT 100,I0,J,A(I0,J)
    GOTO 5
  4 CONTINUE
    PRINT 110
  5 PRINT 100,M,N
    DO 6 I=1,M
```

```
  6 PRINT 120,(A(I,J),J=1,N)
    STOP
100 FORMAT(2(1X,I2)/10(1X,F7.3))
110 FORMAT(' 0')
120 FORMAT(15(1X,F7.3))
    END
```

```
    Вхождение слова в текст                                    OLI
    INTEGER X(1000),Y(1000)
    READ 100,N,(X(I),I=1,N)
    READ 100,K,(Y(I),I=1,K)
    PRINT 100,N,(X(I),I=1,N)
    PRINT 100,K,(Y(I),I=1,K)
    I=-1
  1 I=I+1
    IF(I+K.GT.N) GOTO 3
    DO 2 J=1,K
    IF(X(I+J).NE.Y(J)) GOTO 1
  2 CONTINUE
    I=I+1
    PRINT 110,I
    STOP 1
  3 PRINT 120
    STOP 2
100 FORMAT(1X,I4/10(2X,I5))
110 FORMAT(' ДА ',I4)
120 FORMAT(' НЕТ')
    END
```

ОЛИМПИАДА 83

```
    Бит-реверс                                                 OLI 8
    INTEGER B
    M=512
    B=1
    GOTO 3
  1 B=B-K
    K=K/2
  2 IF(B.GE.K) GOTO 1
    B=B+K
    M=M+1
  3 PRINT 100,M,B
    K=512
    IF(M.LT.1023) GOTO 2
    STOP
100 FORMAT(' M=',I4,'  B(M)=',I4)
    END
```

```
    Треугольник и точка                                        OLI 8
    READ 100,X1,Y1,X2,Y2,X3,Y3,X,Y
    PRINT 100,X1,Y1,X2,Y2,X3,Y3,X,Y
    S=GERON(X1,Y1,X2,Y2,X3,Y3)
    S1=GERON(X1,Y1,X2,Y2,X,Y)
    S2=GERON(X1,Y1,X3,Y3,X,Y)
```

```
      S3=GERON( X2, Y2, X3, Y3, X, Y)
      IF( S1+S2+S3. GT. 1. 000001*S) GOTO 1
      PRINT 110
      STOP 1
    1 PRINT 120
      STOP 2
  100 FORMAT( 8( 2X, F6. 2))
  110 FORMAT(' точка в треугольнике')
  120 FORMAT(' точка вне треугольника')
      END

      FUNCTION GERON( X1, Y1, X2, Y2, X3, Y3)
      DIS( X1, Y1, X2, Y2)=SQRT(( X1-X2)**2+( Y1-Y2)**2)
      A=DIS( X1, Y1, X2, Y2)
      B=DIS( X1, Y1, X3, Y3)
      C=DIS( X2, Y2, X3, Y3)
      P=( A+B+C) /2.
      GERON=SQRT( P*( P-A) *( P-B) *( P-C))
      RETURN
      END


C     Лабиринт
      INTEGER P, P1, P2, Q, Q1, Q2, K*2, A*2( 40, 40)
      READ 100, M, N, I, J, (( A( P, Q), Q=1, N), P=1, M)
      PRINT 100, M, N, I, J
      DO 1 P=1, M
    1 PRINT 110, ( A( P, Q), Q=1, N)
      A( I, J)=2
      K=2
C     GRANICA
    2 IF( I. EQ. 1. OR. I. EQ. M. OR. J. EQ. 1. OR. J. EQ. N)
     #GOTO 7
C     NOVAJA KLETKA
    3 P=I
      Q=J
      P1=P-1
      P2=P+1
      Q1=Q-1
      Q2=Q+1
      DO 4 I=P1, P2
      DO 4 J=Q1, Q2
      IF( I. NE. P. AND. J. NE. Q. OR. A( I, J). NE. 0) GOTO 4
      K=K+1
      A( I, J)=K
      GOTO 2
    4 CONTINUE
      IF( K. GT. 2) GOTO 5
      PRINT 120
      GOTO 9
C     VOZVRASCENIE
    5 A( P, Q)=1
      K=K-1
      DO 6 I=P1, P2
      DO 6 J=Q1, Q2
```

```
      IF(A(I,J).EQ.K) GOTO 3
    6 CONTINUE
C     PECHAT6 DOROWKI
    7 PRINT 130,K,I,J
      IF(K.EQ.2) GOTO 9
      K=K-1
      P=I
      Q=J
      P1=P-1
      P2=P+1
      Q1=Q-1
      Q2=Q+1
      DO 8 I=P1,P2
      DO 8 J=Q1,Q2
      IF(I.LT.1.OR.I.GT.M.OR.J.LT.1.OR.J.GT.N)
     #GOTO 8
      IF(A(I,J).EQ.K) GOTO 7
    8 CONTINUE
    9 STOP
  100 FORMAT(4(1X,I2)/40(I1,1X))
  110 FORMAT(' ',40I1)
  120 FORMAT(' NET VYHODA')
  130 FORMAT(' ',I4,2(2X,I2))
      END

C     Пила
      DIMENSION X(1000)
      READ 100,M,(X(I),I=1,M)
      PRINT 100,M,(X(I),I=1,M)
      K=1
      IF(M.LT.3) GOTO 3
      I=1
      J=1
    1 IF(X(I).LT.X(I+1).AND.X(I+1).GT.X(I+2))
     #GOTO 2
      J=1
      I=I+1
      IF(K+I.LT.M) GOTO 1
      GOTO 3
    2 J=J+2
      IF(J.GT.K) K=J
      I=I+2
      IF(I+1.LT.M) GOTO 1
    3 PRINT 100,K
      STOP
  100 FORMAT(1X,I4/10(1X,F7.3))
      END

C     Сократить дробь
      READ 100,M,N
      M1=M
      N1=N
    1 K=M1-M1/N1*N1
      IF(K.EQ.0) GOTO 2
      M1=N1
```

```
      N1=K
      GOTO 1
    2 M1=M/N1
      N1=N/N1
      PRINT 100,M,N,M1,N1
      STOP
  100 FORMAT(2(1X,I4))
      END
```

ОЛИМПИАДА 84

```
C     Инверсия
      INTEGER TI,P(100),T(100)
      READ 100,N,(T(I),I=1,N)
      PRINT 110,(T(I),I=1,N)
      DO 1 I=1,N
    1 P(I)=0
      DO 3 I=1,N
      TI=T(I)
      K=0
      J=0
    2 J=J+1
      IF(P(J).NE.0) GOTO 2
      K=K+1
      IF(K.LE.TI) GOTO 2
    3 P(J)=I
      PRINT 110,(P(I),I=1,N)
      STOP
  100 FORMAT(I3/20(2X,I2))
  110 FORMAT(30(1X,I3))
      END

C     Дорога
      DIMENSION A(15,15,20),B(15),C(15)
      READ 100,N,M
      PRINT 100,N,M
      N1=N-1
      READ 110,(((A(I,J,K),I=1,M),J=1,M),K=1,N1)
      DO 1 K=1,N1
      DO 1 I=1,M
    1 PRINT 110,(A(I,J,K),J=1,M)
      DO 2 I=1,M
    2 B(I)=0
      DO 6 K=1,N1
      DO 4 J=1,M
      R=B(1)+A(1,J,K)
      DO 3 I=1,M
      X=B(I)+A(I,J,K)
      IF(X.LT.R) R=X
    3 CONTINUE
    4 C(J)=R
      DO 5 J=1,M
    5 B(J)=C(J)
    6 CONTINUE
```

```
      R=B(1)
      DO 7 I=1,M
      IF(B(I).LT.R) R=B(I)
    7 CONTINUE
      PRINT 120,R
      STOP
  100 FORMAT(2(1X,I2))
  110 FORMAT(10(1X,F7.3))
  120 FORMAT(1X,F9.3)
      END

C     Совершенные числа                                          OL
      READ 100,M
      PRINT 100,M
      IF(M.LE.6) GOTO 5
      M1=M-1
      DO 4 I=6,M1
      N=1
      J=2
    1 K=I/J
      IF(K.LE.J) GOTO 3
      IF(J*K.LT.I) GOTO 2
      N=N+J+K
      IF(N.GT.I) GOTO 4
    2 J=J+1
      GOTO 1
    3 IF(J*J.EQ.I) N=N+J
      IF(N.EQ.I) PRINT 100,I
    4 CONTINUE
    5 STOP
  100 FORMAT(1X,I7)
      END

C     Период дроби                                               OL
      READ 100,M,N
      PRINT 100,M,N
      M=M-M/N*N
      DO 1 I=1,N
      M=M*10
    1 M=M-M/N*N
      K=M
    2 K=K*10
      I=K/N
      PRINT 110,I
      K=K-I*N
      IF(K.NE.M) GOTO 2
      STOP
  100 FORMAT(2(1X,I3))
  110 FORMAT(1X,I1)
      END

C     Слияние массивов
      DIMENSION A(1000),B(1000),C(2000)
      READ 100,M,(A(I),I=1,M)
      READ 100,N,(B(J),J=1,N)
```

```
      PRINT 100,M,(A(I),I=1,M)
      PRINT 100,N,(B(J),J=1,N)
      I=1
      J=1
      L=M+N
      DO 3 K=1,L
      IF(I.GT.M) GOTO 1
      IF(J.GT.N) GOTO 2
      IF(A(I).LT.B(J)) GOTO 2
    1 C(K)=B(J)
      J=J+1
      GOTO 3
    2 C(K)=A(I)
      I=I+1
    3 CONTINUE
      PRINT 110,(C(K),K=1,L)
      STOP
  100 FORMAT(1X,I4/10(1X,F7.3))
  110 FORMAT(15(1X,F7.3))
      END

C     Календарь                                                    OL
      INTEGER A,B,C
C     Число месяц год
      DIMENSION M(11)
      DATA M/31,28,31,30,31,30,31,31,30,31,30/
      READ 100,A,B,C
      N=A
      IF(B.EQ.1) GOTO 2
      J=B-1
      DO 1 I=1,J
    1 N=N+M(I)
      IF(B.GT.2.AND.(MOD(C,4).EQ.0.AND.
     #MOD(C,100).NE.0.OR.MOD(C,400).EQ.0))
     #N=N+1
    2 PRINT 100,A,B,C,N
      STOP
  100 FORMAT(2(1X,I2),1X,I4,1X,I3)
      END

C     Квадратики                                                   OL
      INTEGER*2 A(50,50)
      READ 100,M,((A(I,J),J=1,M),I=1,M)
      DO 1 I=1,M
    1 PRINT 110,(A(I,J),J=1,M)
      K=0
      L=M-1
      DO 2 J=1,L
      DO 2 I=1,L
      IF(A(I,J)+A(I+1,J)+A(I,J+1)+A(I+1,J+1).EQ.17)
     #K=K+1
    2 CONTINUE
      PRINT 120,K
      STOP
  100 FORMAT(I2/20(1X,I2))
```

```
110 FORMAT(40(1X,I2))
120 FORMAT(1X,I4)
    END
```

ОЛИМПИАДА 85

```
    Разложение на слагаемые                                    OLI
    INTEGER T,S
    DIMENSION M(1000)
    READ 100,N
    M(1)=N
    I=1
    GOTO 4
  1 T=M(K)-1
    S=T+I-K+1
    DO 3 I=K,N
    IF(S.GT.T) GOTO 2
    M(I)=S
    GOTO 4
  2 M(I)=T
    S=S-T
  3 CONTINUE
  4 PRINT 110,(M(K),K=1,I)
    DO 5 J=1,I
    K=I-J+1
    IF(M(K).GT.1) GOTO 1
  5 CONTINUE
    STOP
100 FORMAT(I3)
110 FORMAT(30(1X,I3))
    END
```

```
    Равные элементы                                            OLI
    INTEGER A(50,50),J(50)
    READ 100,M,N,((A(I,K),K=1,N),I=1,M)
    DO 1 I=1,M
    J(I)=1
  1 PRINT 110,(A(I,K),K=1,N)
    KR=A(1,1)
  2 DO 5 I=1,M
  3 IF(J(I).GT.N) GOTO 6
    L=J(I)
    IJ=A(I,L)
    IF(IJ.LE.KR) GOTO 4
    KR=IJ
    GOTO 2
  4 J(I)=J(I)+1
    IF(IJ.LT.KR) GOTO 3
  5 CONTINUE
    PRINT 120,KR
    GOTO 7
  6 PRINT 130
  7 STOP
100 FORMAT(I2,1X,I2/20(1X,I3))
```

```
110 FORMAT(30(1X,I3))
120 FORMAT(' ',I3)
130 FORMAT(' Нет')
    END

    Несоставляемое число                                        OLI
    INTEGER S,P(100)
    READ 100,N,(P(I),I=1,N)
    PRINT 110,(P(I),I=1,N)
    S=1
    DO 3 I=1,N
    DO 1 J=I,N
    IF(P(J).LE.S) GOTO 2
  1 CONTINUE
    GOTO 4
  2 S=S+P(J)
    P(J)=P(I)
  3 CONTINUE
  4 PRINT 120,S
    STOP
100 FORMAT(I3/20(1X,I3))
110 FORMAT(30(1X,I3))
120 FORMAT(' ',I5)
    END

    Тетраэдры                                                   OLI
    LOGICAL U,D
    D(I,J,K,L)=(M1.EQ.I).AND.
   #((M2.EQ.J).AND.(M3.EQ.K).AND.(M4.EQ.L).OR.
   # (M2.EQ.K).AND.(M3.EQ.L).AND.(M4.EQ.J).OR.
   # (M2.EQ.L).AND.(M3.EQ.J).AND.(M4.EQ.K))
    READ 100,M1,M2,M3,M4,N1,N2 N3,N4
    PRINT 100,M1,M2,M3,M4,N1,N2,N3,N4
    U=D(N1,N2,N3,N4).OR.D(N2,N1,N4,N3)
   #.OR.D(N3,N1,N2,N4).OR.D(N4,N1,N3,N2)
    IF(U) GOTO 1
    PRINT 110
    STOP 1
  1 PRINT 120
    STOP 2
100 FORMAT(8(1X,I2))
110 FORMAT(' Нет')
120 FORMAT(' Да')
    END

    Мода                                                        OLI
    INTEGER AM,A(500)
    READ 100,N,(A(I),I=1,N)
    Массив A упорядочен
    PRINT 100,N,(A(I),I=1,N)
    I=1
    M=0
    DO 2 J=1,N
    IF(A(J).EQ.A(I)) GOTO 2
    IF(J-I.LE.M) GOTO 1
```

```
      AM=A(I)
      M=J-I
    1 I=J
    2 CONTINUE
      IF(N-I+1.GT.M) AM=A(I)
      PRINT 100, AM
      STOP
  100 FORMAT(1X,I3/20(1X,I3))
      END

C     Системы счисления
      DIMENSION M(9)
      READ 100,I,J
      READ 110,(M(K),K=1,9)
      PRINT 110,(M(K),K=1,9)
      N=0
      DO 1 K=1,9
      L=10-K
    1 N=N*I+M(L)
      PRINT 120,N
    2 K=N/J
      L=N-K*J
      PRINT 110,L
      N=K
      IF(N.GT.0) GOTO 2
      STOP
  100 FORMAT(I2,1X,I2)
  110 FORMAT(9(1X,I1))
  120 FORMAT(' ',I10)
      END

C     Побочная диагональ
      INTEGER P,Q
      DIMENSION A(50,50)
      READ 100,K,M,N,((A(I,J),J=1,N),I=1,M)
      DO 1 I=1,M
    1 PRINT 110,(A(I,J),J=1,N)
      P=1-K
      IF(P.LT.1) P=1
      Q=M-K
      IF(Q.GT.N) Q=N
      S=0
      GOTO 3
    2 S=S+A(P+K,P)
      P=P+1
    3 IF(P.LE.Q) GOTO 2
      PRINT 120,S
      STOP
  100 FORMAT(I3,2(1X,I2)/10(1X,F5.2))
  110 FORMAT(20(1X,F5.2))
  120 FORMAT(' ',F7.2)
      END
```

```
      Без тройных повторений                                        OLI
      INTEGER*2 A(50)
      READ 100,N
      I=0
    1 I=I+1
      IF(I.GT.N) GOTO 6
      A(I)=-1
    2 IF(A(I).GT.0) GOTO 5
      A(I)=A(I)+1
      IF(I.LT.3) GOTO 1
      L=I/3
      DO 4 J=1,L
      M=I-J+1
      DO 3 K=M,I
      IF(A(K).NE.A(K-J).OR.A(K).NE.A(K-2*J))
     #GOTO 4
    3 CONTINUE
      GOTO 2
    4 CONTINUE
      GOTO 1
    5 I=I-1
      IF(I.GT.1) GOTO 2
      PRINT 110
      GOTO 7
    6 PRINT 120,(A(I),I=1,N)
    7 STOP
  100 FORMAT(I2)
  110 FORMAT(' НЕТ')
  120 FORMAT(' ',50I1)
      END

      Покер                                                         OLI
      INTEGER P,S
      DIMENSION A(5)
      READ 100,(A(I),I=1,5)
      PRINT 100,(A(I),I=1,5)
      S=0
      DO 1 I=1,4
      I1=I+1
      DO 1 J=I1,5
      IF(A(I).EQ.A(J)) S=S+1
    1 CONTINUE
      P=7-S
      IF(S.EQ.6) P=2
      IF(S.EQ.10) P=1
      PRINT 110,P
      STOP
  100 FORMAT(5(1X,F5.2))
  110 FORMAT(' ',I1)
      END

      Барабан                                                       OLI
      INTEGER P,Q
```

```
      DIMENSION A(50)
      READ 100,N,(A(I),I=1,N)
      PRINT 110,(A(I),I=1,N)
      K=1
      DO 2 P=2,N
      DO 1 I=1,N
      L=K+I-1
      IF(L.GT.N) L=L-N
      Q=P+I-1
      IF(Q.GT.N) Q=Q-N
      IF(A(Q).LT.A(L)) K=P
      IF(A(Q).NE.A(L)) GOTO 2
    1 CONTINUE
    2 CONTINUE
      PRINT 100,K
      STOP
  100 FORMAT(1X,I2/12(1X,F5.2))
  110 FORMAT(20(1X,F5.2))
      END
```

Дважды монотонный

```
      DIMENSION A(50,50)
      READ 100,M,N,((A(I,J),J=1,N),I=1,M)
      READ 110,X
      DO 1 I=1,M
    1 PRINT 120,(A(I,J),J=1,N)
      PRINT 110,X
      I=1
      J=N
    2 IF(A(I,J)-X) 3,6,4
    3 I=I+1
      IF(I-M) 2,2,5
    4 J=J-1
      IF(J.GE.1) GOTO 2
    5 PRINT 130
      GOTO 7
    6 PRINT 100,I,J
    7 STOP
  100 FORMAT(2(1X,I2)/10(1X,F7.3))
  110 FORMAT(1X,F7.3)
  120 FORMAT(15(1X,F7.3))
  130 FORMAT(' НЕТ')
      END
```

Центральное селение

```
      DIMENSION A(50,50)
      READ 100,K,((A(I,J),J=1,K),I=1,K)
      DO 1 I=1,K
    1 PRINT 110,(A(I,J),J=1,K)
      DO 3 I=1,K
      S=0
      DO 2 J=1,K
      IF(J.NE.I.AND.S.LT.A(I,J)) S=A(I,J)
    2 CONTINUE
      S=S+A(I,I)
```

```
      IF(I.GT.1.AND.S.GE.T) GOTO 3
      T=S
      I1=I
    3 CONTINUE
      PRINT 120,I1,T
      STOP
  100 FORMAT(1X,I2/10(1X,F7.3))
  110 FORMAT(15(1X,F7.3))
  120 FORMAT(' ',I2,2X,F8.3)
      END
```

ОЛИМПИАДА 87

```
C     Рюкзак                                                    0
      REAL A(100),B(100)
      INTEGER*2 P(100)
      READ 100,N
C     Число N>=2
      READ 110,(A(I),I=1,N)
      READ 110,(B(I),I=1,N)
      PRINT 110,(A(I),I=1,N)
      PRINT 110,(B(I),I=1,N)
      S=0
      Z=0
      ZM=0
      I=0
      T=30
      N1=N-1
    1 J=I+1
      DO 3 I=J,N
      IF(S+A(I).GE.T) GOTO 2
      S=S+A(I)
      Z=Z+B(I)
      P(I)=0
      GOTO 3
    2 P(I)=1
    3 CONTINUE
      IF(ZM.LT.Z) ZM=Z
      DO 5 J=1,N1
      I=N-J
      IF(P(I+1).EQ.0) GOTO 4
      IF(P(I).EQ.1) GOTO 5
      GOTO 6
    4 S=S-A(I+1)
      Z=Z-B(I+1)
    5 CONTINUE
      PRINT 120,ZM
      GOTO 7
    6 S=S-A(I)
      Z=Z-B(I)
      P(I)=1
      GOTO 1
    7 STOP
  100 FORMAT(I3)
```

```
  110 FORMAT(10(1X,F6.2))
  120 FORMAT(1X,F8.2)
      END

C     Полукратные
      INTEGER A2,A3,A(1000)
      READ 100,N
      PRINT 110
      IF(N.EQ.1) GOTO 3
      A(1)=1
      K2=1
      K3=1
      DO 2 I=2,N
      A2=2*A(K2)+1
      A3=3*A(K3)+1
      IF(A2.GT.A3) GOTO 1
      A(I)=A2
      K2=K2+1
      IF(A2.EQ.A3) K3=K3+1
      GOTO 2
    1 A(I)=A3
      K3=K3+1
    2 PRINT 120,A(I)
    3 STOP
  100 FORMAT(I3)
  110 FORMAT(11H           1)
  120 FORMAT(' ',I10)
      END

C     Сумма кубов
      READ 100,N
      M=0
      I=1
      J=1
    1 IF(J**3+1.GE.N) GOTO 2
      J=J+1
      GOTO 1
    2 K=I*I*I+J*J*J
      IF(K.EQ.N) M=M+1
      IF(K.LE.N) I=I+1
      IF(K.GE.N) J=J-1
      IF(I.LE.J) GOTO 2
      PRINT 100,N,M
      STOP
  100 FORMAT(1X,I4,1X,I2)
      END

C     Перевертыши
      INTEGER LN/1/,PN/2/,MX/1/,Z/1/,P
      DIMENSION A(1000)
      READ 100,N,(A(I),I=1,N)
      PRINT 110,(A(I),I=1,N)
      GOTO 4
    1 L=LN
      P=PN
```

```
  2 IF(A(L).NE.A(P)) GOTO 3
    L=L-1
    P=P+1
    IF(1.LE.L.AND.P.LE.N) GOTO 2
  3 M=P-L-1
    IF(MX.LT.M) MX=M
    IF(Z.GT.0) PN=PN+1
    IF(Z.LT.0) LN=LN+1
    Z=-Z
  4 IF(2*(N-PN+1).GT.MX) GOTO 1
    PRINT 100,MX
    STOP
100 FORMAT(1X,I4/15(1X,F4.1))
110 FORMAT(25(1X,F4.1))
    END
```

```
    Индексы порядка                                          OL
    DIMENSION A(100),I(100)
    READ 100,N,(A(J),J=1,N)
    PRINT 100,N,(A(J),J=1,N)
    DO 1 J=1,N
  1 I(J)=J
    DO 3 K=1,N
    M=K
    DO 2 J=K,N
    IF(A(M).GT.A(J)) M=J
  2 CONTINUE
    R=A(K)
    A(K)=A(M)
    A(M)=R
    L=I(K)
    I(K)=I(M)
    I(M)=L
  3 CONTINUE
    PRINT 110,(I(J),J=1,N)
    STOP
100 FORMAT(1X,I3/15(1X,F4.1))
110 FORMAT(30(1X,I3))
    END
```

ОЛИМПИАДА 88

```
    Правый больший                                           OL
    DIMENSION A(100),B(100)
    READ 100,N,(A(I),I=1,N)
    PRINT 110,(A(I),I=1,N)
    K=N
    B(K)=A(N)
    A(N)=0
    IF(N.EQ.1) GOTO 4
    DO 3 L=2,N
    I=N-L+1
    DO 1 J=K,N
    IF(A(I).LT.B(J)) GOTO 2
  1 CONTINUE
```

```
      K=N
      B(K)=A(I)
      A(I)=0
      GOTO 3
    2 K=J-1
      B(K)=A(I)
      A(I)=B(J)
    3 CONTINUE
    4 PRINT 110,(A(I),I=1,N)
      STOP
  100 FORMAT(I3/10(1X,F5.2))
  110 FORMAT(20(1X,F5.2))
      END

      Многочлен
      DIMENSION A(50),X(50)
      READ 100,N,(X(I),I=1,N)
      PRINT 100,N,(X(I),I=1,N)
      A(1)=1
      DO 3 M=1,N
      IF(M.EQ.1) GOTO 2
      DO 1 J=2,M
      I=M-J+2
    1 A(I)=A(I-1)-A(I)*X(M)
    2 A(M+1)=1
    3 A(1)=-A(1)*X(M)
      PRINT 110,(A(I),I=1,N)
      STOP
  100 FORMAT(1X,I2/10(1X,F5.2))
  110 FORMAT(8(2X,E13.6))
      END

      Простые делители
      READ 100,N
      PRINT 100,N
      I=2
      J=0
      GOTO 4
    1 IF(N-N/I*I.EQ.0) GOTO 2
      I=I+1
      GOTO 1
    2 IF(I.EQ.J) GOTO 3
      PRINT 100,I
      J=I
    3 N=N/I
    4 IF(N.GT.1) GOTO 1
      STOP
  100 FORMAT(1X,I4)
      END
```

```
      Оптовая покупка                                          OL
      READ 100,N
      N1=N/144
      M=N-N1*144
      N2=M/12
      N3=M-N2*12
      IF(N3*1.05.LE.10.25) GOTO 1
      N2=N2+1
      N3=0
    1 IF(N2*10.25+N3*1.05.LE.114) GOTO 2
      N1=N1+1
      N2=0
      N3=0
    2 PRINT 100,N,N1,N2,N3
      STOP
  100 FORMAT(1X,I4,3(2X,I2))
      END

      Обнуление                                                OL
      INTEGER Z,C(50)
      DIMENSION A(50,50)
      READ 100,M,N,((A(I,J),J=1,N),I=1,M)
      DO 1 I=1,M
    1 PRINT 110,(A(I,J),J=1,N)
      DO 2 J=1,N
    2 C(J)=0
      DO 5 I=1,M
      Z=0
      DO 3 J=1,N
      IF(A(I,J).NE.0) GOTO 3
      Z=1
      C(J)=1
    3 CONTINUE
      IF(Z.EQ.0) GOTO 5
      DO 4 J=1,N
    4 A(I,J)=0
    5 CONTINUE
      DO 7 J=1,N
      IF(C(J).EQ.0) GOTO 7
      DO 6 I=1,M
    6 A(I,J)=0
    7 CONTINUE
      DO 8 I=1,M
    8 PRINT 110,(A(I,J),J=1,N)
      STOP
  100 FORMAT(I2,1X,I2/10(1X,F5.2))
  110 FORMAT(20(1X,F5.2))
      END
```

Глава 2

ЛЕКЦИИ ПО ПРОГРАММИРОВАНИЮ

§ 2.1. Программирование перебора вариантов

Программирование перебора вариантов является сердцевиной программ искусственного интеллекта независимо от того, к чему он прилагается — программированию игр, выбору решений, распознаванию образов и т. п. Между тем даже программист, опытный в вычислительных задачах, знающий операционные системы и языки программирования, зачастую оказывается беспомощен в программировании перебора, поскольку схема перебора не укладывается в схемы циклов, имеющихся в языках программирования.

Сейчас мы составим блок-программу перебора. Сделаем это на одной определенной задаче, но сделаем так, что для других задач перебора потребуется менять лишь конкретное содержание блоков, а сама блок-программа сохранится.

Вот наша задача. *Какими способами можно расставить на шахматной доске* 8 *ферзей так, чтобы они не угрожали друг другу*? *).

Прикинем, как искать эти расстановки. Поставим первого ферзя на какую-нибудь клетку. Затем поставим второго ферзя на первую клетку и проверим, что ему не угрожает первый. Если угрожает, то передвинем второго ферзя и снова проверим и т. д. Когда второй ферзь окажется на допустимой клетке, возьмем третьего ферзя и будем двигать его, пока он не окажется на допустимой клетке и т. д.

Уже это побуждает ввести два понятия, фундаментальных для теории перебора: *номер хода* и *номер варианта*.

*) То есть на квадратной клетчатой доске размером 8×8 так, чтобы никакие два ферзя не стояли на одной вертикали, горизонтали или диагонали.

Не принято обозначать буквами i, j. В расстановке ферзей номером хода будет порядковый номер ферзя (которого мы пытаемся поставить), а номером варианта — порядковый номер попытки установить этого ферзя после того, как положение предыдущих фиксировано. Понятие «номер варианта» не следует понимать буквально. Им, например, может служить само поле, на которое предполагается поставить ферзя. Важно лишь, чтобы рассмотрение вариантов было упорядочено.

Сказанное уже стоит записать блок-схемой (рис. 2.1).

Продолжим рассмотрение нашей задачи. Размещая одного ферзя за другим, мы, в лучшем случае, разместим последнего (восьмого) ферзя и сможем вывести (например, отпечатать) получившуюся расстановку ферзей. Однако в

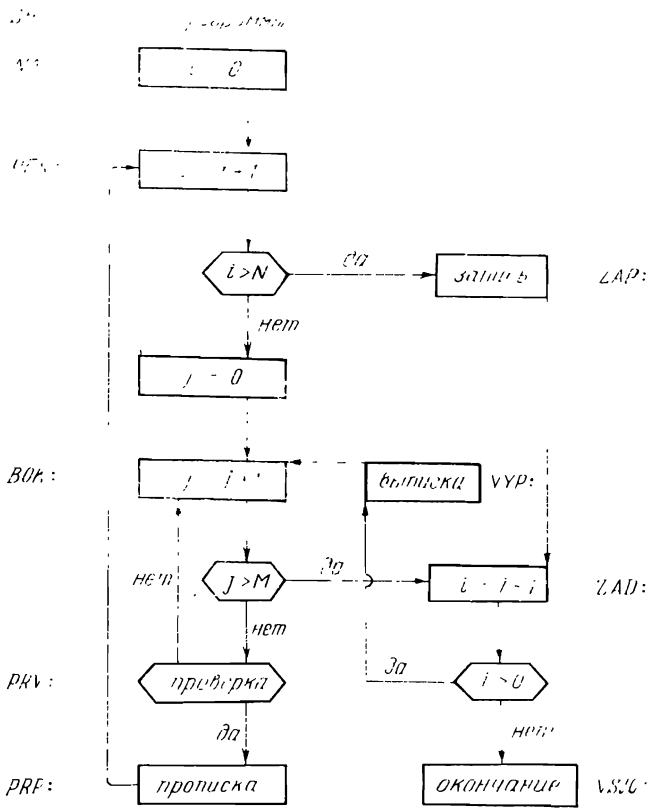


Рис. 2.1

ходов назад, исчерпав все варианты расположения очередного (скажем, i-го) ферзя, то есть не сумев его поставить ни на какое, мы должны будем вернуться на ход назад и перейти к следующему варианту расстановки предыдущего ферзя, $(i-1)$-го. Ясно, что для этого нам придется вспомнить последний рассмотренный вариант установки $(i-1)$-го ферзя. И затем, увеличив номер варианта, продолжить просмотр вариантов установки этого ферзя.

В этих движениях вперед и назад по номерам ходов и состоит особенность схемы перебора, которая не укладывается в готовую схему цикла (однократного по i или двукратного по i и j) обычных языков программирования.

Подытожим грубо наше рассмотрение. Мы движемся *вперед*, увеличивая номер хода. Для каждого (очередного) хода движемся *вбок*, подбирая допустимый вариант, и переходим вперед к следующему ходу, если вариант подобрался. Если вариант подобрать нельзя, то мы возвращаемся на ход *назад* и продолжаем движение вбок, начиная с последнего варианта, рассмотренного на этом ходе. Установив последнего ферзя, мы записываем полученное решение. Все это указывается в блок-схеме и блок-программе (рис. 2.2). (Заметим только, что блок-схема дает лишь представление о последовательности работы разных разделов программы, в то время как блок-программа является точной программой.)

```
SH:   шапка программы
XACH: i:=0; начальная подготовка;
PER:  i:=i+1; if i>n then goto ZAP;
      j:=0; подготовка вариантов;
BOK:  j:=j+1; if j>n then goto ZAD;
      образование очередного варианта;
PRV:  if (вариант недопустим)
      then goto BOK;
PRP:  прописка варианта;
      goto PER;
ZAP:  запись очередного решения;
ZAD:  i:=i-1; if i=0 then goto VSJO;
VYP:  выписка последнего варианта и хода;
      восстановление текущего варианта
      предыдущего хода;
      goto BOK;
VSJO. окончание программы
```

Рис. 2.2. Блок-программа перебора

Наша блок-программа является самой общей, пригодной для любой задачи перебора, а блоки, к которым она обращается, зависят от конкретной задачи. В зависимости от сложности возложенных на них функций они могут реализоваться несколькими командами, помещенными в саму блок-программу (чего, однако, лучше не делать), или представлять подпрограммы, которым передается управление.

1. Чтобы легче разобраться в блок-программе, начнем с простейшего способа решения задачи о расстановке ферзей и посмотрим, какой получатся ее блоки и вся программа. Именно: обозначим клетки доски (как это принято в шахматах) парами чисел (x, y), где x — номер вертикали и y — номер горизонтали. Упорядочим поля по вертикалям, а внутри вертикали — по горизонталям. И будем пытаться ставить очередного ферзя на поля доски, рассматривая их в таком порядке: (1,1), (1,2), ... , (2,1), (2,2), ...

Договоримся об обозначениях. Возьмем доску размером $A*B$, N ферзей и положим $A=B=N=8$. Это делается для ясности программы. Для записи положения ферзей заведем два массива X и Y по N элементов каждый. Общее число найденных вариантов обозначим буквой k. Договоримся еще об одном обстоятельстве. Мы бы не хотели считать разными позиции, в которых ферзей поменяли местами. Поэтому второго ферзя начнем пытаться ставить не с поля (1,1), а с поля, следующего за полем $(X|1|, Y|1|)$, на котором уже находится первый ферзь. Стоит заметить, что порядковым номером поля (x, y) будет $j=B*(x-1)+y$ и что последнее поле будет иметь номер $j=A*B$.

Теперь рассмотрим получившуюся блок-программу и напишем ее блоки.

SH: Шапка программы сильно зависит от языка программирования. В Бейсике достаточно задать A, B, N и массивы X, Y:

```
A=8 : N=A : B=8 : m=A*B 'SH
dim X(A), Y(B)
```

Далее пока пойдет строка

```
i=0 : k=0     'NACH
```

Вся «начальная подготовка» свелась к одному оператору, обнуляющему число k найденных решений.

Раздел «вперед» начинается просто:

```
i=i+1 : if i>N goto ZAP     'PER
if i=1 then j=0
```

Далее в блок-программе идет «подготовка вариантов». Здесь она должна быть такой, чтобы в разделе *BOK* образовался первый вариант размещения i-го ферзя. Значит, в «подготовке вариантов» надо в $(X|i|, Y|i|)$ записать поле, предшествующее первому варианту. Если $i=1$, то для этого удобно взять фиктивное поле (1,0), за которым действительно следует (1,1), а если $i>1$, то можно взять поле, где стоит $(i-1)$-й ферзь. Таким образом, получаем часть программы:

```
if i=1 then X(1)=1 : Y(1)=0
else X(i)=X(i-1) : Y(i)=Y(i-1)
```

Далее идет раздел *BOK*:

```
j=j+1 : if j>m goto ZAD    'BOK
```

Образование «очередного варианта» можно сделать так:

```
if Y(i)<B then Y(i)=Y(i)+1
else X(i)=X(i)+1 : Y(i)=1
```

Проверку допустимости выбранного положения ферзя лучше оформить логической подпрограммой *NEDOP*. Тогда этот раздел в программе будет иметь вид

```
GOSUB NEDOP
if NEDOP goto BOK
```

Прописка варианта у нас произошла автоматически, поскольку мы его сразу занесли в $X|i|$, $Y|i|$, но не для всех алгоритмов это удобно. В данном же случае остается только перейти к метке *PER*:

```
goto PER    'PRP
```

Запись очередного решения состоит в его печати и увеличении числа k найденных решений. Например, так:

```
k=k+1 : print k;    'ZAP
for p=1 to N :  print X(p); Y(p); "␣"; : next
```

Далее идет движение назад:

```
i=i-1 :  if i=0 goto VSJO    'ZAD
j=B*(X(i)-1)+Y(i) : goto BOK
print "вариантов"; k    'VSJO
```

При движении назад нам понадобилось восстановить в j номер последнего варианта предыдущего хода. Вообще же возврат назад — наиболее сложный раздел программы перебора. Двигаясь вперед, мы перерабатываем информа-

нию о позиции и бывает трудно однозначно восст вить предыдущую. Вопрос этот автоматически реша при рекурсивном описании программы. Но тогда движении вперед приходится переписывать столько ин мации, что скорость работы становится недопустимо кой для программирования игр.

Остается написать подпрограмму проверки, би ли поле i-го ферзя предыдущими ферзями.

```
VEDOP=0      'VEDOP
for p=1 to i-1
NEDOP=NEDOP
or X(p)=X(i) or Y(p)=Y(i)
or Y(p)-X(p)=Y(i)-X(i)
or Y(p)+X(p)=Y(i)+X(i)
next : RETURN
```

Сначала проверяется, не стоят ли ферзи p и i на о вертикали, потом на одной горизонтали, потом восходящей диагонали (слева снизу направо ввер наконец, на нисходящей диагонали (слева сверху право вниз).

Остается пронумеровать строки, заменить в опер рах перехода метки номерами строк и вся программа мет вид

```
 10 'PERZI-1
 20 A=8:N=A:B=8:m=A*B                    'SH
 30 dim X(A),Y(B)
 40 i=0:k=0                              'NACH
 50 i=i+1:if i<N goto 120                'PER
 60 if i=1 then j=-1:X(1)=1:Y(1)=0
    else X(i)=X(i-1):Y(i)=Y(i-1)
 70 j=j+1:if j<m goto 140                'BOK
 80 if Y(i)<B then Y(i)=Y(i)+1
    else X(i)=X(i)+1 Y(i)=1
 90 GOSUB 200                            'PRV
100 if NEDOP goto 70
110 goto 50                              'PRP
120 k=k+1:print: printk;                 'ZAP
130 for p=1 to N:print X(p);Y(p);:next
140 i=i-1:if i=0 goto 160                'ZAD
150 j=B*(X(i)-1)+Y(i):goto 70
160 print:print"Вариантов";k              'VSJO
```

```
200 NEDOP=0                                    'NED
210 if i=1 goto 250
220 for p=1 to i-1
230   NEDOP=NEDOP
      or X(p)=X(i) or Y(p)=Y(i)
      or Y(p)-X(p)=Y(i)-X(i)
      or Y(p)+X(p)=Y(i)+X(i)
240 next
250 RETURN
```

Заметьте, что при написании сложной программы на Бейсике метки строк удобно указывать справа в виде комментариев. В самой программе переходы на эти метки сперва пишутся по их названиям, и лишь потом заменяются номерами строк. Так, например, строка 50 у нас сначала имела вид

$$50\ i=i+1:\text{if } i>N \text{ goto } ZAP \quad 'PER$$

и только после того, как были написаны все строки и уже появилась строка

$$120\ k=k+1:print\ k. \quad 'ZAP$$

мы придали строке 50 окончательный вид

$$50\ i=i+1:\text{if } i>N \text{ goto } 120 \quad 'PER$$

Программа *FERZI*-1 верна. Она помогла нам разобрать блок-программу перебора, но на этом ее полезность кончается. В среднем нового ферзя мы пытаемся поставить на 32 поля. А это грозит привести к $8^{32}\approx 10^{30}$ вариантам (!). Бросается в глаза, что если очередной ферзь не установлен на очередной вертикали, то все ферзи заведомо не разместятся, и мы будем продолжать бессмысленную работу.

2. Чтобы ускорить работу программы, будем ставить i-го ферзя на i-ю вертикаль. Тогда число вариантов не превысит $8^8=2^{24}\approx 10^6$ (на самом деле менее 10^4), что уже приемлемо.

Блок-программа, разумеется, сохранится. Выпишем участки, которые надо переделать. Мы откажемся от массива X и всюду заменим $X[i]$ на i. Массив Y, как это принято в шахматах, назовем именем $POZA$, а для краткости записи текущие действия будем выполнять не с $POZA[i]$ и с переменной j (которая теперь равна $POZA[i]$). Теперь

в разделе «прописка» в *POZA*[*i*] будет занесено *j*. Раздел «запись» упростится, а в разделе *ZAD* последний опробованный вариант из *POZA*[*i*] будет занесен в *j*. Программа заметно упростится и примет вид *FERZ1*-2.

```
10 'FERZ1 2
20 A=8:N=A:B=2                                  'SH
30 dim POZA(A)
40 i=0:k=0                                      'NACH
50 i=i+1:j=0:if i>N goto 100                    'PER
60 j=j+1:if j>B goto 120                        'BOK
70 GOSUB 200                                    'PRV
80 if NEDOP goto 60
90 POZA(i)=j:goto 50                            'PRP
100 k=k+1:print, print k;                       'ZAP
110 for p=1 to N:print POZA(p);:next
120 i=i-1:if i=0 goto 140                       'ZAD
130 j=POZA(i):goto 60
140 print:print"Вариантов";k                     'VSJO
150 end
200 NEDOP=0                                     'NED
210 if i=1 goto 250
220 for p=1 to i-1
230   NEDOP=NEDOP
      or POZA(p) = j
      or POZA(p)-p  = j-i
      or POZA(p)+p  = j+i
240 next
250 RETURN
```

3. Стоит подумать об ускорении подпрограммы *NEDOP*. Мы сравниваем положение *i*-го ферзя со всеми предыдущими. Это грозит затратой *i* действий. Не лучше ли будет при установке очередного ферзя отметить поля доски, на которые он бьет, и тогда в одно действие посмотреть, осталась ли свободной клетка, на которую мы собирались поставить нового ферзя. Оказывается, нет. И дело здесь даже не в том, что на отметку полей всех ферзей уйдет *A***N* действий. Важнее, что при ходе назад или вбок нам будет трудно снять метки с освободившихся полей — ведь поле могло биться не только тем ферзем, которого мы снимаем, но еще и другими.

Выход из затруднения состоит в следующем. Заведем три массива *GR*, *VOS*, *ZAK* для списков горизонталей, восходящих и заходящих диагоналей. И при установке очередного ферзя мы сможем в три действия отметить занятые горизонталь и две диагонали. А при попытке установить нового ферзя сможем в три действия проверить, свободно ли поле. Важно заметить, что при движении назад мы также легко можем исправить эти массивы — ведь на данной горизонтали или диагонали может находиться лишь один ферзь!

Новый вариант программы (см. программу *FERZI-3*) потребует введения трех логических массивов, которые нужно будет очистить в разделе *NACH*. В подпрограмме *VEDOP* исчезнет цикл, и она упростится настолько, что ее оператор проще будет поместить в раздел *PRV* (проверка). Но усложнится *PRP* (прописка). Здесь появится занесение отметок в массивы *GR*, *VOS* и *ZAK* о том, что заняты горизонталь и две диагонали той клетки, куда установлен новый ферзь. И последнее — в разделе *ZAD* (возврат назад) появится снятие отметок о занятости:

```
 10 'FERZI-3
 20 A 9:N=A:B=8                                   'SH
 30 dim POZA(A),GR(B+A),
       VOS(B+A),ZAK(B+A)
 40 i=0:k=0                                       'NACH
 50 for p=1 to B:GR(p)=0:next
 60 for p=1 to B+A:VOS(p)=0:next
 70 for p=1 to B+A:ZAK(p)=0:next
 80 i=i+1:j=0:if i>N goto 130                     'PER
 90 j=j+1:if j>B goto 150                         'BOK
100 if GR(j) or VOS(j-i+A) or ZAK(j+i)
    goto 90                                       'PRV
110 GR(j)=1:VOS(j-i+A)=1:ZAK(j+i)=1               'PRP
120 POZA(i)=j:goto 80
130 k=k+1:print:print k;                          'ZAP
140 for p=1 to N:print POZA(p);:next p
150 i=i-1:if i=0 goto 190                         'ZAD
160 j=POZA(i)
170 GR(j)=0:VOS(j-i+A)=0:ZAK(j+I)=0
180 goto 90
190 print:print"Вариантов";k                      'VSJO
200 end
```

4. Полученная программа настолько удовлетворительна, что Э. Дейкстра [6] на ней и остановился. Все же один ее недостаток бросается в глаза: устанавливая последнего ферзя, мы пробуем все 8 полей, тогда как свободным могло остаться только одно. Чтобы избежать этого и тем значительно ускорить вычисления, мы заведем список свободных горизонталей и будем пытаться ставить очередного ферзя только на поля из этого списка. Подчеркнем, что образование списка вариантов i-го хода является приемом мощным и достаточно общим для применения в различных конкретных задачах. Выигрыш от него покрывает издержки на расход места для этого списка и на действия, нужные для его поддержания. Заметим все же, что список свободных горизонталей не является списком полей, куда можно поставить i-го ферзя, так как эти поля могут биться по диагоналям, но он сокращает число проб.

Что касается места, то оно у нас есть. Это (на i-м ходе) — ячейки массива $POZA$, начиная с i-й и до конца. Нужно будет только взять массив $POZA$ длиною B и вначале заполнить его списком свободных (т. е. всех) горизонталей:

$$POZA|i|=i \text{ для } \quad i=1.2....,B.$$

Но нужно поддерживать этот список. При движении вперед это выполняется легко. Найдя на i-м ходе в массиве

$$POZA|1|,...,POZA|i|,\ POZA|i+1|,...,POZA|B|$$

на некотором месте $j \geqslant i$ подходящую для i-го ферзя горизонталь $POZA|j|$, мы меняем местами значения элементов $POZA|i|$ и $POZA\,|j|$ и тем самым выполняем и прописку ферзя i, и образование нового списка свободных горизонталей (начинающегося теперь с элемента $i+1$ массива $POZA$).

Трудность возникает при движении назад, когда нам надо будет вернуть значения $POZA|i|$ и $POZA|j|$ на свои места, чтобы восстановить порядок рассматриваемых вариантов. Предвидя это, заведем массив $Z|1 : N|$ и будем при движении вперед вписывать в элемент $Z|i|$ номер выбранного варианта j хода i. Благодаря этому при движении назад на ходе i мы прочтем в ячейке $Z|i|$ число j, и сможем вернуть на свои места значения $POZA|i|$ и $POZA|j|$.

5. Все это приводит к программе *FERZI-4*. Разберем ее на основе блок-программы перебора, не ссылаясь на предыдущие программы:

```
10 'FEKZI 4
20 A=8:N=A:B=8                                    'GH
30 dim POZA(B+A),N
       VOS(B+A),ZAK(B+A)
40 i=0:k=0                                        'NACH
50 for p=1 to B:POZA(p)=p:next
60 for p=1 to B+A:VOS(p)=0:next
70 for p=1 to B+A:ZAK(p)=0:next
80 i=i+1:j=i-1:if i>N goto 150                    'PER
90 j=j+1:if j>B goto 170                          'BOK
100 p=POZA(j)
110 if VOS(p-i+A) or ZAK(p+i) goto 90             'PRV
120 VOS(p-i+A)=1:ZAK(p+i)=1                       'PRP
130 POZA(j)=POZA(i):POZA(i)=p
140 Z(i)=j:goto 80
150 k=k+1:print:print k,                          'VAR
160 for q=1 to N:print POZA(q);:next
170 i=i-1:if i=0 goto 200                         'ZAD
180 j=Z(i):p=POZA(j)                              'VYT
190 VOS(p-i+A)=0:ZAK(p+i)=0:POZA(j)=POZA(i):p
       =p+1
200 goto 90
210 goto 90
220 print:print"Вариантов";k                       'VSJO
230 end
```

SH: В нашем алгоритме обязательно $N=A$ и надо положить $A=N=B=8$. Следует описать два целых массива: массив $POZA[1:B]$ для записи номера горизонтали ферзя, стоящего на данной вертикали, и для записи списка свободных горизонталей и массив $Z[1:N]$ для записи номера j последнего варианта, рассмотренного на ходе i. Надо описать и два логических массива:

$VOS[1:B+A]$ — для списка диагоналей $y-x=\text{const}$,
$ZAK[1:B+A]$ — для диагоналей $y+x=\text{const}$.

В них будет отмечаться, свободна ли данная диагональ.

NACH: Первоначальная подготовка. Здесь обнуляются i и k (переменная k служит для подсчета числа расстановок ферзей). Массив $POZA$ заполняется списком свободных (то есть всех) горизонталей, массивы VOS и ZAK очищаются числом 0.

PER: Номер хода увеличивается; если он превосходит N то ферзей, то расстановка найдена. Иначе поиск рас-

становки следует продолжить и номер j варианта (предварительно) делается равным $i+1$ с таким расчетом, чтобы в разделе *BOK* он стал равен номеру i, с которого в массиве *POZA* начинается список свободных горизонталей для хода i. Напомним, что номер варианта не должен быть обязательно его порядковым номером — лишь бы варианты были упорядочены по возрастанию своими номерами (у нас здесь номера вариантов хода i суть $j=i, i+1,...$).

BOK: Здесь увеличивается номер варианта j хода i, и если варианты исчерпаны, то следует идти назад. В переменную p, для краткости, заносится номер очередной свободной горизонтали.

PRV: Проверяется, бьется ли клетка (i, p) предыдущими ферзями (с номерами меньше i). Если бьется, то надо идти вбок. Если не бьется, то надо идти к прописке ферзя i на горизонтали p.

PRP: Прописка. Отмечается занятость двух диагоналей в массивах *VOS* и *ZAK*. В элементе *POZA*$|i|$ записывается номер p горизонтали, занятой ферзем i, а свободная горизонталь, записанная раньше в этом элементе, переносится в элемент *POZA*$|j|$, откуда был взят номер p. Это должно работать верно (то есть не делать ничего) и в случае $i=j$. Наконец, в $Z|i|$ запоминается номер варианта j.

ZAP: Печатается найденный вариант расстановки и его порядковый номер k.

6. Мы рассмотрели четыре варианта программы расстановки ферзей. И на этом можно остановиться. Но игровую программу почти всегда удается усовершенствовать. Можно, например, заметить, что программа *FERZI*-4 рассматривает свободные горизонтали в порядке, зависящем от уже выбранных. Читатели, которые захотят рассматривать горизонтали в фиксированном порядке, могут познакомиться с более трудной программой *FERZI*-5 :

```
10 'FERZI-5
20 A=8:N=A:B=8                          'SH
30 dim POZA(N),Z(B),
       VOS(B+A),ZAK(B+A)
40 i=0:k=0                              'NACH
50 for p=0 to B:Z(p)=p+1:next
60 for p=1 to B+A:VOS(p)=0:next
```

```
 70 for p 1 to i+A.ZAK(p) 0.next
 80 i=i+1:p 0:if i<N goto 140                    'PER
 90 j p.p Z j .:if p i goto 10                   'BOK
100 if VOS(p-i+A) or ZAK(p+i) goto 90            'PRV
110 VOS(p-i+A)=1:ZAK(p+i)=1                      'PRP
120 Z(j)=Z(p):Z(p)=j:POZA(i)=p
130 goto 80
140 k=k+1:print:printk;                          'ZAP
150 for q=1 to N:print POZA(q);:next
160 i=i-1:if i=0 goto 200                        'ZAP
170 p=POZA(i):j=Z(p):Z(p)=Z(j):Z(j)=p            'VYP
180 VOS(p-i+A)=0:ZAK(p+i)=0
190 goto 90
200 print.print"Вариантов";k                     'VSJO
210 end
```

В программе *FERZI-5* также четыре массива: *POZ* 1, 1 *OS*, *ZAK* и *Z*. Но в $Z[j]$ помещается номер горизонтали, к горую следует рассматривать после рассмотрения го- ризонтали j (если горизонталь j осталась свободна) [ли же горизонталь j заня ли ферзем, то в $Z[j]$ заносится номер горизонтали, которая была рассмотрена последней перед горизонталью j и осталась свободной.

Вначале массив $Z[0:B]$ заполняется номерами горизонталей так, чтобы первой рассматривалась горизонталь $Z[0]$, второй — $Z[Z[0]]$,..., а следом за последней рассматриваемой шла фиктивная горизонталь $B+1$. Например, так:

$$Z[j]=j+1 \text{ для } 0\leqslant j\leqslant B.$$

Попытка установить очередного ферзя i теперь всегда начинается с горизонтали $Z[0]$. Если она не удалась, то последовательно рассматриваются горизонтали $Z[Z[0]]$, $Z[Z[Z[0]]]$ и т. д. Если, наконец, ферзя i удалось установить на горизонталь $p=Z[j]$, то делается следующее:

$$Z[j]=Z[p]$$

ибо в дальнейшем после горизонтали j должна будет рассматриваться горизонталь $Z[p]$, и далее

$$Z[p]=i:POZA[i]=p$$

Остальное выполняется так же, как и в программе *FERZI-4*.

ZAD: Движение назад. Номер хода уменьшается на 1 и проверяется, не окончена ли задача.

VYP: Выписка. Все возвращается к такому состоянию, которое было при образовании последнего варианта расположения ферзя *i* до его прописки.

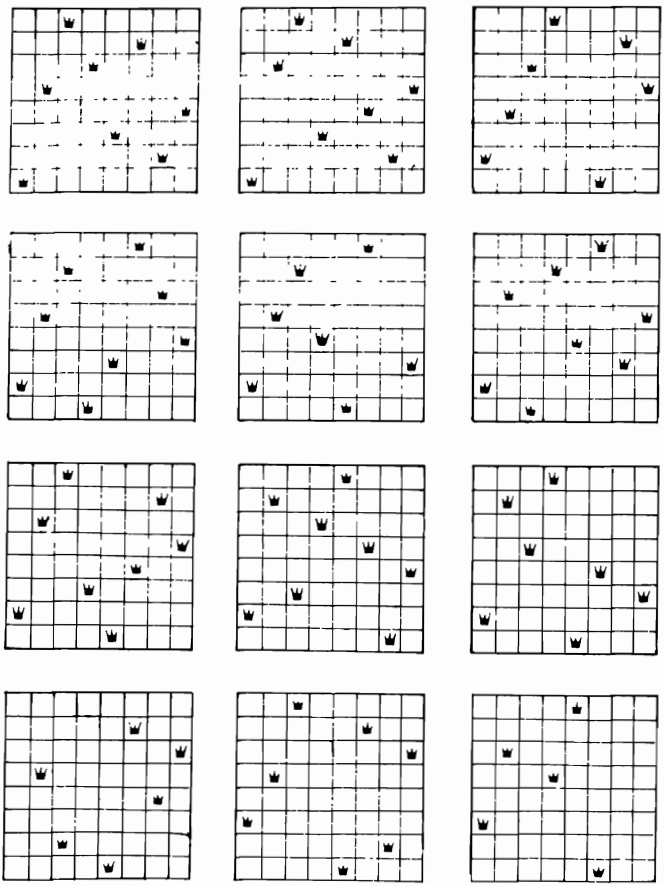


Рис. 23

VSJO: Печать числа вариантов.

7. Посмотрим, сколь универсальна наша программа. Она без изменений идет при любых значениях A, B, N при условии, что

$$A = N \leqslant B$$

Переделка программы на случай $N \leqslant A \leqslant B$ предоставляется читателю.

Для решения задачи 80.1.2 о перестановках достаточно убрать массивы *VOS* и *ZAK*.

Решение других наших задач на частные случаи перебора также не вызывает сложных переделок.

И все же основное содержание этого параграфа — в блок-программе перебора и в том, чтобы научиться ею пользоваться.

Задачи.

(1) Какими способами можно обойти доску 6×6 шахматным конем по замкнутому пути, который совмещался бы с собою при повороте доски на 90°?

(2) Переделайте программу *FERZI* так, чтобы она печатала и считала только те позиции, которые нельзя перевести друг в друга поворотами и зеркальными отражениями доски. Их всего 12 (рис. 2.3).

§ 2.2. Ретроспективный анализ

Есть задачи, в которых перебор оказывается безнадежно долгим, а анализ ситуаций, начиная с заключительных, быстро приводит к цели.

1. Вот самая известная задача такого рода. Два француза решили выяснить, не являются ли они прямыми потомками по мужской линии короля Карла Великого. Были у них все метрические архивы. Первый француз начал с Карла Великого. У короля было 11 сыновей. У сыновей снова были сыновья..., и к концу дня первый француз выяснил, что не сможет закончить анализ за всю свою жизнь. А второй француз начал с себя, нашел в записях своего отца, потом его отца... и в тот же день закончил работу.

2. Вот более трудная задача. Она из сказочных шахмат.

Неприкасаемый король. У белых король и ферзь, у черных — один король. Могут ли белые выиграть, не делая хода своим королем, стоящим на поле c3?

Эта задача была известна еще в прошлом веке, но явилась первой шахматной задачей, решенной машиной раньше всех людьми. Впрочем, зная это решение, человек находит его за 1—2 часа.

2.1. Попытаемся решить задачу перебором. У белых, как правило, больше 20 вариантов хода, у черных — примерно 5. Вариантов ход-ответ более 100. А всего вариантов просмотра на глубину в 20 ходов-ответов получается более $100^{20}=10^{40}$. Если машина будет просматривать миллиард (10^9) позиций в секунду (что уже нереально), то анализ займет миллиарды миллиардов лет (напомним, что возраст галактики менее 100 миллиардов лет). Попытка явно не удалась.

2.2. Можно, однако, заметить, что всего позиций не так уж много. У белого ферзя менее 64 положений, у черного короля — тоже. Всего позиций (считая и невозможные) 4096, а с учетом очереди хода 8192. Значит, сведения о них можно поместить в память машины.

Введем понятие о ранге позиции.

Ранг позиции — это число ходов, которые должны сделать белые, чтобы дать мат.

Заведем два четырехиндексных массива:

$BR[1:8, 1:8, 1:8, 1:8]$

$CR[1:8, 1:8, 1:8, 1:8]$

Индексы F, E, K, R элементов $BR[F, E, K, R]$ и $CR[F, E, K, R]$ будут обозначать:

F и E — номера вертикали и горизонтали, где стоит белый ферзь,

K и R — то же для черного короля.

В самих элементах этих массивов будут записываться:

BR — ранг позиции, если ход белых,

CR — то же, если ход черных.

2.3. Теперь решение состоит в заполнении массивов. Сначала отмечаются невозможные позиции, маты, паты и другие ничьи (король съел ферзя). Для матовых позиций в CR, естественно, заносится 0.

Далее, для каждого $i=0,1,\ldots$ делается следующее.

(1) Просматриваются все позиции (F, E, K, R), еще не имеющие ранга BR. Если из такой позиции можно пойти ферзем в позицию $(F1, E1, K, R)$, где CR равно i, то в $BR[F, E, K, R]$ заносится $i+1$.

(2) Просматриваются все позиции (F, E, K, R), еще не имеющие ранга CR. Если из такой позиции все ходы королем ведут в позиции, уже имеющие ранги BR, то в $CR[F, E, K, R]$ заносится число $i+1$.

Фактически это можно сделать экономнее, но это несущественно важно только принципиальное решение.

2.4. Работа заканчивается, когда при очередном значении I не появится новых значений в элементах BR.

Если все позиции получили ранги BR, то мат всегда возможен, если не все — то не всегда. Так, было выяснено, что при белом короле на $c3$ мат дается не позже 23-го хода.

2.5. После того как массивы заполнены, можно написать программу игры человека с машиной. Если машина играет белыми, то делает очередной ход ферзем так, чтобы ранг CR новой позиции был на единицу ниже ранга BR исходной позиции. Если же машина играет черными, то делает ход королем в позицию с максимальным рангом BR. Любопытно, что эта тактика черных дает человеку, играющему белыми фигурами, немалые трудности.

Чтобы не отыскивать каждый раз нужные ходы, для N удобно завести два четырехиндексных массива BH и CH (белых ход и черных ход). Если в позиции (E, K, R) ферзь должен пойти на $(F1, E1)$, то в $BH(F, E, K, R)$ удобно записать число $10*F1+E1$. Если в этой позиции черный король должен пойти на поле $(K1, k1)$, то в $CH(E, K, R)$ записывается $10*K1+K1$. Массивы BH и CH удобно заполнять одновременно с заполнением массивов BR и CR.

2.6. Для тех, кто захочет написать эти программы, приведем числа, которые удобно предварительно занести в массивы и которыми удобно отметить особые позиции:

Позиция	BR	BH	CR	CH
Предварительно	60	90	60	90
Шах	00	90		
Съел ферзя	50	90	50	90
Ем ферзя	00	90	50	$10*F+E$
Недопустимо	00	90	00	90
Мат	00	90	00	90
Пат			50	90

3. Недавно с помощью ретроспективного анализа было установлено, что король с двумя слонами всегда выигрывает у короля с конем.

§ 2.3. Случайные числа и электронная гадалка

В этом параграфе мы рассмотрим две задачи программирования: получение случайных чисел и угадывание задуманного числа.

1. Случайные числа бывают нужны при решении многих прикладных задач. С их помощью исследуют поведение регуляторов в ответ на случайные отклонения регулируемой величины. Они употребляются для приближенного решения задач вычислительного анализа. Бывают они полезны для проверки правильности работы программы в неожиданных ситуациях. Нужны они в теории игр и других вопросах.

1.1. Для работы ЭВМ со случайными числами вначале пытались вводить эти числа извне. Вводили в память готовые таблицы случайных чисел. Строили приборы, использующие случайные физические процессы, например радиоактивный распад, и вводящие полученные числа в машину. Все это было достаточно плохо. Таблицу случайных чисел ЭВМ быстро исчерпывала, а случайное физическое явление нельзя было повторить, чтобы проверить вычисления.

1.2. Таким образом, возникла парадоксальная задача — вырабатывать в самой ЭВМ числа случайные, но такие, чтобы их можно было повторить и чтобы последующее число вычислялось по предыдущим, но не зависело от них. Задача, разумеется, неразрешимая. Но Дж. Нейман придумал алгоритм, последовательно вычисляющий числа x_1, x_2,..., очень похожие на случайные, равномерно распределенные от 0 до 1. Их называют псевдослучайными, или просто случайными. Сейчас мы изложим этот алгоритм.

Метод середины квадрата. Будем строить четырехзначные псевдослучайные числа. Возьмем произвольное целое k_1 из диапазона $0<k_1<10^4$. Это будет наше первое число. Если нам уже известно k_i, то возведём его в квадрат и возьмем 4 средних разряда. Если, например, $k_i=2251$, то возведя его в квадрат, получим $k_i^2=05067001$, и поэтому $k_{i+1}=0670$.

Чтобы получить из нашей последовательности k_1, k_2,... последовательность действительных чисел x_1, x_2,..., равномерно распределенных между 0 и 1, надо положить $x_i=k_i/10^4$.

Несмотря на очевидные возражения, вроде того, что эта последовательность непременно начнет повторяться или что она может выродиться в сплошные нули, числа $x_1, x_2, \ldots$ похожи на случайные. В среднем в половине случаев $x_{i+1} > x_i$, а в половине — наоборот. Среди первой тысячи чисел $x_1, x_2, \ldots$ примерно половина будет меньше $1/2$, а половина больше, и т. п.

Разработаны и более сложные способы получения псевдослучайных последовательностей, препятствующие обращению членов последовательности в нули и удлиняющие период ее повторения. Но принцип они используют тот же, и для понимания существа дела достаточно изложенного.

1.3. Напишите подпрограмму, доставляющую по $k = k_i$ следующее значение $k = k_{i+1}$ четырехзначных случайных чисел.

Решение.

```
1000 m = k
1010 m = m + 4321
1000 m = m * m
1020 m = m - int(m/1E6)*1E6
1040 k = m
1050 return
```

Здесь k — целая переменная, m — реальная (напомним, что $1E6 = 10^6$). Для борьбы с тенденцией обнуления мы ввели преобразование $m = m + 4321$.

Наши четырехзначные псевдослучайные числа слишком быстро начинают повторяться и склонны к обнулению. Обычно работают с десятизначными числами. Напишите такую программу, учитывая, что во всех промежуточных вычислениях числа не должны иметь более десяти разрядов.

Указание. Представьте десятизначное число парой пятизначных.

Придумайте сами другие методы получения случайных чисел и запрограммируйте их. Например, метод, который строит k_{i+2} по k_{i+1} и k_i. Рассмотрите такой метод ($r = 10^{10}$):

$$k_{i+2} = 7 * k_{i+1} + 3 * k_i + 123456789;$$
$$k_{i+2} = k_{i+2} - (k_{i+2} \backslash r) * r.$$

В нем надо начинать с двух десятизначных чисел, и он будет давать числа от 0 до 10^{10}.

Составьте алгоритм, подающий случайным образом числа 1, 2, 3, 4, 5, 6, взамен бросания игрального кубика.

Указание. Можно рассмотреть наши случайные четырехзначные числа и вычислить $6*k/10^4+1$.

1.4. Случайные числа можно использовать для приближенного интегрирования. Пусть (x_i, y_i) — пара независимых случайных чисел, равномерно распределенных на интервале от 0 до 1. Пусть функция $f(x)$ такова, что $0 \leqslant f(x) \leqslant 1$ при $0 \leqslant x \leqslant 1$. Тогда

$$\int_0^1 f(x)\,dx \approx m/n, \qquad (*)$$

где n — число всех рассмотренных случайных пар (x_i, y_i), а m — число тех из них, для которых $y_i \leqslant f(x_i)$.

Подсчитайте таким способом величину интеграла (*) для $f(x) = x^2$ по $n = 500$ случайным точкам (x, y). Измените программу так, чтобы вычислять с ее помощью объемы трехмерных тел, например,

$$x \geqslant 0,\ y \geqslant 0,\ z \geqslant 0 \text{ и } x^2 + y^2 + z^2 \leqslant 1.$$

2. Электронную гадалку, которую мы сейчас опишем, придумал создатель теории информации К. Шеннон. Работает она так. Человек пишет на бумаге число 0 или 1. Машина этого числа не знает, но печатает 0, 1 или 2. Двойка означает, что машина не берется угадать написанное число, а 0 или 1 — ее предположение о написанном числе. После этого человеку сообщают предположение машины, а в машину вводят число, написанное человеком.

Вначале машина играет неважно, но после двух-трех десятков проб начинает угадывать в 90% случаев, сколько бы человек ни пытался ее запутать. Это производит впечатление.

Устроена программа так. В ней имеется 5-индексный массив $A[0:1, 0:2, 0:1, 0:2, 0:1]$ из 72 элементов. Вначале массив очищен нулями, и машина первые три раза печатает двойки. В дальнейшем машина помнит несколько последних ходов своих и человека. Если человек последними написал числа a_1, a_2, a_3, а машина на это отвечала b_1, b_2, b_3, то в ячейку $A[a_1, b_1, a_2, b_2, a_3]$ добавляется единица, то есть машина запоминает, что после комбинации a_1, b_1, a_2, b_2 человек выбрал число a_3. Чтобы предсказать, что теперь напишет человек, машина сравнивает числа $A[a_2, b_2, a_3, b_3, 0]$ и $A[a_2, b_2,$

a и b. Если первое сильно превосходит второе, то машина предсказывает число 0, если наоборот, то число 1, а если они отличаются мало, то печатает число 2, то есть отказывается угадывать. Можно усовершенствовать программу, добавляя на ходе i в нужную ячейку не единицу, а число $(1.1)^i$, и тем самым уменьшая вес старых событий, которые человек успевает забыть.

Запрограммируйте гадалку так, чтобы цифры, написанные человеком, и цифры, предсказанные машиной, располагались на экране парами и чтобы человек видел последние 10—20 пар. Показывайте все время на экране текущий процент верных угадываний. Испытайте вариант гадалки, не учитывающей своих предсказаний, но зато руководствующейся более длинными сериями чисел человека.

Если бы человек определял свои числа бросанием монеты или с помощью случайных чисел, то программа не смогла бы угадать заметно более 50% чисел. Но человек не умеет задавать числа случайно, и электронная гадалка расшифровывает его тактику или психологию.

§ 2.4. Рекурсии

1. В языках программирования (и даже командах машины) имеется возможность выйти из основной программы, обратиться к подпрограмме и, закончив ее, вернуться в основную программу к оператору, следующему за обратившимся к подпрограмме. Называются такие подпрограммы по-разному — в Бейсике и Фортране это субрутины, в Алголе и Паскале — процедуры, в Си — функции. Сама подпрограмма может, в свою очередь, тоже обратиться к подпрограмме и т. д. При этом возникает следующий вопрос: может ли подпрограмма, сразу или после цепочки промежуточных подпрограмм, обратиться к самой себе? Вопрос решается по-разному в разных языках. В «мощных» языках — Алголе, Паскале, Си,... это разрешено. В Фортране запрещено, а в Бейсике разрешено, но бесполезно (из-за отсутствия механизма передачи параметров).

Если подпрограмма обращается сама к себе как к подпрограмме (непосредственно или через цепочку подпрограмм), то это называется *рекурсией*. В языках, где рекурсии разрешены, все происходит так, как будто бы обращение произошло к другому экземпляру этой же подпрограммы.

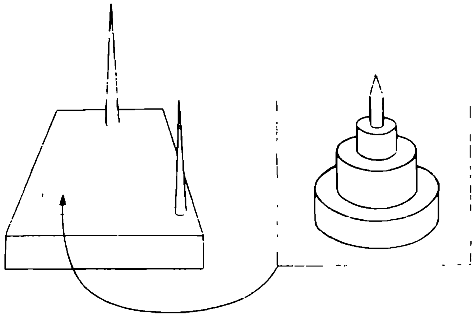


Рис. 2.4

Рекурсий следует избегать. Они замедляют выполнение программы, могут потребовать большой памяти и т. д. Вот, однако, пример, когда рекурсивный алгоритм естествен и прост, а нерекурсивный требует немалой сообразительности.

2. Ханойские башни. В центре мира в вершинах равностороннего треугольника в землю вбиты три алмазных шпиля. На одном из них надето 64 золотых диска убывающих радиусов (самый большой — нижний). Трудолюбивые буддийские монахи день и ночь переносят диски с одного шпиля на другой. При этом их надо переносить по одному и нельзя класть больший диск на меньший (рис. 2.4). Когда все диски перенесут на другой шпиль, наступит конец света (задачу и рассказ придумал математик Э. Люка в 1883 г.).

Оставляя временно вопрос о конце света, поищем алгоритм для перенесения (всех) дисков с одного шпиля на другой. Что это не так просто, вы сможете убедиться, изготовив модель ханойских башен и поупражнявшись с нею.

3. Идея рекурсивного алгоритма не вызывает трудностей. Если алгоритм $P(m, a, b, c)$ должен перенести верхние m дисков со шпиля a на шпиль b, то (в предположении, что все диски лежат правильно — меньший на большем и что остальные диски на всех трех шпилях не меньше чем верхние m на шпиле a) легко пишем алгоритм

$P(m, a, b, c)$
    если $m=1$, то перенеси верхний диск
        со шпиля a на шпиль b
    иначе $P(m-1, a, c, b)$; $P(1, a, b, c)$;
    $P(m-1, c, b, a)$

«По человечески» этот алгоритм очень понятен. Если $m = 1$, то перенеси один диск с a на b. Если же $m > 1$, то перенеси временно $m - 1$ верхних дисков с a на c. Потом перенеси один оставшийся диск с a на b и, наконец, перенеси $m - 1$ дисков, хранящихся на c, на шпиль b. Что касается перенесения $m - 1$ дисков, то для этого подойдет тот же алгоритм, но с уменьшенным (от m до $m - 1$) числом переносимых дисков. Таким образом, мы перейдем от m к $m - 1$, от $m - 1$ к $m - 2$, $m - 3$, ... и дойдем до единицы.

Чтобы записать этот алгоритм на каком-либо языке программирования, обозначим шпили a, b, c цифрами 0, 1, 2 и будем печатать нужные переносы дисков. Перенос со шпиля a на b будет изображаться надписью $0 \to 1$, с b на c — надписью $1 \to 2$, и т. п.

```
program HANOI(input, output);
var n : integer;
procedure P(m, a, b, c: integer);
begin if m=1 then write(a,'→', b,' ')
  else begin P(m-1, a, c, b); P(1, a, b, c);
  P(m-1, c, b, a) end
end;
begin readln (n); write('n=', n); P(n, 0, 1, 2)
end.
```

При $n = 4$ эта программа напечатает

```
0→2 0→1 2→1 0→2 1→0 1→2
0→2 0→1 2→1 2→0 1→0 2→1
0→2 0→1 2→1
```

4. Только теперь, внимательно проследив за перемещениями дисков в рекурсивном алгоритме, мы можем написать алгоритм без рекурсий:

T: *Первый (минимальный) диск переложить по часовой стрелке. Из двух других (верхних) дисков меньший переложить на больший (если остался один, то переложить его на пустой стержень, а если не осталось ни одного, то задача решена).*

Если задача еще не решена, то повторить действия (с метки T :).

Этот алгоритм также приводит к решению — для нечетного числа n диски переложатся на соседний шпиль по часовой стрелке, а для четного — против. Доказательство можно провести по индукции по числу n дисков.

Для реализации алгоритма на Бейсике занумеруем шпили числами 0, 1, 2. Заведем двумерный массив $A(2, n)$ и одномерный $m(2)$. В $m(0)$ будем держать число дисков на шпиле "0", а размеры этих дисков поместим в $A(0, 1)$, $A(0, 2)$,.... Вначале это будут числа n, $n-1$,... Аналогичным образом разместим сведения о дисках на шпилях "1" и "2". В переменные $A(0, 0)$, $A(1, 0)$, $A(2, 0)$ занесем не существующие диски размера $n+1$ (большего, чем все заданные) — это упростит выяснение того, откуда куда надо перенести очередной диск. Программа получится такой:

```
10 input "n";n
20 dim A(2,n),m(2)
30 for i=0 to n:A(0,i)=n+1-i:next
40 A(1,0)=n+1:A(2,0)=n+1
50 m(0)=n:m(1)=0:m(2)=0
60 i=0:j=1:k=2
70 p=i:q=j: gosub 200
80 if A(i,m(i))>=A(k,m(k)) goto 100
90 p=i:q=k: gosub 200: goto 120
100 if A(k,m(k))>=A(j,m(j)) goto 140
110 p=k:q=j: gosub 200
120 t=i:i=j:j=k:k=t
130 goto 70
140 end
200 m(q)=m(q)+1
210 A(q,m(q))=A(p,m(p))
220 m(p)=m(p)-1
230 print p;"->";q;
240 return
```

5. Отладьте на машине одну из программ «Ханойские башни». Только не берите число дисков n слишком большим. Хотя ваша машина работает в миллионы раз быстрее буддийских монахов, но и ей на перекладывание 64 дисков потребуется миллион лет (заметим мы, возвращаясь к вопросу о конце света).

Упражнения.

(1). Докажите, что для переноса башни из n дисков с одного шпиля на другой достаточно 2^n-1 переносов и нельзя сделать этого быстрее.

(2). Попытайтесь переделать не рекурсивную программу так, чтобы обойтись без массивов.

6. Рекурсивные определения функций могут быть и более сложными, когда не только значения, но и аргументы функции определены рекурсивно, т. е. с помощью ее же определения. Так, например, функция Аккермана $\mathrm{Ack}(m, n)$ определяется для целых $m \geqslant 0$, $n \geqslant 0$ условиями:

$$\begin{aligned}
&\mathrm{Ack}(0, n) = n + 1,\\
&\mathrm{Ack}(m, 0) = \mathrm{Ack}(m-1, 1) && \text{при } m > 0,\\
&\mathrm{Ack}(m, n) = \mathrm{Ack}(m-1, \mathrm{Ack}(m, n-1)) && \text{при } m, n > 0.
\end{aligned}$$

Напишите (рекурсивную) программу для вычисления значений $\mathrm{Ack}(m, n)$. Заметьте, что ручное вычисление уже для $\mathrm{Ack}(3, 2)$ оказывается довольно громоздким.

7. Решения к упражнениям из п. 5.

(1) Обозначим через $P(n)$ число переносов, которые тратит наш рекурсивный алгоритм, чтобы перенести башню из n дисков. Тогда, согласно этому алгоритму,

$$\begin{aligned}
&P(1) = 1,\\
&P(n+1) = P(n) + P(1) + P(n).
\end{aligned}$$

Докажем, что $P(n) = 2^n - 1$. При $n = 1$ эта формула верна. Пусть она верна для некоторого n, тогда

$$P(n+1) = 2P(n) + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 1.$$

Формула доказана.

Покажем, что быстрее башню перенести нельзя. Пусть башню из n колец можно перенести за $Q(n)$ переносов. Докажем, что

$$Q(n) \geqslant P(n).$$

Для $n = 1$ это верно, ибо $Q(1) \geqslant 1$. Пусть это неравенство верно для некоторого n. Чтобы перенести башню из $n+1$ дисков, нужно будет в какой-то момент перенести нижний $(n+1)$-й диск. К этому моменту один шпиль должен быть свободен, а верхние n дисков должны быть сняты — и значит перенесены на какой-то третий шпиль. Для этого, по предположению индукции, уйдет не менее $P(n)$ переносов. Теперь нужно истратить один перенос для перемещения $(n+1)$-го диска и не менее $P(n)$ переносов, чтобы перенести башню из n дисков. Следовательно,

$$Q(n+1) \geqslant P(n) + 1 + P(n) = P(n+1).$$

(2) Не рекурсивную программу, без использования массивов, на Бейсике можно сделать такой:

```
10 p 0:q 1:r=2
20 input "n";n
30 k 2^(n 1)+0.1:m 0
40 goto 90
50 i=1:j=m
60 if (j mod 2)=0 tnen
     i=i+1:j=j\2:goto 60
70 if i mod 2
    tnen u=r:v=q else u=q:v=r
80 print u;"->";v;"  ";
90 print p;"->";q;"  ";
100 t=p:p=q:q=r:r=t:m=m+1
110 if m<k goto 50
120 end
```

Для $n=4$ она напечатает

$0\to 1\ \ 0\to 2\ \ 1\to 2\ \ 0\to 1\ \ 2\to 0\ \ 2\to 1$
$0\to 1\ \ 0\to 2\ \ 1\to 2\ \ 1\to 0\ \ 2\to 0\ \ 1\to 2$
$0\to 1\ \ 0\to 2\ \ 1\to 2$

§ 2.5. Структурированное программирование

1. Структурированное программирование — это определенный способ организации программы, коренным образом влияющий на разработку больших программ. Структурированное программирование облегчает написание большой программы, ускоряет его, упрощает отладку, помогает разделению работы между исполнителями и открывает возможности для дальнейших переделок программы.

Структурированное программирование возникло не позже конца пятидесятых годов и рассматривалось в печати уже в середине шестидесятых [1], но была серьезная причина, задержавшая распространение нового метода. При программировании снизу-вверх программистская квалификация требовалась от исполнителей, а руководитель мог быть администратором. При программировании же сверху-вниз руководитель должен быть квалифицированным программистом. Только Э. Дейкстра [6, 7] в семидесятых годах сумел убедить программистов ведущих фирм в категорической необходимости структурированной организации больших программ. С тех пор по

структурированному программированию стали выходить толстые книги и научные статьи.

В настоящее время большая неструктурированная программа воспринимается как написанная архаично или неграмотно.

2. Сформулируем основной признак структурированной программы.

Структурированная программа целиком состоит из блок-программы и блоков (то есть операторов или подпрограмм), к которым блок-программа обращается. Блок должен возвращать продолжение программы к оператору блок-программы, следующему за обратившимся к нему.

Блок-программа должна быть обозримой (достаточно короткой). Она (по возможности) не содержит вычислений и состоит лишь из операторов передач управления (условных или безусловных) на блоки (выполняющие нужные вычисления) и на операторы самой блок-программы. Можно сказать, что выполнение структурированной программы состоит в выполнении ее блок-программы, которая организует работу своих блоков, играющих в ней роль операторов с определенными локальными функциями.

В литературе и устной речи встречаются различные названия для блок-программы — от «собиралки» до «программной записи блок-схемы». Все же блок-программа пишется по более жестким правилам, чем блок-схема, и становится более определенной и обязательной.

3. Чтобы написать структурированную программу, надо разбить задачу на отдельные осмысленные части, и начать писать ее с блок-программы, обращающейся к этим, еще не написанным частям (блокам). Не беда, если при написании блок-программы будет не известно, как выполнить задачи, поставленные перед блоками — решение следует отложить до написания самих блоков. Все, что не ясно, надо исключать из блок-программы и относить к блокам, а в блок-программе выяснять только, когда к какому блоку следует обратиться.

Блок нужно оформлять как отдельную часть программы с одним входом и одним выходом. Он может быть просто частью программы, но может иметь вид процедуры Паскаля, Фортрана, функции Си и т. п. Если при написании отдельного блока снова появятся трудности или его программа окажется плохо обозримой, то программу этого блока следует тоже разбить на более мелкие осмысленные части и написать его блок-

программу. Итак, структурированную программу разбивают на блоки — это называется декомпозицией программы — и пишут сверху, то есть начиная с блок программы.

Отладку программы надо постараться начать до написания всех блоков, вставив вместо отсутствующих блоков какие-нибудь операторы, имитирующие их работу и дающие определенные ответы. Структурированную программу легче будет потом уточнить, усовершенствовать и даже переделать. Это тоже ее важное преимущество перед программой, написанной по старинке «в линеечку», когда все, что выполняется подряд, записывается и в программе подряд.

4. Преимущества структурированного программирования убедительно сказываются только на больших программах, которых мы не можем здесь привести. Поэтому ограничимся тем, что мы декларативно смогли сообщить, и дадим сводку принципов структурированного программирования.

5. Принципы структурированного программирования:

(1) Программа должна состоять из блок-программы и блоков, к которым блок-программа обращается. Так же пишется и всякий сложный блок.

(2) Блок-программа состоит из операторов обращения к блокам и операторов перехода (условных и безусловных) к операторам самой блок-программы.

(3) Блок имеет единственный вход и единственный выход. Блок всегда возвращает управление оператору блок-программы, следующему за тем, который обратился к нему.

(4) Программа пишется и отлаживается сверху, то есть начиная с блок-программы (то же относится к блоку, имеющему блок-программу). Вместо отсутствующих (к данному моменту) блоков пишутся заменители, имитирующие работу этих блоков.

(5) Изложенные здесь принципы можно нарушать только в случаях серьезной надобности.

6. Замечание. Некоторые считают, что структурированная программа, и даже язык программирования (но все же не система команд машины) не должны содержать операторов перехода (подобных goto). В этом случае вся программа будет состоять из операторов присваивания, обращения к процедурам, самих процедур, выбора одного из двух или нескольких вариантов и

операторов циклического повторения до достижения определенного условия.

Мы этого не считаем. Признавая, что злоупотребление операторы перехода служат указанием на неряшливость или низкую квалификацию программиста и затрудняют проверку программы, мы, тем не менее, не запрещаем операторов перехода. Причина этого в том, что нарочитое избегание операторов перехода отдаляет программу от естественного алгоритма решения задачи, а употребление операторов перехода не является единственным способом написания плохих программ.

§ 2.6. Что такое алгоритм

«Алгоритм» является фундаментальным понятием информатики. Представление о нем необходимо для эффективного применения вычислительной техники к решению практических задач. Мы сначала дадим определение алгоритма, потом рассмотрим ситуации, приведшие к такому определению и в конце познакомимся с историей возникновения и установления понятия алгоритма.

1. Начнем с неформального определения. *Алгоритм* — это точное и безотказное предписание действий, которые должны быть выполнены.

Нужно понять смысл слов, входящих в это определение, и то, что никакими иными свойствами алгоритм может и не обладать.

Что означает, что предписание должно быть точным? Оно должно быть настолько точным, чтобы его мог выполнить компьютер с неограниченной памятью. Здесь иногда вместо «выполнить» надо сказать «промоделировать». Возможно здесь и еще одно расширение понятия «точный», к которому мы вернемся в п. 5.

Что означает, что предписание должно быть безотказным? Это значит, что в любых условиях, на которые объявлен алгоритм, он должен выдавать свои предписания. Особую трудность могут представить те условия, которые могут возникнуть в дальнейшем в результате выполнения самого алгоритма.

2. Иногда говорят не «алгоритм», а «алгоритм решения некоторой задачи (или группы задач)». Понятия эти близкие, но разные, ибо алгоритм может и не решать никаких задач. Они могут в нем вовсе не упоминаться.

Алгоритм решения некоторой задачи — это алгоритм, приводящий к решению этой задачи за конечное число действий.

Алгоритм решения группы задач — это алгоритм, приводящий к решению каждой задачи (из этой группы) за конечное число действий.

3. Алгоритм может быть хорошим или плохим (по чьей-то оценке), он может быть ясным или запутанным, коротким или длинным. Он может быть пригоден для решения одной или многих задач, может приводить к решению быстро или медленно. Может быть пригоден для выполнения на простом автомате или требовать компьютера с большой памятью. Можно назвать и иные качества алгоритма, но нужно понимать, что никакими свойствами, кроме указанных в п. 1, алгоритм может и не обладать.

4. Необходимо также понимать, что мы можем уметь решать задачу и не знать алгоритм ее решения, т. е. не знать (достаточно точно), как мы ее решаем. С такими случаями мы познакомимся на примерах.

5. Облегчим применение понятия «алгоритм». По определению, приведенному в п. 1, получается, что алгоритм — это программа для компьютера. Строго говоря, так оно и есть. Но проверять это всякий раз было бы длинно. Поэтому мы будем считать алгоритмом и такие указания, про которые нам будет ясно, как сделать их понятными компьютеру, т. е. как довести их до программы. Это, конечно, не совсем точно, так что в сомнительных случаях следует возвращаться к определению из п. 1.

6. Перейдем к примерам алгоритмов.

6.1. Все мы знаем алгоритм сложения десятичных чисел. Надо заранее выучить таблицу сложения пар однозначных чисел, а затем складывать цифры слагаемых поразрядно, справа налево, учитывая единицы переноса. Вы можете сами аккуратно выписать этот алгоритм. Легко его и запрограммировать. Стоит заметить, что алгоритм решает за конечное число действий каждую задачу, а не все задачи, на которые объявлен.

6.2. Напомним задачу о волке, козе и капусте. По условию волк, коза, капуста, перевозчик и лодка находятся на одном берегу и их надо перевезти на другой. В лодке, кроме перевозчика, может находиться только одно животное или только капуста. Оставлять (без присмотра перевозчика) волка с козой или козу с капустой нельзя. Решите эту задачу. Заметьте, что для самого

решения вам не понадобилось ни лодки, ни животных. Вы «промоделировали» решение. И эту модель можно запрограммировать.

6.3. Алгоритм Евклида. Для двух заданных натуральных чисел m, n требуется найти их наибольший общий делитель $d(m, n)$. Заметим предварительно, что если $m=n$, то $d(m, n)=m$. Если же $m\neq n$ и, для определенности, $m<n$, то всякий общий делитель пары чисел m, n будет и общим делителем пары m, $n-m$. И наоборот. Следовательно, если $m<n$, то $d(m, n)=d(m, n-m)$.

Это подсказывает следующий алгоритм отыскания $d(m, n)$:

(1) если $m=n$, то ответ равен m и вычисление закончено;

(2) иначе надо заменить большее число разностью между ним и меньшим и вернуться к п. (1).

Ясно, что этот алгоритм приводит к решению за конечное число шагов (докажите!). Алгоритм реализуется программой:

```
10 input m, n
20 goto 40
30 if m<n then n=n-m
          else m=m-n
40 if m<>n goto 30
50 print m
60 end
```

Решение можно ускорить. Действительно, если $m<n$, то число m будет вычитаться из n до тех пор, пока разность станет меньше или равна m. Но эту (последнюю) разность можно получить сразу, как остаток (n mod m) от деления n на m. Нужно только учесть, что остаток может обратиться в 0. Программе можно придать вид

```
10 input m, n
20 if m<n goto 40
30 swap m, n
40 n = n mod m
50 if n>0 goto 30
60 print m
70 end
```

Разберите ее работу. Поупражняйтесь с ней на машине, сделайте примеры.

7. Мы успешно программируем задачи, в которых у человека нет интуиции. Но когда переходим к решению задач, где приходится конкурировать с механизмом, созданным самой природой, наши программы значительно уступают решениям людей и животных. Выясняется, что мы не знаем, как мы думаем и не умеем этого запрограммировать — не знаем алгоритмов природы.

7.1. Завязать шнурки на ботинках бантиком нетрудно. Многие дети в 6 лет умеют это делать. Но написание алгоритма может затруднить и взрослого.

7.2. Разбор кучи. В начале автоматизированной линии обработки поковок коленчатых валов стоит человек. Он вынимает заготовки из короба и вставляет в конвейер. Автоматизировать эту работу не удается. Надо узнавать деталь в разных ракурсах, частично заложенную другими деталями. Решать, с какой начать и по какой траектории ее вынимать.

7.3. Чтение. Человек легко читает книгу. Для машины здесь две задачи, и с обеими дело обстоит плохо. До сих пор машина не умеет прочесть страницу книги, то есть ввести в свою память буквы с этой страницы, и не может воспроизвести речью текст, введенный в нее с клавиатуры. Для первой задачи имеется одна, практически работающая программа на большой машине, но выдает текст с ошибками. Человек делает это как-то проще!

7.4. Если в предыдущих примерах можно что-либо свалить на несовершенство внешних устройств, то при программировании шахмат недостатки нашего понимания того, «как мы думаем», проявляются в чистом виде. Рассмотрим для примера одну позицию: Б. Крb5, Фb6; Ч. Крс3. Ее без труда выигрывает едва научившийся играть. Но даже мастер не сможет быстро написать выигрывающий алгоритм. Да и написанный им алгоритм будет играть хуже, чем едва научившийся человек.

8. О неаккуратных и неудачных определениях и пониманиях алгоритмов.

8.1. Начнем с известного примера «бытового алгоритма» перехода улицы.

«Посмотри налево. Если машин нет — дойди до середины улицы. Если есть — подожди, пока они проедут. И т. д.» Представьте себе, что машина слева есть, но она не едет — у нее меняют колесо. Если вы думаете, что надо ждать, то вы поняли этот алгоритм. Если же вы решили, что улицу переходить можно, считая алго-

…ем подправленным ввиду непредвиденных (по вашему мнению¹) обстоятельств, то вы не усвоили понятия алгоритма. Еще раз подчеркнем — алгоритм всегда должен быть рассчитан на выполнение (не размышляющей) машины.

8.2. Некоторые думают, что алгоритм обязательно должен быть «массовым», то есть применимым к группе задач, и вводят это требование в само определение алгоритма. Этого не следует делать. Такого требования нет в п. 1, нет его в понятии алгоритма в математической логике (см. п. 9). Если его ввести, то мы не сможем говорить об алгоритме решения одной определенной задачи (см. п. 6.2).

8.3. Иногда спрашивают, чем же отличается алгоритм решения задачи от решения этой задачи? Разница в том, что решение содержит ответ, а алгоритм может указывать всего лишь способ для получения ответа. Рассмотренный в п. 6.3 алгоритм Евклида дает лишь способ найти наибольший общий делитель чисел 429 и 455, а решение этой задачи должно содержать ответ: 13.

8.4. Некоторые говорят, что алгоритм «должен быть понятен исполнителю, на которого расчитан», и включают это в определение понятия алгоритма (ср. с п. 1). Такое определение алгоритма делает его недостаточно ясным и не универсальным.

8.5. Рассмотрим, для примера, такой «алгоритм» игры в шахматы — поручить провести игру квалифицированному шахматисту. Нас не устраивает ни такая «алгоритмизация», ни такой «исполнитель». Для нас это — не алгоритмическое решение задачи.

9. Из истории понятия «алгоритм».

9.1. Алгоритмы появились с самого зарождения математики. Появились они в качестве правил для вычисления разного рода величин. Например:

формула для вычисления суммы арифметической прогрессии:

$$s := (a + b) \cdot n/2,$$

где a и b — это первый и последний члены;

формула для вычисления площади треугольника:

$$s := ah/2, \text{ или } s := \text{sqrt}(p \cdot (p - a) \cdot (p - b) \cdot (p - c)),$$

где $p := (a + b + c)/2$.

алгоритм Евклида, с которым мы познакомились в п. 6.3, и другие.

9.2. Слово «алгоритм» возникло в средние века, когда европейцы познакомились со способами выполнения арифметических действий (сложения, вычитания, умножения столбиком и деления уголком), описанными математиком из Хорезма. Таким образом, слово «алгоритм» оказывается европеизированным произношением слов «аль Хорезм» и первоначально под алгоритмом понимали способы выполнения арифметических действий, описанные в этом руководстве математика из Хорезма.

9.3. Научное (строгое) определение алгоритма дал А. Черч в 30-е годы. Он придумал логическую систему «рекурсивных функций», назвал алгоритмами построения в этой системе, доказал важные теоремы и высказал «постулат Черча — все, что сможет доказать математик, можно построить в системе рекурсивных функций». После Черча и другие математики (А. Тьюринг, Э. Пост, А. Н. Колмогоров, А. А. Марков и др.) давали определения алгоритма. Все эти определения эквивалентны определению Черча. Нам будет удобно определение Тьюринга: «алгоритм — это программа для машины Тьюринга». Описывать точно машину Тьюринга мы не будем, скажем только, что это простейшая цифровая машина с неограниченной памятью. Таким образом, у нас будет не вполне строгое, но достаточно ясное понятие алгоритма.

10. Машина Тьюринга.

Ученику не нужно знать машины Тьюринга, но преподавателю нужно иметь о ней представление хотя бы для того, чтобы грамотно ответить на неожиданный вопрос ученика.

10.1. Машина Тьюринга состоит из следующих частей:

(1) внешняя память с внешней головкой и внешним алфавитом;

(2) внутренняя память на одно число, которое может меняться от 0 до m;

(3) три функции f, g, h от двух переменных.

10.2. Опишем эти части.

(1) Внешний алфавит содержит конечное число $(n+1)$ букв, которые можно обозначить $a0$, $a1$, ..., an. Букву $a0$ называют пустым знаком. Внешняя память — это лента с бесконечной в обе стороны последовательностью клеток. В каждой клетке записывается одна буква внешнего алфавита. В дальнейшем эта буква может изменяться. Внешняя головка находится на одной клетке внешней памяти; машина видит букву только этой клетки, только ее может менять. Внешняя головка может передвигаться вперед или назад на одну клетку ленты.

(2) Внутренняя память состоит из регистра на одно число от 0 до m. Эти числа называют состояниями машины (или номерами состояний). Число 0 называют остановкой. Числа могут меняться (определенным образом, см. далее).

(3) Каждая таблица:

$f(i, aj)$, $g(i, aj)$, $h(i, aj)$

[illegible] a_0 [illegible] буквы i и буквы алфавита a_j, которую внешняя головка видит на [illegible] ленты.

Таблица $i' = f(i, a_j)$ дает состояние, в которое перейдет машина (ясно, что $f(i, a_j)$ принимает значения лишь от 0 до m).

Таблица $a_j' = g(i, a_j)$ дает новую букву, которую надо вписать в обозреваемую ячейку ленты (таким образом, значения $g(i, a_j)$ — это буквы внешнего алфавита).

Таблица $h' = h(i, a_j)$ принимает три значения $0, -1, +1$, которые означают, оставить головку на месте, передвинуть на одну ячейку влево или вправо.

10.3. Вначале в конечном числе ячеек внешней памяти записаны любые знаки внешнего алфавита, а остальные заполнены знаками a_0; машина находится в состоянии $i = 1$. Дальнейшая работа машины происходит так. Если она в состоянии i читает на ленте знак a_j, то находит в таблицах новые значения i', a_j', h' и запоминает их. Затем переходит в состояние i', вписывает в обозреваемую ячейку букву a_j', передвигается согласно значению h'. После этого цикл повторяется: машина в новом состоянии видит новую букву вместо [illegible] алфавита и т. д.

10.4. Хоть это и может показаться удивительным, но на такой простой машине можно реализовать все арифметические действия и вообще все, что только удается человеку определить строго (в конечное число шагов). Для десятичной арифметики, например, заведомо хватит внешнего алфавита из 15 букв и регистра из 5 внутренних состояний.

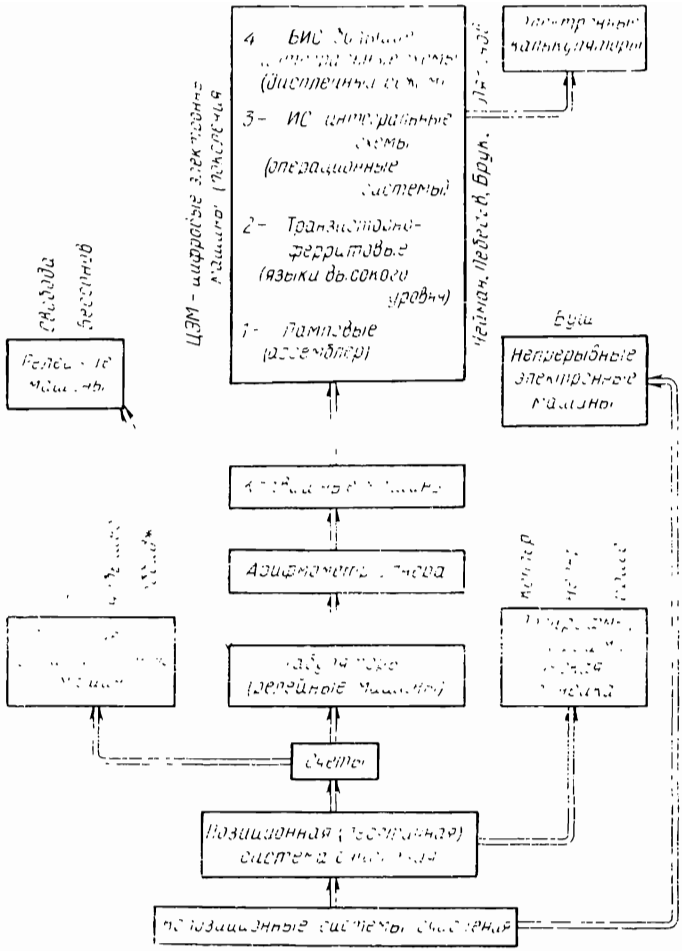
10.5. Замечание. В некоторых книгах вместо трех функций i, g, h [illegible] о наборе пятерок (i, a_j, i', a_j', h'), содержащем все пары (i, a_j) [illegible] одном раз. Это разумеется то же самое.

10.6. Машина Поста устроена, по существу, так же. Только ее внешний алфавит состоит лишь из двух знаков.

§ 2.7. История вычислительной техники

Мы рассмотрим события, лежащие на основной дороге развития вычислительной техники и сознательно опустим (или лишь упомянем) то, что осталось на боковых, не жизнеспособных путях.

1. Крупнейшим событием вычислительной техники было изобретение позиционной (десятичной) системы счисления, которой мы сейчас пользуемся. В этой системе значение цифры зависит от позиции, которую цифра занимает в числе. В самой правой позиции цифра указывает число единиц, в следующей — число десятков, еще левее — число десятков десятков и т. д. Появилась десятичная система, вероятно, в Индии. Выбор графических изображений для цифр, разумеется, не принципиален. Все же попытайтесь догадаться, по какому принципу они были выбраны (см. рис. 2.5). Заметьте, что современные изображения цифр — простая стилизация древних, более удобная для письма.



В непозиционных системах считать трудно. Древние греки построили геометрию, которую изучают в школе, и доказали важные теоремы теории чисел, но считать они не умели. В древнем Риме придумали «римские числа» (см. рис. 2.6), но выполнять арифметические действия с этими числами безнадежно. Попытайтесь решить примеры, приведенные на рис. 2.7, и вы сами убедитесь в этом. Представления о десятичных (или иных позиционных) дробях в Греции и Риме не было.

Рис. 2.5. Древние изображения десятичных цифр (по какому принципу они выбраны?)

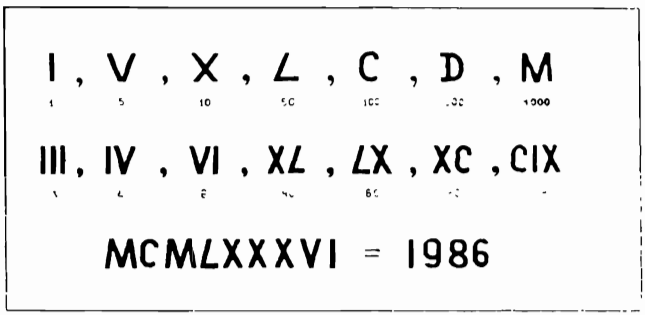


Рис. 2.6. Римские числа

После появления десятичной системы быстро появились правила для выполнения арифметических действий: сложения, вычитания, умножения «столбиком» и деления «уголком». Появились и десятичные дроби. Умение считать широко распространилось.

Изобретение десятичной системы счисления относится к главным достижениям человеческой мысли (наряду с алфавитным письмом). Без нее вряд ли могла существовать, а тем более возникнуть, современная техника. Но положенное в ее основание число десять — дар случайный, дар напрасный — связано со счетом на пальцах рук, практически не употребляемым. Выяснилось, что

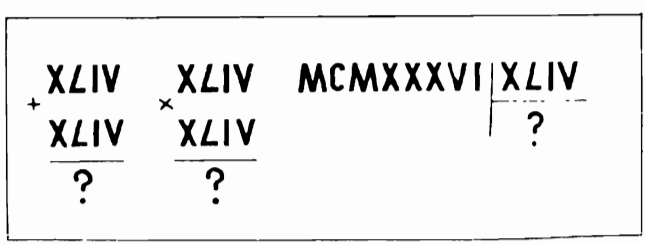


Рис. 2.7. Легко ли считать в римских числах?

для повседневного счета была бы удобнее двенадцатеричная система (в ней хорошо записываются треть и четверть). Были придуманы названия для дополнительных цифр и для круглых чисел (дюжина, гросс). Но на двенадцатеричную систему люди не перешли, чтобы не переучиваться. Иное дело вычислительные машины. Выяснив, что для них удобнее двоичная система, большинство машин стали строить считающими в двоичной системе.

2. Следующий длинный этап вычислительной техники связан с вычислениями на счетах. Счеты оформлялись по-разному в разных странах. Одними из лучших считаются «русские счеты» с десятью косточками на каждом стержне. На счетах нетрудно складывать и вычитать. Некоторые умеют на них умножать и делить, но это уже медленно и утомительно. Просуществовали счеты до наших дней.

3. Особую роль в истории вычислительной техники сыграло изобретение логарифмов. Придумал их известный математик и астроном И. Кеплер, а составили два первых таблицы натуральных логарифмов Дж. Непер и десятичных Г. Бригс. С помощью логарифмов умножение и деление сводятся к сложению и вычитанию и, тем самым, сильно упрощаются. Но поскольку ни логарифмы, ни логарифмические линейки не сказались на развитии вычислительной техники в том направлении, по которому она пошла, то мы останавливаться на них не будем. Отметим только, что они потеряли значение для арифметических вычислений с появлением арифмометра.

4. Полезность вычислительных машин понимали многие выдающиеся ученые. На протяжении веков был предложен целый ряд таких устройств — машина Паскаля, машина Лейбница, арифмометр Чебышева, аналитическая машина Беббиджа... Но все эти предложения были неконструктивны — либо они представляли безнадежно дорогие конструкции, либо вообще не могли быть практически реализованы в полном объеме (по крайней мере в свое время) и тоже не сказались на развитии вычислительной техники.

5. Третий этап вычислительной техники составили машины на электромеханических реле. Были они большими, тяжелыми и стоили дорого. Обрабатываемые числа машина вводила с предварительно набитых перфокарт, а программа для каждой задачи набиралась

вручную проводами на коммутационной доске. На этих машинах обрабатывали большие массивы чисел по несложным фиксированным программам. Скорость вычислений равнялась нескольким действиям в секунду. Сначала релейные машины могли только складывать целые числа, потом — вычитать и умножать, но до деления дело не дошло. На релейных машинах обрабатывали результаты переписей населения. Их широко использовали в бухгалтериях больших заводов. Для инженерных или научных расчетов они не применялись. Вытеснены релейные машины были электронными.

Значительно позже Свобода и Н. И. Бессонов построили релейные вычислительные машины, работавшие с максимальной для них скоростью — около 100 действий в секунду. Эти машины поражают изобретательностью авторов, но не идут в сравнение с электронными.

6. Четвертый этап был открыт появлением ручного арифмометра. Его изобрел в 1874 г. ленинградский (тогда — петербургский) инженер В. Однер. Арифмометр был небольшим и недорогим чисто механическим прибором для индивидуального пользования. На нем легко выполнялись все четыре арифметических действия и вычислялась сумма произведений. Вслед за Россией этот арифмометр начали выпускать во всех развитых странах. Следует отметить, что по простоте изготовления арифмометр могли бы выпускать на несколько веков раньше, если бы придумали. Центральную часть арифмометра составляли колеса с переменным числом зубьев (от 0 до 9) и остроумный механизм, позволяющий устанавливать нужное число зубьев в зависимости от цифр числа. Сначала арифмометры применялись для астрономических, картографических и бухгалтерских расчетов. Затем их стали применять для расчетов деталей машин.

7. Только через 50 лет после ручного арифмометра появились электрические настольные вычислительные машинки, составившие пятый этап развития вычислительной техники. Они работали быстрее ручного арифмометра и меньше утомляли, но были в 100 раз дороже него и применялись, в основном, для научных и бухгалтерских расчетов. На них закончилась «доэлектронная» история вычислительной техники. Вытеснены были клавишные машины микрокалькуляторами.

8. Шестой этап составили электрические и электронные вычислительные и аналоговые машины непрерывного действия. Появились они в конце тридцатых годов на-

шего века. Обрабатываемая величина изображалась в них не числом из цифр, а величиной напряжения на соответствующем проводе. Напряжение могло меняться непрерывно, поэтому такие машины называли непрерывными, в отличие от цифровых. Программы на этих машинах набирались переключателями. На них либо выполняли цепочки арифметических действий, либо собирали схемы, моделирующие изучаемые объекты. Набор программы занимал много времени и это обесценивало скорость работы машин. Использовались машины непрерывного действия только для научных расчетов. На них рассчитывали движения по сложным законам, рассчитывали линии дальних электропередач, моделировали залежи нефти и газа. Но точность промежуточных вычислений была так низка (ошибки доходили до 1% и даже до 10%), что окончательный результат в сложной задаче искажался полностью. Несмотря на усилия энтузиастов, эти машины потеряли значение с появлением цифровых электронных машин и никак не сказались на развитии вычислительной техники в главном направлении. И только их название - ЭВМ (т. е. электронно-вычислительные машины) - сохранилось в вычислительной технике и применяется для обозначения современных цифровых электронных машин (ЦЭМ), пригодных и для вычислений и для управления.

9. Цифровые электронные машины появились в начале сороковых годов нашего века. В них были реализованы основные принципы, предложенные Дж. Нейманом. Вот эти принципы.

1. Память. Машина имеет память, в которой хранит программу, данные (числа) и результаты промежуточных вычислений.

2. Программа. Программа вводится в машину так же, как данные (а не коммутируется проводами).

3. Адресный принцип. В команде указываются не сами числа, над которыми нужно выполнять арифметические действия, а адреса (т. е. номера ячеек памяти), где эти числа находятся.

4. Автоматизм. После ввода программы и данных машина работает автоматически, выполняя предписания программы без вмешательства человека. Для этого машина всегда помнит адрес выполняемой команды, а каждая команда содержит (явное или не явное) указание об адресе команды, которую следует выполнять за нею. Указание может быть одним из трех типов: неявным (не-

... (или к команде, следующей по адресу за выполняемой), безусловным (перейти к команде по такому-то адресу) или условным (проверить некоторое условие и, в зависимости от его выполнения, перейти к команде по тому или иному адресу).

5. Переадресация. Адреса ячеек памяти, указанные в команде, можно вычислять и преобразовывать как числа. Благодаря этому ЦЭМ может сама (по заданной программе) готовить команды, которые выполняет.

Нужно понимать, что главным в этом перечне является пункт о переадресации. Никогда бы ЦЭМ не заняли их нынешнего положения, если бы каждую команду, выполняемую машиной, должен был написать человек. Если бы это было так, то многие программы стали бы безнадежно длинны, а некоторые принципиально невозможны. На самом же деле работа ЦЭМ напоминает движение танка, который выкладывает перед собой рельсы (на внутренней стороне гусениц) и едет по ним.

Для машины, не изменяющей исполнительные адреса своих команд, мы могли бы, конечно, написать программу решения двух линейных уравнений с двумя неизвестными, или для трех уравнений с тремя, и т. д. Но каждая следующая программа была бы длиннее предыдущей и наши возможности быстро бы исчерпались. А универсальную программу решения n линейных уравнений с n неизвестными, пригодную для работы с любым n (которое ей будет указано потом), вообще написать было бы нельзя. Между тем, любая современная машина снабжается такой программой, и программа эта довольно коротка.

С появлением ЦЭМ возникла реальная перспектива переложить на машины часть не энергетического и не вычислительного, а интеллектуального труда людей.

9.1. Первые ЦЭМ строились на радиолампах (содержали они их по несколько тысяч), занимали машины площадь целого зала и потребляли довольно много электроэнергии. Эти (ламповые) машины получили название машин первого поколения. Они делали несколько тысяч операций в секунду и обладали памятью на несколько тысяч слов (т. е. команд и чисел). С их появлением скорость вычислений возросла в большее число раз, чем скорость человека, который сначала шел пешком, а затем полетел на самолете. Строили первые машины для астрономических расчетов, но сразу же применили

для расчетов по атомной энергии, расчета ракет и для других отраслей науки и техники. У истоков советской электронной вычислительной техники стояли С. А. Лебедев, И. С. Брук и А. А. Ляпунов.

Программист в те времена писал свою программу в командах машины и отлаживал ее за пультом машины, которая (на время отладки) была полностью в его распоряжении. Он проводил за пультом час-два и за один-два раза получал результат. Но со стороны это выглядело расточительно - более 90% времени машина простаивала, а программист думал. И когда скорость машин возросла в 100 раз (а цена – в 10), пультовая отладка прекратилась.

9.2. Второе поколение ЦЭМ называют транзисторно-ферритовым, так как эти приборы заменили радиолампы. Транзисторы (твердые диоды и триоды) заменили лампы в процессорах (т. е. устройствах, выполняющих вычисления и управляющих работой машины), а ферритовые (намагничиваемые) колечки - в памяти. Скорость возросла до сотен тысяч операций в секунду, а память — до десятков тысяч слов. Присоединили к машинам и устройства для быстрой печати. В это же время от написания программ в командах машины перешли к записи программ на специальных языках программирования, промежуточных между человеческими языками и командами машин.

9.3. Третье поколение машин характеризуется появлением интегральных схем (вместо отдельных транзисторов), что привело к дальнейшему увеличению скорости (до миллиона операций в секунду) и памяти до сотен тысяч слов. Но еще существеннее было развитие внешних устройств. Появились магнитные диски, на которые можно было быстро помещать промежуточные результаты сотнями тысяч слов.

Изменилось программное обеспечение. Кроме трансляторов (переводивших программы с языков программирования в команды машины) появились и заняли центральное положение операционные системы — специальные программы, организующие всю работу отдельных узлов машины, вплоть до одновременной обработки нескольких программ.

Изменилась и отладка программ. Теперь программиста вообще перестали пускать в машинный зал. Он сдавал программу и получал *листинг* — листы бумаги, на которых программа печатала свои результаты, а опера-

вводная система и транслятор — сообщения об ошибках). Разобрав листинг, программист исправлял программу и снова отправлял ее на машину. Внешне это выглядело здорово. Машина работала не только без простоев, но и с тройной загрузкой: она одновременно одну программу вводила, по другой считала, а результаты третьей печатала. Число программистов, обслуживаемых одной машиной, стало исчисляться сотнями.

И все же пристальному взгляду картина представлялась не столь радужной. Для отыскания ошибок оказывалась нужна печать не тех результатов, которые были запланированы, а транслятор сообщал не столько об ошибках программиста, сколько об ошибках, которые возникли от этого. Число выходов на машину, нужных для отладки, возросло, а календарное время на отладку одной программы (от начала — до конца) выросло катастрофически. Короче говоря, если при пультовой отладке машина в основном простаивала, то в пакетном режиме она в основном работала впустую, выполняя расчеты, которые программист прекратил бы, увидев уже промежуточное. И цифровые электронные машины вышли на второе место в мире по производству бумажной макулатуры (после газет). Все же машинного времени в пакетном режиме стало идти на задачу меньше, чем при пультовой отладке, хотя программистского — больше.

9.4. Четвертое (нынешнее) поколение ЦЭМ началось с появлением больших интегральных схем (БИС), когда на одном кристалле размером 1 см^2 стали размещаться сотни тысяч электронных элементов, подобных радиолампе (диодов, триодов и т. п.). Скорость и объем памяти возросли в тысячу раз по сравнению с машинами первого поколения. Машины стали маленькими, экономичными и дешевыми. Появились мини-, микро- и персональные машины. Появились флоппи-диски — маленькие запоминающие магнитные диски, которые можно носить в кармане.

Программы стали промышленным продуктом, а расходы на программирование дошли до расходов на производство машин. Программисты вернулись к пультовой отладке в режиме диалога, но уже за дисплеем машины, объединяющем клавиатуру пишущей машинки с экраном телевизора. Маленькие машины встраиваются в свои дисплеи, а большие обслуживают сотни дисплеев.

В это же время появились карманные электронные калькуляторы, окончательно вытеснившие счеты, арифмо-

метры и клавишные машины. Цена электронного процессора в калькуляторе, часах или стиральной машине снизилась до нескольких рублей. Если бы последние 15 лет автомобильная промышленность развивалась также как электронная, то автомобиль «Волга» стоил бы теперь рубль, а одной заправки бензином хватало бы ему на весь срок эксплуатации.

10. *Основные этапы развития вычислительной техники.*

1. Изобретение десятичной системы.
2. Счеты.
3. Арифмометр.
4. Релейные и клавишные машины.
5. Электронные машины и их поколения.

5.1. Первое поколение — ламповые машины, программирование в командах машины и отладка за пультом. Скорость — тысячи действий в секунду, память — тысячи слов.

5.2. Второе поколение — машины на транзисторах, программирование на языках высокого уровня.

5.3. Третье поколение — интегральные схемы и диски, операционные системы и пакетный режим.

5.4. Четвертое поколение - большие интегральные схемы (БИС), мини- микро- и персональные машины, дисплеи, флоппи-диски, диалоговый режим. Скорость возросла в тысячу раз по сравнению с электронными машинами первого поколения.

Основные достижения вычислительной техники:

1) скорость вычислений современных ЭВМ в миллионы раз выше, чем у электромеханических; ЭВМ обладает памятью на миллионы и даже миллиарды знаков;

2) ЭВМ автоматически выполняет заданную ей программу вычислений;

3) ЭВМ по заданной программе может сама вырабатывать команды, которые будет выполнять;

4) ЭВМ стали дешевыми.

Перечисленные достижения качественно изменили как вычислительные методы, так и области применения вычислительной техники.

Глава 3

ЗАНИМАТЕЛЬНЫЕ ЗАДАЧИ КИБЕРНЕТИКИ

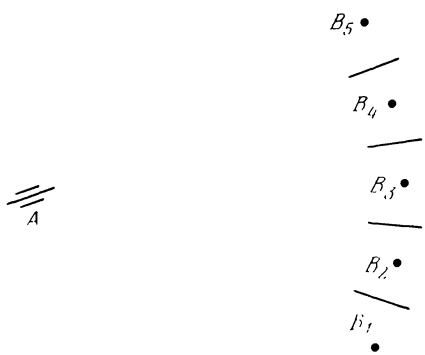
Это — лекции, прочитанные школьникам. Задачи, приведенные в них, известны, но наши решения отличаются от опубликованных.

§ 3.1. Артиллерист и пехотинец

Артиллерист ведет огонь по пехотинцу, который может укрываться в любом из пяти окопов B_1, B_2, B_3, B_4, B_5, расположенных, как показано на рис. 3.1. В каком окопе укрывается пехотинец, артиллеристу не видно. Артиллерист может стрелять в промежуток между любой парой соседних окопов, и пехотинец будет поражен, если окажется в одном из них.

Требуется определить, как вести себя пехотинцу и артиллеристу, если:

а) пехотинец хочет иметь максимальный шанс избежать поражения в наихудшем для него случае;



артиллерист хочет иметь максимальный шанс поразить пехотинца в наихудшем для артиллериста случае.

Это типичная задача оптимального поведения. Если, например, пехотинец решит укрыться в B_1, то (в худшем для пехотинца случае) артиллерист может решить выстрелить как раз между B_1 и B_2. Так же невыгодно и артиллеристу решать заранее, куда он будет стрелять. Поэтому надо решать задачу с помощью случайной стратегии, открытой Дж. Нейманом.

Есть в этой задаче и ловушка: может показаться, что окопы B_1, B_5 равноценны и безопаснее равноценных же окопов B_2, B_3, B_4, которые поражаются выстрелами по двум прицельным точкам. Несмотря на простоту задачи, Р. Айзекс нашел ее достаточно интересной, чтобы начать с нее курс «Оптимального поведения» для офицеров. Впрочем, решение у него сложнее, чем будет у нас.

1. Прежде всего, должно стать ясно, что пехотинцу можно не пользоваться окопом B_2. Действительно, всякий выстрел, поражающий B_1, поразит и B_2. Поэтому положение пехотинца не ухудшится, если он заменит укрытие в окопе B_2 на укрытие в B_1. Аналогично обстоит дело с окопом B_4.

2. Итак, остаются окопы B_1, B_3, B_5, в которых может укрыться пехотинец, а артиллерист может поразить одним выстрелом любой из этих окопов, но только один.

3. Равноценность окопов подсказывает такое решение. Пехотинец случайным и равновероятным образом выбирает один из трех окопов: B_1, B_3, B_5. Артиллерист тоже случайным и равновероятным образом выбирает одну из трех целей: между B_1 и B_2, между B_3 и B_4 или между B_4 и B_5 (вместо цели между B_3 и B_4 годится цель между B_2 и B_3). Чтобы сделать равновероятным выбор между тремя возможностями, можно, например, бросить игральный кубик: если выпадут 1 или 2 очка, то остановимся на первой возможности, если 3 или 4 очка, то — на второй, а если 5 или 6 очков, то — на третьей.

При таком решении вероятность, что пехотинец останется невредимым, равна 2/3, а вероятность, что артиллерист поразит пехотинца, равна 1/3.

4. Это — решение задачи. Действительно, при другом поведении пехотинца он попадет в одну из трех ситуаций:

$$B_1 \text{ или } B_2;\ B_3;\ B_4 \text{ или } B_5 \qquad (*)$$

с вероятностью выше 1/3. И в неудачном для пехотинца случае артиллерист может выбрать цель, при которой пехотинец будет поражен в этой ситуации. Аналогично, невыгодно в поставленных условиях отклониться от рекомендованной ему тактики и артиллеристу — иначе в одном из случаев (*) пехотинец будет поражаться с вероятностью менее 1/3 и может выбрать как раз этот вариант.

§ 3.2. Тройственная дуэль

Три человека — назовем их A, B и C — устроили дуэль по следующему правилу. Они бросают жребий, который равновероятно выдает им номера 1, 2 и 3. После этого они поочередно стреляют в порядке своих номеров: № 1, № 2, № 3, № 1, № 2, ...Каждый раз стреляющий выбирает, в кого он будет стрелять, а убитый, естественно, выбывает из очереди. Дуэль продолжается, пока не останется в живых только один из участников.

Требуется найти для каждого дуэлянта оптимальную тактику поведения и вероятность остаться в живых, если A убивает противника наверняка, B — в 80% случаев, а C — в 50%, и участникам дуэли это известно.

1. Сначала рассмотрим более простую дуэль двух противников, назовем их P и Q, которые попадают с вероятностями p и q соответственно. Пусть они тоже стреляют поочередно, причем первым стреляет P. Обозначим вероятность его выигрыша (в этих условиях) через x. Для вычисления x будем различать два несовместимых события: того, что P попадает первым выстрелом, и того, что и P и Q вначале промахнутся. Вероятность первого события равна p, а второго — равна $(1-p)(1-q)$. В первом случае вероятность выигрыша P равна 1, а во втором — снова равна x. Таким образом,

$$x = p \cdot 1 + (1-p)(1-q)x.$$

Откуда

$$x = \frac{p}{1-(1-p)(1-q)}.$$

Если первым стреляет Q, то вероятность y выигрыша P будет равна произведению вероятности того, что Q промахнется, на вероятность x того, что после этого выиграет P:

$$y = (1-q)x = \frac{(1-q)p}{1-(1-p)(1-q)}.$$

так что $q < x$, и поэтому пропускать свой выстрел при дуэли двух противников невыгодно.

2. Вернемся к тройственной дуэли A, B, C и объявим заранее оптимальную тактику. Пока есть три дуэлянта, A и B должны стрелять в сильнейшего противника (то есть друг в друга), а C должен стрелять в воздух. При этом вероятности выигрыша дуэли для A, B и C, как мы покажем, будут соответственно равны

$$0.3,\ 0.177\ldots,\ 0.522\ldots$$

Таким образом, лучшие шансы имеет худший стрелок (!).

3. Докажем, что при предложенной нами тактике вероятности выигрыша действительно будут указанными. Пока дуэлянтов трое, участник C стреляет в воздух. Поэтому события распадаются на две группы с вероятностями $p_1 = 1/2$ и $p_2 = 1/2$: стрельбу начинает A или B.

Если начинает A, то он убивает B и продолжает дуэль с C. Начинает ее C, и он либо попадает и выигрывает, либо промахивается (с вероятностью $p = 1/2$) и тогда наверняка выигрывает A.

Если начинает B, то либо он промахивается (с вероятностью $p_3 = 1/5$), либо попадает ($p_4 = 4/5$). В первом случае возникает уже рассмотренная ситуация с вероятностью выигрыша для A равной $p_5 = 1/2$. Во втором случае возникает дуэль между B и C, в которой B стреляет вторым. Дуэль двух лиц мы уже подробно рассмотрели и находим, что B имеет шансы на выигрыш с вероятностью p_7, равной ($p = 4/5$, $q = 1/2$):

$$p_7 = \frac{1 - q \cdot p}{1 - (1-p)(1-q)} = \frac{4}{9}.$$

Итак, вероятность выигрыша для A равна

$$P_A = p_1 p_3 + p_2 p_4 p_6 = 3/10.$$

Для B она будет равна

$$P_B = p_2 p_3 p_7 = 8/45.$$

Для C она равна тому, что осталось от единицы:

$$P_C = 1 - P_A - P_B = 47/90.$$

4. Остается показать, что предложенные тактики действительно оптимальны.

Для C это просто. Если он не пропустит свою очередь и попадет в A (в B стрелять еще хуже), то у него воз-

...кнет дуэль с B, в которой тот начинает. Вероятность выигрыша C в этой дуэли равна $p = 1/2$, $q = 4/5$:

$$u = \frac{1 - q}{1 - pq} = \frac{p}{1 - q} = \frac{1}{2}$$

что гораздо меньше, чем $1/2$ — вероятность выигрыша при выстрелах в воздух в худшем случае.

Для B это очевидно. Пропустив свою очередь на выстрел или выстрелив в C, он наверняка будет убит следующим выстрелом A.

Но верно это и для A. Если его очередь стрелять, то он уберет B и выиграет дуэль с C с вероятностью $1/2$. Если же он пропустит свою очередь, то снизит вероятность выигрыша до $1/10$ (с вероятностью $1/5$ промахнется B, и с вероятностью $1/2$ промахнется C).

5. И последнее: не выгоднее ли A и B заранее сговориться, что они будут сначала оба стрелять в C, а после устранения C тот из них, у кого оказался больший номер, сам застрелится. Да, выгоднее, и это очень частый эффект задач теории коллегиальных игр. В ней приходится предполагать, что участник нарушит договор (но не правила игры), когда ему это окажется выгодным.

§ 3.3. Справедливый дележ

Хорошо известен способ справедливого дележа золотого песка между двумя участниками. Способ состоит в том, что один участник делит песок (по своему усмотрению) на 2 кучки, а второй — выбирает себе любую из них. *Как справедливо разделить песок, если число участников равно n?*

1. Пояснение. В отличие от предыдущей задачи способ дележа должен остаться справедливым, даже если несколько участников сговорятся действовать так, чтобы увеличить свою долю за счет других участников.

2. Не торопитесь читать решение. Попытайтесь сделать задачу хотя бы для трех участников.

3. Решение. При $n = 1$ решение тривиально (не надо делить). Поэтому, решая задачу для $n > 1$ участников, можно предполагать, что известен способ справедливого деления песка между $n - 1$ участниками.

Если $n > 1$, то первые $n - 1$ участников (их можно взять какими угодно) делят весь песок справедливо между собою, что по предположению можно сделать.

После этого каждый из них делит свою долю по собственному разумению на n частей и предлагает взять последнему участнику любую из этих частей.

§ 3.4. Выбор шара

В урне находится n шаров. На каждом шаре написано число (все числа различные). Число n велико и известно заранее, а числа, написанные на шарах, любые (не обязательно от 1 до n) и заранее не известны.

Игра состоит в следующем. Играющий выбирает наудачу шар из урны, прочитывает на нем число и решает — берет он этот шар или нет.

Если не берет, то выбранный шар отбрасывается (не возвращается в урну), и игра продолжается. Играющий выбирает наудачу новый шар, читает на нем число и снова решает, брать этот шар или нет, и т. д.

Если играющий берет очередной шар, то игра прекращается, и игрок считается выигравшим в том и только в том случае, если на выбранном им шаре окажется самое большее среди чисел, написанных на всех n шарах, находившихся в урне с самого начала.

Требуется указать тактику играющего, при которой он будет иметь максимальные шансы на выигрыш.

1. Пусть, например, шаров $n=100$, и на первом выбранном шаре оказалось число 1000. Брать его или нет? Может быть, на следующем шаре окажется число 1 000 000, а может, 0.000001.

Если решить брать первый же вынутый шар, то вероятность выигрыша будет равна $1/n$. Но если десятый или последний, то она тоже будет равна $1/n$. Поэтому задача может показаться бессмысленной.

Но это рассуждение поверхностно. С каждым сброшенным шаром растут шансы потерять шар с максимальным номером, но растет и информация о шарах, которые были в урне. Если сброшен шар с числом 5, то в дальнейшем не имеет смысла брать шары с числами меньшими, чем 5.

Попытайтесь сами найти тактику, при которой вероятность выигрыша не стремится к нулю, когда число шаров n стремится к бесконечности (отыскание оптимальной тактики представляет математические трудности).

2. Вот тактика, которая дает вероятность выигрыша больше $1/4$. Пропустим первую половину шаров, запомнив максимальное встретившееся число. Затем возь-

ся первый же шар, на котором будет число больше запомненного. Эта тактика заведомо приведет к выигрышу, если шар с максимальным числом встретится во второй половине, а со вторым по величине числом в первой. Вероятность такого выпадения при большом n примерно равна $1/4$.

3. Поиск оптимальной тактики. Зафиксируем n и обозначим через $a_1 > a_2 > \ldots$ числа, написанные на шарах (в порядке убывания), и сами эти шары. Пусть мы пропускаем первые i шаров, запоминаем максимальное встретившееся число и берем первый же шар с числом, большим того, что запомнили. Вероятность P выигрыша складывается из вероятностей таких несовместимых событий:

— шар a_2 будет выбран; a_1 останется;

— шар a_3 будет выбран; a_1, a_2 останутся и первым из них встретится a_1;

.....

шар a_k будет выбран; a_1, a_2, ..., a_{k-1} останутся и первым из них встретится a_1;

.....

(Здесь «выбран» означает выбран среди первых i шаров, а «встретится» — встретится при дальнейшей выборке.)

Вероятность того, что a_k будет выбран, равна i/n (первых мест i, а всего мест n).

Вероятность того, что при этом a_1, a_2, ..., a_{k-1} останутся, равна ($j = n - i$):

$$\frac{j}{n-1}\,\frac{j-1}{n-2}\cdots\frac{j-(k-2)}{n-(k-1)}$$

(для a_1 подходящих мест j, а всего осталось мест $n-1$; после этого для a_2 подходящих мест $j-1$, а всего осталось мест $n-2$ и т. д.). Наконец, среди шаров a_1, a_2, ..., a_{k-1} любой заданный шар окажется первым с вероятностью $1/(k-1)$.

Следовательно, вероятность P_k того, что шар a_k будет выбран, а шары a_1, a_2, ..., a_{k-1} останутся и первым среди них встретится a_1, равна

$$P_k = \frac{i}{n}\,\frac{1}{k-1}\,\frac{j(j-1)\ldots(j-(k-2))}{(n-1)(n-2)\ldots(n-(k-1))},$$

что можно записать и так ($i/n = x$, $j/n = y$):

$$P_k = \frac{x}{k-1}\,\frac{y(y-1/n)\ldots(y-(k-2)/n)}{(1-1/n)\ldots(1-(k-1)/n)}.$$

Вся вероятность выигрыша P будет равна

$$P = P_2 + P_3 + \ldots + P_{i+1}.$$

При условии, что x постоянно *), получаем

$$P_k \to x\frac{y^{k-1}}{k-1} \text{ при } n \to \infty$$

и можно показать, что

$$P \to x\,|y + y^2/2 + y^3/3 + \ldots| \text{ при } n \to \infty.$$

Хорошо известен ряд

$$\ln(1+y) = y - y^2/2 + y^3/3 - \ldots,$$

значит,

$$\ln(1-y) = -|y + y^2/2 + \ldots|.$$

Следовательно, при $n \to \infty$ предел P (обозначим его через $P(x)$) равен

$$P(x) = -x \ln x.$$

Чтобы найти максимум $P(x)$, продифференцируем $P'(x)$ и приравняем производную нулю:

$$P'(x) = -(\ln x + x/x) = -(\ln x + 1) = 0.$$

Откуда $\ln x = -1$ и $x = 1/e$. При этом и

$$P(1/e) = 1/e = 0.368\ldots$$

Итак, при наилучшем выборе i

$$i/n \simeq 1/e \text{ и } P(i/n) \simeq 1/e$$

нужно пропустить $i \simeq n/e$ шаров, потом взять первый, больший выпавших, и вероятность выигрыша будет приблизительно $1/e$. Так, при $n = 100$ надо пропустить

$i = 37$ шаров и $P \simeq 0.368$.

§ 3.5. Удивительное рядом

Вы, разумеется, знаете об «орлянке» — игре, в которой выигравший определяется бросанием монеты: упадет монета на одну сторону — выиграл первый, упадет на другую — второй. Если монета «правильная» — каждый раз падает равновероятно на обе стороны —

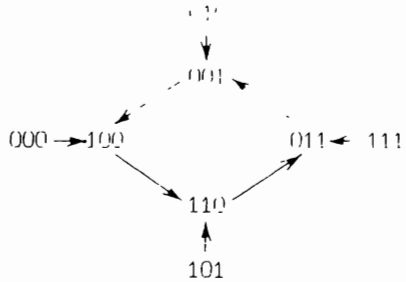
то безразлично какую сторону выбрать для себя выигрышной.

Существует, однако, вариант этой игры, обладающий удивительным свойством. Чтобы сократить изложение, напишем на одной стороне монеты 0, а на другой 1. Пусть теперь, и в этом состоит «тройная орлянка», первый называет последовательность из трех выпадений монеты, например 010, а второй какую-нибудь другую (тоже из трех) — например 001. Монету бросают несколько раз подряд и записывают выпадающие (т. е. оказавшиеся сверху) цифры. Как только последние три выпадания доставят комбинацию, выбранную одним из двух игроков,— этот игрок объявляется выигравшим. Если, например, выпадания будут такими 1011010, то выиграл первый (встретилась тройка 010). Если же выпадет 1011011001 — второй (встретилась тройка 001). На этом партия заканчивается. Если хотят продолжить игру, то начинают ее заново. Опять оба выбирают разные тройки и начинают новую серию бросаний и записей.

Поначалу может показаться, что и в тройной орлянке все комбинации дают одинаковые шансы на выигрыш, или (если уж они не равны), что первый имеет больше шансов на выигрыш, поскольку может выбрать более выигрышную комбинацию. Но на самом деле все обстоит наоборот. Для каждой комбинации (из трех цифр нулей и единиц) можно указать другую комбинацию (тоже из трех цифр нулей и единиц), имеющую больше шансов на выигрыш чем первая. И это несмотря на то, что всего комбинаций 8. Логического противоречия здесь, впрочем, нет, поскольку вторая наша комбинация имеет шансов на выигрыш больше не вообще, а в партии с определенной первой комбинацией. Если от каждой комбинации провести стрелку к той, которая лучше нее, то получится картинка, показанная слева на рис 3.2.

Если же для каждой комбинации указать лучшую чем она, то получится табличка, показанная справа на рис. 3.2. В скобках указано, во сколько раз шансы на выигрыш у правой комбинации выше, чем у левой (в партии, где загаданы две эти тройки). Как видите отношение шансов не опускается ниже, чем 2/1, в то время как на рулетке шансы крупье относятся к шансам игрока как 19/18.

Нетрудно заметить и простое правило для выгодного ответа. Если первый выбрал тройку a, b, c, то второму следует назвать 1 b, a, b.



(7)
(3)
(2)
(2)
(2)
(2)
(3)
(7)

Рис. 32

Любопытно, что шансы на выигрыш:

у 001 в 3 раза ниже, чем у 100,
у 100 в 2 раза ниже, чем у 110,
у 110 в 3 раза ниже, чем у 011,
у 011 в 2 раза ниже, чем у 001
и круг замкнулся!

Посмотрим, как найти эти числа. Пусть, для примера, первый назвал 001, а второй — 100. Обозначим через x_0, x_1, ..., x_7 вероятности выигрыша первого игрока в случаях, если первые три бросания монеты доставили, соответственно, комбинацию 000, 001, ..., 111.

Найдем x_0. Если в первых трех бросаниях выпало 000, то еще ни один не выиграл, монету кинут в четвертый раз и смогут возникнуть две последовательности

0000 или 0001.

В обоих случаях первый бросок не будет уже играть роли, а последние три цифры определяют ситуации

000 или 001.

Вероятность возникновения каждой из них равна 1/2, а вероятность выигрыша (если эта комбинация случилась) обозначена у нас через x_0 и x_1, следовательно

$$x_0 = x_0 \cdot 1/2 + x_1 \cdot 1/2.$$

Найдем x_1. Если в первых трех бросаниях выпало 001, то первый игрок выиграл. Значит

$$x_1 = 1.$$

Для x_2, по аналогии с x_0, получаем

$$x_2 = x_4 \cdot 1/2 + x_5 \cdot 1/2.$$

Продолжая таким образом, мы получим систему из 8 уравнений с 8 неизвестными.

$$\begin{array}{ll} 2x_0 = x_0 + x_1, & 2x_4 = 0, \\ 2x_1 = 2, & 2x_5 = x_2 + x_3, \\ 2x_2 = x_4 + x_5, & 2x_6 = x_4 + x_5, \\ 2x_3 = x_6 + x_7, & 2x_7 = x_6 + x_7. \end{array}$$

Система легко решается:

$$x_0 = x_1 = 1,\ x_2 = x_3 = x_4 = x_5 = x_6 = x_7 = 0.$$

Вероятность выпадения любой комбинации 000, 001, ..., 111 при первых трех бросаниях равна 1/8. Значит, если игроки выбрали комбинации 001 и 100, то вероятность выигрыша для первого равна

$$x_0 \cdot 1/8 + x_1 \cdot 1/8 + \ldots + x_7 \cdot 1/8 = 1/4,$$

а для второго —

$$1 - 1/4 = 3/4.$$

Отношение вероятностей их выигрышей —

$$(3/4):(1/4) = 3,$$

как мы и сообщали.

Составьте таблицу вероятностей выигрышей для всевозможных пар выбранных троек. Учитывая симметрии, достаточно рассмотреть 12 пар. Вычисления можно выполнить так же, как мы это сделали для пары 001, 100. Для отдельных пар вычисления можно сократить. Мы умышленно пользовались общим способом, пригодным всегда. Если вы это проделаете, то убедитесь, что мы указали для каждой тройки наиболее выгодный ответ.

Глава 4

ВЫЧИСЛИТЕЛЬНЫЕ ЗАДАЧИ

§ 4.1. Вычисление корня функции

Излагаемые здесь методы являются одновременно универсальными и устойчивыми. Их недостаток — в большом числе действий, точнее, в вычислении большого количества значений функции.

0. Постановка задачи. Пусть задана функция f и отрезок $|a, b|$, на концах которого она принимает значения разных знаков. Требуется найти точку $x \in |a, b|$, в которой $f(x)=0$.

Мы рассмотрим несколько методов решения этой задачи и начнем с самого простого.

1. Метод деления отрезка пополам. Разделим отрезок $|a, b|$ пополам и возьмем точку x в его середине, т. е.

$$x=(a+b)/2.$$

Если $f(x)=0$, то ответ найден. В противном случае в качестве нового отрезка $|a, b|$ выберем тот из отрезков

$$|a, x| \text{ или } |x, b|,$$

на концах которого функция f принимает значения разных знаков. И с новым отрезком $|a, b|$ поступим так же, как с предыдущим.

В результате итераций этого процесса мы либо наткнемся на точку x, где $f(x)=0$, либо сузим отрезок $|a, b|$ до минимально возможного размера (с точностью до вычислений машины) и тогда, в качестве ответа, будет годиться любая из точек a или b.

Замечание. Изложенный метод настолько общ, что не предполагает даже непрерывности функции f (не

говоря уже о ее дифференцируемости). Даст он, правда, в этом случае не корень функции, а точку x, в любой окрестности которой функция меняет знак.

1.1. Зачастую предложенный метод оказывается недостаточно определен для программирования, т. е. по существу не является алгоритмом. Выясняется это при попытке написать программу, ибо алгоритм, строго говоря, это и есть программа. В данном случае нам нужно рассмотреть несколько дополнительных вопросов и дать на них ясные ответы. Вот эти вопросы.

(1) Что делать, если на концах первоначально заданного отрезка $|a, b|$, значения функции f окажутся одного знака?

Поскольку нам гарантируют, что значения на концах имеют разные знаки, то мы будем считать, что одинаковыми они получились из-за приближенности вычислений, и примем в качестве корня тот из концов отрезка $|a, b|$, в котором функция f меньше по модулю.

если $|f(a)| < |f(b)|$, то $x=a$ иначе $x=b$.

Это сделает наш алгоритм *устойчивым* (т. е. безотказным в практически важной ситуации приближенных вычислений). Зато мы будем «прозевывать» некоторые явные ошибки задания.

(2) Когда кончать измельчение отрезка $|a, b|$?

Здесь универсальный ответ ясен — когда точка $x=(a+b)/2$, вычисленная с машинной точностью, перестанет попадать в интервал

$$a<x<b.$$

Недостаток такого решения тоже ясен — оно ведет к большому числу итераций. Если длина отрезка $|a, b|$ порядка единицы, а вычисления идут с 16 десятичными знаками (50 двоичных), то нам может потребоваться до 50 итераций (и даже больше, если искомый корень близок к нулю).

1.2. Теперь можно написать программу

```
10 input "a=,b=": a,b
20 u = FNA(a): v = FNA(b)
30 if u*v<0 goto 60
40 if abs(u)<abs(v) then x=a:y=u
                    else x=b:y=v
50 print x,y: goto 110
60 x = (a+b)/2: y = FNA(x)
```

```
70 [illegible] n [illegible] ;
80 if (x-a)*(x-b) [illegible] goto 110
90 if [illegible]*y>0 then b [illegible] =y goto 60
100 if u*y<0 then a=x: u=y: goto 60
110 end
05 DEF FNA(x) = x*x-0.25
```

а числа a и b взять соответственно 0.1 и 1.

1.3. Функцию FNA и числа a, b можно, разумеется, задать и иными. Например:

$FNA(x)=\cos(x)-x,\quad a=0.1,\quad b=1,$
$FNA(x)=\sin(x),\quad a=3,\quad b=4,$
$FNA(x)=x*x-100,\quad a=100,\quad b=0,$
$FNA(x)=x*x-0.01,\quad a=0,\quad b=1.$

2. Метод хорд. Обозначим через u и v значения функции f в концах отрезка $[a, b]$, т. е. положим

$$u=f(a)\quad\text{и}\quad v=f(b).$$

Естественно попытаться взять точку дробления x отрезка $[a, b]$ поближе к искомому корню функции f. Но в первом приближении гладкая функция линейна. Поэтому, в рассматриваемом методе, точка x берется на пересечении прямой $a, u - b, v$ с осью Ox (см. рис. 4.1). Из подобия треугольников получаем

$$-u/(x-a)=v/(b-x)$$

и, значит,

$$x=(a*v-b*u)/(v-u)$$

Далее следует поступать так же, как в предыдущем методе (деления отрезка $[a, b]$ пополам).

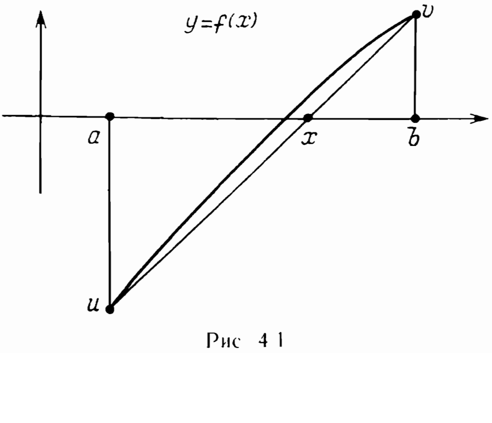


Рис. 4.1

2.1. Таким образом, единственное изменение предыдущей программы будет в строке 60 и она примет вид

60 x = a*(c - b*(c - a)/(c - a))

Разность $(c-a)$ в знаменателе не сулит затруднений, так как числа a и c разных знаков.

3. Смешанные методы. Метод хорд практически приводит к тому, что начиная с некоторого момента движется (изменяется) лишь один конец интервала (все время левый или все время правый), а другой — остается неподвижен. Это хорошо видно на рис. 4.2.

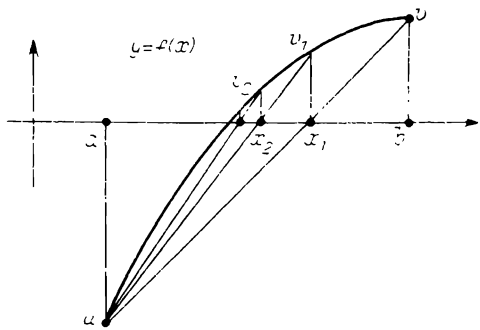


Рис. 4.2

3.1. Поэтому естественно испытать алгоритм, который будет брать точку дробления x попеременно, то в середине отрезка (как в методе 1), то на пересечении с хордой (как в методе 2).

Можно попробовать интерполяцию более высокого порядка и вместо хорд проводить параболы, т. е. вместо хорды, проходящей через две последние вычисленные точки, проводить параболу через три последние вычисленные точки. Читателю предоставляется самому испытать такого рода ускорения.

4. Добавление. Значительного ускорения можно достигнуть, если потребовать, чтобы была задана точность $e>0$, с которой ищется корень и значение функции. Тогда вычисления можно будет прекратить, как только выполнится условие

$$|f(x)|+|b-a|<e.$$

Универсальности метода это повредит, но время на вычисления сильно уменьшит. Составьте такие программы самостоятельно.

§ 4.2. Метод Лобачевского

Создатель неевклидовой геометрии Н. И. Лобачевский придумал и метод приближенного решения алгебраических уравнений (с действительными коэффициентами)

$$a_0+a_1x+a_2x^2+\ldots+a_nx^n=0. \qquad (1)$$

Метод дает не сам корень уравнения, а его абсолютное значение, так что потом (подстановкой) следует определить знак корня. Точнее — метод Лобачевского дает абсолютное значение корня, имеющего максимальный модуль. Единственное ограничение на применение метода — корень, максимальный по модулю, должен быть единственным. Если, например, он комплексный, то их уже два (то есть еще и комплексно-сопряженный), и метод неприменим. В конце мы рассмотрим модификацию метода, приводящую к отысканию минимального (по модулю) корня, или корня, находящегося в окрестности заданного значения.

Идея Лобачевского в следующем. Заменим заданное уравнение (1) другим, корни которого равны квадратам корней заданного, и повторим этот прием многократно. От этого корни уравнения «возведутся в квадрат», потом в четвертую степень, и т. д. В результате максимальный по модулю корень станет играть подавляющую роль перед остальными и его легко будет вычислить (см. далее) по коэффициентам полученного уравнения. Если теперь извлечь из ответа корень нужной степени, то получится (приближенно) абсолютное значение корня исходного уравнения.

Следует заметить, что приближения, получаемые по методу Лобачевского, исключительно быстро сходятся к точному решению.

1. Итак, Лобачевскому нужно было найти многочлен, корни которого равны квадратам корней исходного многочлена, и найти его нужно было, не вычисляя (!) корней исходного многочлена. Посмотрим, как он это сделал.

Будем считать, что нам задан многочлен

$$P(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n. \qquad (2)$$

По теореме Безу

$$P(x)=a_n(x-x_1)(x-x_2)\ldots(x-x_n),$$

где $x_1, x_2, \ldots, x_n$ — корни уравнения $P(x)=0$. Возьмем многочлен $P(-x)$. Его можно записать в виде

$$P(-x)=a_0-a_1x+a_2x^2-\ldots+(-1)^na_nx^n,$$

или

$$P(-x)=a_n(-x-x_1)\ldots(-x-x_n),$$

или даже так:

$$P(-x)=(-1)^na_n(x+x_1)\ldots(x+x_n).$$

Рассмотрим произведение многочленов

$$P(x)\,P(-x)=(-1)^na_n^2(x^2-x_1^2)(x^2-x_2^2)\ldots(x^2-x_n^2).$$

Заметим, что оно не содержит нечетных степеней x. Если в этом произведении заменить x^2 на t, то получится нужный нам многочлен

$$Q(t)=(-1)^na_n^2(t-x_1^2)(t-x_2^2)\ldots(t-x_n^2),$$

корни которого равны квадратам корней заданного многочлена $P(x)$. Для вычисления коэффициентов c_k многочлена $Q(t)$

$$Q(t)=c_0+c_1t+c_2t^2+\ldots+c_nt^n \qquad (3)$$

можно умножить $P(x)$ на $P(-x)$, привести подобные члены при одинаковых степенях x (нечетные степени при этом сократятся) и заменить x^2, x^4, ... на t, t^2, ...

Вот значения коэффициентов c_k для случаев $n=3$ и $n=4$:

$$\begin{aligned}
c_0&=a_0a_0,\\
c_1&=a_0a_2-a_1a_1+a_2a_0,\\
c_2&=\quad -a_1a_3+a_2a_2-a_3a_1,\\
c_3&=\quad\qquad -a_3a_3;
\end{aligned}$$

$$\begin{aligned}
c_0&=a_0a_0,\\
c_1&=a_0a_2-a_1a_1+a_2a_0,\\
c_2&=a_0a_4-a_1a_3+a_2a_2-a_3a_1+a_4a_0,\\
c_3&=\quad\qquad +a_2a_4-a_3a_3+a_4a_2,\\
c_4&=\quad\qquad\qquad\qquad +a_4a_4.
\end{aligned}$$

Коэффициенты c_k несложно получить программой. Для этого нужно сначала обнулить все переменные c_k, а затем умножить каждый коэффициент a_i многочлена $P(x)$ на каждый коэффициент $(-1)^j a_j$ многочлена $P(-x)$. Если $i+j$ нечетно, то результат следует опустить (он все равно сократится). Если же $i+j=2k$, то его следует занести (добавить) в c_k:

$$c_k=c_k+a_ia_j(-1)^j.$$

Стоит учесть, что младший коэффициент $c_0 = a_0 a_0 \ldots$, а затем знаки перед добавляемыми членами будут чередоваться. Тогда программа вычисления c примет вид

```
10 z = 1
20 for i = 0 to n
30   for j = 0 to n
40     k = (i+j)\2
50     if k+k <> i+j goto 70
60     c(k) = c(k) + a(i)*a(j)*z
70 next j: z=-z: next i
```

(программу легко ускорить, но мы сейчас не будем этого делать). После вычисления коэффициентов c_k их можно будет перенести в коэффициенты a_i, вычислить новые c_i и т. д., пока максимальный по модулю корень не станет превалирующим.

Чтобы найти значение этого корня вернемся к формулам

$$Q(t) = c_0 + c_1 t + \ldots + c_{n-1} t^{n-1} + c_n t^n,$$
$$Q(t) = c_n (t - t_1)(t - t_2) \ldots (t - t_n).$$

Если в нижней строчке раскрыть скобки и привести подобные члены, то должна получиться верхняя строчка. Следовательно,

$$c_{n-1} = -(t_1 + t_2 + \ldots + t_n) c_n,$$

или

$$t_1 = -(c_{n-1}/c_n)/(1 + t_2/t_1 + \ldots + t_n/t_1).$$

И если корень t_1 по модулю значительно превосходит остальные, то сумма в скобках близка к 1 и

$$t_1 \simeq -c_{n-1}/c_n;$$

а для корня x_1 исходного уравнения будет

$$t_1 = x_1 \wedge (2 \wedge p),$$
$$|x_1| = t_1 \wedge r, \text{ где } r = 1/2 \wedge p = (1/2) \wedge p,$$

коль скоро многочлен $Q(t)$ получен в результате p итераций (повторений) перехода от многочлена P к многочлену Q и замены корней их квадратами. Приближенные значения корня

$$|x_1| \simeq (-c_{n-1}/c_n) \wedge r$$

удобно выводить на экран на каждой итерации и следить за тем, как они устанавливаются. Именно такой метод,

уравнения

$$a(0)+a(1)x+\ldots+a(n)x^n=0$$

```
10 input "n"; n
20 dim a(n),b(n),c(n)
30 for i = 0 to n
40   print "a(";i;")=";: input a(i)
50 next i
60 for i = 0 to n: b(i)=a(i)/a(n)
70 next i: r=1/2
80 z=1
90 for i = 0 to n
100   for j = 0 to n
110     k = (i+j)\2
120     if k+k <> i+j goto 140
130     c(k) = c(k) + b(i)*b(j)*z
140 next j: z=-z: next i
150 if z*c(n-1)<>0 then print "-"
    else print (z*c(n-1))^r
160 for i = 0 to n: b(i)=c(i): c(i)=0
170 next i: r = r/2
180 goto 80
```

Мы поделили все коэффициенты a_i исходного уравнения на a_n и в дальнейшем работаем с коэффициентами $b_i=a_i/a_n$ и c_k; при этом $|b_n|=|c_n|=1$.

2. Примеры для вычисления корней степенных многочленов:

1) $24-50x+35x^2-10x^3+x^4=0$, корни: 1, 2, 3, 4;

2) $-63.84+124.48x-73.36x^2-4.88x^3+21.61x^4-8.2x^5+x^6=0$, корни: 2.1, 2, 2, 2, 2, -1.9;

3) $90+19x+x^2=0$, корни: -9, -10;

4) $0.009-0.19x+x^2=0$, корни: 0.1, 0.09;

5) $-0.33264+0.4278x+1.129x^2-1.35x^3-0.9x^4+x^5=0$, корни: 1.1, -0.9, 0.8, -0.7, 0.6;

6) $0.76-1.37x-1.3x^2+2x^3=0$, корни: 0.95, -0.8, 0.5.

3. Дополнительные вопросы.

(1) Как методом Лобачевского найти минимальный (по модулю) корень?

(2) Как методом Лобачевского найти корень, находящийся вблизи заданного значения?

(3) Ускорьте вычисления. В операторах 100–140 выносите из циклов по i умножение на z, сделайте цикл по j с шагом 2.

Ответы.

(1) Воспользоваться подстановкой $x=1/y$ и умножить все члены на y^n.

(2) Если задано значение $x0$, то воспользоваться подстановками $x=y+x0$ и $y=1/z$.

(3) Соответствующие изменения сделаны в программе, которая приводится далее.

4. Улучшение вычислительного алгоритма. Этот раздел предназначен только для тех школьников, которые специально заинтересуются методом Лобачевского. Остальным достаточно знать о возможности такого улучшения.

Вычисления на компьютере отличаются от ручных еще и тем, что слишком маленькие числа заменяются нулем, а слишком большие прерывают работу программы. Из-за этого вычисления по нашей программе, как правило, плохо кончаются. Если $|x_i|<0{,}1$, то уже после 7 итераций $c(n-1)=0$ и получить значение x_i невозможно. Если же $|x_i|>10$, то после 7 итераций $c(n-1)>10^{100}$ и программа прерывается.

Попытаемся улучшить алгоритм, сделав искомый корень близким к единице, с помощью подстановки $x=g*\tilde{x}$, где g — приближенное значение этого корня.

Если на очередной p-й итерации будет $z*c(n-1)\leqslant 0$, то не станем вмешиваться в схему вычислений. Но если $z*c(n-1)>0$, то возьмем приближенное значение g искомого корня $|x_1|$:

$$g=d^r, \text{ где } d=c(n-1)*z \text{ и } r=1/2^p,$$

и сделаем (мысленно) в исходном уравнении (1) подстановку

$$x=g*\tilde{x}.$$

При этом и в очередном уравнении (3) для $Q(t)$ произойдет подстановка

$$t=x^{\wedge}(2^p)=g^{\wedge}(2^p)*\tilde{x}^{\wedge}(2^p)=d*\tilde{t}$$

и его коэффициенты приобретут новые значения

$$c(k)=c(k)/d^{\wedge}(n-k).$$

Оно будет сходиться к модулю старшего корня исходного уравнения. Здесь g_1, g_2, ... обозначают значения чисел $g = d^r$ на очередной итерации, где производилась замена x на x^r.

Улучшенная программа метода Лобачевского для вычисления максимального по модулю корня уравнения

$$a(0) + a(1)x + a(2)x^2 + \ldots + a(n)x^n = 0$$

```
10 input "n";n
20 dim a(n),b(n),c(n)
30 for i = 0 to n
40   print "a(";i;")=";: input a(i)
50 next i
60 for i = 0 to n: b(i) = a(i)/a(n)
70 next i: print -b(n-1)
80 r=1/2: g=1
90 z = 1
100 for i = 0 to n
110   bi = z*b(i): k=(i+1)\2
120   for j = i mod 2 to n step 2
130     c(k) = c(k) + bi*b(j): k=k+1
140   next j: z=-z
150 next i
160 d=z*c(n-1): if d<0 goto 200
170 for i = 0 to n
180   b(i)=c(i): c(i)=0
190 next i: print "-": goto 240
200 g=g*d^r: print g
210 for i = 0 to n
220   b(i)=c(i)/d^(n-1): c(i)=0
230 next i: b(n-1)=z: b(n)=-z
240 r = r/2: goto 90
```

5. Добавление. Научно-общественная биография гениального математика Н. И. Лобачевского была трагична. Человек прогрессивных взглядов и независи-

мого поведения *) создал неевклидову геометрию. Передовая научная общественность отвергла ее, сочтя порождением идеализма, а консервативные правительственные чиновники поддержали автора. Создалась парадоксальная ситуация, в которой передовые ученые и общественные деятели писали на Лобачевского рецензии, по сей день остающиеся шедеврами издевательства **), а правительство назначило его ректором Казанского университета и наградило высшим орденом Российской империи.

Началось подспудное соревнование — «кто быстрее разберется». Быстрее «разобрались» чиновники, и Лобачевский умер в изоляции, отстраненный от дел, ослепший, не дожив 15 лет до всеобщего признания. Избранный членом-корреспондентом Геттингенской академии наук, он не удостоился в России даже степени доктора.

§ 4.3. Метод Герона

1. Алгоритм Герона для вычисления квадратного корня. В формуле Герона для площади треугольника встречается квадратный корень. Наверное, поэтому Герон придумал алгоритм извлечения квадратного корня. Этот алгоритм носит имя Герона. Алгоритм чрезвычайно эффективен, прост и употребляется на всех компьютерах.

Алгоритм Герона. Пусть требуется вычислить корень

$$x = \sqrt{a}, \quad \text{где } a > 0 \text{ и } x > 0.$$

---

*) За время обучения в университете Лобачевский получил 33 выговора. Приведем причины некоторых. Лобачевский изготовил ракету и подговорил одного гимназиста запустить ее на университетском дворе, в результате чего сгорела библиотека. Он перепрыгнул на пари через одного тучного профессора. Покатался в городском парке на корове и т. п.

**) Рецензент нашел геометрию Лобачевского «плодом воображения живого и уродливого» и «решил выяснить, с какой целью г. Лобачевский, ординарный профессор, мог напечатать сочинение, не делающее чести и последнему приходскому учителю». Отсюда рецензент заключает, что «автор решил написать сатиру на геометрию», а «безрассудное желание открывать новое, при талантах, едва достаточных для надлежащего усвоения старого, суть те качества, которые автор намерен был изобразить и изобразил как нельзя лучше». Далее рецензент пишет «хвала г. Лобачевскому, принявшему на себя труд обличить наглость и бесстыдство ложных изобретателей и простодушное невежество их почитателей». Известно отрицательное отношение к геометрии Лобачевского со стороны М. В. Остроградского и Н. Г. Чернышевского.

$x_0 > 0$.

Если уже имеется некоторое приближенное значение корня x_n, то следующее приближение вычисляется по формуле

$$x_{n+1} = (a/x_n + x_n)/2.$$

2. Пример. Для $\sqrt{2} = 1.41421\ 35623\ldots$ возьмем $x_0 = 1.5$ и последовательно получим

$$x_1 = 1.417\ldots,$$
$$x_2 = 1.41421\ 6\ldots,$$
$$x_3 = 1.41421\ 35624\ldots$$

3. Оценка погрешности. Положим

$$x = \sqrt{a} \text{ и } x_n = x \cdot (1 + e_n),$$

где e_n — относительная погрешность приближения x_n (со знаком). Выражая e_{n+1} через x_{n+1}, затем x_{n+1} через x_n и x_n через e_n, получаем

$$e_{n+1} = \frac{e_n^2/2}{1+e_n} \text{ или } e_{n+1} = \frac{e_n^2/2}{1+1/e_n} \qquad (*)$$

(проверьте это). Если исходное значение x_0 положительно

$$x_0 > 0, \text{ то } e_0 > -1.$$

И тогда по формулам (*) будет $e_1 \geqslant 0$, и все последующие $e_n \geqslant 0$. Кроме того, из формул (*) получаем

$$0 \leqslant e_{n+1} \leqslant e_n/2,$$
$$0 \leqslant e_{n+1} \leqslant e_n^2$$

при всех $n = 1, 2\ldots$

Следовательно, $e_n \to 0$ и $x_n = x \cdot (1 + e_n) \to x$ при $n \to \infty$. Сходимость очень быстрая. Если даже первое приближение выбрано плохо, то вначале ошибка на каждом шаге (начиная со второго) будет убывать вдвое:

$$|e_{n+1}| \leqslant |e_n|/2,$$

а после того, как станет $e_n < 1/10$, число верных десятичных знаков будет удваиваться на каждой итерации, ибо

$$|e_{n+1}| \leqslant |e_n^2|.$$

Отметим, что $e_1 \geqslant e_2 \geqslant \ldots$, значит, $x_1 \geqslant x_2 \geqslant \ldots$ и последовательность x_n монотонно не возрастает (начиная с x_1).

Замечание. Замечательной особенностью алгоритма Герона, кроме быстрой сходимости итераций, является еще и то, что он не накапливает погрешностей вычислений. Более того - даже умышленно внесенная ошибка будет устранена в процессе итераций, так как сыграет роль смены начального значения x_0.

4. Уточнение алгоритма. Прежде чем писать программу для извлечения корня ($x=\sqrt{a}$), мы должны решить четыре вопроса:

(1) Что делать, если окажется, что $a<0$?

(2) Как выбрать начальное значение x_0?

(3) По какой формуле вычислять приближения x_n, ?

(4) Когда прекратить итерации?

Займемся решением этих вопросов.

(1) Многие следуют принципу Р. Гутера «Лучше лишний стоп, чем лишний ляп», -- и в случае $a<0$ останавливают программу. Но мы придаем бо́льшее значение *устойчивости* программы, т. е. тому, чтобы она безотказно работала при любых возможных ситуациях. Между тем величина a может оказаться отрицательной из-за того, что была мала, а вычислялась приближенно (в каких-то других частях программы).

Поэтому мы принимаем

$x=0$ при $a\leqslant 0$.

(2) Исходное значение мы выберем таким:

$x_0=(a+1)/2$.

Оно получается, если сперва взять $x_0=1$, а потом принять в качестве x_0 получившееся значение $x_0=(a/1+1)/2$. Благодаря такому выбору у нас с самого начала будет $x_0^2\geqslant a$. В практических программах значение x_0 выбирают сложнее. Это ускоряет вычисления, но усложняет программу. Мы этим заниматься сейчас не будем.

(3) Итерации будем вести по алгоритму Герона:

$x_{n+1}=(a/x_n+x_n)/2$.

(4) Пользуясь монотонностью последовательности $x_1\geqslant x_2\geqslant \ldots$, мы будем повторять итерации, пока (с ма-

шинной точностью!) приближения будут убывать. В большинстве «практических» программ итерации ведут до достижения заданной точности или повторяют заданное число раз. Это экономит несколько итераций.

5. Алгоритм Герона реализуется программой

```
10 rem  x = корень квадратный из a
20 input "a="; a
30 if a<=0 then print 0,goto 90
40 x=(a+1)/2
50 y=x
60   x=(a/x+x)/2
70   print x
80 if x<y goto 50
90 end
```

6. Контрольные вопросы.

(1) Можно ли команду 40 заменить более простой:

40 $x=1$

(2) Можно ли заменить команду 80 такой:

80 if $x<>y$ goto 50

то есть прекращать итерации при равенстве двух соседних приближений?

(3) Стоит ли переписать команду 30 так:

30 if $a<=0$ then *print* 0 : end

Ответы. Нет!

При указанной замене команды 40 программа сразу же прекратит работу в случае $a=2$ и даст неверный ответ $x=1.5$.

Указанная замена команды 80 рискованна. Машина считает приближенно. Это может привести к таким колебаниям приближенных значений x_n вокруг искомого корня, при которых два соседних значения никогда не будут совпадать. Например, так: $1+e$, $1-e$, $1+e$, $1-e$, ..., где e — мало.

Указанная замена команды 30 противоречит одному из принципов структурированного программирования — «у программы должен быть один вход и один выход».

7. Геометрический смысл алгоритма Герона. Начертим на плоскости XOY параболу $y=x^2$ и отметим прямую $y=a$. Если в точке (x_n, x_n^2) провести

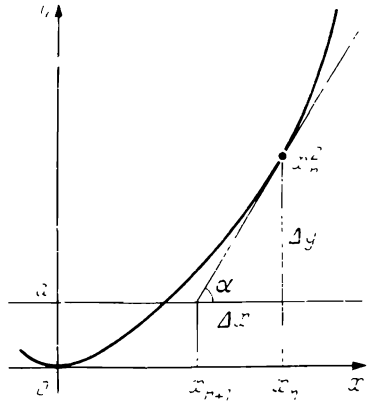


Рис. 4.3

касательную, то ее пересечение с прямой $y=a$ даст значение x_{n+1} (см. рис. 4.3). Действительно, обозначая

$$\Delta x = x_n - x_{n+1} \text{ и } \Delta y = x_n^2 - a,$$

получаем

$$\Delta y / \Delta x = \operatorname{tg} \alpha.$$

По формулам дифференциального исчисления $\operatorname{tg} \alpha = dy/dx = 2x_n$. Поэтому

$$(x_n^2 - a)/(x_n - x_{n+1}) = 2x_n,$$

откуда и получается

$$x_{n+1} = (a/x_n + x_n)/2.$$

Таким образом, алгоритм Герона является частным случаем метода касательных Ньютона.

8. Корень k-й степени. Зная, что производная функции $y=x^k$ равна $y'=kx^{k-1}$, выведите формулу для вычисления последовательных приближений корня k-й степени

$$x = \sqrt[k]{a}.$$

Ответ. Формула имеет вид

$$x_{n+1} = (a/x_n^{k-1} + (k-1)x_n)/k.$$

Нулевое приближение можно взять таким (почему?):

$$x_0 = (a+k-1)/k.$$

9. Упражнение. Составьте программу для вычисления $\sqrt[k]{a}$ и выполните ее для нескольких k и a.

СПИСОК ЛИТЕРАТУРЫ

Брудно А. Л. Введение в программирование. М.: Наука, 1965.

Брудно А. Л., Ландау И. Я. Неприкасаемый король // Шахматы. Рига, 1969. № 19.

Гарднер М. Математические головоломки. М.: Мир, 1971.

Гарднер М. Математические досуги. М.: Мир, 1972.

Гарднер М. Математические новеллы. М.: Мир, 1974.

Дал У., Дейкстра Э., Хоор К. Структурное программирование. М.: Мир, 1975.

Дейкстра Э. Дисциплина программирования. М.: Мир, 197

Дрейфус М., Гангloф К. Практика программирования на фортране: Упражнения с комментариями. М.: Мир, 197

Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си: Задачи по языку Си. М.: Финансы и статистика, 198

Лин К., Ван В. Комбинаторика для программистов. М.: Ми 1988.

Уэзерелл Ч. Этюды для программистов. М.: Мир, 1982.

ОГЛАВЛЕНИЕ