В. М. Котов И. А. Волков А. И. Лапо

Информатика

Методы алгоритмизации

Учебное пособие для 8—9 классов
общеобразовательной школы
с углубленным изучением информатики
с русским языком обучения

Допущено
Министерством образования
Республики Беларусь

МИНСК «НАРОДНАЯ АСВЕТА» 2000

УДК [002.6+681.3](075.3=82)
ББК 32.81я721.6
К95

Рецензенты:

канд. пед. наук, доцент кафедры прикладной математики и информатики БГПУ им. М. Танка *Пупцев А. Е.*; учитель информатики высшей категории СШ № 2 г. Минска *Савченкова Н. И.*

ISBN 985-12-0259-2

ОТ АВТОРОВ

Зачастую решение задачи по информатике протекает следующим образом: вы смотрите на постановку задачи и, используя приобретенные навыки и известные вам методы, выдаете решение. При этом обычно неявно считается, что «первый взгляд — наиболее верный», и если решение получено, то на этом «акт творения» программы завершен. Но насколько это «творение» является законченным и эффективным? Обычно при анализе такого решения оказывается, что первый взгляд может всего не увидеть, и что если после него еще чуть-чуть подумать, то результат получается намного лучше (что верно не только в информатике).

Учебное пособие предназначено для того, чтобы помочь вам выбрать эффективное решение для поставленной задачи и рассказать о стандартных подходах к решению задач по информатике.

Некоторые из рассматриваемых в пособии задач вам уже встречались в курсе математики. Иногда для этих задач предлагаются отличные от уже известных методы решения. Эти методы ориентированы в первую очередь на применение компьютера, позволяющего за небольшое время выполнить большие объемы вычислений.

Книга состоит из 5 глав, в которых приводятся сведения из таких областей математики, как геометрия, арифметика, комбинаторика и теория алгоритмов. Главы содержат теоретический материал, задачи для повторения, задачи повышенной сложности и задачи для самостоятельного решения.

Теоретический материал направлен на то, чтобы дать представление об общих подходах и наиболее распространенных и эффективных методах решения задач. Фрагменты алгоритмов приводятся на алгоритмическом языке, принятом в базовом курсе. Для задач повышенной сложности приводятся указания по их решению. В конце книги даны Приложения, содержащие алгоритмы на языке Паскаль. Теоретический материал, предназначенный для факультативных занятий, обозначен значком *.

При написании была использована отечественная и зарубежная литература по теории алгоритмов.

Глава 1. УРАВНЕНИЕ ПРЯМОЙ

Геометрия развивается по многим направлениям. Возникновение компьютеров привело к появлению такой области математики, как вычислительная геометрия. При создании современных приложений часто требуется разработка эффективных алгоритмов для определения взаиморасположения различных объектов на плоскости, вычисления расстояний между ними, вычисления площадей фигур и др.

В данной главе излагается материал, частично известный вам из курса математики. Мы рассмотрим методы решения геометрических задач, которые эффективно реализуются с помощью компьютера, что позволит вам по другому взглянуть на вопросы, изучаемые в рамках школьного курса геометрии. Для этого придется воспользоваться аналитическим представлением геометрических объектов.

§ 1. ПРЯМЫЕ И ОТРЕЗКИ НА ПЛОСКОСТИ

1.1. Формы записи уравнения прямой

В задачах часто приходится задавать на плоскости различные геометрические объекты. Простейшими геометрическими фигурами на плоскости являются точка и прямая. Точка задается указанием координат, например $A(15;\ -5)$, $B(x_1;\ y_1)$. Прямую можно задавать с помощью уравнения прямой. Существуют различные формы записи уравнения прямой. Выбор какой-то конкретной зависит от исходных данных, задающих прямую на плоскости. (Могут быть заданы координаты двух

точек, через которые проводится прямая, или коэффициенты при неизвестных в линейном уравнении.)

В декартовых координатах каждая прямая определяется уравнением первой степени. Уравнение вида

$$Ax+By+C=0$$

называется ***общим уравнением прямой.***

Если в общем уравнении прямой коэффициент при y не равен нулю, то уравнение можно решить относительно y:

$$y=-\frac{A}{B}x-\frac{C}{B}.$$

Обозначая $k=-\frac{A}{B}$ и $b=-\frac{C}{B}$, получаем уравнение вида $y=kx+b$. Если же $B=0$, то уравнение имеет вид

$$x=-\frac{C}{A}.$$

Уравнение $y=kx+b$ называется ***уравнением прямой с угловым коэффициентом***; k — угловой коэффициент, b — величина отрезка, который отсекает прямая на оси Oy, считая от начала координат (рис. 1).

Уравнение $y-y_0=k(x-x_0)$ — это ***уравнение прямой с угловым коэффициентом k, которая проходит через точку с координатами*** $(x_0;\ y_0)$.

Рассмотрим две точки с координатами $(x_1;\ y_1)$ и

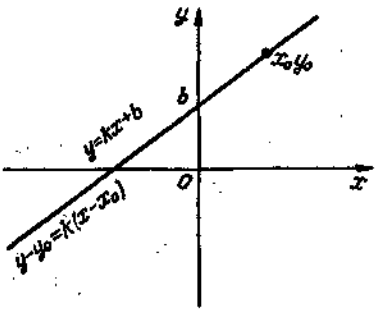


Рис. 1

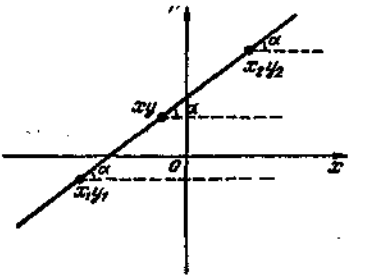


Рис. 2

$(x_2; y_2)$, лежащие на прямой $y=kx+b$. Их координаты удовлетворяют уравнению прямой:

$$y_1=kx_1+b,\ y_2=kx_2+b.$$

Вычитая из второго равенства первое, имеем

$$y_2-y_1=k(x_2-x_1),\ \text{или}\ k=\frac{y_2-y_1}{x_2-x_1}.$$

Пусть точка с координатами $(x; y)$ — произвольная точка на прямой, проходящей через точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$ (рис. 2). Тогда, с учетом того факта, что она имеет тот же коэффициент наклона, получаем

$$k=\frac{y-y_1}{x-x_1}.$$

Поэтому

$$\frac{y-y_1}{x-x_1}=\frac{y_2-y_1}{x_2-x_1}\ \text{или}\ \frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}.$$

Уравнение

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}$$

является **уравнением прямой, которая проходит через точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$.**

Недостатком этой формулы является ее неопределенность при $x_1=x_2$ и (или) $y_1=y_2$. Поэтому ее лучше использовать в виде

$$(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)=0.$$

Нетрудно заметить, что выражение

$$(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)$$

может быть приведено к виду

$$Ax+By+C,$$

где $A=y_2-y_1$, $B=x_1-x_2$, $C=-x_1(y_2-y_1)+y_1(x_2-x_1)$.

Алгоритм для определения значений коэффициентов A, B, C общего уравнения прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$, будет следующим [1]:

---

[1] Здесь и далее приводятся только фрагменты алгоритмов.

```
A:=y2-y1
B:=x1-x2                                    (1.1)
C:=-x1*(y2-y1)+y1*(x2-x1)
```

Рассмотрим пример: $x_1=0$, $y_1=0$, $x_2=1$, $y_2=2$. Уравнение прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$, будет следующим:

$A=y_2-y_1=2-0=2,$
$B=x_1-x_2=0-1=-1,$
$C=-x_1(y_2-y_1)+y_1(x_2-x_1)=0\cdot 2+0\cdot 1=0.$

Следовательно, уравнение прямой будет иметь вид $2x-y=0$.

1.2. Положение точек относительно прямой

Множество точек прямой, проходящей через две точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$, удовлетворяет уравнению

$$(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)=0.$$

Это значит, что если имеется точка с координатами $(x_0; y_0)$ и $(x_0-x_1)(y_2-y_1)-(y_0-y_1)(x_2-x_1)=0$, то эта точка лежит на прямой. В дальнейшем вместо выражения

$$(x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1)$$

мы иногда будем использовать для краткости обозначение

$$Ax+By+C \text{ или } f(x_1, y_1, x_2, y_2, x, y).$$

Прямая $Ax+By+C=0$, проходящая через две заданные точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$, разбивает плоскость на две полуплоскости. Рассмотрим возможные значения выражения $Ax+By+C$.

1) $Ax+By+C=0$ — определяет геометрическое место точек, лежащих на прямой.

Запишем алгоритм для определения, лежит ли точка с координатами $(x_3; y_3)$ на прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$. Переменная P — переменная логического типа, которая имеет значение «истина», если

точка лежит на прямой, и имеет значение «ложь» в противном случае.

```
P:=«ложь»
если (x3-x1)*(y2-y1)-(y3-y1)*(x2-x1)=0
| то P:=«истина»                          (1.2)
все
```

2) $Ax+By+C>0$ — определяет геометрическое место точек, лежащих по одну сторону от прямой.

3) $Ax+By+C<0$ — определяет геометрическое место точек, лежащих по другую сторону от прямой.

Это значит, что если для двух точек с координатами $(x_3;\ y_3)$ и $(x_4;\ y_4)$ значения выражений Ax_3+By_3+C и Ax_4+By_4+C имеют разные знаки, то эти точки лежат по разные стороны от прямой, проходящей через точки с координатами $(x_1;\ y_1)$ и $(x_2;\ y_2)$, а если одинаковые, то эти точки лежат по одну сторону от прямой. При этом число 0 имеет знак и «+» и «—».

На рисунке 3 точки $(x_3;\ y_3)$ и $(x_4;\ y_4)$ лежат по одну сторону от прямой, точки $(x_3;\ y_3)$ и $(x_5;\ y_5)$ — по разные стороны от прямой, а точка $(x_6;\ y_6)$ лежит на прямой.

Рассмотрим пример: $x_1=1$, $y_1=2$, $x_2=5$, $y_2=6$. Уравнение прямой, проходящей через точки $(x_1;\ y_1)$ и $(x_2;\ y_2)$, будет следующим:

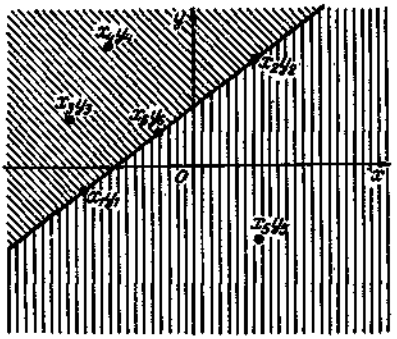


Рис. 3

$$A=y_2-y_1=6-2=4,$$
$$B=x_1-x_2=1-5=-4,$$
$$C=-x_1(y_2-y_1)+y_1(x_2-x_1)=-1\cdot 4+2\cdot 4=4.$$

Следовательно, уравнение прямой будет иметь вид: $4x-4y+4=0$, или $x-y+1=0$. Подставим координаты точек (3; 4), (1; 1), (2; 0), (0; 2) в уравнение прямой. Получим:

$1\cdot 3-1\cdot 4+1=0$, $\quad 1\cdot 2-1\cdot 0+1>0$,
$1\cdot 1-1\cdot 1+1>0$, $\quad 1\cdot 0-1\cdot 2+1<0$.

Следовательно, точка (3; 4) лежит на прямой, точки (1; 1) и (2; 0) лежат по одну сторону от прямой, а точки (1; 1) и (0; 2) — по разные стороны от прямой.

Алгоритм определения взаимного расположения точек $(x_3; y_3)$ и $(x_4; y_4)$ относительно прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$, можно записать следующим образом:

```
L:=«по одну»
Z1:=(x3-x1)*(y2-y1)-(y3-y1)*(x2-x1)
Z2:=(x4-x1)*(y2-y1)-(y4-y1)*(x2-x1)
если Z1*Z2<0
| то L:=«по разные»                    (1.3)
все
```

1.3. Взаимное расположение двух отрезков

Пусть нам необходимо определить взаимное расположение двух отрезков. Отрезки на плоскости заданы координатами своих концевых точек. Предположим, что концевые точки одного из отрезков имеют координаты $(x_1; y_1)$ и $(x_2; y_2)$, а концевые точки другого — $(x_3; y_3)$ и $(x_4; y_4)$. Пусть общее уравнение первой прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$, имеет вид

$$A_1x+B_1y+C_1=0,$$

а уравнение второй прямой, проходящей через точки $(x_3; y_3)$ и $(x_4; y_4)$, выглядит так:

$$A_2x+B_2y+C_2=0.$$

Определим расположение точек $(x_3; y_3)$ и $(x_4; y_4)$ относительно первой прямой. Если они расположены по одну сторону от прямой, то отрезки не могут пересекаться. Аналогично можно определить положение точек $(x_1; y_1)$ и $(x_2; y_2)$ относительно другой прямой.

Таким образом, если значения пары выражений

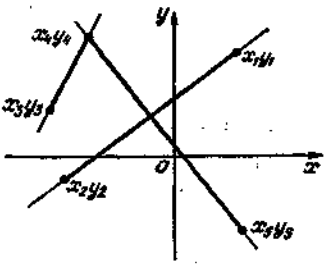


Рис. 4

$$Z_1 = A_1x_3 + B_1y_3 + C_1$$
и $Z_2 = A_1x_4 + B_1y_4 + C_1$
имеют разные знаки или $Z_1 \cdot Z_2 = 0$, а также пары
$$Z_3 = A_2x_1 + B_2y_1 + C_2$$
и $Z_4 = A_2x_2 + B_2y_2 + C_2$
имеют разные знаки или $Z_3 \cdot Z_4 = 0$, то отрезки пересекаются. Если же значения пар выражений Z_1 и Z_2 или Z_3 и Z_4 имеют одинаковые знаки, то отрезки не пересекаются.

Различные случаи расположения отрезков показаны на рисунке 4.

На этом рисунке отрезки с концами в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_4; y_4)$, $(x_5; y_5)$ пересекаются, отрезки с концами в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$, $(x_4; y_4)$ не пересекаются, а отрезки с концами в точках $(x_3; y_3)$, $(x_4; y_4)$ и $(x_4; y_4)$ и $(x_5; y_5)$ имеют общую вершину. Последний случай можно считать частным случаем пересечения.

Алгоритм для определения, пересекаются ли два отрезка с концами в точках $(x_1; y_1)$, $(x_2; y_2)$ и $(x_3; y_3)$, $(x_4; y_4)$, будет следующим:

```
P:=«истина»
Z1:=(x3-x1)*(y2-y1)-(y3-y1)*(x2-x1)
Z2:=(x4-x1)*(y2-y1)-(y4-y1)*(x2-x1)
если Z1*Z2>0
| то P:=«ложь»                                    (1.4)
все
Z3:=(x1-x3)*(y4-y3)-(y1-y3)*(x4-x3)
Z4:=(x2-x3)*(y4-y3)-(y2-y3)*(x4-x3)
если Z3*Z4>0
| то P:=«ложь»
все
```

Приведенный фрагмент алгоритма не учитывает

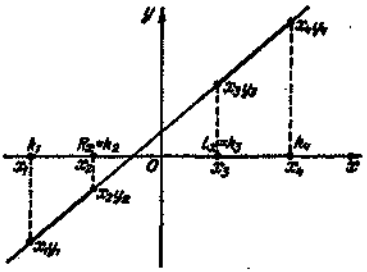


Рис. 5

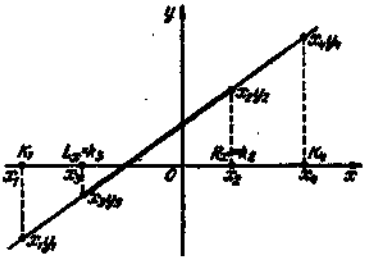


Рис. 6

крайней ситуации, когда два отрезка лежат на одной прямой. В этом случае

$$(x_3-x_1)(y_2-y_1)-(y_3-y_1)(x_2-x_1)=0 \text{ и } (x_4-x_1)\times$$
$$\times(y_2-y_1)-(y_4-y_1)(x_2-x_1)=0.$$

На рисунке 5 отрезки, лежащие на одной прямой, не пересекаются, а на рисунке 6 — пересекаются.

Для того чтобы определить взаимное расположение таких отрезков, поступим следующим образом. Обозначим $k_1=\min(x_1; x_2)$; $k_2=\max(x_1; x_2)$; $k_3=\min(x_3; x_4)$; $k_4=\max(x_3; x_4)$.

Здесь k_1 является левой, а k_2 — правой точкой проекции первого отрезка (отрезка, заданного координатами $(x_1; y_1)$, $(x_2; y_2)$) на ось Ox. Аналогично k_3 является левой, а k_4 — правой точкой проекции второго отрезка (отрезка, заданного координатами $(x_3; y_3)$, $(x_4; y_4)$) на ось Ox. Аналогично ищем проекции на ось Oy.

Отрезки, лежащие на одной прямой, будут пересекаться тогда, когда их проекции на каждую ось пересекаются. (Следует заметить, что если проекции двух произвольных отрезков пересекаются, то это не значит, что и сами отрезки пересекаются, что видно на рисунке 7.)

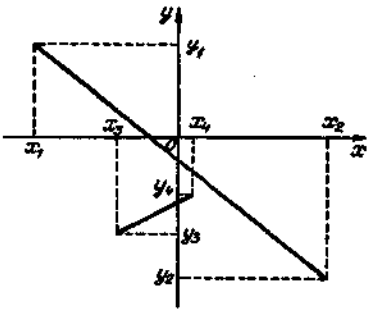


Рис. 7

Для определения взаимного расположения проекций на ось Ox воспользуемся следующим фактом (см. рис. 5 и 6): координата левой точки пересечения проекций L_x равна $\max(k_1; k_3)$, т. е. максимальной из координат левых точек проекций. Рассуждая аналогично для правых точек проекций, получим, что координата правой точки R_x пересечения равна $\min(k_2; k_4)$. Для того чтобы отрезки пересекались, необходимо, чтобы левая координата пересечения проекций была не больше правой координаты пересечения отрезков (такой случай имеет место на рис. 5, когда $L_x=x_3$, а $R_x=x_2$). Поэтому условием пересечения проекций является выполнение неравенства $L_x \leqslant R_x$.

Аналогично можно вычислить величины L_y и R_y, взяв соответствующие проекции на ось Oy.

Следует отметить, что длина пересечения проекций в этом случае равна величине $R_x - L_x$ (если $R_x - L_x = 0$, то проекции имеют только общую точку).

1.4. Точка пересечения отрезков

Для определения места пересечения отрезков (если известно, что они пересекаются), достаточно определить точку пересечения прямых, на которых эти отрезки лежат.

Пусть $A_1x + B_1y + C_1 = 0$ — уравнение прямой, проходящей через концевые точки первого отрезка, а $A_2x + B_2y + C_2 = 0$ — уравнение прямой, проходящей через концевые точки второго отрезка.

Тогда для определения точки пересечения отрезков достаточно решить систему уравнений

$$\begin{cases} A_1x + B_1y = -C_1, \\ A_2x + B_2y = -C_2. \end{cases}$$

Домножив первое уравнение на A_2, а второе — на A_1, получим

$$\begin{cases} A_2A_1x+A_2B_1y=-A_2C_1, \\ A_1A_2x+A_1B_2y=-A_1C_2. \end{cases}$$

Вычитаем из первого уравнения второе и находим значение y:

$$y=\frac{A_1C_2-A_2C_1}{A_2B_1-A_1B_2}.$$

Аналогично вычисляем значение x:

$$x=\frac{B_1C_2-B_2C_1}{B_2A_2-B_1A_2}.$$

Это справедливо в случае, если $A_2\cdot B_1-A_1\cdot B_2\neq 0$. Но мы уже знаем, что отрезки пересекаются и не лежат на одной прямой, а это невозможно, если $A_2\cdot B_1-A_1\cdot B_2=0$.

Вопросы для повторения

1. **Как определить коэффициенты общего уравнения прямой, если известно уравнение прямой с угловым коэффициентом?**
2. **Каким может быть взаимное расположение отрезка и прямой?**
3. **Как определить, что отрезки параллельны?**
4. **Как определить взаимное расположение двух параллельных отрезков?**

§ 2. РАССТОЯНИЕ НА ПЛОСКОСТИ

2.1. Расстояние между точками. Расстояние от точки до прямой

Расстояние между точками $M_1(x_1; y_1)$ и $M_2(x_2; y_2)$ на плоскости (рис. 8) определяется по формуле

$$D=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}.$$

Расстояние от точки до прямой на плоскости определяется как длина отрезка перпендикуляра, опущенного из точки на прямую. Уравнение вида

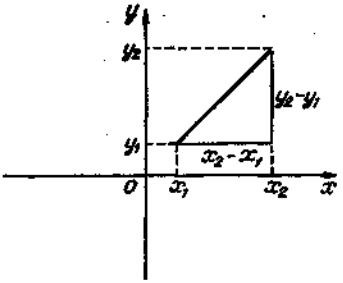


Рис. 8

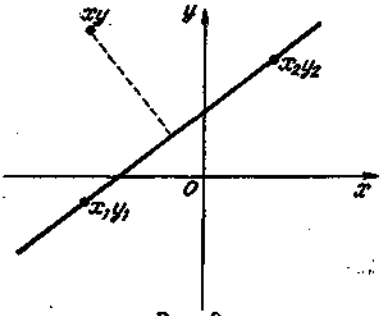


Рис. 9

$$\frac{Ax}{T}+\frac{By}{T}+\frac{C}{T}=0,$$

где $T=\sqrt{A^2+B^2}$, причем $C\leqslant 0$ (чего можно достигнуть изменением знака выражения), называется ***нормальным уравнением прямой.*** Это уравнение обладает тем свойством, что при подстановке координат произвольной точки в выражение $(Ax+By+C)/T$ получается значение, по абсолютной величине равное расстоянию от точки до прямой (рис. 9).

Запишем алгоритм для определения расстояния от точки $(x_3; y_3)$ до прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$:

```
A:=y2-y1
B:=x1-x2
C:=x1*(y2-y1)+y1*(x2-x1)                    (1.5)
T:=SQRT(A*A+B*B)
D:=ABS((A*x3+B*y3+C)/T)
```

Рассмотрим пример. Пусть $x_1=0$, $y_1=0$, $x_2=3$, $y_2=4$, $x_3=-1$, $y_3=7$. Тогда уравнение прямой, проходящей через точки $(x_1; y_1)$ и $(x_2; y_2)$, будет следующим:

$A=y_2-y_1=4-0=4,$
$B=x_1-x_2=0-3=-3,$
$C=-x_1(y_2-y_1)+y_1(x_2-x_1)=0\cdot 4+0\cdot 3=0,$
$T=\sqrt{A^2+B^2}=\sqrt{4\cdot 4+(-3)(-3)}=\sqrt{25}=5,$
$D=|(Ax_3+By_3+C)/T|=|(4(-1)+(-3)7+$
$+0)/5|=5.$

2.2. Расстояние между точкой и отрезком

Для определения расстояния между точкой и отрезком необходимо выяснить, пересекает ли перпендикуляр, опущенный из данной точки на прямую, проходящую через концы отрезка, сам отрезок. Если перпендикуляр пересекает отрезок, то расстояние между точкой и отрезком равно расстоянию между точкой и прямой, проходящей через отрезок. (Эту задачу вы уже умеете решать.)

Если перпендикуляр не пересекает отрезок, то расстояние между точкой и отрезком равно минимальному из расстояний между точкой и одним из концов отрезка.

Для определения взаимного расположения отрезка и перпендикуляра поступим следующим образом.

Рассмотрим треугольник, образованный тремя точками, две из которых $(x_1; y_1)$ и $(x_2; y_2)$ являются концами данного отрезка, а третья — данная точка с координатами $(x_3; y_3)$ (рис. 10). Конечно, может оказаться, что все точки лежат на одной прямой и такого треугольника не существует. В этом случае, однако, мы будем полагать, что треугольник существует, правда он вырожденный (особый). В вырожденном треугольнике вершины могут лежать на одной прямой (см. рис. 10, *а*).

Более того, мы будем полагать, что данный отрезок является основанием рассматриваемого треугольника (см. рис. 10, *б*, *в*).

При таких предположениях для решения исходной задачи нам достаточно определить, является ли один из углов при основании тупым. Действительно, если один из

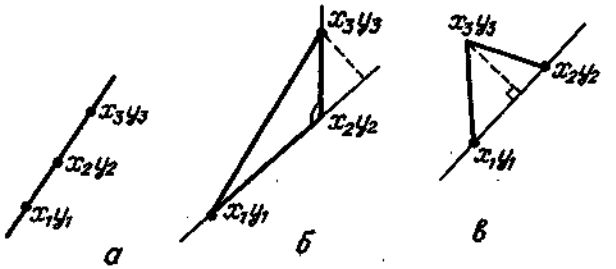


Рис. 10

углов при основании тупой, то перпендикуляр, опущенный из вершины, соответствующей исходной точке, не попадает на основание (отрезок). Если угол не тупой, то перпендикуляр, опущенный из вершины, соответствующей исходной точке, попадает на основание (отрезок).

Для решения последней задачи воспользуемся следующим свойством. Пусть a, b, c — длины сторон треугольника, причем c — длина основания. Тогда треугольник является тупоугольным при основании, если

$$a^2 > b^2 + c^2 \text{ или } b^2 > a^2 + c^2.$$

Поэтому, вычислив значения квадратов длин сторон, нетрудно определить, пересекает ли перпендикуляр, опущенный из точки $(x_3; y_3)$ на прямую, отрезок с концами в точках $(x_1; y_1)$ и $(x_2; y_2)$. И если не пересекает, то расстояние от точки до отрезка равно минимуму из величин a, b. Если же пересекает, то необходимо воспользоваться формулой расстояния от точки до прямой.

Вопросы для повторения

1. Как определить расстояние между двумя точками?
2. Чему равно расстояние между точкой и прямой?
3. Как определяется расстояние между отрезками?

§ 3. МНОГОУГОЛЬНИКИ

3.1. Виды многоугольников

Ломаной называется фигура, которая состоит из точек A_1, A_2, ..., A_n и соединяющих их отрезков A_1A_2, A_2A_3, ..., $A_{n-1}A_n$ (рис. 11, *а*). Точки называются ***вершинами*** ломаной, а отрезки — ***звеньями.*** Наиболее распространенным способом задания ломаной является использование таблицы, элементы которой соответствуют координатам вершин ломаной в порядке ее обхода из одного конца в другой. ***Длиной ломаной*** называется сумма длин ее звеньев.

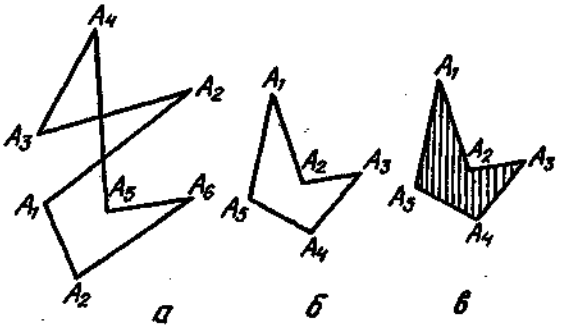


Рис. 11

Многоугольником называется замкнутая ломаная линия без самопересечений (рис. 11, *б*).

Плоским многоугольником называется конечная часть плоскости, ограниченная многоугольником (рис. 11, *в*).

Обход плоского многоугольника называется ***положительным***, если при обходе область расположена по левую руку, и ***отрицательным***, если область остается по правую руку.

Расстояние между фигурами на плоскости определяется как длина минимального отрезка, один конец которого принадлежит одной фигуре, а второй конец — другой фигуре.

3.2. Выпуклость многоугольников

Многоугольник является ***выпуклым***, если для каждой прямой, проходящей через любую его сторону, все остальные вершины лежат в одной полуплоскости относительно прямой. Проверим для каждой прямой, проходящей через вершины $(x_1; y_1)$ и $(x_2; y_2)$, $(x_2; y_2)$ и $(x_3; y_3)$, ..., $(x_{n-1}; y_{n-1})$ и $(x_n; y_n)$, $(x_n; y_n)$ и $(x_1; y_1)$, взаимное расположение вершин многоугольника. Если они каждый раз расположены в одной полуплоскости относительно проведенной прямой, то многоугольник выпуклый. Если же

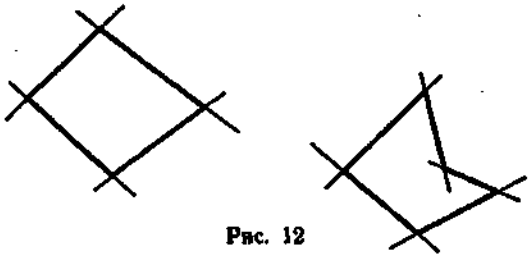


Рис. 12

найдется прямая, проходящая через одну из сторон, и пара вершин многоугольника, лежащих по разные стороны относительно проведенной прямой, то многоугольник не является выпуклым. Случаи выпуклого и невыпуклого многоугольников изображены на рисунке 12.

Можно заметить, что для каждой прямой, проходящей через вершины $(x_1; y_1)$ и $(x_2; y_2)$, $(x_2; y_2)$ и $(x_3; y_3)$, ..., $(x_{n-1}; y_{n-1})$ и $(x_n; y_n)$, $(x_n; y_n)$ и $(x_1; y_1)$, достаточно ограничиться определением взаимного расположения вершин многоугольника $(x_n; y_n)$ и $(x_3; y_3)$, $(x_1; y_1)$ и $(x_4; y_4)$, ..., $(x_{n-2}; y_{n-2})$ и $(x_1; y_1)$, $(x_{n-1}; y_{n-1})$ и $(x_2; y_2)$ соответственно. Если они каждый раз расположены в одной полуплоскости относительно проведенной прямой, то многоугольник выпуклый. Если же найдется прямая и пара вершин многоугольника, лежащих по разные стороны относительно проведенной прямой, то многоугольник не является выпуклым. Поэтому для определения, является ли многоугольник выпуклым, достаточно воспользоваться алгоритмом (1.6).

```
L:=«Выпуклый»
нц для i от 1 до n
| j:=mod(i, n)+1  :номер вершины после вершины i
| k:=mod(j, n)+1  :номер вершины после вершины j
| m:=i-1
| если i=1
| | то m:=n          :номер вершины перед вершиной i
| все                                              (1.6)
| Z1:=(x[m]-x[i])*(y[j]-y[i])-(y[m]-y[i])*(x[j]-
| -x[i])
```

```
| Z2:=(x[k]-x[i])*(y[j]-y[i])-(y[k]-y[i])*(x[j]-
| -x[i])
| если Z1*Z2<0
| | то L:=«Невыпуклый»
| все
кц
```

Вопросы для повторения

1. Какие бывают обходы многоугольника?
2. Какой многоугольник называется выпуклым?

§ 4. ПЛОЩАДИ ФИГУР

4.1. Площадь треугольника

Для вычисления площади треугольника (рис. 13) известна формула Герона:

$$S=\sqrt{p(p-a)(p-b)(p-c)},$$

где a, b, c — длины сторон треугольника, а p — его полупериметр, т. е. $p=(a+b+c)/2$.

Так как при заданных координатах вершин треугольника можно вычислить длины его сторон, то алгоритм поиска площади треугольника сводится к поиску длин сторон и использованию формулы Герона.

```
p:=(a+b+c)/2                                    (1.7)
S:=SQRT(p*(p-a)*(p-b)*(p-c))
```

Однако такой метод вычисления площади имеет один существенный недостаток: необходимо выполнение операции нахождения квадратного корня из числа. При выполнении этой операции часто происходит потеря точности, что может привести к получению не совсем точного результата. Поэтому, чтобы избежать возможных ошибок, в дальнейшем будут использоваться другие формулы.

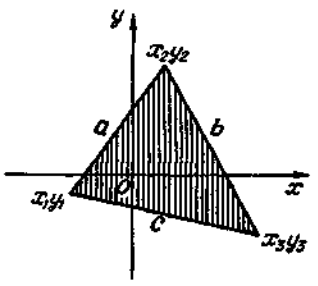


Рис. 13

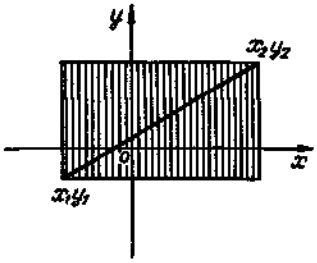


Рис. 14

4.2. Площадь прямоугольника

Мы будем рассматривать прямоугольники, стороны которых параллельны осям координат (рис. 14).

В этом случае прямоугольник может быть определен одной из своих диагоналей. Это значит, что пара точек на плоскости с координатами $(x_1; y_1)$ и $(x_2; y_2)$, соответствующая концам диагонали, однозначно определяет расположение и размер прямоугольника. Такое задание более удобно вместо описания прямоугольника посредством последовательности вершин в порядке обхода. Кроме того, при таком задании легко вычислить площадь прямоугольника по формуле

$$S = |(x_2 - x_1)(y_2 - y_1)|,$$

где $|(x_2 - x_1)|$ — длина проекции прямоугольника на ось Ox (длина стороны, параллельной оси Ox), а $|(y_2 - y_1)|$ — длина проекции прямоугольника на ось Oy (длина стороны, параллельной оси Oy).

4.3. Площадь трапеции

Мы будем рассматривать трапеции, основания которых параллельны оси Oy, одна из боковых сторон лежит на оси Ox, а другая расположена выше оси Ox (рис. 15).

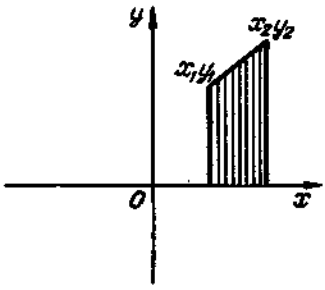


Рис. 15

В этом случае трапеция может быть определена парой точек $(x_1; y_1)$ и $(x_2; y_2)$, соответствующих вершинам трапеции, не лежащим на оси Ox. При таком задании легко вычислить площадь трапеции по формуле

$$S=\frac{|x_2-x_1|(y_2+y_1)}{2},$$

где $|x_2-x_1|$ — высота трапеции, а y_2 и y_1 — длины ее оснований.

4.4. Площадь плоского многоугольника

Сначала мы рассмотрим плоские многоугольники, расположенные выше оси Ox.

Традиционно при подсчете площади произвольного многоугольника его разбивают на треугольники и находят площадь каждого из них. Сумма площадей этих треугольников равна площади данного многоугольника.

Однако при этом возникает вопрос, каким образом делать это разбиение, если многоугольник задан координатами ломаной в порядке ее обхода и при этом не является выпуклым.

Более рациональным способом нахождения площади многоугольника является его представление в виде комбинации трапеций. При этом считается, что если рассматривается трапеция, у которой $x_1<x_2$, то значение ее площади берется со знаком «+» (рис. 16, *а*), а если $x_1>x_2$, то значение ее площади берется со знаком «—» (рис. 16, *б*).

При вычислении значения площади используется формула

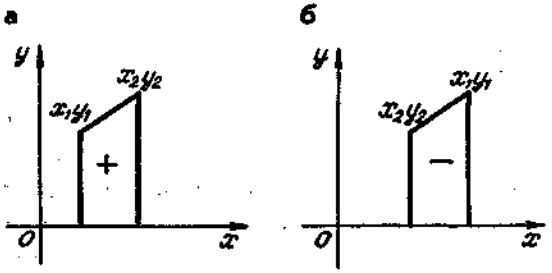


Рис. 16

$$S=\frac{|x_2-x_1|(y_2+y_1)}{2}.$$

Используя такой подход, формула для подсчета площади многоугольника, определяемого ломаной с координатами вершин $(x_1; y_1)$, $(x_2; y_2)$, ..., $(x_{n-1}; y_{n-1})$, $(x_n; y_n)$, примет следующий вид:

$$S=\left|\frac{(x_2-x_1)(y_2+y_1)}{2}+\frac{(x_3-x_2)(y_3+y_2)}{2}+\ldots+\frac{(x_1-x_n)(y_1+y_n)}{2}\right|,$$

или

$$S=\left|\frac{(x_2-x_1)(y_2+y_1)+(x_3-x_2)(y_3+y_2)+\ldots+(x_1-x_n)(y_1+y_n)}{2}\right|.$$

При $n=3$ и $n=4$ эти формулы могут быть использованы для вычисления площадей треугольников и четырехугольников соответственно.

При этом совершенно безразлично, выпуклый многоугольник или нет (рис. 17). Более того, вычисление площади требует выполнения только операций сложения, вычитания, умножения и одной операции деления.

При более детальном изучении оказывается, что приведенная выше формула может использоваться для произвольных точек на плоскости, т. е. точки могут располагаться и ниже оси Ox. Правда, при этом необходимо предварительно осуществить преобразование координат по формуле

$$y_i'=y_i-y_{\min},$$

где $y_{\min}$ — минимальное значение y координаты для вершин многоугольника в исходной системе координат. Такое преобразование соответствует параллельному переносу многоугольника параллельно оси

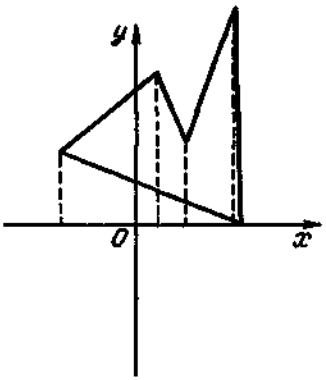


Рис. 17

Oy и гарантирует, что в новой системе координат ни одна вершина многоугольника не будет расположена ниже оси *Ox*.

Алгоритм для определения площади плоского многоугольника, заданного координатами его вершин в порядке их обхода по контуру, будет таким.

(Внимание! Здесь и в дальнейшем мы будем предполагать, что обход многоугольника задается $n+1$ вершиной, причем $(n+1)$-я вершина совпадает с первой вершиной обхода.)

```
ymin:=y[1]
нц для k от 2 до n
| если ymin>y[k]
| | то ymin:=y[k]
| все
кц
нц для k от 1 до n+1
| y1[k]:=y[k]-ymin                                    (1.8)
кц
S:=0
нц для k от 1 до n
| S:=S+(x[k+1]-x[k])*(y1[k+1]+y1[k])
кц
S:=ABS(S)/2
```

Вопросы для повторения

1. Как можно вычислить площадь треугольника, если известны координаты его вершин?
2. Какие преобразования можно сделать, чтобы многоугольник лежал выше оси *Ox*?
3. Какой способ вычисления площади треугольника более эффективен: формула Герона или общая формула?

§ 5. ВЗАИМНОЕ РАСПОЛОЖЕНИЕ ФИГУР НА ПЛОСКОСТИ

5.1. Взаимное расположение многоугольника и точки

Существует много способов определения взаимного расположения многоугольника и точки. Рассмотрим один из наиболее употребительных способов, который называ-

ется ***методом сканирующей прямой***, и основывается на следующем.

Находясь в данной точке $(x_0; y_0)$, мы начинаем двигаться в одном направлении (это может быть любое направление, например, параллельно оси Ox), подсчитывая при этом количество пересеченных сторон многоугольника. Движение заканчивается тогда, когда мы уйдем достаточно далеко, и ни одна сторона многоугольника уже не сможет встретиться на нашем пути. Оказывается, что если при нашем движении было пересечено нечетное число сторон многоугольника, то исходная точка лежит внутри многоугольника (рис. 18, *а*), а если было пересечено четное число сторон, то точка лежит снаружи (рис. 18 *б, в*).

Маршрут нашего движения может быть представлен в виде отрезка, координаты одной концевой точки которого равны координатам исходной точки, а одна из координат другой концевой точки больше (или меньше) любой соответствующей координаты вершин многоугольника.

В этом случае задача взаимного расположения многоугольника и точки сводится к подсчету числа пересечения полученного отрезка и сторон многоугольника. Другая координата может быть любой. Обычно ее берут равной соответствующей координате исходной точки. В этом случае маршрут движения происходит вдоль одной из осей координат.

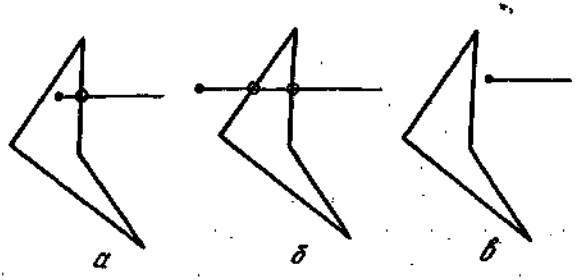


Рис. 18

В этом случае задача взаимного расположения многоугольника и точки сводится к подсчету числа пересечений полученного отрезка и сторон многоугольника. В массивах X и Y хранятся координаты вершин многоугольника в порядке обхода, $x[0]$, $y[0]$ — координаты исходной точки.

```
xmin:=x[1]
нц для k от 2 до n
| если xmin>x[k]
| | то xmin:=x[k]
| все
кц
x:=xmin-1
y:=y[0]
S:=0                                                    (1.9)
нц для i от 1 до n
| Z1:=(x[i]-x[0])*(y-y[0])-(y[i]-y[0])*(x-x[0])
| Z2:=(x[i+1]-x[0])*(y-y[0])-(y[i+1]-
| -y[0])*(x-x[0])
| если Z1*Z2<0
| | то S:=S+1
| все
кц
L:=«внутри»
если mod(S, 2)=0
| то L:=«вне»
все
```

Такой подход требует особого анализа случаев, когда полученный отрезок пересекает сторону многоугольника в концевой точке.

Если отрезок пересекает одну из вершин многоугольника (например, A_1), то может быть 2 случая (рис. 19):

1) Обе стороны многоугольника, входящие в вершину A_1, лежат по одну сторону от отрезка (рис. 19, *а*). Количество пересечений можно считать равным 2 (или 0);

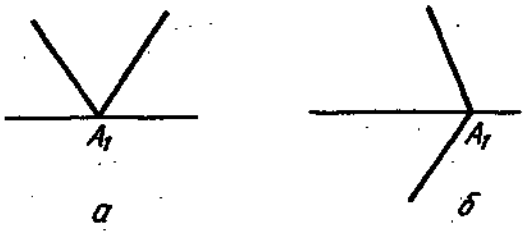


Рис. 19

2) стороны многоугольника, входящие в вершину A_1, лежат по разные стороны отрезка (рис. 19, *б*). Число пересечений примем равным 1.

Для проверки, по разные или по одну сторону от прямой лежат стороны многоугольника, можно использовать алгоритм (1.3).

Если отрезок проходит по стороне, то число пересечений будем считать равным 2.

5.2. Взаимное расположение многоугольников

Возможны 3 варианта взаимного расположения многоугольников: они могут пересекаться (рис. 20, *а*), лежать один внутри другого (рис. 20, *б*) или располагаться каждый вне другого (рис. 20, *в*).

Определение взаимного расположения многоугольников, заданных обходами своих вершин, будет проводиться в два этапа.

Первый этап. Для того чтобы определить взаимное расположение многоугольников, следует проверить взаим-

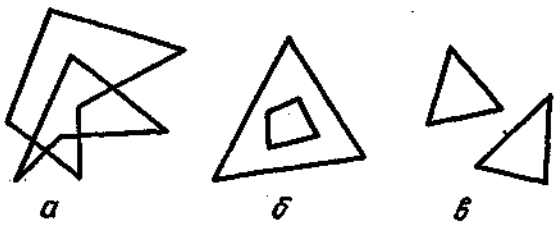


Рис. 20

ное расположение сторон многоугольников. Если две стороны разных многоугольников пересекаются, то и многоугольники пересекаются. Поэтому для определения возможного пересечения сторон многоугольников можно воспользоваться алгоритмом (1.4). Если найдется пара сторон из разных многоугольников, которые пересекаются, то взаимное положение многоугольников определено.

Если оказалось, что в многоугольниках нет взаимно пересекающихся сторон, то переходим ко второму этапу.

Второй этап. Устанавливаем взаимное расположение вершины одного из многоугольников и другого многоугольника. Если оказалось, что вершина одного из многоугольников лежит внутри другого многоугольника, то один из многоугольников лежит внутри другого. Следовательно, взаимное расположение многоугольников установлено.

Пусть для каждого из многоугольников его вершина лежит вне другого многоугольника. В этом случае остается единственно возможное решение: каждый многоугольник лежит вне другого.

Вопросы для повторения

1. Как определить взаимное расположение многоугольника и точки?
2. Зависит ли способ определения взаимного расположения многоугольника и точки от выпуклости многоугольника?
3. Назовите возможные расположения двух треугольников.

ЗАДАЧИ ДЛЯ ПОВТОРЕНИЯ

1. Даны три числа a, b, c. Определить, существует ли треугольник с такими длинами сторон.

2. Даны четыре числа a, b, c, d. Определить, существует ли четырехугольник с такими длинами сторон.

3. Найти взаимное расположение окружности радиуса R с центром в точке $(x_0; y_0)$ и точки A с координатами $(x_1; y_1)$.

4. Найти взаимное расположение двух окружностей

радиуса R_1 и R_2 с центрами в точках $(x_1; y_1)$ и $(x_2; y_2)$ соответственно.

5. Найти взаимное расположение окружности радиуса R с центром в точке $(x_0; y_0)$ и прямой, проходящей через точки с координатами $(x_1; y_1)$ и $(x_2; y_2)$.

6. Определить количество точек с целочисленными координатами, лежащих внутри окружности радиуса R с центром в точке $(x_0; y_0)$.

7. Найти координаты точек пересечения двух окружностей радиусов R_1 и R_2 с центрами в точках $(x_1; y_1)$ и $(x_2; y_2)$ соответственно.

8. Найти координаты точки, симметричной данной точке M с координатами $(x_1; y_1)$ относительно прямой $Ax+By+C=0$.

9. Даны две точки $M_1(x_1; y_1)$, $M_2(x_2; y_2)$ и прямая $Ax+By+C=0$. Найти на этой прямой такую точку $M_0(x_0; y_0)$, чтобы суммарное расстояние от нее до двух данных точек было минимально.

10. Даны три точки с координатами $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$, которые являются вершинами некоторого прямоугольника. Найти координаты четвертой вершины.

11. Даны координаты вершин четырехугольника $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$, $(x_4; y_4)$. Определить, выпуклый ли четырехугольник.

12. Даны координаты вершин четырехугольника $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$, $(x_4; y_4)$. Определить, является ли четырехугольник: а) ромбом; б) квадратом; в) трапецией.

13. Даны координаты двух вершин $(x_1; y_1)$ и $(x_2; y_2)$ некоторого квадрата. Найти возможные координаты других его вершин.

14. Даны координаты двух вершин $(x_1; y_1)$ и $(x_2; y_2)$ некоторого квадрата, расположенных по диагонали, и точка $(x_3; y_3)$. Определить, лежит ли точка внутри квадрата.

15. Даны координаты $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$ вершин треугольника. Найти координаты точки пересечения его медиан.

16. Даны координаты $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$ вершин треугольника. Найти длины его высот.

17. Определить коэффициенты уравнения прямой, параллельной данной прямой, определяемой уравнением $Ax+By+C=0$ и проходящей через точку с координатами $(x_0; y_0)$.

18. Определить коэффициенты уравнения прямой, перпендикулярной данной прямой, определяемой уравнением $Ax+By+C=0$ и проходящей через точку с координатами $(x_0; y_0)$.

ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Определить, пересекаются ли прямая $y=kx+b$ и отрезок с концами $(x_1; y_1)$, $(x_2; y_2)$.

2. Определить, принадлежит ли точка $A(x; y)$ отрезку с концевыми точками $B(x_1; y_1)$ и $C(x_2; y_2)$.

3. а) Выпуклый многоугольник задается координатами своих вершин при его обходе по часовой или против часовой стрелки. Контур многоугольника не имеет самопересечений. Определить направление обхода.

б) Определить направление обхода в случае невыпуклого многоугольника.

4. На плоскости заданы n отрезков координатами концевых точек. Концы отрезков задаются двумя парами координат $(x_1[i]; y_1[i])$, $(x_2[i]; y_2[i])$, $1 \leqslant i \leqslant n$ (концы принадлежат отрезку). Найти прямую, имеющую общие точки с максимальным числом отрезков, и напечатать в порядке возрастания номера тех отрезков, которые эта прямая пересекает.

5. N точек на плоскости заданы своими координатами. Найти такой минимальный по площади выпуклый многоугольник, что все N точек лежат либо внутри этого многоугольника, либо на его границе (такой выпуклый многоугольник называется выпуклой оболочкой).

6. На плоскости заданы своими координатами k точек. Определить, можно ли построить такой выпуклый

многоугольник, что каждая точка принадлежит некоторой стороне.

7. N точек на плоскости заданы своими координатами. Найти порядок, в котором можно соединить эти точки, чтобы получился N-угольник.

8. Представьте себе, что в тетрадке Вы закрасили на листе какое-то количество клеточек и получили клеточную фигуру. Сколько осей симметрии имеет заданная клеточная фигура?

Заданы: N_i — размер фигуры по вертикали, N_j — размер фигуры по горизонтали ($N_i < 101$; $N_j < 81$) и сама фигура в виде N_i строк из пробелов и звездочек по N_j символов в каждой строке.

Звездочка соответствует закрашенной клеточке.

Пример 1.

2 4 (размер фигуры по вертикали и горизонтали)

* * * *

* *

Фигура имеет 1 ось симметрии.

Пример 2.

3 5 (размер фигуры по вертикали и горизонтали)

* * * * *

* * *

* * *

Фигура имеет 0 осей симметрии.

9. Прямоугольник $ABCD$ задан координатами своих вершин. На противоположных сторонах AB и CD заданы последовательности R_1 и R_2 из N точек разбиения, а на сторонах BC и AD — R_3 и R_4 из M точек разбиения. Нумерация элементов последовательностей R_1 и R_2 начинается соответственно от точек A и D, а R_3 и R_4 — от B и A. Соединив отрезками точки с одинаковыми номерами в разбиениях R_1 и R_2, а затем в разбиениях R_3 и R_4, получим разбиение Q прямоугольника $ABCD$ на множество четырехугольников.

Найти четырехугольник разбиения Q с наибольшей площадью при условии, что отрезки, соединяющие точки

разбиений R_1 и R_2 параллельны стороне AD. Последовательности R_1, R_2, R_3 и R_4 задаются как массивы из длин отрезков разбиения соответствующих сторон прямоугольника.

10. На прямой задано N точек с координатами x_1, x_2, ..., x_N. Найти такую точку Z, сумма расстояний от которой до данных точек минимальна.

11. Пусть через административный район проходит по прямой железная дорога. На ней надо построить станцию так, чтобы расстояние от нее до самой дальней деревни было бы минимальным. Написать программу.

12. На плоскости задано N точек с координатами $(x_1; y_1)$, $(x_2; y_2)$, ..., $(x_n; y_n)$. Написать программу, которая из этих точек выделяет вершины квадрата, содержащего максимальное число заданных точек. (Предполагается, что точки, расположенные на сторонах квадрата, принадлежат ему.)

13. На плоскости задано множество из N прямоугольников, стороны которых параллельны осям координат, при этом каждый прямоугольник задается координатами левой нижней и правой верхней его вершин. Составить алгоритм определения наибольшего натурального числа K, для которого существует точка плоскости, принадлежащая одновременно K прямоугольникам.

Примечание. Эффективным считается алгоритм, число действий которого пропорционально N^2.

14. На квадратном торте N свечей. Можно ли одним прямолинейным разрезом разделить его на две равные по площади части, одна из которых не содержала бы ни одной свечи? Свечи будем считать точками, у которых известны их целочисленные координаты $(x_1; y_1)$, ..., $(x_N; y_N)$. Начало координат — в центре торта. Разрез не может проходить через свечу.

15. Даны $N (N > 1)$ прямоугольников, для которых предполагается, что:

а) стороны любого прямоугольника параллельны ко-

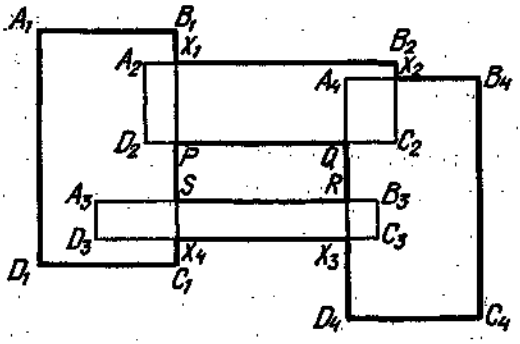


Рис. 21

ординатным осям и прямоугольник задается концами одной из диагоналей;

б) каждый прямоугольник имеет общие внутренние точки с хотя бы одним из остальных и не имеет общих вершин, сторон или частей сторон ни с одним из остальных прямоугольников.

Составить программу, которая даст возможность:

1) Определить внешний контур фигуры F, являющейся объединением прямоугольников (рис. 21).

2) Определить, содержит ли фигура F «дырки», т. е. замкнутые фигуры, которые ей не принадлежат.

3) Разложить фигуру F на наименьшее возможное число непересекающихся прямоугольников, которые могут иметь общие стороны или части сторон, а их объединение дает фигуру F.

4) Вычислить периметр и площадь фигуры F.

Внешний контур объединения прямоугольников $A_iB_iC_iD_i$, $i=1, 2, 3, 4$, есть $A_1D_1C_1X_4X_3D_4C_4B_4X_2B_2X_1B_1$; фигура F содержит единственную «дырку» $PQRS$.

Примечание. Задачи 3) и 4) решаются только для фигур, не содержащих «дырок».

16. Очертание города. Необходимо написать программу, которая должна помочь архитектору в рисовании очертания города. Город задается расположением зданий и рассматривается как двумерный; все здания

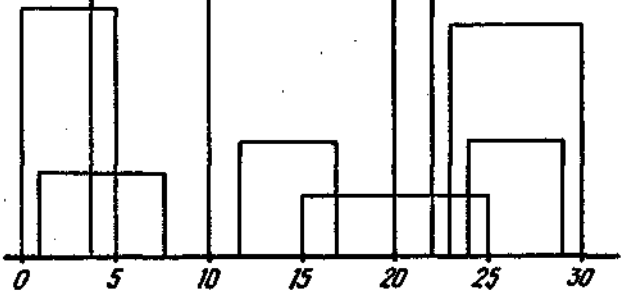


Рис. 22

в нем — прямоугольники, основания которых лежат на одной прямой (город построен на равнине). Здания задаются тройкой чисел (L_i, H_i, R_i), где L_i и R_i — координаты левой и правой стен здания i, а H_i — высота этого здания. На рисунке 22 здания описываются тройками (0, 11, 5), (2, 6, 7), (3, 13, 10), (12, 7, 16), (14, 3, 25), (20, 18, 22), (23, 13, 30), (24, 4, 29), а контур, показанный на рисунке 23, задается последовательностью (1, 11, 2, 13, 10, 0, 12, 7, 16, 3, 20, 18, 22, 3, 23, 13, 30, 0) (о способе формирования этой последовательности см. ниже).

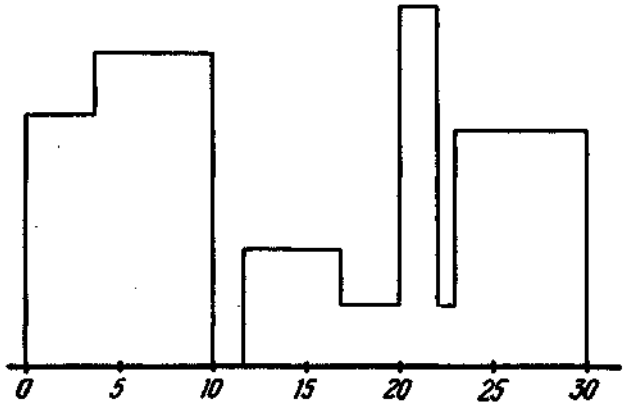


Рис. 23

Ввод представляет собой последовательность троек, задающих дома. Все координаты есть целые числа меньше 10 000. Во входном файле минимум одно и максимум 50 зданий. Каждая тройка, обозначающая здание, находится в отдельной строке во входном файле. Все целые числа в тройке разделены одним или несколькими пробелами. Тройки отсортированы по L_i, т. е. по левой x-координате здания, таким образом, здание с самой маленькой левой x-координатой является первым во входном файле.

Вывод будет состоять из вектора, описывающего очертание, как показано в примере выше. В векторе очертание (v_1, v_2, v_3, ..., v_{n-2}, v_{n-1}, v_n), v_i означает горизонтальную линию (высоту), когда i — четное число, и вертикальную линию (x-координату), когда i — нечетное. Вектор очертания будет определять маршрут, пройденный, к примеру, жуком, начавшим с минимальной x-координаты и путешествующим по всем вертикальным и горизонтальным линиям, определяющим контур. Последний элемент в векторе линии контура будет 0.

17. Нижняя левая и верхняя правая вершины прямоугольника A имеют координаты (0; 0) и (V; W) соответственно. Множество S из N точек задается парами координат (x_i; y_i), $1 \leqslant i \leqslant N$. Найти такой прямоугольник G максимальной площади, что его стороны параллельны сторонам A, G полностью лежит в A (G и A могут иметь общие граничные точки) и ни одна точка из S не лежит внутри G (но может лежать на его стороне). Напечатать величину площади G и координаты нижней левой и верхней правой вершин этого прямоугольника. Если таких прямоугольников несколько, то вывести информацию по каждому.

Примечание. Во множестве S никакие две точки не лежат на одной прямой, параллельной стороне A.

18. В первом квадрате координатной системы Oxy нарисован первый квадрат — $ABCD$, длина стороны которого равна 1 и вершина A находится в начале координат. Потом нарисованы: второй квадрат $BEFC$, третий — $DFGH$, четвертый — $JAHI$, пятый — $KLEJ$ и так далее по спирали (рис. 24). Написать программу, которая для введенных целых чисел x и y определяет и выводит номер квадрата, которому принадлежит точка $P(x; y)$. Если точка P лежит на сторонах квадратов или в вершинах, то будем считать, что она принадлежит квадрату с наименьшим номером из возможных.

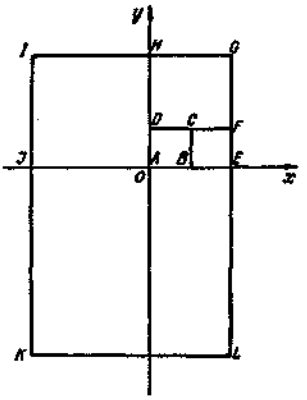


Рис. 24

Примеры.	x	y	результат
	2	1	2
	−1	0	4
	13	2	10

19. На плоскости заданы своими координатами N различных точек. Найти уравнение прямой, делящей это множество точек на два подмножества с одинаковым количеством элементов.

20. Найти пересечение и объединение двух выпуклых многоугольников. Многоугольники задаются координатами вершин в порядке обхода по контуру.

21. N-угольник на плоскости задается координатами вершин в порядке их обхода по контуру. Для точки $Z(x; y)$ найти минимальное расстояние до контура N-угольника.

22. На плоскости своими координатами задаются N точек. Матрица $C[1..N, 1..N]$ задается следующим образом: $C_{ij}=C_{ji}=1$ в случае, если вершины i и j соединены

отрезком, и 0, если не соединены. Известно, что любая вершина соединена, по крайней мере, с двумя другими и что отрезки пересекаются только в концевых точках. Таким образом, вся плоскость разбивается на множество многоугольников. Задана точка $Z(x; y)$. Найти минимальный по площади многоугольник, содержащий Z, или выдать сообщение, что такого не существует. Если Z принадлежит какому-то отрезку, то выдать его концевые точки; если Z лежит в многоугольнике, то выдать его вершины в порядке обхода по контуру.

23. Будем называть два многоугольника подобными, если существует взаимно однозначное отображение сторон этих двух фигур такое, что соответствующие стороны пропорциональны с коэффициентом пропорциональности k, а углы, образованные двумя соответствующими сторонами, равны. Определить, подобны ли два многоугольника. Многоугольники задаются на плоскости координатами вершин контуров. Вершины в контуре перечисляются в порядке обхода против часовой стрелки.

Примечание. Так как все вычисления на ЭВМ проводятся с ограниченной точностью, то считать, что две величины равны, если они совпадают с точностью до двух знаков после запятой.

24. Заданы натуральное число N и две последовательности целых чисел $(a_1, a_2, \ldots, a_N)$ и $(b_1, b_2, \ldots, b_N)$. Заданы также два числа x_0 и x_1, $x_0 < x_1$.

а) Найти числа $t_0, t_1, \ldots, t_p$, $p \leqslant N$, такие, что $x_0 = t_0 < t_1 < \ldots < t_p = x_1$, и указать для каждого отрезка $[t_{j-1}; t_j]$, $1 \leqslant j \leqslant p$, такое число k, $1 \leqslant k \leqslant N$, что для всех i, $1 \leqslant i \leqslant N$, и для всех x из $[t_{j-1}; t_j]$ справедливо неравенство $a_k \cdot x + b_k \geqslant a_i \cdot x + b_i$.

б) Найти числа $s_0, s_1, \ldots, s_Q$, такие, что $x_0 = s_0 < s_1 < \ldots < s_Q = x_1$, и указать для каждого отрезка $[s_{j-1}; s_j]$, $1 \leqslant j \leqslant Q$, такую перестановку $(i_1, i_2, \ldots, i_N)$ чисел

1, 2, 3, ..., N, что для всех x из $[s_{j-1}; s_j]$ справедливо неравенство $a_{i_1} \cdot x + b_{i_1} \leqslant a_{i_2} \cdot x + b_{i_2} \leqslant \ldots \leqslant a_{i_N} \cdot x + b_{i_N}$, и для всех отрезков соответствующие перестановки различны.

25. В правильном n-угольнике провели несколько диагоналей, причем никакие три не пересекаются в одной точке. На сколько частей диагонали разбили n-угольник? Диагонали заданы номерами вершин n-угольника, которые они соединяют, все вершины перенумерованы по порядку числами 1, ..., n.

26. Круг разрезан самонепересекающейся ломаной, координаты вершин которой заданы парами натуральных чисел $(x_1; y_1)$, ..., $(x_k; y_k)$. Первая и последняя вершины лежат на границе круга, а остальные — внутри него. Определить, можно ли разъединить две получившиеся части круга (выход из плоскости и повороты разнимаемых частей не допускаются).

27. На местности, представляющей собой идеально ровную поверхность, стоит высокий забор. План забора — замкнутая ломаная без самопересечений, которая задается N парами координат своих вершин в порядке обхода ограничиваемой забором области против часовой стрелки. Вершины пронумерованы от 1 до N, $N < 100$. В точке $(x; y)$ стоит человек ($(x; y)$ не может лежать на ломаной). Считая, что каждому звену ломаной ставится в соответствие пара номеров концевых вершин, указать, какие звенья человек увидит полностью или частично, а какие — вообще не увидит. Если при взгляде звено видно как точка или как пара точек, то полагаем, что оно не видно.

28. На гранях двух равных правильных тетраэдров N и M написаны числа N_1, N_2, N_3, N_4 и M_1, M_2, M_3, M_4. Можно ли совместить тетраэдры так, чтобы на совпадающих гранях оказались одинаковые числа?

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. Заданы стороны A и B двух квадратов, расположенных так, как показано на рисунке 25 (вертикальные оси симметрии у квадратов совпадают). Определить угол α, под которым шарик надо выпустить из точки A, чтобы он попал в точку B за минимальное количество отражений.

2. Два N-угольника на плоскости задаются координатами вершин в порядке их обхода по контурам. Найти минимальное расстояние между этими N-угольниками.

3. N точек на плоскости заданы своими координатами. Найти порядок, в котором можно соединить эти точки, чтобы получилась ломаная без самопересечений.

4. На плоскости задано N точек с координатами $(x_1; y_1)$, $(x_2; y_2)$, ..., $(x_n; y_n)$. Найти такую точку $Z(x; y)$, сумма расстояний от которой до остальных минимальна и:

а) Z — одна из заданных точек;

б) Z — произвольная точка плоскости.

5. На плоскости задано множество точек A и множество прямых B. Найти две такие различные точки из A, что проходящая через них прямая параллельна наибольшему количеству прямых из B.

6. Среди треугольников с вершинами в заданном множестве точек на плоскости указать такой, стороны которого содержат максимальное число точек заданного множества.

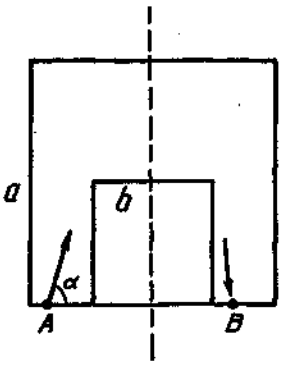


Рис. 25

7. Все стены дома имеют длину 5 м. Северная и южная стороны — белые, западная и восточная — синие. Человек прошел от юго-восточного угла дома A метров на юг, B метров на восток и C метров на север и посмотрел на дом. Написать алгоритм, который определяет, какие стены увидит человек.

8. На столе лежит игральный кубик гранью A_0 к нам, гранью B_0 вверх. Написать программу,

определяющую последовательность «кантования» кубика («на нас», «от нас», «вправо», «влево»), после выполнения которых кубик окажется на прежнем месте, но к нам гранью A_k, вверх — B_k.

Примечание. Под кантованием понимается перекатывание кубика через соприкасающееся со столом ребро без скольжения. Другие способы перемещения кубика запрещены. Нумерация граней кубика такова, что если его положить на грань с цифрой 5, то боковые грани будут иметь номера 1, 6, 4, 3 при обходе по часовой стрелке, а верхняя — номер 2.

9. На плоскости заданы координаты вершин двух треугольников. Требуется найти хотя бы одну прямую, разбивающую каждый из этих треугольников на две равновеликие части. Результат должен выводиться на экран в виде уравнения $y=kx+b$. Если искомая прямая параллельна оси Oy, то уравнение должно быть приведено в виде $x=A$. Коэффициенты A, B, k должны быть приведены не менее чем с четырьмя знаками после десятичной точки.

10. Даны числа $(x_1; y_1)$, $(x_2; y_2)$, $(x_3; y_3)$ — координаты трех каких-то вершин прямоугольника в прямоугольной системе координат. Найти координаты четвертой вершины.

11. Будем отсчитывать углы от положительного направления оси Ox: положительные — против часовой стрелки, отрицательные — по часовой стрелке. Пусть величина угла лежит в пределах от $-180°$ до $180°$. Даны 2 точки $A(x_1; y_1)$ и $B(x_2; y_2)$. Определить, какой из отрезков, OA или OB, образует больший угол с осью Ox.

12. Задается число N и точки плоскости $(x_1; y_1)$, $(x_2; y_2)$, ..., $(x_N; y_N)$, являющиеся серединами последовательных сторон N-угольника. Восстановить по этим точкам исходный N-угольник. Под многоугольником в данной задаче понимается какая угодно (возможно самопересекающаяся) замкнутая ломаная.

Примечание. N — нечетное, $N>1$ и $N<20$, x_i, y_i — вещественные числа. Входные данные будут соответствовать приведенным условиям, причем многоугольник построить можно.

Структура вывода:

точка 1: $A[1]$ $B[1]$

точка 2: $A[2]$ $B[2]$

...

точка N: $A[N]$ $B[N]$,

где $(A[i]; B[i])$ — координаты i-й вершины контура искомого многоугольника.

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. *Вариант 1.* Можно через концы отрезка провести прямую $y=cx+d$ и определить, принадлежит ли точка пересечения двух прямых, если она существует, отрезку, т. е. мы должны решить уравнение $x(c-k)=(b-d)$, найти $y=kx+b$ и проверить выполнение неравенств $x_1\leqslant x\leqslant x_2$, $y_1\leqslant y\leqslant y_2$. Но при нахождении $(x; y)$ в результате деления могут возникнуть большие вычислительные погрешности или даже переполнение или потеря значимости, в результате чего получится неверный ответ.

Вариант 2. Обозначим $F(x; y)=kx+b-y$. Прямая $kx+b=y$ разбивает плоскость на три части: в одной $F(x; y)>0$, в другой $F(x; y)<0$, и на прямой $kx+b=y$ выполняется $F(x; y)=0$. Если прямая $y=kx+b$ пересекает отрезок, то либо концы отрезка лежат в различных полуплоскостях, либо хотя бы одна концевая точка отрезка лежит на прямой. Это равносильно выполнению неравенства $F(x_1; y_1)F(x_2; y_2)\leqslant 0$. Таким образом, не вычисляя точку пересечения, мы по знаку произведения можем определить, имеют ли прямая и отрезок общую точку. Очевидно, что второй вариант решения задачи предпочтительнее первого.

2. Точки отрезка Z можно описать уравнением

$$p\cdot OB+(1-p)\,OC=z, \qquad (*)$$

где p — число $(0\leqslant p\leqslant 1)$, OB и OC — векторы.

Если существует такое число p $(0 \leqslant p \leqslant 1)$, что $p \cdot OB + (1-p) \cdot OC = A$, то A лежит на отрезке, иначе — нет.

Равенство (*) расписывается покоординатно так:

$$px_1 + (1-p)x_2 = x,$$
$$py_1 + (1-p)y_2 = y.$$

Из первого уравнения находим p, подставляем во второе; если получаем равенство и $0 \leqslant p \leqslant 1$, то A лежит на отрезке, иначе — нет.

3. Найдем какую-нибудь внутреннюю точку $A(x; y)$ выпуклого многоугольника, например точку $A(x; y)$ с координатами $((x_1+x_2+x_3)/3; (y_1+y_2+y_3)/3)$. Такая точка называется центром масс треугольника с вершинами в этих трех точках. На контуре выберем произвольно две последовательные вершины L_1 и L_2 и вычислим углы, которые образуют отрезки $(A; L_1)$ и $(A; L_2)$ с осью Ox. Если первый угол меньше второго, то обход против часовой стрелки, иначе — по часовой.

Можно решать задачу и другим способом, который применим и в случае невыпуклой фигуры. Вначале найдем номер вершины, имеющей минимальную y-координату (пусть ее координаты $(x_0; y_{\min})$). Если таких точек несколько, то берем ту, у которой x-координата минимальна (т. е. самую левую). После этого мы знаем координаты двух точек, одна из которых предшествует в заданном обходе найденной точке, а другая — следует за ней. Пусть их координаты $(x_p; y_p)$ и $(x_s; y_s)$ соответственно. Если

$$\frac{y_p - y_{\min}}{\sqrt{(x_p-x_0)^2+(y_p-y_{\min})^2}} > \frac{y_s - y_{\min}}{\sqrt{(x_s-x_0)^2+(y_s-y_{\min})^2}},$$

то обход по часовой стрелке, иначе — против (значения дробей соответствуют косинусам углов, которые образуют соответствующие стороны многоугольника с прямой, определяемой уравнением $y = y_{\min}$. А так как этот угол

лежит в пределах от 0° до 180°, то меньшему углу соответствует большее значение косинуса).

4. Предположим, мы нашли такую прямую. Будем сдвигать ее в направлении, перпендикулярном этой прямой (параллельный перенос) до тех пор, пока она не пересечет какую-нибудь из концевых точек отрезка. За счет поворота прямой вокруг этой точки мы можем добиться того, что прямая будет проходить через 2 концевые точки отрезков и не перестанет быть решением задачи. Следовательно, мы должны рассмотреть прямые, проходящие через все возможные комбинации пар концевых точек отрезков. Всего надо проверить $(2N-1)+$ $+(2N-2)+\ldots+1=N(2N-1)$ прямых и для каждой из них найти число пересечений с отрезками. Та прямая, у которой это число максимальное, и есть искомая.

При решении возникает подзадача пересечения прямой $ax+by+c=0$ и отрезка с концами $(x_1; y_1)$, $(x_2; y_2)$. (См. алгоритм 1.3.)

5. Строим выпуклую оболочку данного множества точек, т. е. такой выпуклый многоугольник, вершинами которого являются некоторые из этих N точек (возможно, не все). Через какое бы ребро этого многоугольника мы не провели прямую, все N точек исходного множества будут лежать по одну сторону от этой прямой и на ней (определение местоположения точек относительно прямой — см. алгоритм 1.3).

Из произвольной точки a_1 множества A из N точек мы можем провести не более $(N-1)$-го отрезка так, чтобы и вторая концевая точка этого отрезка была из множества A. Берем тот отрезок $[a_1; a_2]$, для которого все точки множества A лежат по одну сторону от прямой, проходящей через этот отрезок (если ни один отрезок не удовлетворяет этому условию, то берем другое a_1; с самого начала лучше всего взять в качестве a_1 точку с максимальной абсциссой, а если таких несколько, то среди них берем точку с максимальной ординатой — это гарантирует, что a_1 принадлежит искомому контуру выпуклого

многоугольника). Для точки a_2 ищем точку $a_3 \neq a_1$ так, чтобы все множество A лежало по одну сторону от прямой, определяемой отрезком $[a_2; a_3]$, ..., для точки a_i ищем точку $a_{i+1} \neq a_{i-1}$ так, чтобы все точки A лежали по одну сторону от прямой, содержащей отрезок $[a_i, a_{i+1}]$, и т. д., до тех пор, пока очередной точкой a_{i+1} не станет a_1, — мы замкнули контур и нашли выпуклую оболочку.

Все точки множества A, не лежащие на контуре, лежат внутри выпуклой оболочки.

6. Предположим, что мы построили искомый выпуклый многоугольник. Если какая-то точка a (из данного в условии множества S из k точек) лежит на стороне многоугольника, то считаем ее новой вершиной этого многоугольника. Можем считать, что все вершины данного многоугольника есть точки множества S (если это не так и между двумя точками a и b из S лежит одна или несколько «посторонних» вершин, то мы можем их отбросить и считать, что a и b — две последовательные вершины контура. Многоугольник при этом у нас остается выпуклым (отрезок $[a; b]$ делит фигуру на два выпуклых многоугольника), а так как на контуре между a и b не лежало ни одной точки из S, то полученный многоугольник удовлетворяет требованиям задачи).

Итак, в качестве решения задачи мы получили выпуклую оболочку множества S (о построении ее см. задачу 5).

Если построенная выпуклая оболочка такова, что каждая точка S является ее вершиной (или лежит на стороне), то задача решена, иначе — решения не существует.

7. Рассмотрим два из возможных алгоритмов.

Вариант 1. Строим выпуклую оболочку данного множества точек (см. задачу 5).

Если все точки множества A лежат на контуре, то задача решена. Если же нет, то ищем точку p с минимальным расстоянием до контура (если таких точек несколько, то берем любую из них). Пусть минимальное расстояние до контура есть расстояние до стороны $(u; v)$.

Вставляем в контур точку p: вместо контура ... u, v, ... будет контур ... u, p, v, ...

Для оставшихся точек повторяем описанную выше процедуру, пока все точки не будут вставлены в контур.

Расстояние от точки до стороны — это либо длина перпендикуляра, опущенного из точки на сторону, если проекция точки попадает на отрезок, либо минимальное из расстояний от точки до концевых точек стороны. Расстояние от точки $z(u; v)$ до ее проекции на прямую

$$Ax+By+C=0 \text{ есть } d=|Au+Bv+C|/\sqrt{A^2+B^2}.$$

Вариант 2. Строим выпуклую оболочку V данного множества точек.

Если все точки множества A лежат на контуре, то задача решена. Иначе, обозначим через A_1 все внутренние точки выпуклой оболочки V. Строим для нового множества A_1 выпуклую оболочку V_1 (контуры V и V_1 не пересекаются!).

«Склеиваем» два контура следующим образом.

Выберем по паре последовательных вершин p, s и p_1, s_1 на контурах V и V_1 соответственно так, чтобы в четырехугольнике с вершинами s, p, p_1, s_1 не лежало больше никаких других точек контуров V и V_1. Разрываем контуры V и V_1 (убирая ребра $(p; s)$ и $(p_1; s_1)$) и объединяем их (добавляя ребра $(p; p_1)$ и $(s; s_1)$).

Если внутри V_1 нет внутренних точек, то задача решена, иначе — с внутренними точками V_1 проделываем те же самые операции: находим выпуклую оболочку и пары последовательных точек на контурах, разрываем и «склеиваем» контуры и т. д., пока не получим, что последняя построенная выпуклая оболочка содержит в себе 0, 1 или 2 точки.

Если точек 0, то задача решена. В противном случае присоединяем точки к ранее образованному контуру так, чтобы фигура осталась многоугольником (можно проводить присоединение, как и в варианте 1).

8. У клеточной фигуры могут быть следующие оси симметрии — горизонтальная, вертикальная и идущие

под углом 45° и 135° (т. е. 4 оси симметрии, как у квадрата). Оси могут проходить как через центр какой-то клетки, так и по стороне. Например, фигуры

```
***
*** и ***
```

имеют по 2 оси симметрии — горизонтальную и вертикальную, а фигура * — все 4 оси симметрии.

Введем систему координат таким образом, что каждая зарисованная клетка представляется точкой с целочисленными координатами.

Находим возможный центр симметрии фигуры, имеющий координаты $((x_{max}+x_{min})/2;\ (y_{max}+y_{min})/2)$, где x_{max}, y_{max}, x_{min} и y_{min} соответственно максимальные и минимальные иксовые и игрековые координаты точек в заданной нами системе координат:

$$x_{max}=\max\{x_i\},\qquad y_{max}=\max\{y_i\},$$
$$x_{min}=\min\{x_i\},\qquad y_{min}=\min\{y_i\}.$$

Если у фигуры есть ось симметрии, то она проходит через возможный центр симметрии фигуры.

Рассматриваем 4 возможные оси симметрии, проходящие через этот центр. Определяем, является ли фигура симметричной относительно каждой из осей (для удобства этот центр можно считать началом системы координат). При симметрии относительно горизонтальной (вертикальной) оси каждой клетке фигуры $(x;\ y)$ должна соответствовать клетка с такой же иксовой (игрековой) координатой, но с обратной по знаку другой координатой, т. е. $(x;\ -y)$ (для вертикальной оси соответственно $(-x;\ y)$). При симметрии относительно оси с наклоном 45° (−45°) каждой клетке $(x;\ y)$ фигуры должна соответствовать клетка с координатой $(y;\ x)$ (соответственно $(-y;\ -x)$).

9. Отрезки, соединяющие точки разбиений R_1 и R_2, параллельны стороне AD. Будем брать отрезки, соединяющие последовательные точки разбиений R_3 и R_4, и искать между этими двумя последовательными отрезками

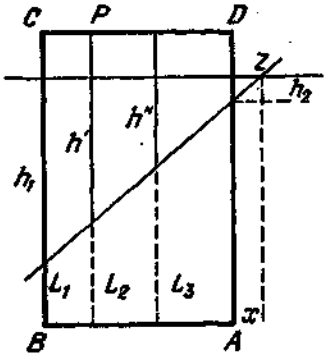


Рис. 26

четырехугольник с наибольшей площадью. Четырехугольник разбиения Q с максимальной площадью есть четырехугольник с максимальной площадью по всем таким разбиениям прямоугольника отрезками.

Пусть последовательные точки разбиения с одинаковыми номерами на сторонах BC и AD есть соответственно f_1, f_2 и g_1, g_2.

Если $f_2-f_1=g_2-g_1$, то анализируемые четырехугольники есть прямоугольники, и прямоугольник с максимальной площадью определяется максимальной длиной отрезка в разбиении R_1 (или, что то же, R_2).

Пусть для определенности $f_2-f_1>g_2-g_1$ (случай обратного неравенства рассматривается аналогично). Обозначим $f_2-f_1=h_1$, $g_2-g_1=h_2$, $p=CD$, h' и h'' — длины левой и правой сторон четырехугольника, L_2 — расстояние между двумя этими сторонами, Z — точка пересечения продолжения верхней и нижней сторон четырехугольника: L_1, L_3 — соответственно расстояния от левой и правой стенок прямоугольника $ABCD$ до четырехугольника, x — расстояние от точки Z до прямоугольника $ABCD$, S — площадь четырехугольника. Получим следующую схему (рис. 26) и равенства:

$$\frac{x}{h_2}=\frac{x+p}{h_1};$$

$$S=(h'+h'')\,L_2/2;$$

$$\frac{x}{h_2}=\frac{x+L_3}{h''}=\frac{x+L_2+L_3}{h'}.$$

Откуда

$$x=ph_2/(h_1-h_2);$$

$$h'=h_2(x+L_2+L_3)/x;$$

$$h''=h_2(x+L_3)/x.$$

Для каждой пары отрезков, определяемых точками разбиений R_3 и R_4, находим максимальную площадь четырехугольника между этими двумя отрезками.

10. Пусть координаты точек $x_1, \ldots, x_N$ не убывают (если это не так, то просто отсортируем предварительно последовательность).

Вариант 1. Предположим, что мы нашли точку Z с координатой x_0 и она лежит на интервале $(x_i; x_{i+1})$. Справа от нее i точек, слева — $(N-i)$. Сумма расстояний $S_{\min}$ будет такой:

$$S_{\min}=(x_0-x_1)+\ldots+(x_0-x_i)+(x_{i+1}-x_0)+\ldots+(x_N-x_0).$$

Предположим, что $i>N-i$ и мы в пределах интервала сдвигаем точку Z влево на какую-то маленькую величину d $(d<x_0-x_i)$. Получаем новую сумму S:

$$(x_0-d-x_1)+\ldots+(x_0-d-x_i)+(x_{i+1}-(x_0-d))+\ldots+ \\ +(x_N-(x_0-d))=S_{\min}-di+d(N-i).$$

А так как по предположению $i>N-i$, то $S<S_{\min}$.

Ситуация, когда $N-i>i$, исследуется аналогично — делаем сдвиг на величину $d<x_{N+1}-x_0$ вправо, не выходя при этом за границы интервала, и опять же получаем, что новая сумма $S<S_{\min}$. Случай, когда Z совпадает с одной из точек x_i, исследуется так же, с использованием маленьких сдвигов.

Следовательно, для того чтобы точка Z была искомой, необходимо и достаточно, чтобы справа и слева от нее лежало одно и то же число точек. Если $N=2k$, то точка Z может быть любой из точек отрезка $[x_k; x_{k+1}]$, если же $N=2k+1$, то точка Z имеет координату x_{k+1}.

Вариант 2. Пусть мы решаем задачу для N точек на прямой.

Точка Z должна, очевидно, лежать на отрезке $[x_1; x_N]$.

Если $N=1$, то данная точка и является искомой.

Если $N=2$, то Z может лежать где угодно на отрезке $[x_1; x_N]$ — суммарное расстояние будет одинаковым и равным длине отрезка.

Если $N>2$, то суммарное расстояние от точки Z до точек с минимальной и максимальной координатами (т. е. до точек x_1 и x_N) не зависит от местоположения точки Z и равно длине отрезка $[x_1; x_N]$.

Так как суммарное расстояние до этих двух точек постоянно, то поэтому мы их можем не рассматривать и решать далее задачу уже для $(N-2)$ точек $x_2, \ldots, x_{N-1}$. Проведя необходимое число раз сокращение количества точек, мы придем к уже рассмотренным случаям одной или двух точек.

Окончательно получаем: если $N=2k$, то точка Z может быть любой из точек отрезка $[x_k; x_{k+1}]$, если же $N=2k+1$, то точка Z имеет координату x_{k+1}.

11. Переформулируем задачу: На плоскости своими координатами задаются N точек $P_i(x_i; y_i)$. Построить окружность минимального радиуса с центром на оси абсцисс так, чтобы она содержала внутри себя и на своей границе все эти точки.

Везде в дальнейшем будем обозначать через $C(P; Z)$ окружность с центром в точке $(P; O)$ и проходящую через точку $Z(x; y)$.

Очевидно, что на искомой окружности лежит, по меньшей мере, одна точка. Действительно, в противном случае мы можем, не меняя центра окружности, уменьшить ее радиус, а это противоречит предположению о том, что нами была построена окружность минимального раднуса.

Докажем следующее простое утверждение: если на искомой окружности лежит единственная точка, то центр окружности есть проекция этой точки на ось абсцисс.

Предположим противное — на минимальной окружности лежит единственная точка $Z(x\ y)$, а центр ее не совпадает с $(x; 0)$. Если мы начнем понемножку двигать центр окружности, проходящей через точку Z, в направлении $(x; 0)$, то, так как все точки, кроме Z, лежат внутри окружности, до какого-то момента они и будут оставаться внутри нее. Таким образом мы можем хоть чуть-чуть,

но сдвинуть центр, уменьшив при этом радиус окружности, содержащей все точки. Получаем противоречие с предположением о минимальности радиуса.

Следствие. Если на искомой окружности лежит только одна точка, то это точка с максимальной по модулю ординатой.

Отметим, что окружность с центром на оси абсцисс единственным образом определяется двумя лежащими на ней точками (центр этой окружности — это точка пересечения оси абсцисс и серединного перпендикуляра к отрезку, соединяющего эти две точки).

Вариант 1.

Шаг 1. Ищем точку $(x_i; y_i)$ с максимальной по модулю ординатой y_i (если таких точек несколько и у них разные абсциссы, то перейти на Шаг 2) и для окружности $C(x_i; (x_i; y_i))$ проверяем, содержит ли она все N точек. Если да, то задача решена, если нет, то переходим к Шагу 2.

Шаг 2. Среди окружностей, определяемых всевозможными парами точек $(P_i; P_j)$, находим те, которые содержат все точки, а затем выбираем из них окружность минимального радиуса.

Пар точек, которые могут определять окружности, всего $N(N-1)/2$, т. е. порядка N^2, следовательно, и возможных окружностей тоже порядка N^2. Для проверки принадлежности N точек каждой окружности требуется порядка N операций. Получаем, что сложность этого алгоритма порядка N^3. (Когда мы говорим о сложности алгоритма, то рассматриваем только зависимость роста числа требуемых операций от числа N, игнорируя все константные множители и медленно растущие слагаемые.) Рассмотрим другой способ решения этой задачи, основанный на более глубоком ее анализе.

Вариант 2.

Проверка по Шагу 1 ранее изложенного алгоритма остается без изменения. Пусть искомая окружность не найдена. Для обоснования Шага 2 докажем следующее утверждение:

Пусть окружность с центром $(P_{ij}; 0)$ определяется точками $P_i(x_i; y_i)$ и $P_j(x_j; y_j)$. Она только тогда может быть содержащей все точки окружностью C минимального радиуса, когда $(P_{ij}; 0)$ лежит на ортогональной проекции отрезка $[(x_i; y_i); (x_j; y_j)]$ на ось абсцисс, т. е. должны выполняться неравенства $x_i \leqslant P_{ij} \leqslant x_j$.

Окружность C с центром $(P_{ij}; 0)$ должна проходить не менее чем через две точки заданного множества из N точек, и при этом из исходных точек всегда можно выбрать две такие (обозначим их $P_i(x_i; y_i)$ и $P_j(x_j; y_j)$), что $x_i < P_{ij} < x_j$. Действительно, если бы абсциссы всех лежащих на окружности точек были, например, меньше P_{ij}, то $(P_{ij}; 0)$ можно было бы сместить влево по оси абсцисс на некоторую величину с уменьшением радиуса охватывающей все точки окружности, что противоречит минимальности найденной ранее окружности.

Ни одна из точек, лежащих на окружности, не может иметь абсциссы P_{ij} вследствие невыполнения условия Шага 1.

Итак: всегда можно найти две лежащие на окружности точки P_i и P_j с абсциссами соответственно меньше и больше абсциссы центра окружности. Эти точки определяют центр окружности $(P_{ij}; 0)$ — точку пересечения серединного перпендикуляра к отрезку $[P_i; P_j]$ с осью абсцисс. При этом точка $(P_{ij}; 0)$, естественно, будет лежать на проекции отрезка $[P_i; P_j]$ на ось абсцисс.

Рассматривая все пары точек $(P_i; P_j)$, таких, что точка пересечения $(P_{ij}; 0)$ серединного перпендикуляра к отрезку $[P_i; P_j]$ с осью абсцисс лежит на проекции отрезка $[P_i; P_j]$ на ось абсцисс, получаем, что центр искомой окружности минимального радиуса совпадает с одной из таким образом полученных точек. Каждая из рассматриваемых пар точек $(P_i; P_j)$ определяет окружность минимального радиуса R_{ij}, содержащую эти две точки.

Из всего вышесказанного получаем, что интересующая нас окружность минимального радиуса, содержа-

щая N точек, должна иметь максимальный из всех полученных радиусов R_{ij}.

Всего пар точек $(P_i; P_j)$ не более $(N^2-N)/2$, и, следовательно, сложность алгоритма:

$$O((N^2-N)/2)=O(N^2-N)=O(N^2).$$

Здесь мы, как и обычно, избавляемся от констант и медленно растущих слагаемых.

Вариант 3.

Все вычисления на машине проводятся с ограниченной точностью, с определенным числом знаков после запятой. Поэтому нам бывает достаточно только указать, что интересующая нас точка лежит внутри отрезка заранее заданной длины *epsilon*. *Epsilon* задается пользователем. Например, если мы хотим найти координату точки с точностью 5 знаков после запятой, то $epsilon=10^{-6}$.

В отличие от варианта 2 мы не будем брать все перпендикуляры, попадающие на проекции отрезков, и искать среди получаемых окружностей окружность с максимальным радиусом. Наоборот, описанным ниже способом будем выбирать точку на оси абсцисс и проверять, является ли она искомой или нет.

Из варианта решения 2 можно сделать вывод, что искомая точка лежит на отрезке $[A_0, B_0]=[\min\{x_i\}, \max\{x_i\}]$. Пусть $C_0=(A_0+B_0)/2$ — середина этого отрезка, а $L=B_0-A_0$ — его длина.

Обозначим:

$Dl(C_0)$ — максимальное из расстояний от точки $(C_0; 0)$ до точек $(x_i; y_i)$ с абсциссами $x_i \leqslant C_0$;

$Dr(C_0)$ — максимальное из расстояний от точки $(C_0; 0)$ до точек $(x_i; y_i)$ с абсциссами $x_i \geqslant C_0$.

Опишем i-й итеративный (повторяющийся) шаг алгоритма $(i=0, 1, \ldots)$.

ЕСЛИ $B_i-A_i \leqslant epsilon$

ТО центр окружности лежит на отрезке $[A_i; B_i]$ и желаемая точность достигнута. Стоп.

ИНАЧЕ

Вычисляем $C_i=(A_i+B_i)/2$

Находим $Dl(C_i)$ и $Dr(C_i)$

ЕСЛИ $Dl(C_i<Dr(C_i)$

ТО искомая точка не может лежать на промежутке $[A_i;\ C_i]$, так как радиус любой содержащей N точек окружности с центром на этом промежутке больше $Dr(C_i)$ (проверьте сами!), а окружность с центром C_i имеет радиус $Dr(C_i)$. Поэтому центр искомой окружности лежит на отрезке $[C_i;\ B_i]$, который мы обозначим $[A_{i+1};\ B_{i+1}]$

ИНАЧЕ

ЕСЛИ $Dr(C_i)<Dl(C_i)$

ТО получаем, что центр искомой окружности лежит на $[A_i;\ C_i]$, который мы обозначим $[A_{i+1};\ B_{i+1}]$

ИНАЧЕ

ЕСЛИ $Dr(C_i)=Dl(C_i)$

ТО C_i — центр искомой окружности. Стоп

Конец i-го итеративного шага. Выполнить шаг $i+1$

Мы видим, что длина L начального отрезка на каждом шаге уменьшается вдвое. Алгоритм, вообще говоря, заканчивает работу при выполнении условия

$$L/(2^S)\leqslant epsilon.$$

Примечание. Таким образом, требуется не более чем $S=[\log_2(L/epsilon)]+1$ шагов, где $\log_2$ — это логарифм по основанию 2.

Так как на каждом шаге (для вычисления Dl и Dr) выполняется не более $O(N)$ операций, то всего их потребуется порядка

$$O(N\log_2(L/epsilon)).$$

12. Это переборная задача. Обратите внимание, что стороны квадрата могут и не быть параллельны осям координат! Каждую из N точек мы последовательно рассматриваем в качестве верхнего левого угла квадрата, каждую из оставшихся $N-1$ — как нижнюю правую вершину и смотрим, есть ли для них в этом множестве из N точек точки, соответствующие верхнему правому и нижнему левому углам. Если да, то подсчитываем, сколько точек лежит в данном квадрате.

Пусть координата левого верхнего угла $(x_1; y_1)$, нижнего правого — $(x_2; y_2)$, тогда координата пересечения диагоналей квадрата:

$$((x_1+x_2)/2;\ (y_1+y_2)/2);$$

координата верхнего правого угла:

$$((x_1+x_2)/2+[y_1-(y_1+y_2)/2];$$
$$(y_1+y_2)/2+[x_1-(x_1+x_2)/2])=((x_1+x_2+y_1-y_2)/2;$$
$$(x_1-x_2+y_1+y_2)/2),$$

нижнего левого:

$$((x_1+x_2-y_1+y_2)/2;\ (-x_1+x_2+y_1+y_2)/2).$$

Для $(x_1; y_1)$ и $(x_2; y_2)$ должны выполняться следующие неравенства: $x_1 \leqslant x_2$, $y_1 \geqslant y_2$ (иначе это будут уже не левый верхний и правый нижний углы квадрата).

13. Из координат вершин прямоугольников формируем массивы: массив X_{kor}, содержащий x-координаты вершин прямоугольников, связанный с ним массив X_{nom}, содержащий соответствующие номера прямоугольников, аналогично формируются массивы Y_{kor} и Y_{nom} для y-координат. При этом координате вершины ставится в соответствие номер со знаком «+», если вершина левая нижняя, и знак «—», если правая верхняя.

Затем массивы координат сортируют в порядке неубывания (при этом соответствие между координатами и номерами прямоугольников сохраняется), причем для одинаковых координат вначале должны располагаться номера левых (нижних) вершин (т. е. со знаком «+»), а затем номера правых (верхних).

Задача решается в два этапа. На первом этапе решается задача нахождения максимального числа взаимно пересекающихся отрезков на оси Ox (в качестве отрезков берутся проекции прямоугольников на ось Ox).

Максимальное число взаимно пересекающихся отрезков находится следующим образом: просматриваем массив X_{kor} от начала к концу и увеличиваем на 1 переменную, соответствующую текущему числу пересечений, ес-

ли просматриваемая координата соответствует левому концу отрезка, и уменьшаем эту переменную на 1, если текущая координата соответствует правому концу отрезка.

Максимальное число взаимно пересекающихся отрезков есть максимальное значение текущего числа пересечений. Заметим, однако, что если для двух прямоугольников их проекции на оси Ox и Oy пересекаются, то прямоугольники пересекаются.

На втором этапе ищется пересечение прямоугольников. Возьмем некоторую прямую вида $y=C$ (параллельную оси Ox) и все прямоугольники, которые она пересекает, будем называть активными. Эти прямоугольники в массиве *ACTIV* пометим 1, остальные — 0.

Теперь, используя результаты первого этапа, с учетов массива *ACTIV* находится максимальное число взаимно пересекающихся прямоугольников, которые являются активными для рассматриваемого значения C (ищется максимальное число пересекающихся активных прямоугольников).

Теперь определим возможные значения C. Можно ограничиться только теми значениями y, которые соответствуют концевым точкам проекций прямоугольников на ось Oy. Следовательно, формирование массива *ACTIV* может осуществляться по следующему принципу: отсортировав y-координаты проекций по неубыванию значений (с учетом того факта, что для нескольких одинаковых координат вначале располагаются координаты, соответствующие верхним концам отрезков, а затем располагаются координаты, соответствующие нижним концам отрезков).

Просматривая массив от начала к концу, мы активизируем прямоугольник, если текущая координата соответствует нижнему концу проекции, или отменяем активность прямоугольника, если текущая координата соответствует верхнему концу проекции. Процесс форми-

рования массива *ACTIV* начинаем при значении переменной *C*, равной минимальной *y*-координате (вначале все элементы массива равны 0). Алгоритм заканчивает работу при значении переменной *C*, равной максимальной *y*-координате.

14. Понятно, что если есть свеча с нулевыми координатами или какие-нибудь две свечи лежат на прямой, проходящей через начало координат, по разные стороны от начала координат, то решения не существует.

Пусть таких свеч нет. Проведем линию через центр и первую свечу (пусть это точка *A*). Если все свечи оказались по одну сторону от прямой, то решение построено. Предположим, что существуют свечи по разные стороны прямой. Определим направление прямой от центра к свече, и пусть *M* — множество точек, лежащих по правую сторону от прямой. Определим среди них точку *B*, для которой угол *AOB* максимальный и лежит в пределах от 0° до 180°. Проведя прямую через точки *O* и *B*, проверяем, лежат ли все свечи по одну сторону от нее. Если да, то решение найдено. Если нет, то решения нет.

15. Припишем сторонам каждого из *N* прямоугольников ориентацию: левая сторона считается идущей сверху вниз (ориентацию обозначим 1), нижняя — слева направо (2), правая — снизу вверх (3), верхняя — справа налево (4). Найдем точки пересечения всех *N* прямоугольников. Обозначим это множество точек *S*. Добавим в *S* угловые точки всех прямоугольников. Каждой из точек *S* припишем пару, состоящую из двух ориентаций, соответствующих ориентации тех ребер, пересечением которых точка является.

Найдем в множестве *S* точку с максимальной ординатой. Если таких точек несколько, то возьмем среди них точку P_0 с минимальной абсциссой. Эта точка лежит на верхней части контура объединения прямоугольников и является левым верхним углом какого-то прямоугольника. Печатаем P_0. Будем двигаться от P_0 вниз по ребру,

пока не встретим одну из точек S (это будет либо точка — вершина прямоугольника, либо точка — пересечение сторон). Обозначим эту точку P_1. Ей приписана пара ориентаций $(O_1; O_2)$, одна из ориентаций (пусть, например, O_1) есть 1 (это то ребро, по которому мы пришли в P_1). Печатаем P_1 — очередную вершину контура, и двигаемся из точки P_1 (по ребру какого-то прямоугольника) в направлении O_2, пока не достигнем еще какой-нибудь вершины из S. Обозначим ее P_2. У нее пара ориентаций $(O_1'; O_2')$. Пусть, например, $O_2' = O_2$, тогда мы из очередной вершины контура P_2 будем двигаться в направлении O_1' и т. д., пока не достигнем вершины P_0. Контур выписан.

Определим, есть ли в контуре «дырки». Занесем в массив $A[1..2, 1..2N]$ в строку 1 все ординаты вершин прямоугольников без повторений и отсортируем этот массив по первой строке по возрастанию. Предположим, что в массиве A хранится всего S различных ординат: $A[1, 1], \ldots, A[1, s]$. Сначала все $A[2, i]=0$, $i=1, \ldots, S$.

Определим массив $B[1..4, 1..N]$. В первой строке массива B располагаются в порядке неубывания x-координаты левых нижних и правых верхних вершин всех N прямоугольников. Пусть $B[1, i]$ — x-координата какой-то вершины P; $B[2, i]$ и $B[3, i]$ — соответственно ординаты нижней и верхней вершин той вертикальной стороны прямоугольника, на которой лежит P; $B[4, i]=0$, если эта вертикальная сторона прямоугольника левая, и 1, если — правая.

Воспользуемся методом сканирующей прямой. Будем брать по возрастанию индекса i элементы $B[1, i]$ массива B.

Если $B[4, i]=0$, то будем увеличивать на 1, а если $B[4, i]=1$, то уменьшать на 1 все элементы $A[2, j]$ такие, что $B[2, i] \leqslant A[1, j] < B[3, i]$ ($A[2, j]$ будет равно количеству прямоугольников, содержащих внутри себя или на границе отрезок $A[1, j] \leqslant y \leqslant A[2, j]$, $x=B[1, i]$).

Если какое-то $A[2, j]=0$, то это означает, что прямоугольники при $x=B[1, i]$ не покрывают интервал $y=(A[1, j]; A[1, j+1])$.

Если мы найдем такие i, j и k, что при $x=B[1, i]$ интервал $(A[1, j]; A[1, k+1])$ не покрыт многоугольниками (т. е. $A[2, j]=A[2, j+1]=\ldots=A[2, k]=0$), а интервалы $(A[1, j-1]; A[1, j])$ и $(A[1, k+1]; A[1, k+2])$ покрыты (т. е. $A[2, j-1]>0$ и $A[2, k+1]>0$), и точка $(x; y)=(B[1, i]; A[1, j])$ не принадлежит внешнему контуру фигуры — объединению прямоугольников, то у фигуры есть по крайней мере одна «дырка». Чтобы, при необходимости, выписать контур «дырки», поступим, как и в случае нахождения внешнего контура,— пойдем по ребрам, образующим контур «дырки», но обход контура необходимо будет осуществлять по часовой стрелке, т. е. против ориентации сторон.

16. Контур может иметь излом лишь в точках, лежащих на стенах зданий. Занесем в массив A координаты $L[i]$ и $R[i]$ и отсортируем его по неубыванию. Заметим также,что между двух соседних стен (определяемых каждыми двумя соседними элементами массива) высота контура остается постоянной. Поэтому для каждых двух соседних элементов массива A найдем их полусумму (координату точки, лежащей между стенами) и вычислим высоту контура в этой точке; после этого мы будем знать высоты всех горизонтальных площадок и координаты начала и конца этих площадок.

Будем выписывать контур точка за точкой, начиная с самой левой точки контура.

Если две соседние горизонтальные площадки имеют одинаковую высоту, то мы их «склеиваем», т. е. рассматриваем как одну площадку.

Если же две соседние площадки различаются по высоте, то, следовательно, надо выписать вертикальный излом контура.

17. Считаем, что точки в S не дублируются (так как S — множество). Введем в множество S точки $(0; w)$, $(0; 0)$, $(v; 0)$, $(v; w)$ — вершины A. Будем использовать два двумерных массива — BX и BY; в массиве BX располагаются координаты точек множества S в порядке

неубывания абсциссы, в BY — по невозрастанию ординаты. Пара ($BX[i, 1]$; $BX[i, 2]$) (аналогично ($BY[j, 1]$; $BY[j, 2]$)) есть x- и y-координаты точки из S.

Рассмотрим множество прямоугольников P_i, удовлетворяющих условию задачи. Тот из них, который имеет максимальную площадь, и является искомым. Очевидно, что на каждой из сторон P_i должна лежать точка из S либо сторона P_i должна лежать на стороне A. Рассмотрим следующие случаи.

1) Верхняя сторона P лежит на верхней стороне прямоугольника A. Для каждой точки $BX[i]$ ищем, двигаясь по массиву BX вправо и влево от элемента $BX[i]$, такие первые $BX[j]$ и $BX[k]$, $j<i$, $k>i$, что $BX[j, 2]>BX[i, 2]$, $BX[k, 2]>BX[i, 2]$.

Считаем, что стороны прямоугольника P_i проходят: нижняя — через точку $BX[i]$, левая — через $BX[j]$, правая — через $BX[k]$. Верхняя сторона лежит на верхней стороне A.

Таким образом находим все прямоугольники, примыкающие к верхней стороне.

Прямоугольники, примыкающие к нижней, левой и правой сторонам A, находятся аналогично, но в двух последних случаях надо вместо BX использовать массив BY.

2) Ни одна из сторон P_i не лежит на стороне A. Берем последовательно точки массива $BY[i]$, $i=1, \ldots, N$, и считаем, что верхняя сторона P_i проходит через точку $BY[i]$. Для этой точки полагаем сначала, что $X_{Begin}=0$, $X_{End}=V$. При просмотре массива BY вправо от элемента $BY[i]$ находим такие точки $BY[j]$ и $BY[k]$, которые первыми удовлетворят условиям $BY[j, 1]\geqslant X_{Begin}$, $BY[k, 1]\leqslant X_{End}$, $BY[j, 1]<BY[i, 1]$, $BY[k, 1]>BY[i, 1]$ (объяснение смотрите ниже), т. е. мы находим точки с максимальной ординатой, через которые можно провести правую и левую стороны прямоугольника P_i. Внутри интервала ($BY[j, 1]$; $BY[k, 1]$) на оси Ox ищем точку $BY[r, 1]$ такую, что y-координата этой точки $BY[r, 2]$ макси-

мальная, меньше величины $\max\{BY[j, 2], BY[k, 2]\}$. Через эту точку $BY[r]$ проведем нижнюю сторону прямоугольника. Полагаем $X_{Begin}=BY[j, 1]$, $X_{End}=BY[k, 1]$.

Но этим прямоугольником может не исчерпываться все множество прямоугольников, у которых точка $BY[i]$ лежит на верхней стороне. Попытаемся найти еще один из таких прямоугольников. Очевидно, что левая его сторона имеет x-координату не меньше чем X_{Begin}, а правая — не больше чем X_{End}. Будем искать такую точку $BY[r]$, что $X_{Begin}\leqslant BY[r, 1]\leqslant X_{End}$ (теперь уже понятно почему), $BY[r, 2]$ — максимальная из всех ординат, меньше $\max\{BY[j, 2], BY[k, 2]\}$ (мы «сужаем» прямоугольник как можно незначительнее). Если $BY[r, 1]<BY[i, 1]$, то это новая левая сторона, иначе — новая правая. Находим новую нижнюю сторону и т. д. Если мы не можем найти нового значения $BY[r]$, то просмотр прямоугольников с точкой $BY[i]$ на верхней стороне закончен, и мы переходим к $BY[i+1]$.

18. Длина стороны первого квадрата — 1, второго — 1, третьего — 2, четвертого — 3 и т. д. Видно, что длины сторон есть числа Фибоначчи, определяемые следующим рекуррентным соотношением:

$$u(1)=1,\ u(2)=1,\ u(N)=u(N-1)+u(N-2).$$

Будем хранить координаты четырехугольника A_i — объединения квадратов с номерами от 1 до i.

Второй квадрат A_2 рисуется справа от первого A_1, A_3 — сверху от A_2, A_4 — слева от A_3, A_5 — снизу от A_4, A_6 — опять справа от A_5 и т. д.

Как только точка P впервые попадает в A_i, распечатываем номер i. Проверка принадлежности точки P четырехугольнику с параллельными сторонами: пусть левый верхний угол $(x_1; y_1)$, правый нижний $(x_2; y_2)$; точка $P(P_x; P_y)$ принадлежит четырехугольнику, если одновременно $x_1\leqslant Px\leqslant x_2$ и $y_2\leqslant Py\leqslant y_1$.

19. Отсортируем координаты точек в порядке неубывания x-координат, а в случае одинаковых x-координат в порядке невозрастания y-координат. Находим коорди-

наты средней точки (находящейся в позиции (n *div* 2+ +1) отсортированного массива координат). Пусть эта точка имеет координаты (x_0; y_0). При этом множество точек оказалось разбитым на 3 части: точки, лежащие на прямой $x=x_0$; точки, лежащие левее прямой $x=x_0$; точки, лежащие правее прямой $x=x_0$. Представим, что точки, лежащие левее прямой $x=x_0$, лежат в пределах прямоугольника, x-координата правого края которого равна x_1 ($x_1<x_0$), а точки, лежащие правее прямой $x=x_0$, лежат в пределах прямоугольника, x-координата левого края которого равна x_2 ($x_2>x_0$). При этом верхний и нижний края обоих прямоугольников имеют координаты y_{max} и y_{min} соответственно, где y_{max} — максимальная y-координата точек, y_{min} — минимальная y-координата.

Тогда существует прямая с достаточно большим углом наклона (например, с углом наклона, тангенс которого превышает величину $(y_{max}-y_{min}+2)/Z$, где $Z=$ $=\min(x_0-x_1, x_2-x_0)$, которая разделяет эти части. Осталось разделить только точки на прямой так, чтобы количество точек в получившихся частях было равным (т. е. найти точку пересечения разделяющей прямой с прямой $x=x_0$). Если количество точек нечетно, то разделяющая прямая проходит через среднюю точку, иначе — над средней точкой отсортированного массива, но под предыдущей, если та лежит на прямой $x=x_0$.

Таким образом:

x_1 — может быть найдена просмотром массива x-координат справа налево от средней точки до нахождения первой координаты, отличной от x_0; если такая координата не найдена, то $x_1=x_0-1$;

x_2 — может быть найдена просмотром массива x-координат слева направо от средней точки до нахождения первой координаты, отличной от x_0; если такая координата не найдена, то $x_1=x_0+1$;

y_1 — это y-координата точки, предшествующей средней точке, если ее x-координата равна x_0 или равна y_0+1, если x-координата точки, предшествующей средней точке, не равна x_0.

После того как x_0, y_0, x_1, y_1 найдены, осталось написать уравнение прямой с тангенсом угла наклона $(y_{max}-y_{min}+2)/Z$, проходящей через точку с координатами $(x_2; y_2)$, определяемую по следующему правилу:

если N — четно,

то $x_2=x_0$; $y_2=(y_0+y_1)/2$,

иначе $x_2=x_0$; $y_2=y_0$.

20. Проведем через каждую вершину этих двух выпуклых многоугольников прямые, параллельные оси Oy. Эти прямые разбивают всю плоскость на полосы. Пересечение каждой полосы с выпуклым многоугольником образует трапецию, поэтому внутри каждой полосы пересечением двух выпуклых многоугольников будет пересечение двух четырехугольников. Собираем все эти пересечения в одну фигуру, удаляя при этом ложные вершины, которые возникают на границах между полосами.

Объединение делается аналогично.

21. Если проекция точки Z попадает на сторону многоугольника, а не на ее продолжение, то минимальное расстояние от точки Z до стороны есть длина проведенного перпендикуляра. Если же проекция точки Z попадает на продолжение стороны, то минимальное расстояние есть минимум из расстояний от Z до концевых точек этой стороны.

Минимальное расстояние от точки Z до контура есть минимум из расстояний от точки Z до каждой из сторон.

22. Проверяем, лежит ли точка Z на каком-либо отрезке. Если нет, то проводим отрезок, концевые точки которого Z, и, например, точка с номером 1. Находим ближайшую к Z точку пересечения этого отрезка и сторон многоугольников, на которые разбивается плоскость. Пусть эта точка принадлежит стороне Ab. Для треугольника ZAB сначала определим направление обхода контура от A к B (задача 3). Для стороны AB ищем следующую, смежную с ней, сторону BC контура, которая образует максимальный по величине угол с отрезком BA

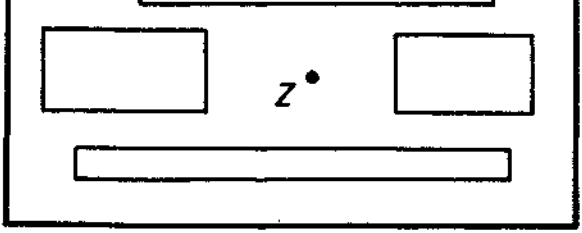


Рис. 27

(угол отсчитывается от отрезка AB по часовой или против часовой стрелки в зависимости от того, по или против часовой стрелки обход). Находим замкнутый контур и определяем, находится точка Z внутри него или снаружи; если снаружи, то удаляем из фигуры все ребра этого контура и повторяем процесс (рис. 27).

23. Подобные многоугольники могут быть зеркально симметричны. Фигура на плоскости полностью характеризуется матрицей расстояний $C[i, j]$, где $C[i, j]$ — расстояние от вершины i до вершины j. Для каждой из введенных фигур строим свою матрицу расстояний и проверяем, можем ли мы получить из одной матрицы другую перестановкой строк и столбцов и умножением всех элементов матрицы на одно и то же число.

24. Дадим другую интерпретацию этой задачи: есть N отрезков, описываемых уравнениями

$$a_i x + b_i = y,\ x_0 \leqslant x \leqslant x_1,\ i = 1, \ldots, N.$$

Предположим, что отрезки не совпадают.

1) Найти верхний контур объединения фигур

$$a_i x + b_i \leqslant y,\ i = 1, \ldots, N,\ x_0 \leqslant x \leqslant x_1,$$

(т. е. найти такую разбивку $t_0, \ldots, t_p$ отрезка $[x_0, x_1]$ и те кусочки отрезков $a_i x + b_i = y$, $t_j \leqslant x \leqslant t_{j+1}$, $j = 0, \ldots, p-1$, что отрезок $a_i x + b_i$ лежит не ниже любого другого отрезка при $t_j \leqslant x \leqslant t_{j+1}$).

2) Найти такую разбивку отрезка $[x_0, x_1]$ точками S_j, что в каждом секторе $S_i < x < S_{i+1}$ кусочки отрезков $a_i x + b_i$, $i = 1, \ldots, N$ не пересекаются, а на границах сектора, при $x = S_i$ и при $x = S_{i+1}$, лежит по крайней мере по одной точке пересечения отрезков $a_i x + b_i$, $i = 1, \ldots, N$.

Решим сначала пункт 2) задачи, затем пункт 1).

Найдем все точки пересечения отрезков $a_i x + b_i$, $i = 1, \ldots, N$, друг с другом. Добавим в это множество точки x_0 и x_1. Упорядочим точки пересечения по возрастанию (если в последовательности встречаются несколько точек с одним и тем же значением, то оставляем из них только одну). Получаем таким образом последовательность $S_0, \ldots, S_Q$.

Для каждого отрезка $[S_j, S_{j+1}]$, $j = 0, \ldots, Q-1$ находим его середину $Z_j = (S_j + S_{j+1})/2$, вычисляем значения $f_{ij} = a_i Z_j + b_i$, $i = 1, \ldots, N$, сортируем их по возрастанию. Индексы i значений f_{ij} в отсортированной последовательности для фиксированного j как раз и есть искомая перестановка $(i_{1j}, i_{2j}, \ldots, i_{Nj})$ чисел 1, 2, 3, ..., N, упомянутая в формулировке задачи в пункте 2 для отрезка $[S_j, S_{j+1}]$.

Для решения пункта 1 выпишем по порядку для каждого отрезка $[S_j, S_{j+1}]$, $j = 0, \ldots, Q-1$, величины i_{Nj} (это индекс самого верхнего отрезка в секторе $[S_j, S_{j+1}]$). Отрезок с номером k может быть самым верхним для нескольких смежных секторов $[S_j, S_{j+1}]$, ..., $[S_r, S_{r+1}]$. Поэтому мы просматриваем номера i_{Nj}, соответствующие отрезкам $[S_j, S_{j+1}]$, $j = 0, \ldots, Q-1$, и определяем последовательные максимальные отрезки, помеченные одним и тем же номером. Концевые точки этих максимальных отрезков и есть искомые точки $t_0, \ldots, t_p$ (они выбираются по указанному выше методу из точек $S_0, \ldots, S_Q$).

25. Пусть в фигуре уже проведены L диагоналей, и они разбивают n-угольник на K частей. Проведем еще одну $(L+1)$-ю диагональ. Подсчитаем, сколько ранее проведенных диагоналей пересекает во внутренних точ-

ках эта диагональ. Обозначим количество пересечений через S. Проведение диагонали:

1) увеличивает количество разбивок n-угольника на 1;
2) каждое пересечение этой диагонали с ранее проведенной диагональю также увеличивает количество разбивок на 1 (по условию никакие 3 диагонали не пересекаются в одной точке).

Итак, после проведения $(L+1)$-й диагонали количество частей станет $K+S+1$ (предполагается, что $(L+1)$-я диагональ не совпадает ни с одной ранее проведенной).

Как определить, пересекается ли диагональ, соединяющая вершины i и j, с диагональю, заданной вершинами m и p? Вершины i и j разбивают контур многоугольника на 2 части: множество A — вершины, лежащие на контуре между вершинами i и j, и множество B — вершины контура между j и i (множества A и B не включают i и j). Если m принадлежит одному из этих множеств, а p — другому, то диагонали пересекаются, иначе — нет.

26. Будем обозначать через V_i вершину ломаной с координатами (x_i, y_i).

Сначала рассмотрим разрез круга ломаной, состоящей только из двух ребер $R_1=(V_1; V_2)$ и $R_2=(V_2; V_3)$. Для того чтобы разнять этот круг, необходимо тянуть в направлении вектора S, выходящего из точки V_2 и лежащего либо внутри угла $V_1V_2V_3$ (вершина угла — точка V_2), либо внутри центрально-симметричного ему относительно точки V_2 угла. Будем говорить, что вектор S лежит в конусе C_2 с вершиной V_2.

Аналогично, если ломаная определяется k вершинами, то для каждой пары ребер $(V_i; V_{i+1})$ и $(V_{i+1}; V_{i+2})$, $i=1, \ldots, k-2$, определяем конус C_{i+1} возможных направлений перемещения, затем считаем, что параллельным переносом вершины всех конусов совмещены в одной точке. Пересечение всех C_i, $i=2, \ldots, k-1$, и даст искомое возможное направление разнимания круга. Если это

пересечение пусто, то круг разнять нельзя, иначе — можно.

27. Пусть точка Z — центр координат (если это не так, сделаем параллельный перенос). Для того чтобы звено забора было полностью видно, необходимо и достаточно, чтобы из точки Z, где стоит человек, были видны обе вершины этого звена и еще какая-нибудь его внутренняя точка. Будем считать, что вершина P звена видна, если интервал $(Z; P)$ не пересекает никаких звеньев забора или же если обе концевые вершины пересекаемого звена k лежат на $[Z; P]$, т. е. человек смотрит вдоль звена k.

Отсортируем по неубыванию углы, образуемые с осью Ox отрезками, одна концевая точка которых Z, а вторая пробегает все вершины звеньев (углы отсчитываются от точки Z в положительном направлении, т. е. против часовой стрелки). Получаем последовательность углов $a_1, a_2, \ldots, a_n$. Добавляем в эту последовательность угол $a_{n+1} = a_1$. Из точки Z в направлении между прямыми, идущими под углами a_i и a_{i+1}, может быть виден кусок только одного единственного звена.

Из точки Z под углами $(a_i + a_{i+1})/2$, $i = 1, \ldots, n$, проводим лучи и для каждого луча смотрим, какое звено k этот луч пересекает первым (пересечение по вершине не учитывается). В том случае, если у этого звена k видны обе вершины, то звено видно полностью, если хотя бы одна вершина не видна, то k видно частично.

После анализа точек пересечения всех n лучей те звенья, которые не видны ни полностью, ни частично, получают пометку невидимых.

28. Рассматриваем номера граней как элементы массивов. Сортируем каждый из массивов с помощью некоторого обменного алгоритма (например, с помощью «пузырьковой» сортировки), подсчитывая количество обменов (пусть это KN и KM соответственно). Если отсортированные массивы совпадают и $(KN - KM)$ кратно 2, то тетраэдры совпадают. (Обмен двух граней можно трактовать как отражение тетраэдра в зеркале.)

Глава 2. ПОИСК И СОРТИРОВКИ

§ 1. ПОСЛЕДОВАТЕЛЬНЫЙ ПОИСК НЕОБХОДИМОГО ЭЛЕМЕНТА В МАССИВЕ

Человеку постоянно приходится сталкиваться с задачами поиска требуемой информации. Типичными примерами может служить работа с тем или иным справочником, телефонной книгой, библиотечной картотекой.

В программировании ***поиск*** является одной из наиболее часто выполняемых операций. Будем считать, что множество из N элементов задано в виде массива (таблицы) $A[1..N]$.

Среди разновидностей простейших задач поиска, встречающихся на практике, можно выделить следующие типы:

1. Найти хотя бы один элемент, равный заданному элементу X. В результате необходимо получить i — индекс (номер) элемента массива такой, что $A[i]=X$.

2. Найти все элементы, равные заданному X. В результате необходимо получить количество таких элементов и (или) их индексы.

Иногда поиск организуется не по совпадению с элементом X, а по выполнению некоторых условий. Примером может служить поиск элементов, удовлетворяющих условиям $X_1 \leqslant A[i] \leqslant X_2$, где X_1 и X_2 заданы.

Если у нас нет никакой добавочной информации о разыскиваемых данных, то очевидный подход — это последовательный просмотр массива. Такой метод называется ***линейным*** или ***последовательным поиском.***

Рассмотрим сначала реализацию последовательного поиска для задач типа 1.

Поиск заканчивается при выполнении одного из двух следующих условий:

1. Элемент найден, т. е. в массиве есть такой элемент $A[i]$, что $A[i]=X$.

2. Весь массив просмотрен и совпадения не обнаружено.

Одним из возможных решений данной задачи может быть следующее: пусть P — переменная логического типа, которая имеет значение «истина», если элемент в массиве найден, и «ложь» — в противном случае.

```
P:=«Ложь»
нц для i от 1 до N
| если A[i]=X
| | то
| | P:=«Истина»
| все
кц
```
(2.1)

Если после выполнения алгоритма P=«Истина», то элемент найден, если переменная P не изменила своего значения, то элемента нет.

Часто требуется не только определить, есть ли в массиве искомый элемент, но и установить, на каком месте он находится. Будем хранить индекс найденного элемента в переменной K:

```
K:=0
нц для i от 1 до N
| если A[i]=X
| | то
| | K:=i
| все
кц
```
(2.2)

После выполнения данного алгоритма по значению переменной K можно определить, есть ли в массиве искомый элемент, и если есть, то где он стоит. Если в массиве несколько таких элементов, то в переменной K будет храниться номер последнего из них. Если такого элемента нет, то значение переменной K не изменится (останется равным нулю).

На практике операцию поиска приходится выполнять достаточно часто, и скорость работы программы находится в прямой зависимости от используемого алгоритма поиска.

В рассмотренных выше алгоритмах требуется просмотреть весь массив даже в том случае, если искомый элемент находится в массиве на первом месте.

Для сокращения времени поиска можно останавливаться сразу после того, как элемент найден. В этом случае весь массив придется просмотреть только тогда, когда искомый элемент последний или его нет вообще.

В результате получим следующий алгоритм:

```
i:=1
нц пока (i<=N) и (A[i]<>X)
| i:=i+1                                    (2.3)
кц
```

Цикл заканчивает работу, либо когда будет найден искомый элемент, либо когда $i=N+1$ (элемента, совпадающего с X, не существует).

На каждой итерации цикла требуется увеличивать индекс i и вычислять логическое выражение. Давайте попытаемся ускорить поиск, упростив логическое выражение. Поместим в конец массива дополнительный элемент со значением X. Тогда совпадение с X обязательно произойдет, и мы можем отбросить проверку условия ($A[i]<>X$). Такой вспомогательный элемент часто называют **«барьером»** или **«часовым»**, так как он препятствует выходу за пределы массива. В исходном массиве теперь будет $N+1$ элемент.

Алгоритм поиска с «барьером» выглядит следующим образом:

```
A[N+1]:=X
i:=1
нц пока A[i]<>X                             (2.4)
| i:=i+1
кц
```

Если по выходу из цикла $i=N+1$ (X равен «барьеру»), то элемента X в массиве $A[1..N]$ нет.

Для реализации поиска в задачах типа 2 (нахождение количества элементов, равных X) в любом случае придется просматривать весь массив.

Если требуется определить количество элементов, то заводят переменную, значение которой увеличивают на 1 каждый раз, когда найден нужный элемент. Такую переменную называют ***«счетчиком»***. До начала просмотра элементов массива «счетчику» нужно задать начальное значение или, другими словами, *инициализировать* значение переменной. В данном случае это начальное значение равно нулю. Алгоритм подсчета элементов, равных заданному, составьте самостоятельно.

Для получения индексов искомых элементов создадим новый массив $B[1..N]$. Как только будет найден необходимый элемент, его индекс будет заноситься в массив B. В переменной K будет храниться номер последнего занятого места в массиве B. Сначала $K=0$.

```
K:=0
нц для i от 1 до N
| если A[i]=X
| | то
| | K:=K+1
| | B[K]:=i
| все
кц
```
(2.5)

После завершения работы первые K элементов массива B будут содержать индексы искомых элементов.

Вопросы для повторения

1. Что называют последовательным поиском?
2. Как определить, что в массиве был найден элемент?
3. Какой из алгоритмов (2.2), (2.3) или (2.4) работает быстрее и почему?
4. Что такое поиск с «барьером»?
5. Для чего используют переменные «счетчики»?
6. Что такое инициализация переменной?

§ 2. ПОИСК МАКСИМАЛЬНОГО И МИНИМАЛЬНОГО ЭЛЕМЕНТОВ В МАССИВЕ

Очень часто для решения задачи требуется находить не заданный элемент массива, а максимальный (наибольший) или минимальный (наименьший) элемент.

Рассмотрим задачу нахождения максимального элемента. Если в массиве один единственный элемент, то он и есть максимальный. В противном случае воспользуемся тем, что максимальным в массиве из i элементов является максимум из $A[i]$ и максимального средн первых i-1 элементов.

```
max:=A[1]
нц для i от 2 до N
| если A[i]>max                    (2.6)
| | то
| | max:=A[i]
| все
нц
```

После завершения работы в переменной max будет храниться значение максимального элемента массива. Однако данный алгоритм не позволяет определить, на каком месте в массиве находится этот максимальный элемент. Будем использовать перменную K для хранения индекса максимального элемента.

```
max:=A[1]
K:=1
нц для i от 2 до N
| если A[i]>max                    (2.7)
| | то
| | max:=A[i]
| | K:=i
| все
кц
```

После выполнения алгоритма переменная K будет содержать значение индекса максимального элемента.

Если в массиве несколько элементов имеют максимальное значение, то в переменной K будет запоминаться индекс первого из них. Если использовать условие $A[i] \geqslant \max$, то будет запоминаться индекс последнего из максимальных.

Для поиска минимального элемента необходимо заменить знак $>$ в ЕСЛИ на знак $<$.

Вопросы для повторения

1. Какой элемент массива является максимальным, какой — минимальным?
2. Как найти максимальный элемент в массиве? Как найти минимальный элемент в массиве?
3. Как определить номер первого элемента, равного максимальному?
4. Как определить номер последнего элемента, равного минимальному?

§ 3. УПОРЯДОЧЕНИЕ ЭЛЕМЕНТОВ МАССИВА

Сортировка *массива* — это расстановка элементов массива в некотором порядке. В отсортированном массиве, за счет предварительно выполненной работы по упорядочению, поиск элемента (даже в худшем случае) можно осуществлять, не просматривая весь массив. Пример отсортированного массива — это, например, список телефонов в справочнике, список фамилий в адресной книге и т. д.

Примером задачи сортировки может служить следующая: в массиве из N различных чисел необходимо осуществить их перестановку так, чтобы после нее в массиве первым элементом было самое большое число. Каждое следующее должно быть меньше предыдущего, а последнее — самое маленькое из чисел данного массива. Такой порядок расположения чисел называют ***убывающим.***

Если в массиве есть равные числа, то задачу можно переформулировать следующим образом: в массиве из

N чисел осуществить их перестановку так, чтобы после перестановки в массиве первым элементом было самое большое число. Каждое следующее должно быть не больше предыдущего, а последнее — самое маленькое из чисел данного массива. Такой порядок расположения чисел называют ***невозрастающим.***

Порядок, при котором в массиве первым элементом является самое маленькое число, каждое следующее число больше предыдущего, а последнее — самое большое из чисел данного массива называют ***возрастающим.***

Если в массиве есть равные числа и они расположены так, что первым элементом является самое маленькое число, каждое следующее число не меньше предыдущего, а последнее — самое большое из чисел данного массива, то такой порядок называют ***неубывающим.***

Задача сортировки, как и любая другая задача, может решаться множеством способов, каждый из которых имеет как достоинства, так и недостатки. Выбор способа сортировки определяется особенностями решаемой задачи.

При выборе метода сортировки необходимо учитывать объем требуемой памяти и скорость работы. При сортировке массива желательно использовать как можно меньше дополнительной памяти, поэтому обычно рассматриваются алгоритмы, которые упорядочивают массив перестановками его элементов (без использования еще одного массива). Оценить скорость работы метода сортировки можно, оценив количество требуемых операций сравнения и (или) операций перестановок элементов.

Ниже рассматриваются методы сортировки линейного массива по убыванию (невозрастанию). Сортировка по возрастанию (неубыванию) производится аналогично.

3.1. Сортировка выбором

Давайте представим, что перед нами поставлена задача расставить N чисел по убыванию. Как бы мы ее решали?

Наверное, пришлось бы сначала найти максимальное из всех чисел и поменять местами с первым числом. Затем из еще неотсортированных элементов надо было бы опять выбрать максимальный элемент и поменять местами с первым элементом из еще не отсортированной части.

Примененный нами метод и называется ***сортировкой выбором.***

Формально его можно описать следующим образом. На i-м шаге ($i=1, \ldots, N-1$):

выбираем из элементов с индексами от i до N максимальный элемент;

меняем местами найденный максимальный и элемент $A[i]$; на i-м месте оказывается максимальный элемент из еще неотсортированной части массива.

После выполнения $N-1$-го шага в позиции $A[N]$ будет находиться самый маленький элемент массива.

Название метода, ***«сортировка выбором»*** определяется тем, что на каждом шаге мы находим (выбираем) максимальный элемент из еще неотсортированной части массива.

Запишем алгоритм сортировки выбором:

```
нц для i от 1 до N-1
| K:=i
| max:=A[i]
| нц для j от i +1 до N
| | если A[i]>max
| | | то                                  (2.8)
| | | max:=A[j]
| | | K:=j
| | все
| кц
| A[K]:=A[i]
| A[i]:=max
кц
```

Внутренний цикл ДЛЯ j ОТ i+1 ДО N является

ничем иным, как алгоритмом поиска максимального алгоритма среди элементов с номерами от $i+1$ до N.

Рассмотрим работу данного метода на массиве $A=\{0, 1, 9, 2, 4, 3, 6, 5\}$. Максимальный элемент будем подчеркивать.

1) 0, 1, 9, 2, 4, 3, 6, 5
2) 9, 1, 0, 2, 4, 3, 6, 5
3) 9, 6, 0, 2, 4, 3, 1, 5
4) 9, 6, 5, 2, 4, 3, 1, 0
5) 9, 6, 5, 4, 2, 3, 1, 0
6) 9, 6, 5, 4, 3, 2, 1, 0
7) 9, 6, 5, 4, 3, 2, 1, 0
8) 9, 6, 5, 4, 3, 2, 1, 0

Подсчитаем количество сравнений, которые пришлось сделать для упорядочения массива.

На первом шаге для нахождения максимального элемента необходимо $(N-1)$ сравнение, на втором $(N-2)$, на третьем $(N-3)$, ..., на последнем шаге — одно сравнение. Найдем сумму:

$$N-1+N-2+N-3+\ldots+1=N(N-1)/2=(N^2-N)/2.$$

Примечание. Сумму можно посчитать исходя из следующих соображений. Количество слагаемых равно $(N-1)$, сумма первого и последнего, второго и предпоследнего и т. д. равна N. Произведение $N(N-1)$ даст удвоенную сумму, так как каждое слагаемое будет входить в эту сумму дважды, поэтому его нужно разделить на 2.

Количество перестановок элементов равно $(N-1)$. Это количество определяется внешним циклом ДЛЯ.

3.2. Сортировка обменом

Рассмотрим еще один метод сортировки, который формально можно описать так:

На i-м шаге ($i=1, \ldots, N-1$) выполняем:

1. Сравниваем первые два элемента. Если первый меньше второго, то меняем их местами.

2. Сравниваем второй и третий, третий и четвертый, ..., $N-i$ и $N-i+1$, при необходимости меняя элементы местами. Самый маленький окажется на i-м месте в массиве.

После первого шага самый маленький элемент массива помещается на N-е место. Массив будет отсортирован после просмотра, в котором участвуют только первый и второй элементы.

Название метода ***«сортировка обменом»*** определяется тем, что алгоритм основывается на обмене местами двух элементов массива.

Описанный метод сортировок обменом называют также ***пузырьковой сортировкой.***

Алгоритм метода сортировки обменом:

```
нц для i от 1 до N-1
| нц для j от 1 до N-i
| | если A[j]<A[j+1]
| | | то
| | | x:=A[j]
| | | A[j]:=A[j+1]
| | | A[j+1]:=x
| | все
| кц
кц
```

(2.9)

Рассмотрим его на примере того же массива $A=\{0, 1, 9, 2, 4, 3, 6, 5\}$.

1) 1, 9, 2, 4, 3, 6, 5, 0
2) 9, 2, 4, 3, 6, 5, 1, 0
3) 9, 4, 3, 6, 5, 2, 1, 0
4) 9, 4, 6, 5, 3, 2, 1, 0
5) 9, 6, 5, 4, 3, 2, 1, 0
6) 9, 6, 5, 4, 3, 2, 1, 0
7) 9, 6, 5, 4, 3, 2, 1, 0

Число сравнений в данном алгоритме равно также $(N^2-N)/2$.

При каждом сравнении возможна перестановка двух

элементов в массиве. Поэтому количество перестановок (в худшем случае) будет равно количеству сравнений, т. е. $(N^2-N)/2$.

* На последних двух проходах в приведенном выше примере массив не менялся. Заметим, что если на каком-то шаге алгоритма элементы массива уже упорядочены, то при последующих проходах по массиву перестановки больше выполняться не будут. Следовательно, как только количество выполненных на последнем проходе перестановок станет равным 0, алгоритм можно заканчивать.

Если запоминать положение (индекс) K последнего обмена, то все пары соседних элементов дальше индекса K уже находятся в желаемом порядке. Поэтому на следующем проходе просмотр можно заканчивать на индексе K.

```
P:=«Истина»
K:=N-1
нц пока P=«Истина»
| P:=«Ложь»
| R:=K
| нц для j от 1 до R
| | если A[j]<A[j+1]
| | | то
| | | x:=A[j]                    (2.10)
| | | A[j]:=A[j+1]
| | | A[j+1]:=x
| | | P:=«Истина»
| | | K:=j
| | все
| кц
кц
```

В приведенном фрагменте переменная P логического типа используется для определения, были перестановки или нет, а переменная K — для хранения индекса по-

следнего обмена. Переменная R является границей, на которой заканчивается просмотр.

Если проанализировать пузырьковую сортировку, то можно заметить, что самое маленькое число занимает свое место за один проход по массиву, а самое большое перемещается по направлению к своему месту на одну позицию при каждом проходе. Это наводит на мысль чередовать направление проходов. Такая сортировка называется ***шейкер-сортировкой.*** Рассмотрим ее работу на том же массиве $A=\{0, 1, 9, 2, 4, 3, 6, 5\}$. ($L$ — левая граница просмотра, R — правая.)

1. 1, 9, 2, 4, 3, 6, 5, 0 $L=1$, $R=7$
2. 9, 1, 6, 2, 4, 3, 5, 0 $L=2$, $R=7$
3. 9, 6, 2, 4, 3, 5, 1, 0 $L=2$, $R=6$
4. 9, 6, 5, 2, 4, 3, 1, 0 $L=3$, $R=6$
5. 9, 6, 5, 4, 3, 2, 1, 0 $L=3$, $R=5$
6. 9, 6, 5, 4, 3, 2, 1, 0 $L=4$, $R=5$

Алгоритм данной сортировки попытайтесь написать самостоятельно.

Все эти улучшения сокращают количество операций сравнения для частных случаев, однако при неблагоприятной начальной расстановке элементов массива (подумайте какой) приходится проделать все $(N^2-N)/2$ операции сравнения*.

Вопросы для повторения

1. На каких принципах основана сортировка выбором?
2. На каких принципах основана сортировка обменом?
3. За какое количество операций сравнения будет отсортирован массив, если применять сортировку обменом? сортировку выбором?
4. Сколько перестановок будет сделано для упорядочения массива, если применять сортировку обменом? сортировку выбором?
5. Как можно сократить время работы алгоритма сортировки обменом?
6. Алгоритм какой из сортировок будет работать быстрее? Почему?

§ 4. СОКРАЩЕНИЕ ОБЛАСТИ ПОИСКА. ДВОИЧНЫЙ ПОИСК

Рассмотрим следующую задачу.

В отделение милиции аэропорта поступило сообщение о том, что некто пытается провезти на самолете бомбу. Диверсант пока находится в здании аэропорта. Допустим, что в нашем распоряжении имеется прибор, который может определить, есть ли бомба в комнате. Необходимо как можно быстрее определить, у кого находится бомба.

Самым простым (и самым продолжительным по времени) решением будет последовательная проверка каждого из пассажиров в комнате с прибором. Однако такой подход вряд ли поможет сэкономить время.

Поступим по-другому. Разделим всех пассажиров на две наиболее равные по численности группы. Разведем группы по двум комнатам. Прибор сможет определить, в какой из комнат находится бомба. В результате такого действия количество подозреваемых уменьшится вдвое. С оставшимися подозреваемыми поступим аналогичным образом: разделим их на 2 части и разведем по двум комнатам. При этом опять количество подозреваемых сократится. Продолжаем так, пока не найдем диверсанта.

Такой метод поиска называется ***двоичным поиском.*** Встречаются и другие названия этого метода: *бинарный поиск, логарифмический поиск, метод деления пополам, дихотомия.*

В программировании такой поиск применяют для нахождения элемента X в отсортированном массиве.

Пусть массив отсортирован в порядке убывания. Находим средний элемент массива $A[m]$ ($m = \mathrm{div}(N+1, 2)$) и сравниваем его с элементом X. Если он равен X, то поиск заканчивается. Если он меньше X, то все элементы с индексами, большими или равными m, можно не рассматривать: если же он больше X, то исключаются элементы с индексами, меньшими или равными m.

При выполнении данного алгоритма нам придется на каждом шаге пересчитывать границы поиска. Так, на первом шаге левая граница $L=1$, правая — $R=N$. На втором шаге либо левая, либо правая граница поменяет свое значение.

Поиск будет продолжаться до тех пор, пока элемент не будет найден, либо когда левая и правая границы поиска не совпадут, что соответствует отсутствию элемента в массиве.

Пусть натуральное число K есть количество операций сравнения, которые необходимы для нахождения элемента в упорядоченном массиве методом дихотомии. Число K определяется из следующего неравенства: $N \leqslant 2^K$, причем K — минимальное из всех возможных.

Примечание. Обычно в математике для определения числа K пользуются функцией log[1]. Тогда число K будет вычислено по формуле $K=[\log_2 N]+1$, где квадратные скобки обозначают целую часть числа, находящегося в скобках.

Алгоритм имеет следующий вид:

```
L:=1
R:=N
P:=«ложь»
нц пока (L<=R) и (P=«ложь»)
| m:=div(R+L, 2)
| если A[m]=X
| | то
| | P:=«истина»
| иначе
| | если A[m]>X                    (2.11)
| | | то
| | | L:=m+1
| | иначе
| | | R:=m-1
| | все
| все
кц
```

[1] Если $2^a=b$, то $\log_2 b=a$.

После выполнения алгоритма в переменной m будет храниться номер найденного элемента, если он есть в массиве (P = «истина»).

Пример. В массиве A найти элемент x:

A (9, 6, 5, 4, 3, 2, 1, 0); $x=1$

1) $L=1$; $R=8$; $m=4$; $A[4]=4$; $4>1$; $P=$ «ложь»
2) $L=5$; $R=8$; $m=6$; $A[6]=2$; $2>1$; $P=$ «ложь»
3) $L=7$; $R=8$; $m=7$; $A[7]=1$; $1=1$; $P=$ «истина»

A (9, 8, 7, 6, 5, 4, 3, 2; 0); $x=1$

1) $L=1$; $R=9$; $m=5$; $A[5]=5$; $5>1$; $P=$ «ложь»
2) $L=6$; $R=9$; $m=7$; $A[7]=3$; $3>1$; $P=$ «ложь»
3) $L=8$; $R=9$; $m=8$; $A[8]=2$; $2>1$; $P=$ «ложь»
4) $L=9$; $R=9$; $m=9$; $A[9]=0$; $0<1$; $P=$ «ложь»
5) $L=9$; $R=8$; $L>R$; $P=$ «ложь»

A (9, 8, 7, 6, 5, 4, 3, 2, 0); $x=7$

1) $L=1$; $R=9$; $m=5$; $A[5]=5$; $5<7$; $P=$ «ложь»
2) $L=1$; $R=4$; $m=2$; $A[2]=8$; $8>7$; $P=$ «ложь»
3) $L=3$; $R=4$; $m=3$; $A[3]=7$; $7=7$; $P=$ «истина»

Можно несколько сократить запись алгоритма, если как и в случае линейного поиска попытаться упростить составное условие цикла ПОКА.

```
L:=1
R:=N
нц пока (L<R)
| m:=div(R+L, 2)
| если A[m]>X
| | то
| | L:=m+1
| иначе
| | R:=m
| все
кц
```
(2.12)

```
если A[R]=X
| то
| P:=«истина»
иначе
| P:=«ложь»
все
```

После выполнения цикла к переменной R хранится номер элемента, равного X, если такой элемент в массиве есть. Если элемента нет, то переменная R показывает номер места в массиве, куда можно вставить элемент X, не нарушая упорядоченности массива.

Вопросы для повторения

1. **В каких случаях можно использовать метод двоичного поиска? Приведите примеры из жизни.**
2. **В чем суть метода двоичного поиска?**
3. **Почему алгоритм (2.12) будет работать быстрее, чем алгоритм (2.11)?**

§ 5*. ДРУГИЕ ВИДЫ СОРТИРОВОК

Все сортировки, рассмотренные раньше, требуют $(N^2-N)/2$ операций сравнения, или, другими словами, порядка N^2 операций. Используя дихотомический поиск, количество операций сравнения можно сократить.

5.1. Сортировка вставками

Будем просматривать элементы массива A, начиная со второго. Каждый новый элемент $A[i]$ будем вставлять на подходящее место в уже упорядоченную совокупность $A[1], \ldots, A[i-1]$. Это место определяется последовательными сравнениями элемента $A[i]$ с упорядоченными элементами $A[1], \ldots, A[i-1]$.

Такой метод сортировки называется ***сортировкой простыми вставками.*** Он также требует порядка N^2 операций сравнения и столько же перестановок элементов в массиве. Попытайтесь составить алгоритм метода самостоятельно.

Если для поиска места элемента $A[i]$ в упорядоченную совокупность $A[1], \ldots, A[i-1]$ воспользоваться методом двоичного поиска, то количество операций сравнения будет порядка $N \log_2 N$, что существенно меньше чем N^2. Количество же перестановок будет порядка N^2. Такой метод сортировки называется ***сортировкой бинарными вставками.***

Алгоритм сортировки бинарными вставками:

```
нц для i от 2 до N
| R:=i
| L:=1
| нц пока L<R
| | m:=div(L+R, 2)
| | если A[m]>A[i]
| | | то
| | | L:=m+1                                    (2.13)
| | иначе
| | | R:=m
| | все
| кц
| k:=R
| x:=A[i]
| нц для j от i до k+1 шаг -1
| | A[j]:=A[j-1]
| кц
| A[k]:=x
кц
```

Цикл ПОКА $L<R$ — цикл поиска места вставки. Он основан на методе двоичного поиска. (Сравни с алгоритмом 2.12.) Цикл ДЛЯ j ОТ i ДО $k+1$ ШАГ -1 сдвигает элементы для освобождения места вставки.

Пример. (Элемент, который вставляем, подчеркнут.)

1. 1, 9, 2, 4, 3, 6, 5, 0
2. 9, 1, 2, 4, 3, 6, 5, 0
3. 9, 2, 1, 4, 3, 6, 5, 0
4. 9, 4, 2, 1, 3, 6, 5, 0

5. 9, 4, 3, 2, 1, 6, 5, 0
6. 9, 6, 4, 3, 2, 1, 5, 0
7. 9, 6, 5, 4, 3, 2, 1, 0
8. 9, 6, 5, 4, 3, 2, 1, 0

5.2. Сортировка слияниями

Сначала рассмотрим следующую задачу:

Есть два отсортированных в порядке невозрастания массива $A[1..N]$ и $B[1..M]$. Получить отсортированный по невозрастанию массив $C[1..N+M]$, состоящий из элементов массивов A и B («слить» вместе массивы A и B).

Можно в массив C записать сначала элементы массива A, затем массива B, затем применить любой алгоритм сортировки. Но в этом случае мы не используем того, что A и B уже отсортированы. Будем просматривать элементы массивов A и B, начиная с $A[1]$ и $B[1]$. Если, например, $A[1]>B[1]$, то $C[1]:=A[1]$, и на следующем шаге сравниваем уже $A[2]$ и $B[1]$, занося больший элемент пары в ячейку $C[2]$, и т. д.

Запись алгоритма:

```
Ai:=1
Bi:=1
Ci:=1 :текущие индексы в массивах A, B, C
нц пока Ci<=N+M
| если A[Ai]>B[Bi]
| | то
| | C[Ci]:=A[Ai]
| | Ai:=Ai+1
| иначе
| | C[Ci]:=B[Bi]
| | Bi:=Bi+1
| все
| Ci:=Ci+1                                    (2.14)
| если (Ai>N) и (Ci<>N+M) :Проверка окон-
|                          чания одного из массивов
```

```
    то
    нц для i от Bi до M
    | C[Ci]:=B[i]
    | Ci:=Ci+1
    кц
  все
  если (Bi>M) и (Ci<>N+M)
    то
    нц для i от Ai до N
    | C[Ci]:=A[i]
    | Ci:=Ci+1
    кц
  все
кц
```

При решении данной задачи мы выполнили «слияние» двух отсортированных массива в один. Количество операций сравнения, которые понадобились для получения упорядоченного массива, равно $N+M$.

На приведенном алгоритме основывается ***сортировка слияниями.***

1. Разобьем все элементы массива на пары (группы) и упорядочим элементы в парах. Если последнему элементу нет пары, то он просто остается на своем месте, образуя отдельную группу.

2. Выполним «слияние» соседних упорядоченных групп чисел, полученных на предыдущем шаге (первой и второй, третьей и четвертой и т. д.; если количество групп нечетное, то с последней группой ничего не делаем). Для «слияния» придется воспользоваться вспомогательным массивом. Количество элементов во всех группах, за исключением, может быть, последней, равно четырем.

3. Выполним «слияние» соседних упорядоченных групп чисел, полученных на предыдущем шаге. Получим упорядоченные группы элементов массива, содержащие по восемь чисел, кроме, может быть, последней группы.

4. Продолжаем действовать таким образом до тех пор, пока не получим две отсортированные части массива, которые после слияния и дадут упорядоченный массив.

Алгоритм сортировки слияниями также имеет название алгоритма фон Неймана.

Алгоритм метода запишите самостоятельно.

Алгоритм сортировки слияниями требует порядка $N \log_2 N$ операций сравнения и столько же перестановок элементов. Данный алгоритм является самым быстрым из всех рассмотренных выше алгоритмов сортировок (проверьте!).

Рассмотрим работу алгоритма на примере массива 1, 9, 2, 4, 3, 6, 5, 0, 7, 11, 8. (Сортируем массив в порядке убывания.)

1. 1, 9, 2, 4, 3, 6, 5, 0, 7, 11, 8
2. 9, 1, 4, 2, 6, 3, 5, 0, 11, 7, 8
3. 9, 4, 2, 1, 6, 5, 3, 0, 11, 8, 7
4. 9, 6, 5, 4, 3, 2, 1, 0, 11, 8, 7
5. 11, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

ЗАДАЧИ ДЛЯ ПОВТОРЕНИЯ

1. Дан линейный массив $A[1..N]$, содержащий целые числа. Определить:

а) имеется ли в массиве хотя бы одно число с указанными ниже свойствами;

б) посчитать количество чисел, которые
   1) являются положительными;
   2) являются нечетными;
   3) отличны от $A(K)$; K вводится;
   4) делятся на 3 и на 5;
   5) делятся на 7 и не делятся на 4;
   6) при делении на 3 и на 5 дают одинаковые остатки;
   7) при делении на 7 дают в остатке 1, 2 или 3;
   8) являются квадратом целого числа;
   9) являются степенью 5.

2. Дан линейный массив $A[1..N]$, содержащий целые числа. Определить, сколько в нем соседств

1) двух положительных чисел;
2) двух разных чисел;
3) двух нечетных чисел;
4) двух нулевых элементов;
5) двух чисел одного знака, причем модуль первого должен быть больше модуля второго;
6) трех упорядоченных по неубыванию чисел.

3. Дан линейный массив $A[1..N]$, содержащий целые числа. Получить те элементы массива, индексы которого являются

1) степенями двойки;
2) полными квадратами;
3) числами Фибоначчи.

4. Дан линейный массив $A[1..N]$, содержащий целые числа.

1) Определить, сколько раз встречается максимальный элемент в этом массиве.
2) Получить те элементы массива, которые находятся между минимальным и максимальным.
3) Найти длину наименьшего отрезка числовой прямой, содержащего все элементы массива.
4) Найти минимальный, максимальный, наименьший больше минимального, наибольший меньше максимального за один просмотр массива.

5. Дан линейный массив $A[1..N]$, содержащий целые числа. Посчитать в нем

1) наибольшее количество одинаковых чисел;
2) количество различных чисел.

6. При поступлении в институт лица, получившие двойку на первом экзамене, ко второму не допускаются. Считая фамилии абитуриентов и их оценки после первого экзамена исходными данными, составить список абитуриентов, допущенных ко второму экзамену.

7. Дан прямоугольный массив $A[1..M, 1..N]$. Получить номер

1) столбца, содержащего максимальный элемент;
2) строки, содержащей минимальный элемент;
3) столбца, сумма элементов которого максимальна;
4) строки, сумма элементов которой минимальна.

8. Дан прямоугольный массив $A[1..M, 1..N]$. Получить номера
1) строк, в которых есть равные элементы;
2) строк, суммы элементов в которых равны.

9. Дан прямоугольный массив $A[1..M, 1..N]$. Получить линейный массив B, в котором
1) $B[i]$ — максимальный элемент i-й строки массива A (i изменяется от 1 до M);
2) $B[i]$ — минимальный элемент i-го столбца массива A (i изменяется от 1 до N).

10. Дан линейный массив $A[1..N]$, содержащий целые числа. Отсортировать его
1) методом «пузырька» в порядке возрастания;
2) методом «пузырька» в порядке возрастания модулей;
3) методом выбора, при помощи поиска минимального элемента в порядке возрастания;
4) методом выбора при помощи поиска одновременно минимального и максимального элементов в порядке возрастания;
5) переставив все нулевые элементы в конец массива; порядок ненулевых — произвольный;
6) так, чтобы все четные элементы стояли в начале массива, а нечетные — в конце;
7) так, чтобы все положительные элементы стояли в начале массива, отрицательные — в середине, а нули — в конце;
8) так, чтобы все положительные элементы стояли в начале массива, а отрицательные — в конце (относительный порядок следования элементов должен сохраниться);
9) в порядке возрастания количества цифр в числах, входящих в массив (сначала однозначные, затем

двузначные и т. д.); порядок расположения чисел с одинаковым количеством цифр должен сохраниться неизменным;

10) в порядке возрастания количества цифр в числах, входящих в массив (сначала однозначные, затем двузначные и т. д.); числа с одинаковым количеством цифр располагаются в порядке убывания.

11. Имеются весы без гирь и четыре различных груза. На каждую из двух чашек весов можно класть по одному грузу. Разложить грузы по убыванию массы, использовав пять взвешиваний.

12. Дан прямоугольный массив $A[1..M, 1..N]$. Получить номера

1) строк, элементы которых расположены в возрастающем порядке;
2) столбцов, элементы которых расположены в убывающем порядке;
3) столбцов, элементы которых являются частью последовательности Фибоначчи;
4) строк, элементы которых являются последовательными степенями двойки.

13. Фамилии участников соревнований по фигурному катанию после короткой программы расположены в порядке, соответствующем занятому месту. Составить список стартовых номеров участников для произвольной программы (участники выступают в порядке, обратном занятым местам).

14. Дан отсортированный в порядке возрастания линейный массив $A[1..N]$.

1) Определить, имеется ли в данном массиве число, равное среднему арифметическому элементов массива.
2) Вставить в данный массив некоторое число X так, чтобы упорядоченность массива сохранилась.

15. Таблица выигрышей лотереи представлена в виде двух массивов $A[1..N]$ и $C[1..N]$. В массиве A, упорядо-

ченном по убыванию, хранятся выигрышные номера, а в массиве C — выигрыши в рублях, выпавшие соответственно на номера $A[1]$, $A[2]$, ..., $A[N]$. Требуется найти выигрыши, выпавшие на ряд номеров, хранящихся в массиве $B[1..M]$ (если номера нет в таблице, выигрыш считается равным нулю).

16. Заданы массивы $A[1..N]$ и $B[1..M]$. Массив A упорядочен по убыванию. Подсчитать количество тех $B[i]$, $1 \leqslant i \leqslant M$, для которых нет равных среди элементов массива A.

17. Задан массив $A[1..N]$. Получить в порядке возрастания все различные числа, входящие в массив A.

18. Лыжные гонки проводятся двумя группами по 10 человек. Результаты соревнований представлены списками участников по каждой группе, в порядке занятых ими мест. Необходимо получить общий список, в котором участники расположены в порядке, соответствующем показанным результатам.

19. Соревнования по плаванию проводятся отдельно в Европе и Америке. Результаты 100 лучших спортсменов каждого континента представлены в виде таблиц, содержащих в порядке занятых мест фамилии спортсменов и их результаты. Составить список 100 лучших спортсменов мира в порядке, определяемом результатами.

20. В памяти ЭВМ хранится список фамилий абонентов в алфавитном порядке и номера их телефонов. Составить программу, обеспечивающую быстрый поиск абонента по номеру телефона.

21. В памяти ЭВМ хранятся списки номеров телефонов и фамилий абонентов, упорядоченные по номерам телефонов, для каждого из пяти телефонных узлов города. Один телефонный узел включает несколько АТС (не более 10). Номера АТС (первые две цифры номера телефона), относящиеся к каждому телефонному узлу, также хранятся в упорядоченном виде в памяти ЭВМ. Составить программу, обеспечивающую быстрый поиск фамилии абонента по заданному номеру телефона.

22. Даны N монет, среди которых одна фальшивая. Определить фальшивую монету с помощью чашечных весов без гирь за минимальное количество взвешиваний, если фальшивая монета

1) тяжелее остальных;
2) легче остальных;
3) отличается от остальных по массе и при этом не известно — тяжелее она или легче настоящей монеты.

23. Некто задумал число из промежутка [0, 100]. Он может правдиво отвечать «да» или «нет» на задаваемые ему вопросы. Какие вопросы нужно задавать, чтобы отгадать задуманное число за минимальное количество попыток.

ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Задан массив чисел $A[1..N, 1..M]$, упорядоченный по возрастанию по строкам и по столбцам, так что

$$A[i, 1] < A[i, 2] < \ldots < A[i, M] \text{ (при всех i)},$$
$$A[1, j] < A[2, j] < \ldots < A[N, j] \text{ (при всех j)}.$$

Найти индексы (i, j) элемента массива, равного заданному числу X, или напечатать слово «нет», если такого элемента не окажется.

2. Задан массив $A[1..N, 1..N]$, элементы которого равны 0 или 1, причем $A[i, i]=0$ для любого i. Необходимо найти, если они есть, такие строку i_0 и столбец j_0, чтобы в столбце j_0 были все 0, а в строке i_0 — все 1 (кроме элемента $A[i_0, i_0]$, равного 0).

3. На плоскости своими координатами задается выпуклый N-угольник. Координаты N-угольника целочисленны и перечисляются в порядке обхода по контуру. Вводятся координаты точки $(x; y)$. Определить

а) является ли она вершиной N-угольника;

б) принадлежит ли она N-угольнику.

4. На каждой из двух параллельных прямых слева направо заданы по N точек (рис. 28).

Их координаты хранятся в массивах $A[1..N]$ и $B[1..N]$. Точки с одинаковыми номерами соединены отрезками. Таким образом, полоса между прямыми разбивается на $N-1$ конечную и 2 бесконечные трапеции. Расстояние между прямыми единичное. Вводится точка $(x; y)$, $0<y<1$. Определить, в какой из трапеций лежит точка.

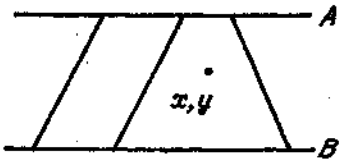


Рис. 28

5. Вводится последовательность из N натуральных чисел. Определить наименьшее натуральное число, отсутствующее в последовательности.

6. На длинной перфоленте записаны N попарно различных положительных целых чисел. Ваша ЭВМ может перематывать ленту на начало и считывать числа одно за другим. Внутренняя память ЭВМ может хранить только несколько целых чисел. Требуется найти наименьшее положительное целое число, которого нет на ленте. Опишите алгоритм, который сделает это за небольшое количество перемоток ленты.

7. На столе в двух столбиках лежат 64 золотые и 64 серебряные монеты соответственно. Как серебряные, так и золотые монеты упорядочены по убыванию масс (самая тяжелая — вверху, самая легкая — внизу). Массы всех монет разные. Какое наименьшее количество взвешиваний необходимо для определения 64-й монеты в порядке убывания масс среди всех 128 монет? (За один раз можно взвешивать две монеты.)

8. Задано N наборов монет из различных стран. Наборы упорядочены по невозрастанию массы монет. В i-м наборе a_i монет. Определить за как можно меньшее число взвешиваний на чашечных весах k-ю по массе монету среди всех монет.

9. Заданы массивы $A[1..N]$ и $B[1..M]$. Найти такие индексы i_0 и j_0, что

$$A[i0]+B[j0]=\max(A[i]+B[j]),$$

где $1\leqslant i\leqslant N$, $1\leqslant j\leqslant M$, $i<j$.

10. Решить уравнение $f(x)=0$ на промежутке $[a, b]$, на котором функция $f(x)$ является монотонной и непрерывной. Корень найти с точностью n знаков после запятой.

11. В романе N глав. В i-й главе a_i страниц. Требуется издать роман в K томах так, чтобы объем самого толстого тома был минимален. Описать алгоритм, отвечающий на вопрос, каким будет объем V самого толстого тома. Делить и переставлять главы нельзя.

12. Имеется N камней массой $A_1, A_2, \ldots, A_N$. Необходимо разбить их на две кучи таким образом, чтобы массы куч отличались не более чем в 2 раза. Выдать сообщение, если этого сделать нельзя.

13. Условие задачи 12, только массы куч отличаются не более чем в 1,5 раза.

14. Даны две целочисленные таблицы $A[1..10]$ и $B[1..15]$. Разработать алгоритм и написать программу, которая проверяет, являются ли эти таблицы похожими. (Две таблицы называются похожими, если совпадают множества чисел, встречающихся в этих таблицах.)

15. Задается словарь. Найти в нем все анаграммы (слова, составленные из одних и тех же букв).

16. На прямой окрасили N отрезков. Известны координата $L[i]$ левого конца и координата $R[i]$ правого конца i-го отрезка для $i=1, \ldots, N$. Найти сумму длин всех окрашенных частей прямой.

Примечание. Число N столь велико, что на выполнение N^2 даже простейших операций не хватит времени.

Модификация. На окружности окрасили N дуг. Известны угловая координата $L[i]$ начала и $R[i]$ конца i-й дуги (от начала к концу двигались, закрашивая дугу, против часовой стрелки). Какая доля окружности окрашена?

17. Имеется $2N$ чисел. Известно, что их можно разбить на пары таким образом, что произведение чисел

в парах равны. Сделать разбиение, если числа: а) натуральные; б) целые.

18. Имеются числа $A_1, A_2, \ldots, A_N$ и $B_1, B_2, \ldots, B_N$. Составить из них N пар (A_i, B_j) таким образом, чтобы сумма произведений пар была максимальна (минимальна). Каждое A_i и B_j в парах встречаются ровно по одному разу.

19. Упорядочить по невозрастанию 5 чисел за 7 операций сравнения.

20. Пусть A — множество из N натуральных чисел. Ваша программа должна определить, существует ли по крайней мере одно подмножество B множества A, удовлетворяющее следующим условиям для любых X, Y, Z из B $(X \neq Y \neq Z \neq X)$:

$$X+Y+Z \leqslant \mathrm{SUM}\{t : t \text{ из } B \backslash \{X, Y, Z\}\}, \qquad (*)$$

где $B\backslash\{X, Y, Z\}$ означает множество B без элементов X, Y и Z, SUM — сумма элементов соответствующего множества.

В случае положительного ответа программа должна найти подмножество B, удовлетворяющее условию $(*)$ и состоящее из максимально возможного числа элементов.

21. Дано положительное целое число K и K целых чисел $A_1, \ldots, A_K$. Вычислить значение суммы $S(M, N) = A_M + A_{M+1} + \ldots + A_{N-1} + A_N$, где $1 \leqslant M \leqslant N \leqslant K$:

а) наибольшее;

б) наименьшее;

в) наиболее близкое к нулю;

г) наиболее близкое к заданному числу P.

Примечание. Число K столь велико, что числа $A_1, A_2, \ldots, A_K$ занимают примерно пятую часть памяти, отводимой для хранения данных, а на выполнение K^2 даже простейших операций не хватает времени.

22. Даны целые числа M и N и массив действительных чисел $X[1..N]$. Найти целое число i $(1 \leqslant i \leqslant N-M)$, для которого сумма $X[i]+\ldots+X[i+M]$ ближе всего к нулю по модулю.

23. Дан массив $X[1..N]$. Необходимо циклически сдвинуть его на k элементов влево (т. е. элемент $X[i]$ после сдвига должен стоять на месте $X[i-k]$; тут мы считаем, что за $X[1]$ следует $X[N]$). Разрешается использовать только несколько дополнительных переменных (дополнительного массива заводить нельзя!).

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. Чему будут равны x_1 и x_2, если: а) $x_1 = P(1.0, 2.0)$, б) $x_2 = P(3.0, 4.0)$?

```
Алг вещ P(вещ a, b)
нач вещ c
| c:= (b-a)/3+a
| если ABS(b-a)<0.0001
| | то знач:=c
| иначе
| | если sin(a)*sin(c)<0
| | | то знач:=P(a, c)
| | иначе
| | | знач:=P(c, b)
| | все
| все
кон
```

2. Определить, можно ли представить заданное натуральное число в виде произведения четырех последовательных натуральных чисел. Длина числа не более 250 символов. Конец числа — пробел.

3. Написать программу, которая в двумерном массиве $A[1..N, 1..M]$ целых чисел, таком, что для всех $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M-1$ выполняется $A[i, j] > A[i, j+1]$ и для всех $1 \leqslant i \leqslant N-1$ выполняется $A[i, M] > A[i+1, M]$, находит все элементы $A[i, j]$, равные $j+i$, или устанавливает, что таких элементов нет.

4. Написать программу, которая в двумерном массиве $A[1..N, 1..M]$ целых чисел, таком, что для всех

$1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M-1$ выполняется $A[i, j] < A[i, j+1]$ и для всех $1 \leqslant i \leqslant N-1$ выполняется $A[i, M] < A[i+1, M]$, находит все элементы $A[i, j]$, равные $j+i$, или устанавливает, что таких элементов нет.

5. На прямой своими концами заданы N отрезков и точка X. Определить, принадлежит ли точка межотрезочному интервалу. Если да, то указать концевые точки этого интервала. Если нет, то найти: а) какому количеству отрезков принадлежит точка; б) каким именно отрезкам принадлежит точка?

6. На прямой своими концами заданы N отрезков. Найти точку, принадлежащую максимальному числу отрезков.

7. Пусть $F(x)$ — процедура, вычисляющая значение непрерывной функции, которая определена на всей действительной оси. Эта функция убывает при всех x меньше некоторого числа x_0 и возрастает при всех x больше x_0. Составить алгоритм, который определяет x_0 с точностью n знаков после запятой.

8. Дан прямоугольный треугольник ABC: $\angle ABC = 90°$; $\angle BAC = a°$ $(0° < a° < 90°)$. Катет BC разделен на n равных частей:

$$|BD_1| = |D_1D_2| = \ldots = |D_{n-2}D_{n-1}| = |D_{n-1}C|.$$

Каждая из точек D_i $(1 \leqslant i \leqslant n-1)$ соединена отрезком с вершиной A таким образом, что полученные отрезки разделяют угол BAC на n частей:

$$\angle BAD_1 = a_1°, \quad \angle D_1AD_2 = a_2°, \quad \ldots, \quad \angle D_{n-2}AD_{n-1} = a_{n-1}°,$$
$$\angle D_{n-1}AC = a°.$$

Для введенных a (в градусах) и n определить k $(1 \leqslant k \leqslant n)$, для которого значение выражения $|a_k - a/n|$ будет наименьшим.

9. В треугольном королевстве, расположенном между тремя пирамидами, где-то под пылью столетий спрятан вход в гробницу фараона Тутрамзеса. Многие кладо-

искатели отправлялись в дорогу, чтобы найти вход в гробницу и обнаружить несметные сокровища.

Напрасно! Проклятие фараона гласило: как только кладоискатель попадает в треугольное королевство, он должен двигаться только прямолинейно в направлении к вершине одной из пирамид. При этом, преодолев половину пути, он должен некоторое время отдохнуть и затем снова отправиться в путь по направлению к какой-нибудь из трех пирамид. Каждый раз направление выбирается случайным образом. Существует ли в треугольном королевстве точка, в которую кладоискатель никогда не сможет попасть? Если да, то вход в гробницу останется навсегда скрытым и фараон навеки будет оставлен в покое.

Написать программу для кладоискателя, показывающую на экране только точки остановки путника. (Задаются координаты трех угловых точек треугольного королевства и начальное положение кладоискателя.)

Получить как минимум 5 графиков. Один из них должен быть для следующих начальных условий: начальная точка и три вершины лежат в 4 углах экрана; проделать 5000 итераций.

10. Предположим, что имеется некоторый кусок ленты, разделенный на кадры. Кадры занумерованы с двух сторон. Полоска ленты склеена в лист Мебиуса. Необходимо составить алгоритм упорядочения этой последовательности, предположив, что соседние кадры можно переставлять (естественно, в упорядоченной последовательности будет один «скачок» от минимального элемента к максимальному). Следует учесть, что при перестановке кадров переставляются числа с обеих сторон кадров.

Например: есть 2 кадра, A_1, B_1 — одна сторона кадров, A_2, B_2 — другая. Пусть $A_1=1$, $A_2=2$, $B_1=7$, $B_2=3$. Тогда после перестановки содержимого A и B будет $A_1=7$, $A_2=3$, $B_1=1$, $B_2=2$.

Всегда ли такое упорядочение возможно?

11. Имеется N человек и целые числа $A_1, \ldots, A_N$. Может ли число знакомых i-го человека в этой группе равняться A_i?

12. Условие задачи 11. Указать один из возможных вариантов знакомств.

13. Задано семейство множеств букв. Найти такое максимальное k, для которого можно построить множество, состоящее из k букв, причем каждая из них принадлежит ровно k множествам заданного семейства.

14. В музее регистрируется в течение дня время прихода и ухода каждого посетителя. Таким образом за день получены N пар значений, где первое значение в паре показывает время прихода посетителя, а второе значение — время его ухода. Найти промежуток времени, в течение которого в музее одновременно находилось максимальное число посетителей.

15. Задается число $n>1$, которое определяет размерность пространства и размеры M n-мерных параллелепипедов $(a_{i1}, \ldots, a_{in})$, $i=1, \ldots, M$. Параллелепипед может располагаться в пространстве любым из способов, при которых его ребра параллельны осям координат. Найти максимальную последовательность вкладываемых друг в друга параллелепипедов.

16. Построить максимальное множество, состоящее из попарно не сравнимых векторов v. Векторы v определяются парами чисел, выбираемых из данной последовательности чисел $a_1, \ldots, a_n$, $n \geqslant 1$. Два вектора $v=(a, b)$ и $v'=(a', b')$ называются сравнимыми, если $a \leqslant a'$ и $b \leqslant b'$ или $a \geqslant a'$ и $b \geqslant b'$, в противном случае — несравнимыми.

17. Пусть группа состоит из N человек. Назовем одного из этих людей знаменитостью, если он не знает никого из оставшихся, а его знают все. Задача состоит в том, чтобы в этой группе определить знаменитость (если она там есть). При этом разрешается задавать только вопросы вида: «Извините, знаете ли Вы вон того человека?» (Предполагается, что все ответы правдивы,

и что даже знаменитость ответит на поставленный ей вопрос.)

Найти минимальное необходимое число вопросов, которые надо задать, и описать алгоритм опроса.

Входные данные: матрица $C[i, j]$, такая, что $C[i, j]=1$, если i знает j, и $C[i, j]=0$, если иначе.

18. На бирже есть N продавцов и M покупателей и один вид товара. Каждый продавец определяет для себя и объявляет минимальную цену, по которой он согласен продать товар, а каждый покупатель называет максимальную цену, по которой он еще согласен купить товар.

Найти такую единую цену на товар, чтобы сумма сделок купли-продажи была максимальной.

Найти такую максимальную единую цену на товар, чтобы общее количество продавцов и покупателей, участвующих в торгах, было максимальным.

19. Пусть *blackbox*(v) — функция аргумента v; *blackbox*(v)=1, если слово v является частью неизвестного слова w, и 0 — в противном случае.

Разработать программу нахождения слова w с помощью как можно более короткой последовательности вызовов функции *blackbox*.

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Для решения задачи необходимо сравнить число X с элементами массива. Если сравнивать X со всеми элементами массива A, количество операций сравнения равно $M \cdot N$.

Существенно сократить количество сравнений можно, если рассматривать первую строку и последний столбец массива A как один массив. Элементы в нем упорядочены по возрастанию.

Рассмотрим элемент $A[1, M]$.

Возможны следующие ситуации.

1) $X=A[1, M]$.

В этом случае заданный элемент найден.

2) $X<A[1, M]$.

Тогда элемента, равного X, в столбце с номером M быть не может. Поэтому можно ограничиться поиском заданного элемента в массиве A с выброшенным M-м столбцом.

3) $X>A[1, M]$.

В этом случае элемента, равного X, в строке с номером 1 быть не может. Поэтому в дальнейшем поиск заданного элемента будет проводиться в массиве A без первой строки.

Таким образом, на каждом шаге, после сравнения заданного элемента X со значением элемента из правого верхнего угла массива, из массива выбрасывается либо строка, либо столбец. В результате этого суммарное число строк и столбцов уменьшается на 1. Поэтому для решения задачи потребуется не более чем $N+M$ сравнений.

2. Номера строки i_0 и столбца j_0 должны совпадать, иначе в строке будет недиагональный нулевой элемент.

Простейшим решением является просмотр элементов i-й строки и i-го столбца, $i=1, 2, \ldots, N$, массива до нахождения индекса k такого, что элементы k-й строки и k-го столбца удовлетворяют требуемому свойству или до установления факта, что такого индекса нет. Очевидно, что в последнем случае необходим просмотр почти всех элементов массива.

Однако возможно существенно сократить количество операций, используя следующий факт.

Рассмотрим элемент $A[k, j]$, $k\neq j$. Возможны две ситуации:

1) $A[k, j]=0$.

В этом случае можно заметить, что индекс k не подходит, так как в строке с индексом k стоит 0. Поэтому можно ограничиться поиском заданного элемента в подмассиве без k-го столбца и k-й строки.

2) $A[k, j]=1$.

В этом случае можно заметить, что индекс j не подходит, так как в столбце с индексом j стоит 1. Поэтому можно ограничиться поиском заданного элемента в подмассиве без j-го столбца и j-й строки.

Таким образом, рассматривая на каждом шаге значения элементов $A[k, j]$ таких, что $A[k, j]$ является элементом интересующего нас подмассива, потребуется просмотр ровно $N-1$ элемента для установления единственного индекса k, который может удовлетворить требуемому свойству. Остается проверить только элементы этого столбца и строки.

```
k:=1
для j от 2 до N
 если A[k, j]=0
 то k:=j
```

После окончания цикла остается проверить только элементы k-го столбца и k-й строки.

Суммарная трудоемкость описанного алгоритма не превосходит $3N$ операций.

3. Простейшим решением является просмотр координат вершин N-угольника и определение, совпадают ли координаты одной из них с координатами точки $(x; y)$. В этом случае необходим просмотр координат всех N вершин многоугольника.

Сократить количество операций просмотра можно, используя метод дихотомии.

Пронумеруем вершины от 1 до N в порядке обхода по контуру и занесем координаты вершин в массив.

Сравним вначале координату первой вершины в обходе с координатами точки $(x; y)$. Если они совпадают, то задача решена. Если не совпадают, то точка (x, y) может совпадать с одной из оставшихся $N-1$ вершиной. Номер начальной вершины оставшегося участка контура равен 2, конечной — N.

Пусть на некотором шаге номера начальной и конечной вершин рассматриваемого контура равны j и q соответственно. Найдем номер средней вершины $m=$

$=(f+q)$ div 2. Определим теперь положение точки $(x; y)$ и вершины с номером f относительно прямой, проходящей через вершины с номерами 1 и m. Возможны следующие ситуации:

1) Точки лежат по одну сторону от прямой. В этом случае нас не интересует часть контура от вершины m до вершины q. Поэтому последней интересующей нас вершиной в контуре можно считать вершину $m-1$. Полагаем $q:=m-1$.
2) Точки лежат по разные стороны от прямой. В этом случае нас не интересует часть контура от вершины f до вершины $m-1$. Поэтому первой интересующей нас вершиной в контуре можно считать вершину m. Полагаем $f:=m$.

Процесс заканчивается, когда $q-f=1$ (т. е. осталось только две точки). Проверка их координат на совпадение с точкой $(x; y)$ и дает ответ на поставленный в задаче вопрос.

Примечание. Для решения задачи нам понадобилось проанализировать $O(\log_2 N)$ вершин.

Этот пример показывает, что метод дихотомии можно применять не только к числовым упорядоченным множествам.

4. Пронумеруем трапеции слева направо от 1 до $N+1$. Левая бесконечная трапеция имеет номер 1, правая — $(N+1)$, конечные трапеции — от 2 до N.

Простейшим решением является последовательное определение положения точки $(x; y)$ относительно прямых, проходящих через точки с координатами $(A_{i-1}; 1)$, $(B_{i-1}; 0)$ и $(A_i; 1)$, $(B_i; 0)$ соответственно, $i=2, \ldots, N$. Возможны следующие ситуации.

1) Точка лежит по одну сторону от прямых. Полагаем $i:=i+1$.
2) Точка лежит по разные стороны от прямых. Тогда интересующей нас трапецией является конечная трапеция с индексом i.

Если мы проанализировали все конечные трапеции, а ситуация 2) не возникла, то точка лежит в одной из бесконечных трапеций. Проверим, лежат ли точка $(x; y)$ и точка из левой бесконечной трапеции по одну сторону от прямой, проходящей через точки $(A_1; 1)$, $(B_1; 0)$. Если да, то точка принадлежит левой бесконечной трапеции, если нет — правой.

Однако можно существенно сократить количество операций, используя метод дихотомии.

Определим индексы начального и конечного номеров интересующих нас трапеций, среди которых будет искомая. Вначале индексы равны 1 и $N+1$ соответственно.

Пусть на некотором шаге индексы начального и конечного номеров интересующих нас трапеции равны f и q соответственно. Вычислим индекс среднего номера $m=(f+q)$ div 2.

Определим взаимное расположение точки $(x; y)$ и точки, лежащей заведомо в левой бесконечной трапеции, относительно прямой, проходящей через точки с координатами $(A_m; 1)$, $(B_m; 0)$.

Возможны следующие ситуации.

1) Точки лежат по одну сторону от прямой. В этом случае нас не интересуют трапеции с номерами от m до q.
   Полагаем $q:=m$.
2) Точки лежат по разные стороны от прямой. В этом случае нас не интересуют трапеции с номерами от f до m. Полагаем $f:=m+1$.

Алгоритм заканчивает работу, когда $f=q$. Искомая трапеция имеет номер f.

5. Среди N введенных натуральных чисел отсутствует по крайней мере одно число из интервала $[1, N+1]$. Идея решения состоит в использовании массива из N чисел, в котором элемент с индексом i «регистрирует», пришло ли число со значением i. После «регистрации» всех элементов последовательности остается только проверить, какое минимальное число не «зарегистрировано».

В качестве признака «регистрации» числа i можно заносить в i-й элемент массива 1. Первоначально все числа «незарегистрированы» — все элементы массива равны 0.

Если все числа от 1 до N «зарегистрированы», то минимальное отсутствующее натуральное — $N+1$.

6. Очевидно, что при вводе N натуральных чисел по крайней мере одно число из интервала $[1, N+1]$ отсутствует.

Определим начало a и конец b некоторого интервала индексов, который нас интересует. Возможно ли за один просмотр ленты установить, все ли числа из интервала $[a, b]$ присутствуют на ленте? Учитывая тот факт, что записанные числа различны, можно определить, сколько чисел, записанных на ленте, попадают в интересующий нас интервал. С другой стороны, нетрудно определить количество натуральных чисел на интервале $[a, b]$ — это $(b-a+1)$.

Алгоритм состоит в следующем.

Определим значения начала и конца интересующего нас интервала. Очевидно, что вначале они равны 1 и $N+1$ соответственно.

Пусть на некотором шаге значения начала и конца интересующего нас интервала равны f и q соответственно. Определим индекс среднего элемента интервала $m = (f+q) \operatorname{div} 2$.

Найдем теперь количество элементов k на ленте, лежащих в интервале $[f, m]$.

Возможны следующие ситуации.

1) Количество элементов $k < m-f+1$. В этом случае нас не интересует интервал от m до q, так как на интервале $[f, m]$ хотя бы одно число отсутствует. Поэтому интересующим нас интервалом можно считать $[f, m]$. Поэтому полагаем $q := m$.
2) Количество элементов $k = m-f+1$. В этом случае нас не интересует интервал от f до m, так как на интервале $[f, m]$ все натуральные числа присутствуют. Таким образом, интересующим нас интерва-

лом можно считать $[m+1, q]$. Поэтому полагаем $q := m+1$.

Процесс оканчивается, когда $f=q$. Значение f является искомым.

7. Создадим два массива, каждый из которых будет содержать массы монет из первого и второго столбиков соответственно.

Очевидное решение задачи — «слияние» этих двух массивов в общий упорядоченный массив весов и нахождение в нем 64-го по величине элемента.

Однако задачу можно решить быстрее, используя метод дихотомии. При этом не возникает необходимости объединять исходные массивы.

Обозначим соответственно через N_i и K_i начальный и конечный номера элементов в рассматриваемых на данном шаге массивах, $i=1, 2$. Определим номера средних элементов $S_i=(N_i+K_i)$ div 2. На первом шаге $N_1=1$, $K_1=64$, $N_2=1$, $K_2=64$, $S_1=32$, $S_2=32$. Значения элементов, стоящих на месте S_i, $i=1, 2$, обозначим X и Y.

Сравнив средние элементы X и Y, найдем больший из них. Пусть это X. В силу упорядоченности массива каждый из элементов, стоящих перед X, больше X по величине. С другой стороны, только элементы, стоящие перед Y во втором массиве, могут быть больше X. Поэтому X больше искомого 64-го элемента. Следовательно, в первом массиве можно не рассматривать первые 32 элемента.

Аналогично можно показать, что элементы, стоящие после Y, можно также не рассматривать, так как они меньше искомого элемента. После таких действий в массивах осталось по 32 элемента, и при этом необходимо найти 32-й по величине элемент.

Повторяем описанный выше процесс с измененными значениями начальных N_i, конечных K_i и средних S_i номеров элементов в массивах по следующему правилу: если больший из X и Y находятся в первом массиве, то $N_1=S_1+1$, $K_2=S_2$, иначе $N_2=S_2+1$, $K_1=S_1$. Процесс

заканчивается, когда в каждом массиве останется по одному элементу. При этом больший из них и будет искомым.

Примечание. Поскольку количество монет после каждого шага уменьшается в два раза, то количество взвешиваний — $\log_2 128 = 7$.

8. Очевидным решением задачи является создание общего упорядоченного массива, содержащего элементы всех исходных массивов, и нахождение в нем k-го по величине элемента.

Примечание. В этом случае требуется порядка $(a_1+a_2+\ldots+a_n)\log_2(a_1+a_2+\ldots+a_n)$ операций.

Однако можно существенно сократить количество операций, используя дихотомический способ поиска.

Пусть для каждого из наборов i определены начальный N_i и конечный K_i индексы. Определим индексы средних элементов $S_i=(N_i+K_i)\,\mathrm{div}\,2$ для массивов с $K_i \geqslant N_i$. Найдем максимальное max и минимальное min среди значений средних элементов. Определим также S_j для массивов с $K_j < N_j$ по правилу $S_j = K_j$.

На каждой итерации вычисляем значение

$$S = S_1 + S_2 + \ldots + S_N.$$

Возможны три ситуации.

1) $S > k$.

В этом случае искомый элемент не меньше минимального среднего элемента min (пусть он находится в наборе с номером j). Тогда элементы с индексами $S_j, \ldots, K_j$ заведомо не являются решением и их можно игнорировать. Поэтому пересчитываем K_j по правилу $K_j = S_j - 1$ (напомним, что $K_j \geqslant N_j$).

2) $S < k$.

В этом случае искомый элемент не больше максимального элемента max (пусть он находится в наборе с номером t). Тогда элементы с индексами

$N_i, \ldots, S_i$ заведомо не являются решением и их можно игнорировать. Поэтому пересчитываем N_i по правилу $N_i = S_i + 1$ (напомним, что $K_i \geqslant N_i$).

3) $S = k$.

В этом случае можно пересчитать $K_j = S_j$ и $N_i = S_i + 1$.

Процесс заканчивается, когда начальный N_i и конечный K_i индексы удовлетворяют условию $K_i < N_i$ для каждого i. При этом меньший из элементов с индексами K_i и будет искомым (если $K_i = 0$, то этот элемент рассматривать не нужно).

9. Для решения задачи достаточно осуществлять просмотр элементов массивов A и B, фиксируя две величины:

1) положение p максимального просмотренного элемента из A;

2) значение индексов i_0, j_0 для максимальной найденной суммы, удовлетворяющей требованию условия.

На начальном этапе $p = 1$, $i_0 = 1$, $j_0 = 2$.

При просмотре очередного элемента i из A определяется:

максимальная из сумм

$\{A[i_0] + B[j_0],\ A[p] + B[i+1],\ A[i] + B[i+1]\}$

пересчетом индексов i_0, j_0 для максимальной найденной суммы;

индекс p максимального просмотренного элемента из A (при сравнении значений $A[p]$ и $A[i]$).

Действия выполняются для i от 2 до $m - 1$.

10. Если функция является монотонной и непрерывной на промежутке $[a, b]$ и имеет на концах промежутка значения разных знаков, то она имеет на этом промежутке единственный корень, который можно найти методом дихотомии.

Если функция имеет на промежутке единственный корень, то значения функции на концах этого промежутка имеют разные знаки, т.е. $f(a) f(b) < 0$. Найдем середину отрезка $[a, b]$: $c = (a + b)/2$. Точка c разделит

наш промежуток на две части. Вычислим значение $f(c)$. Если $f(c)=0$, то c — корень уравнения. Если $f(c)\neq 0$, то либо $f(a)f(c)<0$, либо $f(c)f(b)<0$. В качестве нового промежутка $[a, b]$ берем тот, на концах которого функция принимает значения разных знаков. Длина промежутка сократилась вдвое. Разделим $[a, b]$ пополам и из двух полученных промежутков выберем тот, на концах которого функция принимает значения разных знаков, и т. д.

Процесс продолжается до тех пор, пока модуль значения функции в найденной на данном шаге точке x не будет меньше 10^{-n} (если $n=5$, то $|f(x)|<0{,}00001$).

11. Попытаемся распределить главы по томам равномерно. На каждый том придется не более $P=\lceil N/K\rceil$ глав, где $\lceil\ \rceil$ — округление по избытку. Тома с 1-го по $(K-1)$-й будут содержать по P глав, а том с номером $K-(N-PK)$ глав. Найдем максимальный объем тома при таком распределении. Запомним его в переменной *high*. Это будет оценка минимального объема V сверху.

В переменной *low* будем хранить оценку V снизу. Выберем ее как максимум из двух величин: объема наибольшей главы и частного от деления суммарного объема всех глав на количество томов (что соответствует идеальному равномерному распределению страниц по томам). Обозначим через *level* полусумму оценок сверху и снизу:

$$level=(low+high)\ \text{div}\ 2.$$

Попытаемся распределить главы по томам так, чтобы объем каждого тома не превосходил *level*. Будем заносить в очередной том главы до тех пор, пока добавление следующей главы не приведет к превышению *level*. После этого перейдем к формированию следующего тома.

Если нам удалось распределить все главы по томам, то мы нашли новую оценку для V сверху. В этом случае полагаем $high=level$.

Если же мы сформировали последний том и после этого осталась еще хотя бы одна глава, то с максималь-

ным объемом тома не более *level* главы распределить нельзя. Требуется увеличить нижнюю оценку. Полагаем $low := level + 1$.

Повторяем этот процесс до тех пор, пока не станет $low \geqslant high$ (тем самым объем искомого тома V уточнен до одной страницы). Значение *high* и будет искомым минимальным объемом самого толстого тома.

Применение метода дихотомии к решению данной задачи позволяет значительно сократить перебор возможных вариантов.

12. Основная стратегия решения заключается в том, что каждый следующий камень кладется в кучу с меньшей текущей массой. При этом в первую кучу надо положить камень максимальной массы. Покажем, что этого достаточно, чтобы гарантировать правильное решение задачи. По окончании распределения камней по кучам возможны 2 ситуации:

1) Все камни попали во вторую кучу, а ее масса осталась меньше половины массы первой кучи. Понятно, что в этом случае камни требуемым образом разбить нельзя, следовательно, решения не существует.

2) Случай 1 не выполняется. Тогда возможны следующие ситуации.

а) Все камни попали во вторую кучу. В этом случае ясно, что массы куч отличаются не более чем на половину первой кучи, если масса первой кучи больше, или не более чем на массу последнего камня, положенного во вторую кучу. В любом из этих случаев требуемое условие выполняется.

б) В первую кучу попали и другие камни. Тогда ясно, что массы куч отличаются не более чем на массу самого тяжелого камня, кроме первого. Следовательно, и в этом случае условие задачи выполняется.

13. На начальном этапе в первую кучу кладется самый тяжелый камень, а во вторую — два следующих

по массе камня. Для оставшихся камней реализуется описанная для задачи 12 стратегия.

14. Мы можем отсортировать оба массива — и *A*, и *B* (например, по неубыванию), далее, если первые элементы массивов *A* и *B* совпадают, то ищем и в *A*, и в *B* минимальные элементы больше данного и повторяем сравнения; если же элементы не совпадают либо один из массивов уже закончился, а другой еще нет, то массивы не похожи.

```
{A и B уже отсортированы}
i:=1; {смотрим массивы A и B, начиная с первых}
j:=1; {элементов}
while (i<=10) and (j<=15) and (A[i]=B[j]) do
begin
  element:=A[i];
 while (i<=10) and (A[i]=element) do
    i:=i+1; {поиск несовпадающего элемента}
 while (j<=15) and (B[j]=element) do
    j:=j+1; {поиск несовпадающего элемента}
end;
if (i=11) and (j=16) {просмотрели все элементы
A и B}
then writeln ('Массивы похожи')
else writeln ('Массивы непохожи');
```

15. Каждому слову приписываем номер в словаре. Сначала сортируем буквы в каждом слове по (например) неубыванию. Получаем какой-то «ключ», который совпадает у всех слов-анаграмм (например, слова «лом» и «мол» преобразуются в одни и те же ключи «лмо»).

Далее мы сортируем ключи слов (совместно с приписанными номерами) по неубыванию. Все одинаковые ключи будут размещаться в отсортированной последовательности слов друг за другом. Мы просматриваем полученную последовательность, ищем совпадающие ключи и по приписанным им номерам находим в словаре соответствующие слова-анаграммы.

16. Отсортируем отрезки в порядке неубывания координат левых концов и будем моделировать последовательное их закрашивание, начиная с самого левого отрезка. Назовем закрашиваемым тот отрезок, который является объединением одного или нескольких отрезков. Каждый раз, когда берем новый отрезок из упорядоченной последовательности, мы анализируем следующие возможные ситуации.

1) Если новый отрезок пересекается с закрашиваемым отрезком (его левая координата не больше координаты правого конца закрашиваемого отрезка), то новым правым концом закрашиваемого сейчас отрезка становится более правый из концов закрашиваемого и нового отрезков.
2) Если новый отрезок не пересекается с закрашиваемым отрезком, то закраска предыдущего отрезка закончена, его длина суммируется с длиной уже закрашенной части, а закрашиваемым отрезком становится новый отрезок.

Процесс продолжается до тех пор, пока не будут просмотрены все отрезки. После этого длина последнего закрашенного отрезка суммируется с длиной ранее закрашенной части.

17. Основная идея состоит в том, чтобы не использовать операцию умножения двух чисел. Если числа натуральные, то одна из пар должна содержать максимальное и минимальное числа. Рассуждая таким образом для оставшихся чисел, приходим к простому алгоритму. Сортируем числа в порядке неубывания. Тогда пары составляют первое и последнее числа, второе и предпоследнее и т. д. Ситуация немного изменяется, если числа целые. В этом случае возможны три варианта:

1) Произведение равно 0. В этом случае существует хотя бы N нулевых элементов. Поэтому пары будут организованы из одного ненулевого элемента и одного нулевого или из двух нулевых элементов.

2) Произведение положительно. В этом случае перемножаются положительные числа с положительными, а отрицательные — с отрицательными по правилу, как и в случае с натуральными числами.
3) Произведение отрицательно. В этом случае перемножается минимальное положительное число с минимальным отрицательным и т. д.

Для определения ситуаций достаточно подсчитать количество нулевых, положительных и отрицательных элементов.

Если есть нулевые элементы, то возможен только вариант 1). Если количество положительных элементов не равно количеству отрицательных, то возможен только вариант 2). В других случаях возможна ситуация 2) и 3).

Для определения знака произведения рассмотрим четыре элемента массива: максимальный положительный (пусть это a), минимальный положительный (b), минимальный отрицательный (c), максимальный отрицательный (d). Понятно, что решением могут быть только пары (a, b), (c, d) или (a, d), (b, c). Если $a \neq -c$, то в случае $a > -c$ в паре с элементом a должен быть меньший по модулю из элементов b и d, а если $a < -c$, то — больший по модулю. В случае, если $a = -c$ и $b = -d$, эта четверка не дает никакой информации о знаке произведения, поэтому можно перейти к следующей четверке чисел и т. д., пока не будет установлен знак произведения. Если же просмотрены все числа, а знак не установлен, то он может быть как плюс, так и минус.

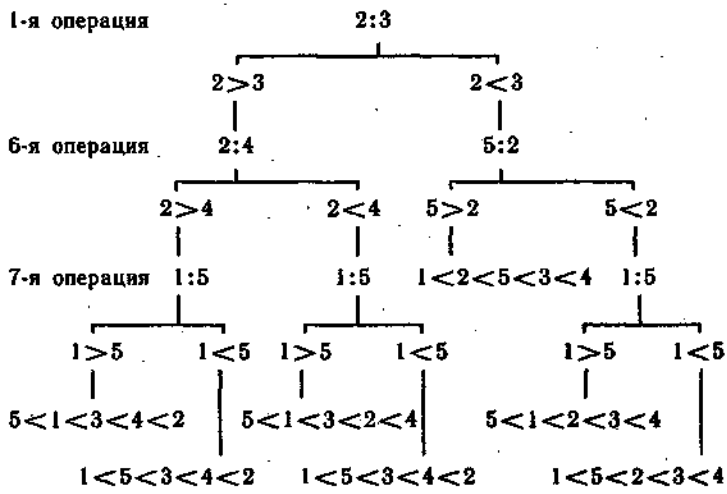
18. Чтобы сумма произведений пар была максимальна (минимальна), необходимо упорядочить наборы A и B одинаковым (различным) образом, и пары будут составлять элементы, стоящие на одинаковых позициях в упорядоченных наборах. Это следует из того, что если $a < b$ и $c < d$, то $ac + bd \geqslant ad + bc$.

19. Предположим, что среди пяти чисел нет одинаковых (случай совпадающих чисел рассматривается аналогично. В дальнейшем будем обозначать операцию срав-

нения значком «:». Например, 5:3 означает, что мы сравниваем пятое и третье числа. Запись $5<3$ означает, что пятое число меньше третьего.

Сначала выполним операции 1:2, 3:4. При необходимости, перенумеровывая числа, получаем, что $1<2$, $3<4$. Далее выполняем операцию 1:3. Опять же при необходимости, перенумеровывая числа, получаем $1<3$. Выполняем операцию 3:5. При этом возможны следующие ситуации:

1) $3>5$. Подводя итог четырех проделанных операций сравнения, имеем: $1<2$; $1<3<4$; $5<3$.



2) $3<5$.

Сравниваем 4 и 5.

Пусть $4<5$. Тогда $1<2$; $1<3<4<5$.

Выполняем 2:4. В зависимости от результата делаем либо 2:3, либо 2:5. Оставшиеся варианты рассматриваем аналогично.

20. Без потери общности предположим, что элементы массива A упорядочены в возрастающем порядке (во множестве нет дублирующихся элементов, поэтому массив A упорядочен именно в возрастающем, а не только

в неубывающем порядке). Если это не так, то добавляем в программу подпрограмму сортировки.

Если свойство (*) выполняется для подмножества B, то оно выполняется и для трех наибольших по величине элементов B. Обратно, из выполнимости (*) для трех максимальных элементов B следует выполнимость (*) и для B. Мы будем включать в B в порядке их возрастания элементы из A и проверять для трех максимальных выполнение условия (*).

21. а) Пусть в переменной *MaxEndHere* хранится сумма элементов максимального подвектора, заканчивающегося в позиции $i-1$. Для того чтобы переменная *MaxEndHere* начала хранить сумму элементов максимального подвектора, оканчивающегося в позиции i, необходимо выполнить следующую операцию присваивания:

$$MaxEndHere := \max(MaxEndHere + A[i],\ A[i]);$$

а для того чтобы найти максимальную сумму *MaxSoFar* элементов подвектора, встретившегося до позиции i, надо выполнить операцию:

$$MaxSoFar := \mathrm{Max}(MaxSoFar,\ MaxEndHere).$$

Программа:

```
MaxSoFar: = A [1];
MaxEndHere: = A [1];
for i: = 2 to K do
begin
  MaxEndHere: = max(MaxEndHere + A[i], A [i]);
  MaxSoFar: = max(MaxSoFar, MaxEndHere);
end;
```

б) Для поиска минимальной суммы мы можем сначала умножить все элементы массива A на

-1, а затем искать, как и в пункте а), максимальную сумму.

в) Определим массив $C[0..k]$ такой, что $C[i]=A[1]+\ldots+A[i]$, $C[0]=0$. Заметим, что

$$S(M, N)=C[N]-C[M-1].$$

Сумма $S(M, N)$ элементов вектора $A[M]+\ldots+A[N]$ равна нулю, если $C[M-1]=C[N]$. Исходя из этого соображения, возьмем массив C, отсортируем его, затем найдем минимальную по модулю разность двух соседних элементов отсортированного массива (т. е. найдем два наименее отличающихся элемента массива C). Эта разность как раз и будет наиболее близким к нулю значением суммы $S(M, N)$.

г) Как и в предыдущем пункте, сформируем массив C, затем его отсортируем. Нам надо найти в этом массиве два элемента $C[i]$ и $C[j]$, значение разности которых наиболее близко к P. Пусть массив C упорядочен по неубыванию, и i и j — индексы текущих просматриваемых элементов массива C.

```
i:=1; j:=1;
MinSoFar:=abs(C[2]-C[1]-P); {Текущее значение минимальной разности}
while (i<=k) and (j<=k) do
begin
  if i<> j {если это не один и тот же элемент массива C}
  then MinSoFar:=min(MinSoFar, abs(C[j]-C[i]-P));
  if C[j]-C[i]>P
  then i:=i+1; {увеличиваем вычитаемое}
  else j:=j+1; {увеличиваем уменьшаемое}
end;
```

22. Заметим, что если мы знаем сумму $S[i]=X[i]+\ldots+X[i+M]$, то можем вычислить $S[i+1]$ по очевидной формуле

$$S[i+1]=S[i]+X[i+M+1]-X[i],$$

и нет необходимости во вложенном цикле для вычисления $S[i+1]$.

23. Пусть A — это K первых элементов массива X, а B — последних $N-K$. Необходимо из массива AB получить массив BA. Пусть есть подпрограмма *REVERSE* (i, j), которая реверсирует (меняет порядок элементов на обратный) часть массива X с индексами от i до j. Начав с массива AB, реверсируем часть A, получаем $(A')B$; реверсируем B, получаем $(A')(B')$; реверсируем весь массив, получаем $((A')(B'))'=BA$.

Продемонстрируем описанный алгоритм на примере. Пусть X есть последовательность 1, 2, 3, 4, 5, $K=3$:

REVERSE (1, K);	{3, 2, 1, 4, 5}
REVERSE (K+1, N);	{3, 2, 1, 5, 4}
REVERSE (1, K);	{4, 5, 1, 2, 3}

Глава 3. АЛГОРИТМЫ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ

§ 1. ПОИСК ДЕЛИТЕЛЕЙ ЧИСЛА. ПРОСТЫЕ ЧИСЛА

Натуральное число b называют ***делителем*** натурального числа a, если a представимо в виде произведения

$$a = bc, \qquad (1)$$

где c — натуральное число. В этом случае говорят, что число a делится без остатка на число b, или, короче, число a делится на число b. Из формулы (1) следует, что число a делится также на число c, т. е. c — делитель числа a. Например, $15 = 3 \cdot 5$, 3 и 5 — делители числа 15.

Число 1 является делителем любого натурального числа, поскольку любое натуральное число делится на 1 ($a : 1 = a$).

Число, делящееся на 2, называют ***четным;*** число, не делящееся на 2, называют ***нечетным.***

Кратным числа b называют число a, которое делится на b. Множество чисел, кратных данному числу b, бесконечно.

В программировании для определения того, является ли число b делителем числа a или нет, пользуются следующим свойством: если число a делится на число b, то остаток от деления равен 0.

```
если mod(a, b)=0
| то
|  P:=«Делится»                    (3.1)
иначе
|  P:=«Не делится»
все
```

Натуральное число a, не равное 1, называется ***простым,*** если оно делится только на себя и на 1, т. е. имеет

только два делителя. Натуральное число, отличное от 1 и не являющееся простым, называется ***составным.*** Другими словами, натуральное число называется составным, если оно имеет более двух делителей. Число 1 не относится ни к простым, ни к составным числам, поскольку имеет лишь один делитель. Наименьшим простым числом является число 2. Это единственное четное простое число. Остальные простые числа являются нечетными.

Для того чтобы найти делители числа $a>1$, мы должны попытаться делить данное число на все отличные от 1 числа, меньшие a. Если a не разделилось ни на одно из них, следовательно, оно является простым.

Массив B будем использовать для хранения делителей числа a, переменную P — для определения того, простое число или составное, переменную k — для обозначения номера текущего делителя числа a.

```
P:=«простое»
B[1]:=1
B[2]:=a
k:=2
нц для i от 2 до a-1                                    (3.2)
| если mod(a, i)=0
| | то
| |  k:=k+1
| |  B[k]:=i
| |  P:=«составное»
| все
кц
```

После выполнения алгоритма первые k элементов массива B будут содержать все делители числа a.

В приведенном алгоритме мы делили число a на все отличные от 1 числа, меньшие a. На самом деле делитель не может превышать $a/2$ (почему?). Учет этого очевидного факта приводит к увеличению скорости работы программы в два раза.

Алгоритм (3.2), кроме нахождения делителей числа a, определяет, является ли число a простым или составным. Если требуется только определить, простое число или составное, то данный алгоритм можно улучшить. В самом деле, цикл для i от 2 до $a-1$ выполняется $a-2$ раза. Однако для проверки простоты числа a достаточно выполнить данный цикл $[\sqrt{a}]$ раз ($[\sqrt{a}]$ — целая часть от корня квадратного из a). Это вытекает из следующих соображений. Если a — число составное, его можно представить в виде $a=b\cdot c$, где $b\leqslant\sqrt{a}$, $c\geqslant\sqrt{a}$. Если a делится на b, то оно будет делиться и на c. Если a не делится ни на одно число, меньшее или равное $\sqrt{a}$, то оно не будет делиться ни на какое другое число.

Запишем алгоритм проверки, является ли число a простым:

```
P:=«простое»
нц для i от 2 до int(sqrt(a))+1                    (3.3)
| если (mod(a, i)=0) и (i<a)
| | то
| | P:=«составное»
| все
кц
```

В цикле ДЛЯ к величине $int(sqrt(a))$ добавлена 1, так как значение $sqrt(a)$ является вещественным и может вычисляться с погрешностью.

Идеей уменьшения количества выполнений цикла можно воспользоваться и для нахождения делителей числа a. Попытайтесь сделать это самостоятельно.

Алгоритм определения простого числа можно еще улучшить, если останавливаться сразу после того, когда установили, что число не является простым (т. е. если a разделилось на какое-либо число).

```
P:=«простое»
i:=2
нц пока (i<=int(sqrt(a))+1) и (mod(a, i)<>0)
| i:=i+1                                          (3.4)
```

```
кц
если (mod(a, i)=0) и (i<a)
| то
|  P:=«составное»
все
```

* Множество простых чисел является бесконечным. Очевидно, что множество простых чисел, не превосходящих некоторого числа N, будет конечным. Возникает вопрос — как же найти эти простые числа?

Можно, конечно же, просмотреть первые N натуральных чисел и для каждого проверять, является ли оно простым. Однако данный метод, хоть и прост с первого взгляда, не является эффективным, так как программа будет работать очень медленно даже для небольших N. (Можете проверить!)

Лучше всего делать это методом, который предложил Эратосфен Киренский (ок. 276—194 гг. до н. э.), друг Архимеда. Называется метод — решето Эратосфена.

Суть метода следующая: выпишем подряд все числа от 1 до N. Первым стоит число 1. Оно не является простым. Вычеркнем это число. Следующее число 2. Это простое число. Оставляем его и вычеркиваем все числа, кратные 2. Для этого достаточно вычеркнуть каждое второе число, начиная счет с 3. Первым невычеркнутым числом будет 3. Это простое число. Оставляем его и вычеркиваем все числа, кратные 3, т. е. каждое третье число, начиная счет с 4. (При счете необходимо учитывать и ранее вычеркнутые числа, поэтому некоторые числа вычеркиваются второй раз: такими числами будут 6, 12, 18... .) После этой операции первым невычеркнутым, а значит простым, будет число 5. Оставляем это число и вычеркиваем все числа, кратные 5, т. е. каждое пятое число, начиная счет с 6. Затем переходим к следующему невычеркнутому числу 7 и так далее. Таким способом мы вычеркиваем все составные числа, останутся лишь простые. В таблице приведены результаты указанных действий для $N=50$; вычеркнутые числа подчеркнуты.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Метод Эратосфена получил название решета по следующим причинам. Древние греки рабочие записи вели заостренной палочкой на восковых дощечках. Такой палочкой Эратосфен прокалывал те места, где были написаны составные числа. После этого дощечка становилась похожей на решето. Применяя метод Эратосфена, как бы отсеивают, пропускают через решето все составные числа и оставляют только простые.

Для хранения чисел создадим массив $B[1..n]$. Вначале занесем в него число 2 и все нечетные числа, так как все четные, кроме числа 2, являются составными. Затем будем заменять нулями все числа, кратные 3. После этого найдем первое ненулевое число после числа 3 и будем заменять нулями все числа, кратные ему. Так продолжаем до тех пор, пока не просмотрим весь массив.

```
N:=div((N+1), 2)
B[1]:=2
нц для i от 2 до N                  :первоначальное за-
| B[i]:=2*i-1                       :полнение массива
кц
j:=2
нц пока j<=N                        :индекс первого
| i:=j+B[j]                         :обнуляемого
| нц пока i<=N
| | B[i]:=0                         :обнуление чисел,
| | i:=i+B[j]                       :кратных текущему
| кц
| j:=j+1                                          (3.5)
```

```
| нц пока (j<=N) и (B[j]=0)   :поиск очередного
| | j:=j+1                    :ненулевого числа
| кц
кц
```

Этот алгоритм работает достаточно быстро. Для $N=10\,000$ результат получается через 3 секунды (для *IBM PS/2—286*). Однако у него есть один недостаток: для хранения всех чисел требуется массив слишком большой размерности (5000 элементов для $N=10\,000$). Поэтому можно воспользоваться другим алгоритмом, который работает медленнее, однако не требует сохранения в массиве всех нечетных чисел.

В массиве будем хранить только простые числа. Вначале в массив поместим число 2. Рассматривая очередное нечетное число X, будем делить его на предшествующие ему простые числа. И если ни на одно из предшествующих простых чисел текущее не делится, то оно само является простым. Проверку на делимость числа нужно заканчивать после того, как проверена делимость на простое число, не превосходящее $[sqrt(X)]$.

```
B[i]:=2
j:=2
i:=3
нц пока i<=N
| k:=1
| d:=int(sqrt(i))+1
| нц пока (B[k]<=d) и (mod(i, B[k])<>0) и (k<j)
| | k:=k+1
| кц                                              (3.6)
| если B[k]>d
| | то
| | B[j]:=i
| | j:=j+1
| все
| i:=i+2
кц
```

Данный алгоритм на компьютере с 286-м процессором при $N=10\,000$ выдает результат через 12 секунд. Следует обратить внимание и на необходимость введения переменной d. Выражение, значение которого присваивается переменной d, можно вычислять непосредственно в условии цикла, однако это удлинит работу программы до 100 секунд. (Проверьте и подумайте, почему так происходит.)*

Вопросы для повторения

1. Как определить, делится ли данное число a на число b?
2. Как найти все делители числа?
3. Как определить, является ли число простым?
4. Как получить все простые числа, не превосходящие данного числа N?

§ 2. РАЗЛОЖЕНИЕ ЧИСЛА НА ПРОСТЫЕ МНОЖИТЕЛИ

Всякое число можно представить в виде произведения простых множителей, причем такое представление является единственным с точностью до порядка сомножителей. Этот факт, называемый основной теоремой арифметики, известен из курса математики 6-го класса. Напомним алгоритм разложения числа на простые множители.

Пусть число a разложено в произведение $p_1p_2\ldots p_k$, т. е.

$$a=p_1p_2\ldots p_k. \qquad (2)$$

Находить p_i будем по следующему алгоритму: разделим число a на наименьшее простое число 2, если остаток от деления равен 0, то $p_1=2$ и число a уменьшаем в 2 раза, т. е. $a:=a$ div 2. Затем опять делим a на 2, если разделилось, то $p_2=2$ и a опять уменьшаем в два раза. Так продолжаем до тех пор, пока a делится на 2. Если число a не разделилось на 2, то пытаемся делить a на следующее простое число 3. Если разделилось, то нахо-

дим очередное p_i и уменьшаем a в 3 раза. Продолжаем так до тех пор, пока число делится на 3. Затем пытаемся делить на 5, 7, 11, ... Закончим тогда, когда $a=1$.

В качестве примера рассмотрим разложение на простые множители числа 2520. Запись будем производить по знакомой схеме:

2520	2
1260	2
630	2
315	3
105	3
35	5
7	7
1	

$2520=2\cdot2\cdot2\cdot3\cdot3\cdot5\cdot7$

Для реализации данного алгоритма можно вначале сгенерировать все простые числа, не большие данного числа a, и затем делить a поочередно на них. Однако на практике это занимает достаточно времени, кроме того, понадобится дополнительный массив для хранения простых чисел.

Несколько изменим описанный алгоритм. Данное число будем делить на 2, а затем только на нечетные числа. Если какое-то нечетное число X является составным, то число a на него не разделится (так как число a уже разделилось на простые числа, произведение которых равно X, и они меньше X). Простые множители будем хранить в массиве B, k — номер текущего простого множителя, d — число, на которое делим.

```
k:=0
d:=2
нц пока a>1
| если mod(a, d)=0
| | то
| | k:=k+1                    (3.7)
| | B[k]:=d
| | a:=div(a, d)
```

```
    иначе
    если d=2
    | то
    | d:=d+1
    | иначе
    | d:=d+2
    все
  все
кц
```

После выполнения данного алгоритма первые k элементов массива B будут содержать простые числа, произведение которых равно a.

В разложении числа на простые множители могут быть равные числа. Объединяя равные множители, из формулы (2) получим формулу вида

$$a = p_1^{a_1} p_2^{a_2} \ldots p_m^{a_m}, \qquad (3)$$

где p_1, p_2, ..., p_m — различные простые числа, a_1, a_2, ..., a_m — некоторые натуральные числа. Каждое из a_i показывает, сколько раз входит множитель p_i в разложение числа a. Произведение в правой части равенства (3) называют ***каноническим разложением*** натурального числа a.

Так, число 2520 из нашего примера будет записано следующим образом: $2520 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$.

Вопросы для повторения

1. Всякое ли число при разложении на простые множители имеет больше одного множителя?
2. Как разложить число на простые множители?
3. Почему не применяют в информатике алгоритм разложения на простые множители, известный из курса математики?
4. Что такое каноническое разложение?

§ 3. ПОИСК НАИБОЛЬШЕГО ОБЩЕГО ДЕЛИТЕЛЯ (НОД) И НАИМЕНЬШЕГО ОБЩЕГО КРАТНОГО (НОК)

Если a и b — два натуральных числа и если число c таково, что a делится на c и b делится на c, то число c называют общим делителем чисел a и b. Произвольные два числа всегда обладают ***общим делителем.*** Таким делителем является число 1. Если других общих делителей нет, то числа a и b называют ***взаимно простыми.***

3.1. Поиск НОД

Число d называют ***наибольшим общим делителем*** **(НОД)** чисел a и b, если d является общим делителем чисел a и b; d делится на любой другой общий делитель. Другими словами, d — наибольший из всех общих делителей чисел a и b.

В курсе математики приводился следующий алгоритм нахождения НОД. Необходимо разложить числа a и b на простые множители и затем выбрать те из них, которые входят и в одно и в другое разложение. Рассмотрим пример: НОД (2520, 2475).

2520	2	2475	3
1260	2	825	3
630	2	275	5
315	3	55	5
105	3	11	11
35	5	1	
7	7		
1			

Общие множители подчеркнуты. НОД (2520, 2475) = $= 3 \cdot 3 \cdot 5 = 45$.

Данный алгоритм на практике обычно не применяется, так как разложение числа на простые множители уже является достаточно трудоемкой задачей, а ведь еще потребуется найти среди них одинаковые. Поэтому очень

часто для нахождения НОД применяют метод, который называется ***алгоритмом Евклида.***

Это один из самых древних и популярных алгоритмов. Он описан еще в «Началах» Евклида.

Прежде чем описать сам алгоритм, укажем несколько свойств НОД, на которые опирается алгоритм Евклида.

Пусть a и b — отличные от нуля натуральные числа, причем $a > b$, тогда:

1) НОД $(a, b) =$ НОД $(a-b, b)$;
2) НОД $(a, a) = a$;
3) НОД $(a, 0) = a$.

Свойства 2) и 3) являются очевидными.

Примечание. Докажем свойство 1).

Свойство будет доказано, если мы установим, что множество общих делителей чисел a и b совпадает с множеством общих делителей чисел $a-b$ и b, т. е. всякий общий делитель X чисел a и b является делителем чисел $a-b$ и b, и наоборот. Если X — делитель a и b, то $a = kX$ и $b = lX$ для некоторых k и l. Тогда $a-b=(k-l)X$, т. е. X является общим делителем чисел $a-b$ и b. Наоборот, пусть $a-b=mX$ и $b=nX$ для некоторых m и n. Складывая, получаем $a=(n+m)X$. Таким образом, X является общим делителем a и b.

Перечисленные равенства подсказывают идею алгоритма нахождения НОД: нужно от большего отнимать меньшее, до тех пор, пока числа не станут равными (свойство 1)), после чего НОД найдется по свойству 2).

Пример: Найти НОД (530, 155), $a = 530$, $b = 155$.

a	b
530	155
375	155
220	155
65	155
65	90
65	25
40	25
15	25
15	10

5	10
5	5

Алгоритм будет следующим:

```
нц пока a<>b
| если a>b
| | то
| | a:=a-b                                    (3.8)
| | иначе
| | b:=b-a
| все
кц
NOD:=a
```

Если проанализировать работу данного алгоритма, то можно заметить, что одно и то же число нужно отнимать несколько раз. Этого можно избежать, если вместо нескольких вычитаний находить остаток от деления большего числа на меньшее (подумайте почему). В этом случае свойство 1) можно записать в виде: НОД (a, b)= =НОД $(a \bmod b, b)$. Для окончания работы алгоритма нужно воспользоваться свойством 3).

Пример: Найти НОД (530, 155), $a=530$, $b=155$.

a	b
530	155
65	155
65	25
15	25
15	10
5	10
5	0

Алгоритм будет выглядеть следующим образом:

```
нц пока (a<>0) и (b<>0)
| если a>b
| | то
| | a:=mod(a, b)
| | иначе
| | b:=mod(b, a)
| все                                         (3.9)
кц
```

```
если a=0
| то
| NOD:=b
| иначе
| NOD:=a
все
```

Последний оператор ЕСЛИ можно заменить оператором NOD:=a+b. Подумайте почему.

3.2. Поиск НОК

Если a и b — два натуральных числа и если число c таково, что c делится на a и c делится на b, то число c называют ***общим кратным*** чисел a и b. Произвольные два числа всегда обладают общим кратным. Таким кратным является число, равное произведению ab.

Число d называют ***наименьшим общим кратным*** **(НОК)** чисел a и b, если d является общим кратным чисел a и b; на d делится любое другое общее кратное. Другими словами, d — наименьшее из всех общих кратных чисел a и b.

Для нахождения НОК можно воспользоваться алгоритмом, известным из курса математики. Нужно разложить числа a и b на простые множители, а затем из двух разложений выбрать те сомножители, которые входят хотя бы в одно разложение.

Пример: НОК(52, 65).

52	2	65	5
26	2	13	13
13	13	1	
1			

$52=2\cdot2\cdot13$; $65=5\cdot13$; НОК(52, 65)$=2\cdot2\cdot13\cdot5=260$.

Однако пользоваться этим алгоритмом на практике не следует (причины описаны выше). Для нахождения НОК можно воспользоваться следующим свойством: НОК$(a, b)=a\cdot b/$НОД(a, b) (докажите самостоятельно).

Для нахождения НОД следует воспользоваться алгоритмом Евклида.

Вопросы для повторения

1. Что такое НОД?
2. Какие способы вычисления НОД вам известны?
3. Что такое НОК?
4. Как найти НОК трех чисел?

§ 4. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ. ВЫДЕЛЕНИЕ ЦИФР ЧИСЛА

В десятичной системе счисления, как и в любой другой позиционной системе счисления, натуральное число может быть записано в виде суммы разрядных слагаемых, т. е. в виде суммы единиц, десятков, сотен, тысяч и т. д. Так, например, число 6487 содержит 7 единиц, 8 десятков, 4 сотни и 6 тысяч и может быть записано в виде выражения:

$$6487 = 6 \cdot 1000 + 4 \cdot 100 + 8 \cdot 10 + 7.$$

Цифры 6, 4, 8, 7 являются цифрами числа 6487, а 1000, 100, 10 — разрядные единицы. Каждая разрядная единица является степенью числа 10, поэтому представление числа в виде суммы разрядных слагаемых чаще записывают так:

$$6487 = 6 \cdot 10^3 + 4 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0.$$

Вид числа в левой части равенства будем называть ***обычным*** представлением числа в десятичной форме. При написании программ такие числа хранятся в стандартных числовых типах, например: ЦЕЛ, ВЕЩ. При работе с числами в обычном представлении мы не имеем возможности непосредственно обращаться к цифрам этого числа, однако иногда это бывает необходимо. Например, если требуется вычислить значение выражения, которое не помещается в стандартном числовом типе (100!), то число представляют в виде массива цифр. Такое представление будем называть ***табличным*** представлением числа.

Размещение цифр числа в массиве возможно двумя способами. Связано это с тем, что разряды чисел нумеруются справа налево, а число читается слева направо, поэтому цифры числа можно располагать в массиве, начиная с первой (старший разряд), а можно — начиная с последней (младший разряд). Порядок, при котором цифра старшего разряда является первым элементом массива, назовем ***прямым***. ***Обратным*** назовем порядок, при котором первый элемент в массиве является цифрой младшего разряда.

Можно задавать число и как литерную величину, тогда мы получим доступ к каждой отдельной цифре, однако с ними, без преобразования в числовой тип, нельзя выполнять арифметические операции.

Рассмотрим способы преобразования обычного представления числа в табличное и наоборот.

4.1. Преобразование числа из обычного представления в табличное

Пусть задано число в обычном представлении, необходимо получить табличное представление данного числа.

Как видно из представления числа в виде суммы разрядных слагаемых, каждое слагаемое, кроме последнего, разделится на 10 без остатка. Поэтому остаток от деления числа на 10 будет равен последней цифре числа. Эту цифру мы помещаем в массив. Если мы размещаем в массиве цифры числа, начиная с последней, то первому элементу массива присваивается значение полученной цифры. Если размещаем цифры, начиная с первой, то полученную цифру помещаем в элемент массива с индексом, равным количеству цифр в числе (количество цифр необходимо подсчитать до того, как начнем получать сами цифры). Далее находим целую часть от деления исходного числа на 10 (или, говоря другими словами, отбрасываем последнюю цифру числа) и снова на-

ходим остаток от деления данного числа на 10, это и будет следующая цифра. Так продолжаем до тех пор, пока в числе не останется ни одной цифры.

Данный алгоритм размещает цифры числа в массив в обратном порядке. Для хранения цифр числа N будем использовать массив B, k — номер текущей цифры в массиве B.

```
k:=0
нц пока N>=1
| k:=k+1
| B[k]:=mod(N, 10)                    (3.10)
| N:=div(N, 10)
кц
```

Первые k элементов массива B содержат цифры числа N, записанные в обратном порядке.

Изменим алгоритм для получения цифр числа в прямом порядке.

```
k:=0
L:=N
нц пока L>=1
| L:=div(L, 10)
| k:=k+1
кц
m:=k                                  (3.11)
нц пока N>=1
| B[k]:=mod(N, 10)
| N:=div(N, 10)
| k:=k-1
кц
```

Первый цикл ПОКА считает количество цифр в числе, второй — получает цифры числа. Переменная L служит для того, чтобы сохранить значение исходного числа, так как внутри цикла переменная изменяет свое значение. Если бы в первом цикле использовалась переменная N, то после выполнения цикла ее значение

равнялось бы 0 (почему?) и второй цикл не выполнился бы ни разу. В переменной m сохраняется количество цифр числа, что, вообще говоря, не всегда необходимо.

4.2. Преобразование табличного представления числа в обычное

Решим обратную задачу, т. е. получим обычное представление числа из табличного.

Пусть некоторое число a задано своими цифрами:

$$a=\overline{a_1a_2a_3\ldots a_n}.$$

Черта сверху обозначает, что a_1, a_2, a_3, ..., a_n — цифры числа a (a_1 — цифра старшего разряда). Запишем это число в виде суммы разрядных слагаемых:

$$a=a_1\cdot 10^{n-1}+a_2\cdot 10^{n-2}+a_3\cdot 10^{n-3}+\ldots+a_{n-1}\cdot 10+a_n. \quad (4)$$

Для того чтобы получить значение числа a, необходимо вычислить значение выражения, стоящего в правой части равенства (4). Для этого проделаем следующие преобразования: вынесем 10 за скобки из тех слагаемых, для которых это возможно.

$$a=a_1\cdot 10^{n-1}+a_2\cdot 10^{n-2}+a_3\cdot 10^{n-3}+\ldots+a_{n-1}\cdot 10+a_n=$$
$$=10(a_1\cdot 10^{n-2}+a_2\cdot 10^{n-3}+a_3\cdot 10^{n-4}+\ldots+a_{n-1})+a_n.$$

Затем с выражением в скобках проделаем то же самое, т. е. вынесем 10 за скобки из тех слагаемых, для которых это возможно. Продолжим действовать таким образом до тех пор, пока это будет возможно:

$$a=a_1\cdot 10^{n-1}+a_2\cdot 10^{n-2}+a_3\cdot 10^{n-3}+\ldots+a_{n-1}\cdot 10+a_n=$$
$$=10(a_1\cdot 10^{n-2}+a_2\cdot 10^{n-3}+a_3\cdot 10^{n-4}+\ldots+a_{n-1})+a_n=$$
$$=10(10(a_1\cdot 10^{n-3}+a_2\cdot 10^{n-4}+a_3\cdot 10^{n-5}+\ldots)+a_{n-1})+a_n=$$
$$\ldots$$
$$=10(10\cdot\ldots(10(a_1\cdot 10+a_2)+a_3)+\ldots+a_{n-1})+a_n.$$

Теперь для вычисления значения числа a достаточно:

1) первую цифру числа умножить на 10 и к произведению прибавить вторую цифру;
2) полученную сумму умножить на 10 и прибавить следующую цифру;
3) пункт 2) выполнять до тех пор, пока не используем все заданные цифры числа.

Описанный выше алгоритм называется ***схемой Горнера.*** Работа алгоритма рассмотрена для случая, когда цифры числа расположены в массиве в прямом (a_1 — цифра старшего разряда) порядке.

Число будем получать в переменной a. Цифры числа хранятся в массиве B (элемент $B[1]$ содержит цифру старшего разряда).

```
a:=0
нц для i от 1 до N
| a:=a*10+B[i]                    (3.12)
кц
```

Для получения числа a, цифры которого заданы в обратном порядке (первый элемент массива содержит цифру младшего разряда), можно воспользоваться схемой Горнера, начиная обработку с последнего элемента массива.

Можно воспользоваться и другим алгоритмом. Каждая цифра должна умножаться на соответствующую ей разрядную единицу: первая — на 1, вторая — на 10, третья — на 100 и т. д. Каждая следующая разрядная единица в 10 раз больше предыдущей. Число a будем накапливать следующим образом: будем брать очередную цифру, умножать ее на соответствующую ей разрядную единицу и прибавлять полученное произведение к уже накопленному числу. Разрядную единицу увеличим в 10 раз. Так продолжаем до тех пор, пока не используем все цифры числа.

В переменной r будем хранить значение разрядной единицы, цифры числа будем хранить в массиве B.

```
a:=0
r:=1
нц для i от 1 до N
| a:=a+B[i]*r
| r:=r*10                                    (3.13)
кц
```

Проанализируйте алгоритмы (3.12) и (3.13) на скорость работы (посчитайте количество выполненных арифметических операций).

Вопросы для повторения

1. Какой порядок расположения цифр числа в массиве называется прямым? Обратным?
2. Как получить цифры числа в обратном порядке? В прямом порядке?
3. Как получить число из массива цифр?
4. Что такое схема Горнера?

§ 5. ПЕРЕВОД ЧИСЕЛ ИЗ ОДНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДРУГУЮ

Задачи перевода чисел из одной системы счисления в другую можно разделить на две группы: перевод из десятичной системы счисления в какую-то другую и из какой-то в десятичную (рассматриваем только позиционные системы счисления). Если требуется, например, перевести число из троичной системы счисления в семеричную, это можно сделать через десятичную систему счисления. Если делать это напрямую, то придется моделировать арифметические действия либо в троичной, либо в семеричной системе счисления.

Для записи числа в любой системе счисления, кроме десятичной, мы будем пользоваться только табличным представлением числа.

Алгоритм перевода числа из десятичной системы счисления в какую-то другую известен. Нужно разделить число на основание системы счисления, в кото-

рую переводят, и найти остаток. Если целая часть частного получилась не равной нулю, то ее делят на основание системы счисления. Так продолжают до тех пор, пока целая часть частного не станет равной нулю. Затем выписывают остатки в порядке, обратном их получению. Эти остатки являются цифрами искомого числа.

Пример. Переведем число 345 в семеричную систему счисления.

$$\begin{array}{r|l} 325 & 7 \\ \hline \mathbf{28} & 46 \\ \hline 45 & \mathbf{42} \\ \hline \mathbf{42} & \mathbf{4} \\ \hline \mathbf{3} & \end{array} \qquad \begin{array}{r|l} 46 & 7 \\ \hline \mathbf{42} & 6 \\ \hline \mathbf{4} & \end{array} \qquad \begin{array}{r|l} 6 & 7 \\ \hline \mathbf{0} & 0 \\ \hline \mathbf{6} & \end{array}$$

Остатки выделены жирным шрифтом.

$$325_{10}=643_7.$$

По аналогии с записью числа в виде суммы разрядных слагаемых в десятичной системе счисления, в семеричной системе счисления число можно записать следующим образом:

$$6437=6\cdot 7^3+4\cdot 7^1+3\cdot 7^0.$$

Вообще натуральное число a в позиционной системе счисления может быть записано следующим образом:

$$a=a_1\cdot p^{n-1}+a_2\cdot p^{n-2}+a_3\cdot p^{n-3}+\ldots+a_{n-1}\cdot p+a_n, \qquad (5)$$

где a_1, a_2, a_3, ..., a_n — цифры числа a, p — основание системы счисления, причем $0\leqslant a_i<p$, $1\leqslant i\leqslant n$.

Запись (5) отличается от (4) только тем, что основание десятичной системы счисления 10 заменим на произвольное число p (основание новой системы счисления).

Цифры числа в новой системе счисления можем получать, пользуясь алгоритмом (3.10), опять же заменив число 10 числом p. Цифры числа, записанного в системе счисления с основанием p, будут храниться в массиве B в обратном порядке.

Запишем алгоритм перевода числа N из десятичной системы счисления в систему счисления с основанием p:

```
k:=0
нц пока N>=1
| k:=k+1
| B[k]:=mod(N, p)                    (3.14)
| N:=div(N, p)
кц
```

С помощью алгоритмов (3.10) и (3.12) можно осуществить и перевод из какой-то системы счисления (с основанием меньше 10) в десятичную. Для этого нужно вычислить значение выражения (5) (алгоритм (3.12) или (3.13), заменяя 10 на p). Число получим в обычном представлении. Табличное представление можно получить, применив алгоритм (3.10) к полученному результату.

Этот алгоритм сформулируйте самостоятельно.

Вопросы для повторения

1. **Как перевести число из десятичной системы счисления в какую-то другую?**
2. **Как перевести число в десятичную систему счисления?**

§ 6. ДЕЛИМОСТЬ ЧИСЕЛ

Для определения, делится ли одно число на другое, нужно, как упоминалось выше, найти остаток от деления одного числа на другое и проверить, равен ли он нулю. Если остаток равен нулю, то число делится, если не равен нулю — нет. Таким образом, проверка на делимость сводилась к нахождению остатка от деления одно-

го числа на другое. Для чисел в обычном представлении остаток находится с помощью стандартной операции mod. При решении этой задачи вручную мы делим одно число на другое «в столбик».

Рассмотрим, как найти остаток от деления числа, заданного таблично, на число в обычном представлении.

В десятичной системе счисления число представимо в виде (4) (§ 4, п. 4.2):

$$a = a_1 \cdot 10^{n-1} + a_2 \cdot 10^{n-2} + a_3 \cdot 10^{n-3} + \ldots + a_{n-1} \cdot 10 + a_n,$$

где $a_1, a_2, a_3, \ldots, a_n$ — цифры числа a, причем $0 \leqslant a_i < 10$, $1 \leqslant i \leqslant n$.

Пусть нам требуется найти остаток от деления числа a на число m.

Прежде чем это делать, укажем на некоторые свойства остатков. Пусть c, d, n — некоторые натуральные числа, тогда:

1) $(c+d) \bmod n = ((c \bmod n) + (d \bmod n)) \bmod n$;

2) $c \cdot d \bmod n = c\,(d \bmod n) \bmod n = ((c \bmod n)\, d) \bmod n$;

3) $c^d \bmod n = ((c^{d-1} \bmod n)\, c) \bmod n$.

Примечание. Докажем свойство 1). Пусть $c_1 = c \operatorname{div} n$, $c_2 = c \bmod n$, $d_1 = d \operatorname{div} n$, $d_2 = d \bmod n$; тогда $c = c_1 \cdot n + c_2$, $d = d_1 \cdot n + d_2$;

$$(c+d) \bmod n = (c_1 \cdot n + c_2 + d_1 \cdot n + d_2) \bmod n = ((c_1 + d_1)\, n + c_2 + d_2) \bmod n =$$
$$= (c_2 + d_2) \bmod n = ((c \bmod n) + (d \bmod n)) \bmod n.$$

Доказательства свойств 2) и 3) аналогичны. Попытайтесь доказать их самостоятельно.

Найдем остаток от деления числа a (представление (4), п. 4.2) на число m, пользуясь свойствами 1) и 2) (сначала применим свойство 1), затем к каждому слагаемому свойство 2)):

$$a \bmod m = (a_1 \cdot 10^{n-1} + a_2 \cdot 10^{n-2} + a_3 \cdot 10^{n-3} + \ldots +$$
$$+ a_{n-1} \cdot 10 + a_n) \bmod m = ((a_1 \cdot 10^{n-1}) \bmod m +$$
$$+ (a_2 \cdot 10^{n-2}) \bmod m +$$
$$+ \ldots + (a_{n-1} \cdot 10) \bmod m + a_n \bmod m) \bmod m =$$
$$((a_1 (10^{n-1} \bmod m) \bmod m) + (a_2 (10^{n-2} \bmod m) \bmod m) +$$
$$+ \ldots + (a_{n-1} (10 \bmod m) \bmod m) + a_n \bmod m) \bmod m.$$

Из последнего равенства вытекает алгоритм нахождения остатка. Находим остаток от деления очередной разрядной единицы на число m, пользуясь свойством 3). Умножаем полученное число на значение цифры, соответствующей данной разрядной единице, прибавляем к тому, что получили на предыдущих шагах, и снова находим остаток от деления на m. Так продолжаем до тех пор, пока не используем все цифры числа a.

Значение искомого остатка будем накапливать в переменной s, значение остатка разрядной единицы — в переменной r, цифры числа будут храниться в массиве B (в обратном порядке, т. е. элемент $B[1]$ соответствует цифре младшего разряда).

```
s:=0
r:=1
нц для i от 1 до N                                    (3.15)
| s:=mod((s+B[i]*r), m)
| r:=mod(r*10, m)
кц
```

Алгоритм (3.15) можно применять для получения остатка от деления числа a на число m и в том случае, если число a, заданное в виде массива цифр, записано в любой позиционной системе счисления (цифры числа должны быть записаны своими десятичными аналогами). Для этого достаточно число 10 в алгоритме (3.15) заменить основанием системы счисления p. Остаток от деления будет получен в десятичной системе счисления. Если есть необходимость, его можно перевести в систему счисления с основанием p.

Для получения остатка от деления можно также воспользоваться схемой Горнера. Для этого представление числа ((4), п. 4.2) необходимо преобразовать по схеме Горнера, а затем применить свойства остатков так, как это сделано выше. (Данный алгоритм напишите самостоятельно.)

Вопросы для повторения

1. Какие свойства остатков вы знаете?
2. Как найти остаток от деления числа, заданного в виде массива цифр, на число, заданное непосредственно?
3. Как можно использовать схему Горнера для нахождения остатка от деления числа, заданного в виде массива цифр, на число, заданное непосредственно?

§ 7. ДЕЙСТВИЯ С МНОГОЗНАЧНЫМИ (БОЛЬШИМИ) ЧИСЛАМИ

В данном пункте под многозначными числами мы будем понимать числа, заданные табличным представлением в десятичной системе счисления.

Работа с многозначными числами предполагает моделирование арифметических действий над двумя числами, заданными массивами своих цифр.

Алгоритмы работы с такими массивами будут напоминать знакомые из курса математики действия в столбик.

Пусть два положительных числа A и B заданы как массивы цифр, причем $A[1]$ и $B[1]$ содержат цифры младшего разряда. (Подумайте, почему такой порядок цифр для данной задачи удобнее.) Число A состоит из N цифр, а число B — из M цифр.

7.1. Сложение многозначных чисел

Найдем сумму чисел A и B. Будем складывать элементы массивов с одинаковыми номерами и хранить полученные результаты, как элементы массива C. Если на каком-то шаге получим $C[i] \geqslant 10$, то на месте $C[i]$ оставляем остаток от деления $C[i]$ на 10 (количество единиц данного разряда), а к элементу $C[i+1]$ прибавим 1 (перенос 1 в следующий разряд).

Если число A состоит из N цифр, а число B — из M цифр, то число C имеет либо $\max(N, M)$ цифр, либо

$\max(N, M)+1$ цифру. Обозначим через K величину $\max(N, M)+1$.

Перед выполнением сложения следует дополнить незначащими нулями число с меньшим количеством цифр так, чтобы количество цифр уравнялось. Этого можно достигнуть и обнулив массивы A и B до ввода цифр.

Алгоритм сложения двух многозначных чисел будет следующим:

```
если N>M
| то
|  K:=N
| иначе
|  K:=M
все
K:=K+1
нц для i от 1 до K
| C[i]:=0                                        (3.16)
кц
нц для i от 1 до K
| C[i]:=A[i]+B[i]+C[i]
| если C[i]>=10
| | то
| | C[i+1]:=C[i+1]+1
| | C[i]:=mod(C[i], 10)
| все
кц
если C[K]=0
| то
| K:=K-1
все
```

После выполнения данного алгоритма в первых K элементах массива C будет храниться сумма чисел A и B, $C[1]$ — цифра младшего разряда. Последняя проверка позволяет убрать незначащий нуль из старшего разряда числа C.

7.2. Вычитание многозначных чисел

Для определенности будем считать, что $A > B$. Если нет, то необходимо поменять числа местами, а результат сделать отрицательным. (Подумайте, как определить, какое из чисел больше.)

Если число A состоит из N цифр, тогда число B не может иметь более чем N цифр. Разность будет содержать не более чем N цифр.

Найдем разность чисел A и B. Будем вычитать элементы массива B из элементов массива A с соответствующими номерами. Полученные результаты будем хранить в массиве C. Если на каком-то шаге получим $C[i] < 0$, то к элементу $C[i]$ прибавляем 10, а от элемента $C[i+1]$ отнимаем 1 (забираем 1 из следующего разряда).

Алгоритм будет следующим:

```
нц для i от 1 до N
| C[i]:=0
кц
нц для i от 1 до N
| C[i]:=A[i]-B[i]
| если C[i]<0
| | то
| | C[i]:=C[i]+10                    (3.17)
| | C[i+1]:=C[i+1]-1
| все
кц
нц пока C[N]=0
| N:=N-1;
кц
```

Последний цикл позволяет убрать незначащие нули в начале числа C. Данный алгоритм после некоторых дополнений можно применять для вычитания чисел с разными знаками. Подумайте, что нужно изменить.

7.3*. Произведение многозначных чисел

Алгоритм нахождения произведения будет заключаться в следующем: число A будем поочередно умножать на каждую из цифр числа B. Полученный на данном шаге результат будем накапливать в массиве C, прибавляя к тому, что уже получили на предыдущих шагах. При умножении на $B[i]$ результат начинаем прибавлять к $C[i]$, т. е. учитываем сдвиг на разряд.

При такой организации вычислений значения элементов массива C могут стать большими либо равными 10. Можно после получения очередного $C[i]$ проверять его и, если $C[i] \geqslant 10$, поступать так, как поступали при вычислении суммы. Тогда количество проверок будет равно $N \cdot M$. Разумнее разнос по разрядам выполнить после того, когда вычисление произведения закончено. Количество проверок будет равно количеству цифр в произведении $A \cdot B$. Количество цифр произведения двух чисел не превосходит $K = N + M$. (Почему?)

Алгоритм вычисления произведения будет следующим:

```
K:=N+M;
нц для i от 1 до K
| C[i]:=0
кц                                                    (3.18)
нц для i от 1 до M
| нц для j от 1 до N
| | C[i+j-1]:=A[j]*B[i]+C[i+j-1]
| кц
кц
нц для i от 1 до K-1
| C[i+1]:=C[i+1]+C[i] div 10;
| C[i]:=C[i] mod 10;
кц
нц пока C[k]=0
| K:=K-1;
кц
```

Полученные алгоритмы сложения, вычитания, умножения многозначных чисел можно использовать для моделирования действий над числами в какой-либо системе счисления. Для этого необходимо помнить, что если у нас система счисления имеет основание p, то p единиц одного разряда образуют единицу нового разряда. Другими словами, необходимо число 10 в алгоритмах (3.16), (3.17), (3.18) заменить основанием системы счисления.

Вопросы для повторения

1. Как сложить два многозначных числа?
2. Как найти разность двух многозначных чисел?

ЗАДАЧИ ДЛЯ ПОВТОРЕНИЯ

1. Определить, является ли данное число четным.

2. Два натуральных числа называют дружественными, если каждое из них равно сумме всех делителей другого, кроме самого этого числа. Найти все пары дружественных чисел, лежащих в диапазоне от n до k.

3. Найти натуральное число из диапазона от n до k, которое имеет наибольшее количество делителей.

4. Разложить дробь p/q на сумму дробей вида $1/n$. Например, при $p=3$, $q=7$

$$\frac{3}{7}=\frac{1}{3}+\frac{1}{11}+\frac{1}{231}.$$

5. Найти все совершенные числа, меньшие N. Число совершенно, если оно равно сумме всех своих делителей, за исключением самого числа.

6. Любую целочисленную денежную сумму, большую $7p.$, можно выплатить без сдачи трешками и пятерками (докажите). Для данного $n>7$ найти все такие целые неотрицательные a и b, что $3a+5b=n$.

7. Сообщество роботов живет по следующим законам:

1) один раз в начале года они объединяются в группы по 3 или по 5 роботов;

2) за год группа из 3 роботов собирает 5 новых, а группа из 5 роботов собирает 9 новых;

3) роботы объединяются так, чтобы собрать за год наибольшее количество роботов; каждый робот живет 3 года после сборки.

Известно начальное количество роботов. Все они только что собраны.

Сколько роботов будет через N лет?

8. Даны два натуральных числа n и b, причем b — нечетное число больше 1.

Определить, что делает следующий ниже фрагмент программы:

```
нц пока n >= b
| если mod(n, 2)=0
| | то
| |  n:=div(n, 2)
| | иначе
| |  n:=n-b
| все
кц
если n=0
| то
|  a:=«да»
| иначе
|  a:=«нет»
все
```

9. Два двузначных числа, записанных одно за другим, образуют четырехзначное число, которое делится на их произведение. Найти эти числа.

10. Найти натуральные числа из отрезка $[n; k]$, количество делителей у которых является произведением двух простых чисел.

11. Дано натуральное число n. Найти четверки про-

стых чисел и меньших n, принадлежащих одному десятку (например 11, 13, 17, 19).

12. Дано натуральное число n. Выяснить, имеются ли среди чисел n, $n+1$, ..., $2n$ числа-близнецы, т. е. простые числа, разность между которыми равна 2.

13. Дано натуральное число n. Получить все числа, взаимнопростые с n и меньше его.

14. Дано натуральное число n. Получить все его простые делители.

15. Даны натуральные числа p и q. Получить все делители числа p, взаимнопростые с q.

16. Найти k-е простое число в арифметической прогрессии 11, 21, 31, 41, 51, 61, ... Привести ответ для $k=$ $=1$, 10, 100, 1000 и т. д.

17. Сократить дробь a/b (a, b — натуральные числа).

18. Даны 4 целых числа a, b, c, d. Написать программу, вычисляющую сумму обыкновенных дробей $a/b+$ $+c/d$ в виде x/y.

19. Дано натуральное число n. Определить количество сотен (тысяч, миллионов) в нем.

20. Ввести период дроби. Напечатать числитель и знаменатель.

21. Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведенных в n-ю степень, равна самому числу ($153=1^3+5^3+3^3$). Получить все числа Армстронга для $n=2$, 3, 4.

22. Дано натуральное число n. Чему равна сумма его цифр?

23. Дано натуральное число n. Верно ли, что сумма цифр этого числа является нечетной?

24. Дано натуральное число n. Верно ли, что это число содержит ровно три одинаковые цифры? Если да, то укажите эти цифры.

25. Посчитать сумму цифр всех целых чисел от 1 до N.

26. Дано натуральное число n. Выбросить из записи числа n все цифры, равные 1, оставив при этом прежним

порядок остальных цифр. Например, из 5101234 → 50234.

27. Из записи натурального числа выбросить цифры 1 и 5, оставив прежним порядок цифр. Например, число 527012 преобразуется в число 2702.

28. Найти такие две различные наименьшие степени натурального числа n, у которых три последние цифры одинаковы.

29. Дано натуральное число n. Выбросить из записи числа все четные цифры.

30. Дано натуральное число n. Определить, является ли оно палиндромом (читается одинаково с начала и с конца).

31. Найти все числа из диапазона от n до m, которые при возведении в квадрат дают палиндром.

32. Найти все числа-палиндромы из диапазона от n до m, которые при возведении в квадрат также дают палиндромы.

33. Найти остаток от деления числа, записываемого с помощью k семерок, на число a, k и a — заданные натуральные числа.

34. Все натуральные числа выписаны подряд, начиная с единицы. Определить, какая цифра стоит на N-м месте.

35. На интервале (1000; 9999) найти все простые числа, каждое из которых обладает тем свойством, что сумма первой и второй цифр записи этого числа равна сумме третьей и четвертой.

36. Высадившись на планете Альфа, населенной разумными «осьминогами», космонавты увидели написанную на стене близлежащего строения формулу $99 \cdot 99 = 1210$. Сколько щупальцев было у населявших планету Альфа «осьминогов»?

37. Среди простых чисел, не превосходящих N, найти такое, в двоичной записи которого максимальное число единиц.

38. Найти двоичное представление для чисел Ферма, которые представимы в виде $2^{2^l}+1$.

39. Задана последовательность длины N, состоящая из единиц и нулей. Определить количество M-значных двоичных чисел ($M \leqslant N$), входящих в указанную последовательность, которые делятся на 21.

40. Напечатать шестнадцатеричную таблицу умножения так же, как печатают обычную на обложке тетради.

41. Десятичная дробь определяется следующим образом:

$$[-]abc...d[.efg...h],$$

где a, b, c, ..., d, e, f, g, ..., h — десятичные цифры от 0 до 9, элементы в квадратных скобках не являются обязательными. Найти корень уравнения $X/A=B$ (A и B — десятичные дроби), представив его в виде десятичной дроби, сохранив все ее десятичные знаки.

42. Написать программу, которая вычисляет значение выражения.

$$1-\frac{1}{2}+\frac{1}{3}-\ldots-(-1)^n\frac{1}{n}.$$

Ответ представить в виде несократимой дроби p/q, где p и q — натуральные числа.

43. Дана последовательность

$$A[1]=1,\ A[N+1]=A[N]+1/(A[N]+1).$$

Составить программу, в которой используются только целые числа, для печати N-го члена последовательности в виде обыкновенной дроби. Например:

при $N=3$ должно быть напечатано 19/10,

при $N=4$ должно быть напечатано 651/290.

44. Вывести на экран картинку, изображающую умножение «столбиком» двух заданных целых чисел.

45. Написать программу, которая выводит картинку, изображающую умножение «столбиком» двух данных чисел.

Например, сомножители: 398.56, 85.03.

$$
\begin{array}{r}
39856 \\
\times\ 8503 \\
\hline
119568 \\
+199280 \\
318848 \\
\hline
338895568
\end{array}
$$

Произведение: 33889.5568.

46. Определить количество повторений каждой из цифр 0, 1, 2, ..., 9 в числе N^N, $N \leqslant 1000$.

47. Найти частное от деления a/b с точностью k цифр после запятой. Числа a и b вводятся с не более чем 200 цифрами.

48. Даны две таблицы: $M_1, M_2, \ldots, M_{10}$; $N_1, N_2, \ldots, N_{10}$. Каждый элемент либо одна из цифр 0, 1, ..., 9, либо десятичная точка (она появляется не более одного раза).

Написать алгоритм сравнения «записанных» в таблицу величин. Например:

	1	2	3	4	5	6	7	8	9	10
M	3	4	1	.	7	4	9	0	0	0
N	0	0	0	0	0	0	0	3	4	2

Результат: «$M < N$», так как $341.749 < 342$.

49. Даны целые числа m и n ($0 < m \leqslant 12$, $0 \leqslant n < 60$), указывающие момент времени m часов n минут. Определить наименьшее время (число полных минут), которое должно пройти до того момента, когда часовая и минутная стрелки на циферблате

совпадут;

расположатся перпендикулярно друг другу.

50. Даны n положительных чисел $A_1, A_2, \ldots, A_n$ и m положительных чисел $B_1, B_2, \ldots, B_m$. Сумма всех A_i равна сумме всех B_j. Доказать, что можно построить (и описать как) массив $C[i, j]$; $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, который удовлетворяет следующим требованиям:

1) все (i, j) неотрицательные числа;
2) в массиве по крайней мере $(m-1)(n-1)$ нулей;
3) сумма каждой i-й строки равна A_i;
4) сумма каждого j-го столбца равна B_j.

ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Число задано своим двоичным представлением (длина числа не превышает 10 000 двоичных разрядов). Необходимо определить, делится ли число на 15.

2. а) Как-то раз в аптеку доставили 10 флаконов по 1000 пилюль. Не успел провизор расставить флаконы на полке, как почтальон принес телеграмму, в которой было следующее:
«Воздержаться от продажи лекарства. По ошибке фармацевта в одном из флаконов каждая пилюля содержит на 1 мг лекарства больше допустимой дозы. Просьба незамедлительно вернуть флакон с повышенной дозой лекарства».

Сколько взвешиваний придется сделать провизору для определения флакона с повышенной дозой лекарства, если масса пилюли с допустимой дозой лекарства равна 10 мг?

б) Через какое-то время в аптеку доставили еще 10 флаконов того же лекарства. И на этот раз не успели распаковать коробку с флаконами, как почтальон принес телеграмму с извещением о том, что на этот раз фармацевт допустил более серьезную ошибку. В посылке могли оказаться от 1 до 10 флаконов с пилюлями, каждый из которых на 1 мг тяжелее нормы.

Сколько взвешиваний придется сделать провизору в этот раз для определения флаконов с повышенной дозой лекарства?

3. Предположим, что у нас есть весы с двумя чашами и по одной штуке гирь 1, 3, 9, 27, ..., 3^k, ... Уравновесить груз массой M на весах.

4. Сосчитать количество единиц в двоичной записи числа i, заданного в десятичной системе счисления.

5. Последовательность 0112122012202001 12... строится следующим образом. Сначала пишется 0, затем повторяется следующее действие: уже написанную часть приписывают справа с заменой 0 на 1, 1 на 2, 2 на 0, т. е.

$$0 \to 01 \to 0112 \to 01121220 \to \ldots .$$

Составить алгоритм, который по введенному N ($0 \leqslant N \leqslant 2000000000$) определяет, какое число стоит на N-м месте в последовательности.

6. Даны массивы $X[1..100]$ и $Y[1..100]$. Записать алгоритм, последовательно меняющий местами значения элементов $X[k]$ и $Y[k]$, $k = 1, 2, \ldots, 100$, не используя промежуточных переменных.

7. Точки с целочисленными координатами из 1-го квадранта помечаются числами из множества $\{0, 1, 2, \ldots\}$. Очередная точка помечается в том случае, если все точки ниже и левее ее уже помечены. При этом точке приписывается минимальное число, отсутствующее в вертикали и горизонтали, проходящей через точку. Первой помечается точка (0; 0).

Написать программу, которая:

1) по заданным координатам x и y, $x \geqslant 0$, $y \geqslant 0$, x, y — целые, определяет пометку точки;

2) по заданным координате x и пометке точки k ($x \geqslant 0$, $k \geqslant 0$, x, k — целые) определяет вторую координату точки.

8. Найти длину периода и сам период дроби в P-ичной системе счисления, представляющей рациональное число N/M (для конечных дробей считать, что длина периода равна 1). M, N, P — целые десятичные числа, $0 < N < M$, $P > 1$.

9. Для введенных действительного числа $r > 0$ и натурального числа q_{max} необходимо найти наилучшее приближение r в виде рациональной дроби p/q, где $q \leqslant q_{max}$.

10. Пусть запись числа A в позиционных системах

счисления с основанием p и q имеет вид бесконечной периодической дроби с периодом 2:

$$A=0,\ (ab)_p=0,\ (ba)_q, \qquad (*)$$

где $a \neq b$.

Написать программу, которая для введенных натуральных чисел p и q ($2 \leqslant p, q \leqslant 30$, $p > q$) находит и выводит все возможные пары значений цифр a и b, удовлетворяющих соотношению (*). Если таковых нет, вывести сообщение «Пригодных цифр нет».

Примечание. Значением числа, запись которого в позиционной системе счисления с основанием S есть 0, $cdef$ (где c, d, e, f — цифры), является $\frac{c}{S^1}+\frac{d}{S^2}+\frac{e}{S^3}+\frac{f}{S^4}$.

11. Определим множества K_i рекуррентно. Пусть $K_0=[0, 1]$. Разделим отрезок [0, 1] на три части точками 1/3 и 2/3 и удалим из него интервал (1/3, 2/3). Получим множество K_1, состоящее из двух оставшихся отрезков [0, 1/3] и [2/3, 1]. Каждый из них разделим на три части (точками 1/9 и 2/9 для первого отрезка и точками 7/9 и 8/9 — для второго) и удалим средние интервалы (1/9, 2/9) и (7/9, 8/9). Таким образом, получаем множество K_2 и т. д. Пусть мы построим множество K_i. Поделим каждый оставшийся отрезок из K_i на 3 части и удалим из этих сегментов средние интервалы. Получим, таким образом, из K_i множество K_{i+1}.

Вводятся 3 целых числа n, a, b. Необходимо определить, принадлежит ли точка с координатой a/b множеству K_n.

12. Число называется совершенным, если оно равно сумме всех своих делителей, за исключением его самого. Любое четное совершенное число представимо в виде $2^{p-1}(2^p-1)$, где p — простое число.

Найти двоичное представление для максимального совершенного четного числа меньше введенного N.

13. Заданы натуральные числа e, k, m, t в записи химической реакции $X_eA_k+Y\to Y_mA_t+X$, где A, X, Y — атомы или группы атомов. Написать алгоритм для прохождения таких натуральных коэффициентов, при которых стрелку можно было бы заменить знаком равенства.

14. Вводятся целые числа a и b. Пусть у треугольника ABC координаты вершин $A(0;\ 0)$, $B(a;\ b)$, а обе координаты $C(x;\ y)$ — целые числа, и площадь треугольника ABC не равна нулю. Какую минимальную площадь может иметь треугольник ABC?

15. Имеется N банок с целочисленными объемами V_1, ..., V_n литров, пустой сосуд и кран с водой. Можно ли с помощью этих банок налить в сосуд ровно V литров воды?

16. Вычислить число e (основание натурального логарифма) с точностью n значащих десятичных цифр после запятой. Можно использовать числовой ряд

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\ldots+\frac{1}{A!}+\ldots\ .$$

17. Вывести на экран число 2^n, $n<10\,000$, n — натуральное.

18. Вводится N. Необходимо найти, на сколько нулей оканчивается $N!=1\cdot2\cdot3\cdot\ldots\cdot N$.

19. Вводятся два числа N и P. Найти такое максимальное число M, что $N!$ делится на P^M, но не делится на P^{M+1}.

Примечания. 1. Числа N и P так велики, что нет смысла считать значение $N!$.

2. Числа N и P — натуральные.

20. Натуральное число $N>1$ представить в виде суммы натуральных слагаемых так, чтобы их произведение было максимальным.

21. Задается любое положительное действительное число R. Найти положительные действительные R_1, R_2, ..., R_n, $R_i<4$, $i=1,\ \ldots,\ n$, такие, что $R=R_1\cdot R_2\cdot\ldots\cdot R_n=R_1+R_2+\ldots+R_n$.

22. Вывести в порядке возрастания все обыкновенные несократимые дроби, заключенные между 0 и 1, знаменатели которых не превышают 15. Массив при этом заводить не следует.

23. Дан многогранник, в вершинах которого записаны целые числа. Одним ходом можно выбрать одно ребро, и к числу, записанному в одном из его концов, прибавить 1, а из числа, записанного в другом конце,— вычесть 1.

Какому необходимому и достаточному условию должны удовлетворять записанные числа, чтобы с помощью таких ходов можно было добиться, чтобы во всех вершинах был одновременно записан нуль? Ответ обосновать.

24. В нулевой момент времени мастеру одновременно поступает N работ. Работы пронумерованы от 1 до N. Для каждой работы i заранее известно следующее:

1) время выполнения работы t_i, кратное суткам;
2) штраф c_i за каждые сутки ожидания работы i до момента начала ее выполнения.

Одновременно может выполняться только одна работа, и если мастер приступает к выполнению некоторой работы, то он продолжает выполнять ее, пока не закончит. Суммарный штраф, который надо будет уплатить, выражается следующим образом:

сумма $c_i \cdot$(время начала выполнения работы i) по всем i.

Найти такой порядок выполнения работ, чтобы штраф оказался минимальным.

25. В ряд лежат N арбузов, пронумерованных от 1 до N. Нам известно, что:

1) массы первого и N-го арбузов m_1 и m_N соответственно;

2) масса i-го арбуза m_i есть среднее арифметическое масс двух соседних арбузов, увеличенное на d:

$$m_i = d + (m_{i-1} + m_{i+1})/2;$$

По введенным m_1, m_N, N, d и j найти m_j. Ограничение: $N<200$.

Необходимо в рамках формулировки задачи предусмотреть проверку корректности данных программы.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. Натуральное число, записанное в десятичной системе счисления, называется сверхпростым, если оно остается простым при любой перестановке своих цифр.

Определить, является ли данное число сверхпростым.

2. Последовательность, состоящую из четырех целых чисел a, b, c, d, одним ходом можно преобразовать в последовательность $|a-b|$, $|b-c|$, $|c-d|$, $|d-a|$.

Доказать, что какой бы ни была начальная последовательность, многократным повторением ходов можно получить последовательность, состоящую из четырех нулей.

3. Дано целое неотрицательное число K, не превышающее миллиона. Напечатать фразу «K ворон» русскими словами. (Например, если $K=23$, то должно быть напечатано «двадцать три вороны», если $K=1$, то — «одна ворона».)

4. Написать программу, переводящую заданную сумму в рублях (до 30 000 тысяч включительно) в ее полное название. (Например:

СУММА? 29320,25 — ДВАДЦАТЬ ДЕВЯТЬ ТЫСЯЧ ТРИСТА ДВАДЦАТЬ РУБЛЕЙ ДВАДЦАТЬ ПЯТЬ КОПЕЕК

Сокращать слова нельзя.

5. Написать программу, которая выдаст словами название введенного действительного числа. (Например: 25.23 — двадцать пять целых двадцать три сотых.)

6. Число является палиндромом, если оно читается одинаково слева направо и справа налево. Рассмотрим некоторое натуральное число N. Если это число не палиндром, то изменим порядок его цифр на обратный и сложим исходное число с получившимся. Если сумма

не палиндром, то над ней повторяем то же действие и т. д., пока не получится палиндром. До настоящего времени неизвестно, завершается ли этот процесс для любого натурального N.

Даны натуральные числа K, L, M ($K \leqslant L$). Проверить, верно ли, что для любого натурального числа из диапазона от K до L процесс завершается не позднее, чем после M таких действий. Если нет, то выдать числа, для которых процесс не завершился, и количество таких чисел.

7. Все квадраты натуральных чисел выписаны подряд, начиная с единицы. Определить, какая цифра стоит на N-м месте.

8. Доказать, что в 16-разрядной ЭВМ с объемом ОЗУ 192 Кбайт в любой момент времени t существуют две ячейки, содержащие одинаковые числа.

9. Римские числа.

а) Проверить, правильна ли запись числа римскими цифрами.

б) Записать данное целое число из диапазона от 1 до 1999 римскими цифрами.

в) Перевести число, записанное римскими цифрами, в десятичную систему счисления.

10. Можно ли разбить все натуральные числа 1, 2, 3, 4, 6, ... на две такие возрастающие последовательности $a_1, a_2, a_3, \ldots$ и $b_1, b_2, b_3, \ldots$, что $b_k - a_k = k$ для любого $k = 1, 2, 3, \ldots$? По введенному числу N определить, какому номеру оно соответствует и какое число будет с ним в паре (не выписывая все предыдущие a_k и b_k).

11. Послание от внеземной цивилизации представляет собой набор из N символов, каждый из которых является нулем или единицей. Число N является произведением двух простых чисел и ученые предполагают, что эта строка — закодированная прямоугольная «картинка», размеры которой — множители числа N.

Составить программу, которая производит перекодировку послания и печатает картинки, заменяя каждый нуль пробелом, а единицу — звездочкой.

Пример:

$N=55$

Послание:

10100010111101001000000001011110100010100100100101001000100111

12. Числа 46816_M и 160125_N представлены в разных системах счисления с неизвестными основаниями $M \leqslant \leqslant 10$ и $N \leqslant 10$. Существуют ли такие M и N, что эти числа равны?

13. Рассмотрим числовую последовательность, определенную следующим образом: $a_1=10$, $a_{i+1}=a_i+1/a_i$, если $i \geqslant 1$.

Вычислить все цифры перед запятой a_{1000} члена.

14. Написать алгоритм, который для вещественного числа x находит сумму $x^{k \cdot k}$, где $k=1, 2, \ldots, 128$, используя не более 258 умножений и 256 сложений, учитывая операции организации цикла (возведение в степень не использовать).

15. Составить программу, которая выполняет сложение двух пятиразрядных двоичных чисел, пользуясь операциями «И», «ИЛИ», «НЕ» и «СДВИГ ВЛЕВО» (на один разряд), которые выполняются так:

1) $C=\overline{A}$; переменной C присваивается значение числа A, в котором нули заменяются на 1, а единицы — на 0 (Операция «НЕ»).
2) $C=A \vee B$ — поразрядная операция, результат выполнения которой определяется таблицей 1 (Операция «ИЛИ»).

ТАБЛИЦА 1

$0 \vee 0=0$

$0 \vee 1=1$

$1 \vee 0=1$

$1 \vee 1=1$

3) $C=A \wedge B$ — поразрядная операция, результат выполнения которой определяется таблицей 2 (Операция «И»).

ТАБЛИЦА 2

$0 \wedge 0 = 0$

$0 \wedge 1 = 0$

$1 \wedge 0 = 0$

$1 \wedge 1 = 1$

4) $C := \leftarrow A$ — поразрядный сдвиг влево, т. е. i-й разряд C получает значение $(i+1)$-го разряда числа A, 5-й (младший) разряд C становится равным 0. Каждое из исходных чисел меньше $10\,000_2$.

16. Для заданного K найти такое N, что в десятичном числе 2^N встретится K нулей подряд.

17. По заданному n составить программу вычисления

$$A = \sum_{i=1}^{n} \frac{i!}{(2i)!}.$$

Оценить, для каких достаточно больших n будет работать эта программа.

18. Составить программу, которая в полной записи числа 77! (77 факториал) укажет в порядке убывания номера разрядов, содержащих цифру 7.

Примечание. Разряды числа нумеруются в следующем порядке: разряд единиц имеет номер 1, десятков — 2, сотен — 3 и т. д.

19. Из заданного натурального числа вычеркнуть k цифр так, чтобы полученное число было возможно большим.

20. Найти все десятичные цифры числа $100! - 2^{100}$.

21. На правильно идущих часах с 12-часовым циферблатом имеются часовая, минутная и секундная стрелки, движущиеся плавно (без скачков).

В какое время все стрелки образуют между собой углы 120°? Как долго за сутки углы, образованные между всеми стрелками, одновременно будут в пределах от 119° до 121°?

22. Задана таблица чисел. Алгоритм на каждом шаге выбирает в этой таблице строку (или столбец), сумма

элементов которой меньше нуля, и умножает все эти элементы на (−1).

Доказать, что алгоритм не может работать бесконечно долго.

23. Дано натуральное число n, являющееся квадратом некоторого другого натурального числа.

Составить программу, которая вычисляет $n^{1/2}$, используя только операции сложения и вычитания.

24. Найти значение функции $F(1990)$, вычисляемой по следующему алгоритму:

```
цел алг F(цел k)
нач
цел c, m, a1, a2
| c:=0; a1:=1, a2:=−1; m:=1
| нц пока m<k
| | если a1>0
| | | то
| | |  c:=c+m
| | | иначе
| | |  c:=c−m
| | все
| | a1:=a1+a2; a2:=a1−a2; a1:=a1−a2;
| | m:=m+1
| кц
| знач:=c
кон
```

Придумать алгоритм для вычисления этой функции, в записи которого не используются команды ветвления и цикла.

25. Вычислить значение функции $F(n)=m$, где m — число знаков, содержащихся в десятичной записи числа $n!$

26. Известно, что для любых натуральных чисел a и b существуют такие целые числа u и v, что справедлива формула: $au+bv=$НОД(a, b). По введенным a и b найти какие-нибудь u и v.

27. В круге стоит N человек, пронумерованных по порядку от 1 до N. При расчете на первый-второй, начиная с первого человека, каждый второй выходит из круга, до тех пор, пока не останется один человек.

Доказать, что его номер можно найти следующим образом: представить N в двоичной системе счисления, затем поставить крайний левый единичный бит числа после самого правого разряда числа.

Например: $N=12=1100_2$, номер последнего человека $1001_2=9$.

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Вспомним, что признаком деления на 9 в десятичной системе счисления является делимость на 9 суммы цифр числа. Действительно, пусть есть число

$$S=a_n\cdot 10^n+a_{n-1}10^{n-1}+\ldots+a_1 10+a_0,$$

тогда

$$S \bmod 9=(a_n(10^n-1)+a_n+a_{n-1}(10^{n-1}-1)+a_{n-1}+ \\ +\ldots+a_1(10-1)+a_1+a_0) \bmod 9.$$

А так как 10^k-1 делится на 9 нацело, то и

$$S \bmod 9=(a_n+\ldots+a_1+a_0) \bmod 9.$$

Аналогично получаем, что признаком деления на 15 в системе счисления с основанием 16 будет делимость на 15 суммы всех шестнадцатеричных цифр числа.

Разбиваем двоичное число справа налево на тетрады, которые однозначно можно преобразовать в шестнадцатеричные цифры, находим их сумму и делим ее на 15. Если остаток 0, то введенное число делится на 15, иначе — не делится.

2. а) Для решения задачи не нужно взвешивать пилюли из каждого флакона, т. е. производить 10 взве-

шиваний. Достаточно одного взвешивания для определения флакона с повышенной дозой лекарства.

Для этого необходимо взять 1 пилюлю из первого, 2 пилюли из второго, 3 пилюли из третьего ... 10 пилюль из десятого флакона. Затем следует положить 55 отобранных пилюль на одну чашу весов и взвесить их. Если бы все пилюли были с допустимой дозой лекарства, их масса составила бы 550 мг. Предположим, что пилюли весили 551 мг, или на 1 мг больше, чем следует. Это значит, что имеется ровно одна пилюля с повышенной дозой лекарства, а ровно одна пилюля была извлечена из первого флакона. Если бы масса 55 пилюль оказалась на 3 мг больше нормы, то это означало бы, что среди отобранных пилюль имеются 3 пилюли с повышенной дозой лекарства. Их можно было извлечь только из третьего флакона. Таким образом количество взвешиваний можно понизить до 1.

б) Метод решения, позволивший за одно взвешивание определить флакон, содержащий лекарства с повышенной дозой, в данном случае не применим, однако его можно модифицировать.

Для того чтобы решить задачу, необходимо воспользоваться последовательностью, которая бы сопоставляла каждому флакону отличный от других номер и обладала бы еще одним дополнительным свойством: сумма членов любой ее подпоследовательности должна быть отличной от суммы членов любой другой ее подпоследовательности. Примером такой последовательности может служить следующая: 1, 2, 4, 8, ..., все члены ее — степени числа 2. Эта последовательность лежит в основе двоичной системы счисления.

Решение задачи состоит в том, чтобы взять 1 пилюлю из первого закона, 2 пилюли из второго, 4 пилюли из третьего и т. д., затем собрать все отобранные пилюли и взвесить. Предположим, что пилюли оказались на 27 мг тяжелее, чем нужно. Так как каждая пилюля с повышенной дозой лекарства тяжелее нормальной на

1 мг, то, разделив 27 на 1, мы получим 27 — число более тяжелых пилюль.

Запишем число 27 в двоичной системе: 11011. Двоичные разряды, в которых стоят единицы, говорят нам, какие степени числа 2 в сумме дают двоичное число 11011 (или десятичное число 27): 1, 2, 8, 16. Единицы стоят в первом, втором, четвертом и пятом двоичных разрядах. Следовательно, непригодные пилюли с повышенным содержанием лекарства находятся в первом, втором, четвертом и пятом флаконах.

3. При представлении числа в троичной системе счисления обычно используются цифры 0, 1 и 2. Однако можно использовать цифры 1, 0 и -1. Для получения такой записи поступим следующим образом: будем переводить M в троичную систему счисления, и каждый раз, когда при делении на 3 будет получаться 2, будем увеличивать частное на 1, а в остатке писать (-1). Например:

$$24_{10} = 1000_3 - 10_3 = 1 \cdot 3^3 + 0 \cdot 3^2 - 1 \cdot 3^1 + 0 \cdot 3^0.$$

Груз V положим на первую чашу весов. Грузы с коэффициентом в разложении -1 поставим на эту же чашу, а грузы с коэффициентом 1 — на другую.

Объясните, почему данный алгоритм работает.

Опишите процесс уравновешивания груза неизвестной массы на чашечных весах, используя ту же систему гирь.

4. Алгоритм:

```
cnt:=0;                 {cnt — счетчик единиц в i.}
while (i<>0) do         {цикл повторяется число раз,}
begin                   {равное числу единиц в i.}
  i:=(i-1) and i;       {«Убираем» крайнюю справа}
  cnt:=cnt+1;           {единицу в двоичной записи}
end;                    {числа.}
```

Пример. $110 = i$

$101 = i - 1$

$100 = i \text{ and } (i-1)$

5. Пусть a_k — k-й член последовательности. Рассмотрим последовательность, формируемую по следующему правилу: $a_0=0$; фрагмент $a_0 \ldots a_{2^{(k-1)}}$ получаем приписыванием к фрагменту $a_0 \ldots a_{2^{(k-1)}-1}$ справа аналогичного фрагмента, в котором каждый член увеличен на единицу. Получаем $0 \to 01 \to 0112 \to 01121223 \to \ldots$.

Докажем, что a_k есть сумма единиц в двоичном представлении числа k. Доказательство проведем по индукции. Для $a_0=0$ это справедливо. Пусть предположение справедливо для всех a_i, $0 \leqslant i \leqslant 2^{k-1}-1$ (т. е. для всех чисел i, состоящих из не более чем $(k-1)$ двоичных разрядов). Тогда в двоичном разложении числа l, $2^{k-1} \leqslant l < 2^k$, в k-м разряде появляется добавочная единица, и поэтому $a_l = 1 + a_l - 2k - 1$.

Возьмем $a_i \bmod 3$ и получим число, стоящее на i-м месте в последовательности, описанной в условии задачи.

Для того чтобы найти a_i, необходимо по доказанному сосчитать количество единиц в двоичной записи числа i (см. задачу 4).

6. Будем менять местами содержимое переменных A и B. Существует несколько способов сделать это.

1)

Оператор	Значение в A	Значение в B
$A=A+B;$	$A+B$	B
$B=A-B;$	$A+B$	A
$A=A-B;$	B	A

2) Можно использовать логическую операцию XOR (исключающее ИЛИ). Таблица истинности для XOR следующая:

1 XOR 1=0 1 XOR 0=1
0 XOR 0=0 0 XOR 1=1

Операция XOR над двумя переменными в машине реализуется как побитовая операция над двоичным представлением чисел. Поэтому, в частности, A XOR $A=0$, A XOR $B=B$ XOR A, A XOR $0=A$.

Оператор	Значение в A	Значение в B
$A=A\ \mathrm{XOR}\ B$	$A\ \mathrm{XOR}\ B$	B
$B=A\ \mathrm{XOR}\ B$	$A\ \mathrm{XOR}\ B$	$(A\ \mathrm{XOR}\ B)\ \mathrm{XOR}\ B=$ $=A\ \mathrm{XOR}\ (B\ \mathrm{XOR}\ B)=$ $=A\ \mathrm{XOR}\ 0=A$
$A=A\ \mathrm{XOR}\ B$	$(A\ \mathrm{XOR}\ B)\ \mathrm{XOR}\ A=$ $=B\ \mathrm{XOR}\ (A\ \mathrm{XOR}\ A)=B$	A

7. Если рассмотреть битовые представления числа $A[i, j]$, помечающего точку (i, j), и чисел i и j, то обнаруживается, что $A[i, j]=i\ \mathrm{XOR}\ j$, откуда получается, что $A[i, j]\ \mathrm{XOR}\ i=j$, $A[i, j]\ \mathrm{XOR}\ j=i$.

Покажем, что $A[i, j]=i\ \mathrm{XOR}\ j$.

1) Число $A[i, j]=i\ \mathrm{XOR}\ j$ не встречалось еще ни в строке i, ни в столбце j. От противного: существует такое j', что

$$i\ \mathrm{XOR}\ j=i\ \mathrm{XOR}\ j'\Rightarrow i\ \mathrm{XOR}\ j\ \mathrm{XOR}\ i=$$
$$=i\ \mathrm{XOR}\ j'\ \mathrm{XOR}\ i\Rightarrow j'=j;$$

2) Пусть существует такое $k<i\ \mathrm{XOR}\ j$, что $k=$ $=i\ \mathrm{XOR}\ L=j\ \mathrm{XOR}\ M$, и k еще не встречалось в строке i и столбце j (напомним, что по предположению все остальные уже заполненные элементы равны $i\ \mathrm{XOR}\ j$, поэтому $L>j$ и $M>i$).

Тогда, так как $M>i$, то существует бит с номером t такой, что для любого $R>t$ биты M_r и i_r равны, t-й бит $M_t=1$, $i_t=0$. Но так как

$$j\ \mathrm{XOR}\ M<j\ \mathrm{XOR}\ i,\ \text{то}\ j_t=1.$$

Так как $L>j$ и $L\ \mathrm{XOR}\ i=j\ \mathrm{XOR}\ M$, то $L=$ $=j\ \mathrm{XOR}\ M\ \mathrm{XOR}\ i$. Рассмотрим $i\ \mathrm{XOR}\ M$. В силу вышесказанного для любого бита с номером R, $R>t$,

$$(i\ \mathrm{XOR}\ M)_r=0,\ \text{а}\ (i\ \mathrm{XOR}\ M)_t=1.$$

При этом $j_t=1$, следовательно

$$(i\ \mathrm{XOR}\ j\ \mathrm{XOR}\ M)_r=j_r\ \text{для}\ r>t$$

и

$$(i\ \mathrm{XOR}\ j\ \mathrm{XOR}\ M)_t=0\ \text{для}\ r=t,$$

т. е. $L<j$?! Получили противоречие.

8. Введем переменную $N_1 = N$. Пусть N_1 и M заданы в десятичной системе счисления. Переведем дробь M_1/M в систему счисления с основанием p.

Пусть в системе с основанием p искомая дробь $0, a_1 a_2 \ldots$. Получаем:

$$a_1 \cdot p^{-1} + a_2 \cdot p^{-2} + \ldots = N_1/M.$$

Умножим правую и левую части равенства на p: $a_1 + a_2 \cdot p^{-1} + \ldots = N_1 \cdot p/M$. Выделяя целую часть выражений слева и справа от знака равенства, получаем: a_1 есть целая часть от $(N_1 \cdot p/M)$. Обозначим $N_2 = N_1 \cdot p \bmod M$; очевидным образом получаем $a_2 \cdot p^{-1} + \ldots = N_2/M$. Домножая на p и находя целую часть, опять же имеем: a_2 есть целая часть от $(N_2 \cdot p/M)$; продолжая аналогично, определяем коэффициенты a_3, a_4 и т. д.

В ходе выделения цифр a_i мы можем получить различных значений N_i не более чем M (по приведенному алгоритму выше у нас всегда $N_i < M$).

Если вдруг какие-то два остатка совпадают: $N_i = N_j$, $i \neq j$, то совпадают и цифры разложения: $a_{i+1} = a_{j+1}$, $a_{i+2} = a_{j+2}$, ..., т. е. цифры $(a_{i+1}, \ldots, a_j)$ образуют один из кратных периодов. Нам надо найти минимальную длину такой периодически повторяющейся последовательности, которая равна количеству цифр между двумя ближайшими повторяющимися остатками, и сами цифры.

Поступаем следующим образом. Выделяем M цифр p-ичной дроби (исходя из вышесказанного, к этому моменту период уже обязан начаться). Запоминаем N_m и ищем первый такой остаток N_k, $k > m$, что $N_m = N_k$. Величина $k - m$ как раз и есть искомая длина периода.

9. Обозначим: p — числитель, а q — знаменатель искомой дроби. Вначале $p = 0$; $q = 1$. Если p/q меньше r, то увеличим числитель, если p/q меньше искомой дроби, то увеличим знаменатель. Если $p/q = r$, то решение найдено. Если $q > q_{max}$, то из всех полученных дробей выбираем ту, которая ближе всех к r.

Фрагмент программы:

```
write ('r,qmax='); readln(r,qmax);
p:=0; q:=1; min:=r;
REPEAT
   IF p/q<r THEN p:=p+1 ELSE q:=q+1;
   d:=abs(r-p/q);
   IF d<min THEN
         BEGIN
            min:=d; writeln(p:7,'/',q)
         END
UNTIL (q>=qmax)OR(d=0);
```

10. Так как $q<p$, то цифры a и b должны лежать в пределах от 0 до $q-1$. Распишем A в системе с основанием p:

$$A=\frac{a}{p}+\frac{b}{p^2}+\frac{a}{p^3}+\frac{b}{p^4}+\ldots=\frac{1}{p^2}(ap+b)+\frac{1}{p^4}(ap+b)+\ldots=$$
$$=(ap+b)(p^{-2}+p^{-4}+\ldots)=$$

{находим сумму бесконечно убывающей геометрической прогрессии со знаменателем p^{-2}}

$$=(ap+b)\frac{p^{-2}}{1-p^{-2}}=\frac{ap+b}{p^2-1}.$$

Аналогично для A в системе с основанием q выполняется равенство

$$A=\frac{bq+a}{q^2-1}.$$

Получаем

$$(bq+a)(p^2-1)=(ap+b)(q^2-1),$$
$$a[p(q^2-1)-(p^2-1)]=b[q(p^2-1)-(q^2-1)].$$

Вычисляем выражения в квадратных скобках, обозначив их соответственно u и v: $au=bv$.

Находим НОД$(u, v)=s$; обозначим $r=u/s$, $f=v/s$.

Получаем $ar=bf$, r и f взаимно простые числа, следовательно, решениями этого равенства будут числа

$$a=f_k,\ b=r_k,\ k=1,\ 2,\ \ldots,$$

при этом мы берем только те a и b, для которых одновременно выполняется $a\neq b$, $a<q$, $b<q$. Для нахождения НОД используется алгоритм (3.8).

11. Непосредственное вычисление координат концов отрезков, принадлежащих множеству K_n, и определение, принадлежит ли a/b одному из этих отрезков, дает неверный ответ для большинства входных данных. Это связано с тем, что даже для не слишком больших из-за ограниченного числа знаков в машинном представлении чисел с плавающей точкой происходит потеря точности при вычислении. При больших n вообще будет наблюдаться потеря значимости: число при делении на 3 станет таким малым, что в машине оно будет представляться нулем.

Рассмотрим другой метод решения этой задачи. Так как мы постоянно должны делить отрезки на три части, то это наталкивает на мысль использовать троичную систему счисления и троичные дроби.

В первый из удаленных интервалов (1/3, 2/3) попадают только те точки $x=0, a_1a_2a_3\ldots$, в троичном разложении которых $a_1=1$, кроме точки $1/3=0, 1000\ldots$— правого конца отрезка [0, 1/3]. Таким образом, в K_1 остаются все те точки, у которых $a_1\neq 1$, либо $a_1=1$, $a_2=a_3=\ldots=0$. Аналогично, в множестве K_i, $i\geqslant 0$, содержатся точки, у которых ни одно из чисел a_j, $1\leqslant j\leqslant i$, не равно 1, а также точки, удовлетворяющие условию: $a_j=1$, j — фиксировано, $1\leqslant j\leqslant i$, $a_l\neq 1$, $l<j$, и $a_l=0$ для любого $l>j$ (т. е. в записи троичной дроби только одна позиция равна 1, после нее все остальные позиции нулевые. Эти дроби соответствуют правым концам отрезков из множества K_i).

Запись алгоритма на языке Паскаля имеет вид:

```
x:=a;i:=1;
while (i<=n) and (x<>1) and (a<>0) do
begin
  a:=a*3 mod b;
  x:=a*3 div b;
  i:=i+1;
end;
if(x=1) and (a<>0) and (n<>0)
then writeln('Не принадлежит')
else writeln('Принадлежит');
```

12. Число $K=2^{p-1}(2^p-1)$ в двоичном представлении имеет p единиц и $p-1$ нулей. Максимальное значение p определяется как $[\log_2(N)/2]+1$. Затем проверяется, является ли число совершенным для полученного значения p. Оно является совершенным, если для простого значения p число 2^p-1 является простым. Докажем этот факт.

Пусть $2^p-1=q$. Делителями числа K, включая само число K, являются следующие числа:

$$1,2,2^2,\ldots,2^{p-1},$$
$$q,2q,2^2q,\ldots,2^{p-1}q.$$

Запишем сумму этих делителей

$$1+2+\ldots+2^{p-1}+q(1+2+\ldots+2^{p-1}),$$

которая равна

$$(1+2+\ldots+2^{p-1})(q+1)=(1+2+\ldots+2^{p-1})\cdot 2^p.$$

Сумма в скобках есть сумма первых p членов геометрической прогрессии с первым членом 1 и знаменателем 2, она равна $2^p-1=q$.

Таким образом, сумма всех делителей числа K есть $2^p\cdot q=2\cdot 2^{p-1}q$, а сумма всех делителей, кроме самого числа $K=2^{p-1}q$, равна

$$2\cdot 2^{p-1}q-2^{p-1}q=2^{p-1}q=K.$$

Число P равно сумме всех своих делителей, за исключением его самого, следовательно, оно является совершенным.

Если получили несовершенное число, уменьшаем p на 1 и снова проверяем, является ли число совершенным.

Совершенные числа получаются для значений p, равных, например, 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127.

13. Запишем уравнение в виде

$$sX_eA_k+pY=nY_mA_t+rX.$$

Приравнивая коэффициенты при X, A и Y, получаем:

$$\begin{aligned} X|se&=k\\ A|sk&=nt\\ Y|p&=nm \end{aligned} \qquad (*)$$

Так как коэффициенты в формуле должны быть взаимно простыми, то следует найти НОД чисел k и t (алгоритм 3.8). Пусть $(k, t)=d$. Тогда $s(k/d)=n(t/d)$, и числа k/d и t/d являются взаимно простыми, следовательно, $n=k/d$, а $s=t/d$.

Используя формулы (*), находим остальные коэффициенты.

14. По заданным координатам трех вершин мы можем найти площадь треугольника ABC:

$$S_{abc}=(bx-ay)/2.$$

Если $a=0$, то минимальная площадь $S_{min}=b/2$, если $b=0$, то $S_{min}=a/2$. Если же обе координаты отличны от нуля, то из алгоритма Евклида для нахождения НОД (a, b) следует существование таких целых x и y, что $|bx-ay|=$НОД(a, b), и именно эти x и y минимизируют площадь треугольника ABC.

Нахождение НОД — алгоритм 3.8.

15. Обозначим $S=$НОД$(V_1, \ldots, V_n)$. Докажите, что если V делится нацело на S, то в сосуд с помощью банок можно налить V литров воды, иначе — нет.

16. Заведем два массива: $E[0..N+1]$ и $S[0..N+1]$. Элементы массивов — байты. В E будем хранить текущую частичную сумму

$$e_n=1+1/1!+1/2!+\ldots+1/(k-1)!, \qquad (*)$$

вычисленную с точностью $N+1$ знаков после запятой (в ячейке $E[i]$ хранится значение i-го разряда после запятой, т. е. коэффициент при 10^{-i}). В массиве S будет храниться последнее слагаемое частичной суммы, т. е. $1/(k-1)!$ (опять же в $S[i]$ находится i-я цифра после запятой, все незначащие цифры, разумеется, нули).

Оценим погрешность формулы (*) с учетом равенства

$$e=1+1/1!+1/2!+\ldots+1/(k-1)!+\ldots .$$

Погрешность представления числа e будет

$$\begin{aligned} F[n]=e-e_n&=1/(n+1)!+1/(n+2)!+\ldots=\\ &=1/(n+1)!\,(1+1/(n+2)+1/(n+2)(n+3)+\ldots)=\\ &=1/(1+n)!\,(1+1/(n+2)+1/(n+2)^2+\ldots)=\\ &=\frac{1}{(n+1)!}\cdot\frac{1}{1-(n+2)^{-1}}=\frac{1}{(n+1)!}\cdot\frac{n+2}{n+1}. \end{aligned}$$

Так как $F_{450}<10^{-1002}$, то для вычисления, например, 1000 знаков после запятой достаточно $C=450$ итераций.

Будем делить число, представленное массивом S, на очередное число k. Результат $1/k!$ прибавим к массиву E. Сложение реализуется так:

```
perenos:=0;          {перенос из предыдущего разряда}
for i:=N+1 downto 0 do
begin
  E[i]:=E[i]+S[i]+perenos;
  perenos:=E[i] div 10;  {формируем новый перенос}
  E[i]:=E[i] mod 10;     {корректируем E[i]}
end;
```

Вычисление новых слагаемых и суммирование проводится C раз.

Рассмотрим процедуру деления S на k. Делить будем так, как это обычно делается вручную — «столбиком»:

```
m:=0;              {m — текущее делимое, m — Longint}
for i:=0 to N+1 do
begin
```

```
    m:=m*10+a[i];   {дописываем к делимому сзади}
                    {одну цифру}
    a[i]:=m div k   {находим очередную цифру частного}
    m:=m mod k;     {и очередное делимое}
end;
```

Объясните, почему $a[i]$ лежит в пределах от 0 до 9. Попытаемся вычислить e_n, используя схему Горнера:

$$e_n=\left(\ldots\left(\left(\frac{1}{n}+1\right)\cdot\frac{1}{n-1}+1\right)\cdot\frac{1}{n-2}+\ldots\right)\cdot\frac{1}{1}+1.$$

Сначала полагаем $E[0]=1$; затем делим «столбиком» число, представляемое E, на 450 и запоминаем цифры частного снова в E. Затем выполняем $E[0]:=E[0]+1$, делим E на 449 и т. д.

Обратите внимание на то, что во втором способе при использовании схемы Горнера не надо производить суммирование массивов E и S, поэтому скорость работы увеличивается в 2 раза.

17. В переменной стандартного типа такое большое число не поместится. Будем моделировать возведение 2 в степень n, вычисляя последовательно 2^1, 2^2, ..., 2^n. Цифры полученных на каждом шаге степеней двойки будем хранить в массиве. В каждой ячейке массива будем хранить по (например) 4 десятичных цифры числа (т. е. в элементе $A[1]$ — 4 последних цифры числа (разряды 0—3), в $A[2]$ — 4 предпоследних (разряды 4—7) и т. д.).

Оценим количество десятичных цифр в числе 2^n, $n\leqslant 1000$. Это $10\,000\cdot\log_2 10+1<15\,000$ цифр. Количество элементов массива возьмем равным $15\,000/4=3750$. Введем переменную *Nach*, в которой будем хранить индекс элемента массива A, в котором находятся старшие значащие разряды вычисляемого сейчас числа. При умножении на 2 текущего числа будем умножать на 2 каждый элемент массива, начиная с первого и заканчивая элементом с номером *Nach*. Если какой-то из элементов массива больше 9999, то производим перенос единицы

в следующий разряд. Если был перенос единицы в разряд $Nach+1$, то значение переменной $Nach$ увеличим на 1.

18. Мы можем представить $N!$ в виде произведения простых сомножителей:

$$N!=2^{A_2}\cdot 3^{A_3}\cdot 5^{A_5}\cdot 7^{A_7}\cdot\ldots,$$

где A_p — показатель степени, с которой простое число p входит в разложение. Видно, что нулей в конце числа столько же, сколько нулей в конце произведения $2^{A_2}\cdot 5^{A_5}$, но так как $A_2>A_5$, то количество нулей равно A_5.

Для того чтобы найти A_5, необходимо вычислить сумму

$$A_5=\left[\frac{N}{5^1}\right]+\left[\frac{N}{5^2}\right]+\left[\frac{N}{5^3}\right]+\ldots, \qquad (*)$$

где [ ] — целая часть числа.

Каждое пятое число в произведении $N!$ делится на 5, каждое двадцать пятое число еще раз делится на 5, каждое 5^3 число еще раз делится на 5 и т. д. Таким образом в (*) мы находим, сколько чисел в произведении $N!$ делится на 5.

Фрагмент программы выглядит следующим образом:

```
k:=5;
s:=0;
repeat
  s:=s+N div k;
  k:=k*5;
until (k>N);
```

После работы цикла в переменной S будет находиться A_5.

19. Найдем простые множители числа P. Пусть это будут $p_1, \ldots, p_k$. Для каждого множителя p_i найдем число s_i — степень, с которой p_i входит в разложение P на простые сомножители. Как и в задаче 18, найдем макси-

мальные числа $m_1, \ldots, m_k$, такие, что $N!$ делится на p_i в степени m_i, но не делится на p_i в степени m_{i+1}.

Получаем для нахождения m следующее уравнение:

$$\frac{N!}{P^M} = \frac{p_1^{m_1} \cdot p_2^{m_2} \cdot \ldots \cdot p_k^{m_k} \cdot R}{(p_1^{s_1} \cdot p_2^{s_2} \cdot \ldots \cdot p_k^{s_k})^M},$$

где $R = N!/[p_1^{m_1} \cdot p_2^{m_2} \cdot \ldots \cdot p_k^{m_k}]$.

Минимальное из чисел $m_i \operatorname{div} s_i$ и даст искомую степень M.

Рассмотрим пример: $N=15$, $P=135$. $P=3\cdot3\cdot3\cdot5=$ $=3^3\cdot5$, $p_1=3$, $s_1=3$, $m_1=15 \operatorname{div} 3+15 \operatorname{div}(3\cdot3)=6$; $p_2=5$, $s_2=1$, $m_2=15 \operatorname{div} 5=3$.

Получаем, что $M=\min\{6 \operatorname{div} 3,\ 3 \operatorname{div} 1\}=2$.

Объясните, почему мы не можем применить формулу

$$M = N \operatorname{div} P + N \operatorname{div} P_2 + N \operatorname{div} P_3 + \ldots .$$

20. Воспользуемся тем, что для $N \leqslant 4$ выполняется неравенство $N \leqslant (N-2)\cdot 2$, т. е. разбивать число на слагаемые, большие 3, не имеет смысла. Выделяем из числа N слагаемые-двойки, пока не получим остаток меньший либо равный 3 (остаток может быть либо 3, либо 2). Так как $2\cdot2\cdot2<3\cdot3$, то заменим каждые три двойки на две тройки. Полученное разложение и является искомым.

Разберите самостоятельно случаи:

1) когда необходимо максимизировать произведение и слагаемые в разложении числа N должны принадлежать промежутку $[A, B]$, A и B вводятся пользователем.

2) когда необходимо минимизировать произведение и слагаемые в разложении числа N должны принадлежать промежутку $[A, B]$, A и B вводятся пользователем.

21. Если $S \geqslant 4$, то существуют единственные R_1 и R_2 такие, что $R = R_1 \cdot R_2 = R_1 + R_2$. Более того, наименьшее из R_1 и R_2 больше 1 и меньше или равно 2:

$$R_1 = (R - \sqrt{R^2 - 4R})/2,\ R_2 = (R + \sqrt{R^2 - 4R})/2,$$
$$(R-4)^2 = R_2 - 8R + 16 \leqslant R^2 - 4R < (R-2)^2,$$
$$1 < R_1 = (R - \sqrt{R^2 - 4R})/2 \leqslant 2.$$

Итак, если $r<4$, то разложение на множители закончено, если иначе, то проводим разложение r на два множителя, один из них меньше либо равен 2 (и тем более меньше 4), если другой меньше 4, то процесс не закончен, если иначе, то повторяем факторизацию R до тех пор, пока не получим искомое разложение.

22. Пусть m/n — текущая несократимая дробь. Покажем, как найти следующую по значению дробь. Понятно, что она будет среди несократимых дробей вида k/p, где p может принимать значения от 2 до 15. Учитывая условие $k/p>m/n$ можно для каждого p прямо вычислять минимальное значение k следующим образом: $k=m\cdot p/n+1$. При этом каждая дробь k/p, полученная описанным выше образом, несократима.

```
m:=0; n:=1
repeat
i:=1; j:=1;
for p:=2 to 15 do
begin
  k:=m*p div n+1;
  if k*j<p*i then
    begin
      i:=k;
      j:=p;
    end;
  end;
  m:=i; n:=j;
until i>=j
```

23. Это условие — равенство нулю суммы всех чисел. Мы всегда можем «перетащить» с помощью последовательности ходов все ненулевые числа, помечающие вершины, в одну какую-либо вершину. Если сумма всех чисел равна 0, то после этих ходов окажется, что во всех вершинах записан 0.

24. Пусть имеется оптимальное расписание, в котором номера выполняемых работ совпадают с порядковы-

ми номерами работ. Тогда этому расписанию будет соответствовать минимальное значение штрафа. Всякая перестановка в порядке выполнения двух или более работ уже не может привести к уменьшению штрафа, поэтому, переставив местами работы с номерами k и $k+1$, мы получим штраф не меньше.

Обозначим через q_i время начала выполнения работы i. С учетом представления штрафа это можно записать так:

$$\sum_{i=1}^{k-1} c_i \cdot q_i + c_k \cdot q_k + c_{k+1} \cdot q_{k+1} + \sum_{i=k+2}^{n} c_i \cdot q_i \leqslant$$
$$\leqslant \sum_{i=1}^{k-1} c_i \cdot q_i + c_k \cdot q'_k + c_{k+1} \cdot q'_{k+1} + \sum_{i=k+2}^{n} c_i \cdot q_i.$$

Здесь q'_k и q'_{k+1} — время начала выполнения соответственно $(k+1)$-й и k-й работы после перестановки. Заметим, что k-я работа в обоих расписаниях выполнится после того, как будут выполнены предшествующие k-1 работы, поэтому

$$q_k = q'_{k+1},\ q_{k+1} = q_k + t_k,\ q'_k = q'_{k+1} + t_{k+1} = q_k + t_{k+1}.$$

В результате получаем:

$$c_k q_k + c_{k+1}(q_k + t_k) \leqslant c_{k+1} q_k + c_k (q_k + t_{k+1}),$$
$$c_{k+1} t_k \leqslant c_k t_{k+1},\ t_k c_k \leqslant t_{k+1}/c_{k+1}.$$

Алгоритм:

1) для всех работ вычислить отношение t_i/c_i;

2) упорядочить работы по возрастанию этого отношения.

25. В рамках формулировки задачи под корректностью данных программы понимается то, что масса каждого арбуза m_i должна быть положительным числом. Мы знаем m_1 и m_n. Пусть действительные массы арбузов в ряде m_i. Будем обозначать текущие вычисляемые массы m'_i. Зададим произвольное m'_2 — масса второго арбуза.

Пусть $m_2'-m_2=s$, $m_1'=m_1$ и $m_i'=d+(m_{i-1}'+m_{i+1}')/2$, тогда, так как

$$m_i=d+(m_{i-1}+m_{i+1})/2, \text{ то}$$

$m_{i+1}=-2\cdot d+2\cdot m_i-m_{i-1}$, $m_{i+1}'=-2\cdot d+2\cdot m_i'-m_{i-1}'$, и

$m_3'-m_3=(-2\cdot d+2\cdot m_2'-m_1')-(-2\cdot d+2\cdot m_2-m_1]=2s$,

$m_4'-m_4=(-2\cdot d+2\cdot m_3'-m_2')-(-2\cdot d+2\cdot m_3-m_2]=3s$,

...

$$m_n'-m_n=(n-1)\,s.$$

Находим разность $m_n'-m_n=(n-1)\,s$. Вычисляем s.

Глава 4. РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ И ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

§ 1. ПОНЯТИЕ ЗАДАЧИ И ПОДЗАДАЧИ

При формулировке любой задачи необходимо определить исходные данные, которые мы будем называть параметрами задачи.

Например, если мы решаем задачу нахождения корней квадратного уравнения $ax^2+bx+c=0$, то эта задача определяется тремя параметрами — коэффициентами a, b и c.

Если же мы хотим решить задачу нахождения среднего арифметического некоторого набора чисел, то параметрами задачи будут количество чисел и их значения.

При этом нас пока не интересует конкретный алгоритм решения задачи. Мы хотим научиться решать задачу, сводя ее к решению подзадач. Здесь удобно думать об алгоритме как о некотором устройстве или некоторой функции, которые преобразуют входные параметры в некоторые выходные данные, являющиеся решением задачи.

Поэтому при описанном выше подходе любая задача может быть формализована в виде некоторой функции, аргументами которой могут являться такие величины, как:

количество параметров;

значения параметров.

Здесь и далее в качестве параметров будут рассматриваться целые неотрицательные числа.

Как правило, одним из аргументов задачи является количество параметров задачи. В том случае, когда по

значению этого параметра можно определить конкретные значения других параметров, мы эти параметры будем опускать. Это обычно делается в случае, когда параметры заданы таблицей. Например, если нам необходимо найти сумму первых K элементов таблицы, то для решения задачи достаточно знать один параметр K, а все остальные параметры можно выбрать из таблицы.

После того как задача формализована (представлена) в виде функции с некоторыми аргументами, определим понятие ***подзадачи***. Здесь и далее в этой главе под подзадачей будем понимать ту же задачу, но с меньшим числом параметров или задачу с тем же числом параметров, но при этом хотя бы один из параметров имеет меньшее значение.

Пример. Найти самую тяжелую из 10 монет.

Для формализации задачи определим функцию «Самая тяжелая монета», аргументами которой являются количество монет (10) и масса каждой из монет. Пока нас не интересует конкретный вид этой функции, для нас важнейшим фактором является то, что она дает правильное решение.

Для данной задачи можно рассмотреть 9 подзадач, которые имеют меньшее число аргументов:

«самая тяжелая монета» из 1 монеты,
«самая тяжелая монета» из 2 первых монет,
«самая тяжелая монета» из 3 первых монет,

...

«самая тяжелая монета» из 9 первых монет.

Таким образом, у нашей функции «Самая тяжелая монета» аргументом является количество имеющихся монет, по которому можно определить массу каждой монеты. Следовательно, рассмотренные подзадачи имеют меньшее количество аргументов, чем исходная задача.

Надо отметить, что под подзадачей не следует пони-

мать некоторые этапы решения задачи, такие, как организация ввода и вывода данных, их упорядочение или решение некоторой части поставленной задачи.

Вопросы для повторения

1. Определить параметры следующих задач:
   а) решить линейное уравнение $ax+b=0$;
   б) найти наименьшее из N чисел $a_1, a_2, \ldots, a_N$.
2. Что такое подзадача?
3. Что может являться аргументами функции, формализующей задачу?

§ 2. СВЕДЕНИЕ ЗАДАЧИ К ПОДЗАДАЧАМ

Одним из основных способов решения задач является их сведение к решению такого набора подзадач, чтобы, исходя из решений подзадач, было возможно получить решение исходной задачи.

При этом для решения исходной задачи может потребоваться решение одной или нескольких подзадач.

Пример. Задачу, сформулированную в примере в предыдущем параграфе, можно свести к различным наборам подзадач, например:

найти самую тяжелую из 9 монет, а затем найти самую тяжелую из 2 монет (найденной из 9 и оставшейся) или

найти самую тяжелую из 5 монет, затем самую тяжелую из других 5 монет, а затем самую тяжелую из 2 монет, найденных на предыдущих шагах.

Возможны и другие наборы, но нетрудно заметить, что все они основываются на одной подзадаче: найти самую тяжелую из 2 монет.

В приведенном примере исходная задача сводится к подзадачам с меньшим числом параметров, в данном случае — с меньшим количеством монет.

Используя этот же принцип, можно решить задачу нахождения НОД двух чисел, которая рассматривалась в п. 3 гл. 3.

Пример. Найти НОД двух натуральных чисел N и M.

Если числа равны, то их НОД равен одному из чисел, т. е. НОД$(N, M)=N$.

Рассмотрим случай, когда числа не равны. Известно, что

НОД(N, M)=НОД$(N, M+N)$=НОД$(N+M, M)$.

Кроме того, при $N>M$ НОД(N, M)=НОД$(N-M, M)$, а при $M>N$ НОД(N, M)=НОД$(N, M-N)$.

Последние соотношения и обеспечивают основной принцип сведения решения задачи к подзадачам: значение одного из параметров стало меньше, хотя их количество и осталось прежним.

Таким образом, решение задачи нахождения НОД(N, M) при различных значениях N и M сводится к двум подзадачам:

НОД$(N-M, M)$, если $N>M$;

НОД$(N, M-N)$, если $M>N$.

Вопросы для повторения

Сформулируйте основной принцип сведения задачи к подзадачам.

§ 3. ПОНЯТИЕ РЕКУРРЕНТНОГО СООТНОШЕНИЯ

Найденный способ сведения решения исходной задачи к решению некоторых подзадач может быть записан в виде соотношений, в которых значение функции, соответствующей исходной задаче, выражается через значения функций, соответствующих подзадачам. При этом важнейшим условием сведения является тот факт, что значения аргументов у любой из функций в правой части соотношения меньше значения аргументов функции в левой части соотношения. Если аргументов несколько, то достаточно уменьшения одного из них.

Следует обратить внимание на то, что соотношения

должны быть определены для всех допустимых значений аргументов.

Пример. Найти сумму N элементов таблицы A.

Пусть функция $S(N)$ соответствует решению исходной задачи. Эта функция имеет один аргумент N — количество суммируемых элементов таблицы A. Понятно, что для поиска суммы N элементов достаточно знать сумму первых $N-1$ элементов и значение N-го элемента. Поэтому решение исходной задачи можно записать в виде соотношения

$$S(N)=S(N-1)+a_N.$$

Следует отметить, что это соотношение справедливо для любого количества элементов $N>1$. Его можно переписать в виде

$$S(i)=S(i-1)+a_i \text{ при } i>1.$$

Однако пока это соотношение не определено при $N=1$. К приведенному выше соотношению необходимо добавить соотношение $S(1)=a_1$.

Заметим, что на практике применяются имеющие тот же смысл соотношения

$$S(i)=S(i-1)+a_i \text{ при } i\geqslant 1,\ S(0)=0.$$

Последовательное применение первого соотношения при $i=1, 2, \ldots, N$ и используется при вычислении суммы N элементов.

```
S[0]:=0
нц для i от 1 до N                    (4.1)
| S[i]:=S[i-1]+a[i]
кц
```

В $S[i]$ хранится значение функции $S(i)$.

Здесь и далее в круглых скобках будут записываться аргументы функции, а в квадратных — индексы элементов массива. При этом имя функции и имя массива,

в котором хранится значение этой функции, могут совпадать.

Примечание. Индекс у S может быть опущен, но смысл соотношения при этом остается прежним. Это связано с тем, что для вычисления следующего элемента таблицы S необходимо знать только предыдущий.

Пример. Вычислить сумму $S=1+1/x+1/x^2+\ldots+1/x^N$ при x, не равном 0.

Как и в предыдущем примере, можно записать следующее соотношение:

$$S(i)=S(i-1)+a(i),\ i\geqslant 1,$$

где $a(i)=1/x^i$, $S(0)=1$.

Конечно, можно и эти соотношения использовать для написания программы. При этом у нас возникла новая задача — найти способ вычисления $a(i)$, для чего можно воспользоваться тем же приемом — попытаться вычислить $a(i)$ через значение $a(i-1)$. Соотношение между значениями $a(i)$ и $a(i-1)$ имеет вид

$$a(i)=a(i-1)/x,\ i\geqslant 1,\ a(0)=1.$$

Поэтому поставленную задачу можно решить следующим образом:

```
S[0]:=1
a[0]:=1
нц для i от 1 до N                    (4.2)
| a[i]:=a[i-1]/x
| S[i]:=S[i-1]+a[i]
кц
```

Примечание. Отметим, что и в этом случае индексы при S и a можно опустить в связи с тем, что для вычисления текущего элемента каждой из таблиц достаточно знать только значение предыдущего элемента.

Соотношения, связывающие одни и те же функции, но с различными аргументами, называются ***рекуррентными соотношениями*** или ***рекуррентными уравнениями.***

Вопросы для повторения

Являются ли рекуррентными уравнениями следующие соотношения, где i — натуральное число:

а) $D(i)=D(i-1)/a_i$;

б) $S(i)=S(i-1)+S(i+1)$ **для** $i\geqslant 2$, $S(1)=1$;

в) $P(i)=P(i-2)\cdot i$ **для** $i\geqslant 3$, $P(1)=1$, $P(2)=2$;

г) $S(i)=S(i \operatorname{div} 2)+S(i+1)$ **для** $i\geqslant 2$, $S(0)=1$;

д) $S(i)=S(i-1)+1/i$ **для** $i\geqslant 1$, $S(0)=0$;

е) $S(i)=S(i-1)+(-1)^i x^i/i$ **для** $i\geqslant 1$, $S(0)=1$;

ж) $F(i)=F(i \operatorname{div} 2)+1$ **для** $i\geqslant 2$, $F(1)=0$?

§ 4. ПРАВИЛЬНЫЕ РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ

Правильными рекуррентными соотношениями (уравнениями) будем называть такие рекуррентные соотношения, у которых количество или значения аргументов у функций в правой части соотношения меньше количества или, соответственно, значений аргументов функции в левой части соотношения. Если аргументов несколько, то достаточно уменьшения одного из них.

Следует обратить внимание на то, что соотношения должны быть определены для всех допустимых значений аргументов. Поэтому должны быть определены значения функций при начальных значениях параметров.

В приведенных примерах соотношения связывали функции только с двумя различными параметрами: $S(i)$ и $S(i-1)$, а также $a(i)$ и $a(i-1)$ для любого натурального i. При этом были определены начальные значения $S(0)$ и $a(0)$.

Отметим, что без этих начальных значений рекуррентное соотношение

$$S(i)=S(i-1)+a_i,\ i\geqslant 1,$$

было бы неправильным, так как оно не определено при $i=1$.

Конечно, могут быть и более сложные соотношения, связывающие более двух функций.

Пример. Одной из наиболее известных числовых последовательностей являются числа Фибоначчи, которые определяются следующим рекуррентным соотношением:

$F(0)=1$,
$F(1)=1$,
$F(i)=F(i-1)+F(i-2)$ для натурального $i>1$.

В этом случае для вычисления значения $F(N)$, которое хранится в $F[N]$, можно воспользоваться следующим алгоритмом:

```
F[0]:=1
F[1]:=1
нц для i от 2 до N                                    (4.3)
| F[i]:=F[i-1]+F[i-2]
кц
```

Примечание. При этом в приведенном фрагменте уже нельзя просто опустить индексы, хотя в принципе можно вычислить значение $F(N)$ без использования таблицы, но это уже вопрос способа реализации алгоритма.

Пример такой реализации приводится ниже.

```
a:=1
b:=1
нц для i от 2 до N                                    (4.4)
| c:=b+a
| a:=b
| b:=c
кц
```

Здесь используется тот факт, что для вычисления текущего элемента таблицы нам достаточно знать только значения двух предыдущих.

Вопросы для повторения

Являются ли правильными следующие рекуррентные уравнения:

а) $S(i)=S(i-1)+a_i/2$, i — натуральное;

б) $S(i)=S(i-1)+(S(i-1))^i$ для $i\geqslant 2$,
$S(1)=1$;

в) $P(i)=P(i-1)\cdot i$ для $i\geqslant 2$,
$P(1)=1$;

г) $S(i)=S(i \operatorname{div} 2)\cdot a_i$ для $i\geqslant 2$,
$S(0)=1$;

д) $S(i)=S(i-1)+S(i-2)/i$ для $i\geqslant 2$,
$S(0)=0$, $S(1)=1$;

е) $S(i)=S(i-1)+(-1)^i x^i/i$ для $i\geqslant 1$,
$S(0)=1$;

ж) $F(i)=F((i \operatorname{div} 2)+1)+1$ для $i\geqslant 2$,
$F(1)=0$?

§ 5. СПОСОБ ОРГАНИЗАЦИИ ТАБЛИЦ

Из рассмотренных примеров видно, что важнейшим моментом при решении задачи является способ сведения задачи к подзадачам. Но не менее важным вопросом является и способ построения решения исходной задачи из решений подзадач. Одним из наиболее эффективных способов построения решения исходной задачи является использование таблиц для запоминания решений подзадач. Такой метод решения задач называется ***методом динамического программирования.***

Как уже говорилось раньше, подзадача может быть формализована в виде функции, которая зависит от одного или нескольких аргументов. Если мы возьмем таблицу, у которой количество элементов равно количеству всех возможных различных наборов аргументов функции, то каждому набору аргументов может быть поставлен в соответствие элемент таблицы. Вычислив элементы таблицы (решения подзадач), можно найти и решение исходной задачи.

5.1. Организация одномерных таблиц

Одним из способов организации таблиц является такой, когда размерность таблицы определяется количеством аргументов у функции, соответствующей подзадаче.

Пример. Рассмотрим задачу нахождения произведения 10 элементов таблицы A.

Пусть функция $P(10)$ соответствует решению нашей исходной задачи. В данном случае у функции только один параметр — количество элементов. Для поиска произведения 10 элементов достаточно знать произведение первых 9 элементов и значение 10-го элемента. Поэтому решение исходной задачи можно записать в виде соотношения $P(10)=P(9)\cdot a_{10}$. Следовательно, это соотношение может быть определено для любого i, $2\leqslant i\leqslant 10$:

$$P(i)=P(i-1)\cdot a_i \text{ и } P(1)=a_1.$$

На практике для рассматриваемой задачи чаще используется рекуррентное соотношение, имеющее тот же смысл, но с другим начальным значением:

$$P(i)=P(i-1)\cdot a_i \text{ при } i\geqslant 1,\ P(0)=1.$$

Приведем алгоритм, реализующий данное рекуррентное соотношение:

```
P[0]:=1                                          (4.5)
нц для i от 1 до N
| P[i]:=P[i-1]*a[1]
кц
```

Так как у нашей функции один аргумент — количество сомножителей, то для решения задачи достаточно использовать одномерную таблицу. При этом количество элементов таблицы определяется количеством различных значений аргумента. В приведенном примере размерность массива равна 10. Если бы мы решали задачу поиска произведения 20 элементов, то для реализации рекуррентного соотношения нам было бы достаточно одномерной таблицы с 20 элементами.

Таким образом, размерность таблицы, достаточная для реализации рекуррентных соотношений, определяется количеством аргументов у функций, соответствующих подзадачам. Количество же элементов по каждой размерности (количество элементов в строках, столбцах) определяется количеством возможных значений соответствующего аргумента.

5.2. Организация двумерных таблиц

Еще раз обратим внимание на то, что нас пока не очень интересует реализация алгоритма, при которой минимизируется такая характеристика, как размер используемой оперативной памяти. Будем считать, что памяти компьютера достаточно для хранения соответствующей таблицы.

Пример. Для данной прямоугольной таблицы A размера 5×6 построить прямоугольную таблицу B того же размера, элементы которой обладают следующим свойством: элемент $B[i, j]$ равен максимальному из элементов таблицы B, которые расположены левее и выше позиции (i, j), включая также позицию (i, j). При этом считается, что позиция $(1, 1)$ — верхняя левая позиция прямоугольной таблицы, интересующая нас часть таблицы выделена при $i=3$ и $j=4$.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}

Пусть $T(i, j)$ обозначает функцию, вычисляющую элемент $B[i, j]$.

Определим сначала значения элементов таблицы B, расположенных в первой строке и в первом столбце. Получим:

$T(1, 1)=A[1, 1]$,

$T(1, j)=\max\{T(1, j-1), A[1, j]\}$ при $j\geqslant 2$,
$T(i, 1)=\max\{T(i-1, 1), A[i, 1]\}$ при $i\geqslant 2$.

Эти соотношения следуют из того, что в этих случаях интересующая нас область матрицы A ограничена только элементами первой строки или первого столбца матрицы.

При $2\leqslant i\leqslant 5$ и $2\leqslant j\leqslant 6$ для этой функции можно записать следующее рекуррентное соотношение:

$$T(i, j)=\max\{T(i-1, j), T(i, j-1), A[i, j]\}.$$

Действительно, величина $T(i-1, j)$ соответствует максимальной величине элементов таблицы A в той ее части, которая определяется значением индексов $i-1$ и j, а величина $T(i, j-1)$ — максимальной величине элементов таблицы A, определяемой индексами i и $j-1$. Поэтому эти величины учитывают значения всех элементов матрицы A в той ее части, которая определяется значениями индексов i и j за исключением одного элемента $A[i, j]$.

Для реализации этих рекуррентных соотношений достаточно двумерной (прямоугольной) таблицы, так как у функции T два аргумента. Важно отметить, что при этом мы можем отождествить величины $T(i, j)$ и $B[i, j]$. Тогда фрагмент алгоритма можно записать следующим образом:

```
B[1, 1]:=A[1, 1]
нц для j от 2 до 6
| B[1, j]=max(B[1, j-1], A[1, j])
кц
нц для i от 2 до 5
| B[i, 1]=max(B[i-1, 1], A[i, 1])                    (4.6)
кц
нц для i от 2 до 5
| нц для j от 2 до 6
| | B[i, j]=max(B[i, j-1], B[i-1, j])
| | B[i, j]=max(B[i, j], A[i, j])
| кц
кц
```

Вопросы для повторения

1. Определить размеры одномерных таблиц для следующих рекуррентных уравнений, где i — натуральное:
   а) $S(i)=S(i-1)+S(i-1)$ для $2\leqslant i\leqslant 10$, $S(1)=1$;
   б) $P(i)=P(i-1)\cdot i$ для $2\leqslant i\leqslant 20$, $P(1)=1$;
   в) $S(i)=P(i \operatorname{div} 2)+1$ для $1\leqslant i\leqslant 30$, $S(0)=1$;
   г) $S(i)=S(i-1)+1/i$ для $1\leqslant i\leqslant 100$, $S(0)=0$;
   д) $S(i)=S(i-1)+(-1)^i x^i/i$ для $1\leqslant i\leqslant 1000$, $S(0)=1$.
2. Определить размеры двумерных таблиц для следующих рекуррентных уравнений:
   а) $S(i, j)=\min(S(i-1, j), S(i, j-1), a_{ij})$ для $2\leqslant i\leqslant 5$, $2\leqslant j\leqslant 6$, $S(1, j)=a_{1j}$, $S(i, 1)=a_{i1}$;
   б) $S(i, j)=\max(S(i-1, j), S(i-1, j-1))+a_{ij}$ для $2\leqslant i\leqslant 7$, $2\leqslant j\leqslant 13$, $S(1, j)=a_{1j}$, $S(i, 1)=a_{i1}$;
   в) $S(i, j)=S(i-1, j)+a_{ij}+S(i, j-1)$ для $2\leqslant i\leqslant 10$, $2\leqslant j\leqslant 6$, $S(1, j)=a_{1j}$, $S(i, 1)=a_{i1}$;
   г) $S(i, j)=S(i-1, j)-S(i-1, j-1)+a_{ij}$ для $1\leqslant i\leqslant 9$, $1\leqslant j\leqslant 9$, $S(0, j)=0$, $S(i, 0)=0$, $S(0, 0)=0$.

§ 6. СПОСОБ ВЫЧИСЛЕНИЯ ЭЛЕМЕНТОВ ТАБЛИЦЫ

После того как найдено сведение задачи к подзадачам и определены рекуррентные соотношения, соответствующие этому сведению, необходимо определить наиболее рациональный способ вычисления элементов таблицы.

6.1. Вычисление элементов одномерной таблицы

Для одномерной таблицы таким способом обычно является последовательное вычисление элементов, начиная с первого.

Пример. Определить, сколькими различными способами можно подняться на 10-ю ступеньку лестницы, если за один шаг можно подниматься на следующую ступеньку или через одну.

Пусть $K(10)$ — задача поиска количества способов подъема на 10-ю ступеньку. Определим i-ю подзадачу нашей задачи как задачу поиска количества способов подъема на i-ю ступеньку.

Исходя из условия задачи, на 10-ю ступеньку можно подняться непосредственно с 8-й и 9-й. Поэтому, если мы знаем количество способов подъема $K(8)$ и $K(9)$ на 8-ю и 9-ю ступеньки, то количество способов подъема на 10-ю может быть определено как $K(10)=K(8)+K(9)$.

Такое соотношение получается потому, что любой способ подъема на 8-ю ступеньку превращается в способ подъема на 10-ю добавлением перешагивания через 9-ю, а любой способ подъема на 9-ю ступеньку превращается в способ подъема на 10-ю добавлением подъема с 9-й на 10-ю. Все эти способы различны.

Аналогичное соотношение справедливо для любой ступеньки i, начиная с третьей:

$$K(i)=K(i-2)+K(i-1).$$

Осталось определить значения $K(1)$ и $K(2)$, которые равны: $K(1)=1$, $K(2)=2$.

Следовательно, для решения задачи достаточно одномерной таблицы с 10 элементами, для которой необходимо последовательно вычислить значения элементов таблицы согласно приведенным выше рекуррентным соотношениям.

```
K[1]:=1
K[2]:=2
нц для i от 3 до 10                    (4.7)
| K[i]:=K[i-1]+K[i-2]
кц
```

Примечание. Полученные рекуррентные соотношения не отличаются от рекуррентных соотношений примера 6, поэтому могут быть реализованы без использования таблицы.

6.2. Вычисление элементов двумерной таблицы

Пример. В таблице с N строками и M столбцами, состоящей из 0 и 1, необходимо найти квадратный блок максимального размера, состоящий из одних единиц. Под блоком понимается множество элементов соседних (подряд идущих) строк и столбцов таблицы. Интересующая нас часть таблицы выделена.

1	1	1	1	1	1
0	1	1	1	0	1
1	1	1	1	1	1
1	1	0	1	1	1
1	0	1	1	0	1

Положение любого квадратного блока может быть определено его размером и положением одного из его углов.

Пусть $T(i, j)$ — функция, значение которой соответствует размеру максимального квадратного блока, состоящего из одних единиц, правый нижний угол которого расположен в позиции (i, j). Функция $T(i, j)$ вычисляет элемент таблицы $B[i, j]$. Для приведенной выше таблицы значения $T(i, j)$ будут иметь вид:

i \ j	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	2	2	0	1
3	1	1	2	3	1	1
4	1	2	0	1	2	2
5	1	0	1	1	0	1

Таким образом, наша задача свелась к вычислению максимального значения функции T при всевозможных значениях параметров i и j. Этой функции может быть поставлена в соответствие таблица размера $N \cdot M$.

Определим сначала значение элементов таблицы B,

расположенных в первой строке и в первом столбце. Получим:

$B(1, 1)=A[1, 1]$,

$B(1, j)=A[1, j]$ при $j\geqslant 2$,

$B(i, 1)=A[i, 1]$ при $i\geqslant 2$.

Данные соотношения следуют из того, что в этих случаях рассматриваемая область матрицы A содержит только один элемент матрицы.

При $2\leqslant i\leqslant N$ и $2\leqslant j\leqslant M$ для этой функции можно записать следующие рекуррентные соотношения:

$B[i, j]=0$, если $A[i, j]=0$

и

$B[i, j]=\min\{B[i-1, j], B[i, j-1], B[i-1, j-1]\}+1$, если $A[i, j]=1$.

Первое соотношение показывает, что размер максимального единичного блока с правым нижним углом в позиции (i, j) равен нулю в случае $A[i, j]=0$.

Убедимся в правильности второго соотношения. Действительно, величина $B[i-1, j]$ соответствует максимальному размеру единичного блока таблицы A с правым нижним углом в позиции $(i-1, j)$. Тогда размер единичного блока с правым нижним углом в позиции (i, j) не превышает величину $B[i-1, j]+1$, так как к блоку в позиции $(i-1, j)$ могла добавиться только одна строка.

Величина $B[i, j-1]$ соответствует максимальному размеру единичного блока таблицы A с правым нижним углом в позиции $(i, j-1)$. Тогда размер единичного блока с правым нижним углом в позиции (i, j) не превышает величину $B[i, j-1]+1$, так как к блоку в позиции $(i-1, j)$ мог добавиться только один столбец.

Величина $B[i-1, j-1]$ соответствует максимальному размеру единичного блока таблицы A с правым нижним углом в позиции $(i-1, j-1)$. Тогда размер единичного блока с правым нижним углом в позиции (i, j) не превышает величину $B[i-1, j-1]+1$, так как к блоку в позиции $(i-1, j-1)$ могли добавиться только одна строка и один столбец.

Итак, размер единичного блока с правым нижним углом в позиции (i, j) равен $\min\{B[i-1, j], B[i, j-1], B[i-1, j-1]\}+1$.

```
B[1, 1]:=A[1, 1]
нц для j от 2 до 6
| B[1, j]:=A[1, j]
кц
нц для i от 2 до 5
| B[i, 1]:=A[i, 1]
кц                                                    (4.8)
нц для i от 2 до 5
| нц для j от 2 до 6
| |если A[i, j]:=1
| | | то
| | | B[i, j]:=min(B[i, j-1], B[i-1, j])
| | | B[i, j]:=min(B[i, j], B[i-1, j-1])+1
| | | иначе
| | | B[i, j]:=0
| | все
| кц
кц
```

6.3. Вычисление элементов двумерной таблицы с дополнительными ограничениями

Пример. На складе имеется 5 неделимых предметов. Для каждого предмета известна его стоимость (в рублях) и масса (в килограммах). Величины стоимости и массы являются натуральными числами. Наша цель состоит в том, чтобы определить максимальную суммарную стоимость предметов, которые можно унести со склада при условии, что суммарная масса предметов не должна превышать 16 кг.

Пусть элемент C_i таблицы C соответствует стоимости i-го предмета, а элемент M_i таблицы M — массе i-го предмета. Будем считать, что предметы пронумерованы в порядке их следования в таблицах.

Пусть T обозначает функцию, значение которой соответствует решению нашей задачи. Аргументами у этой функции является количество предметов (по этому аргументу можно определить их стоимости и массы соответствующих предметов), а также максимальная суммарная масса, которую можно унести.

Для нашей задачи $T(5, 16)$ определим подзадачи $T(i, j)$, где i обозначает количество начальных предметов, из которых можно осуществлять выбор, а j определяет максимально возможную суммарную массу уносимых предметов. Отметим, что введенный таким образом первый параметр i определяет как количество предметов для подзадачи, так и значения их стоимостей и масс из таблиц C и M.

Определим сначала начальные значения функции T. При нулевых значениях одного из аргументов значение функции равно нулю:

$T(0, 0)=0$,

$T(0, j)=0$ при $j \geqslant 1$,

$T(i, 0)=0$ при $i \geqslant 1$.

Определим возможные значения функции $T(i, j)$ при ненулевых значениях аргументов.

Решение подзадачи, соответствующей функции $T(i, j)$, может быть сведено к двум возможностям: уносится ли при наилучшем решении предмет с номером i или нет.

Если предмет не уносится, то решение задачи с i предметами сводится к решению подзадачи с $i-1$ предметами, т. е.

$$T(i, j)=T(i-1, j).$$

Если предмет с номером i уносится, то это уменьшает максимально возможную суммарную массу для $i-1$ первых предметов на величину $M[i]$, одновременно при этом увеличивая значение решения для оставшихся предметов $T(i-1, j-M[i])$ на величину $C[i]$, т. е.

$$T(i, j)=T(i-1, j-M[i])+C[i].$$

При этом необходимо учитывать, что вторая ситуация возможна только тогда, когда масса i-го предмета не больше значения j.

Теперь для получения наилучшего решения нам необходимо выбрать лучшую из этих двух возможностей. Поэтому рекуррентное соотношение при $i \geqslant 1$ и $j \geqslant 1$ имеет вид:

$$T(i, j) = T(i-1, j)$$

при $j < M[i]$,

$T(i, j) = \max(T(i-1, j), \quad T(i-1, j-M[i]) + C[i]$

при $j \geqslant M[i]$.

Пусть заданы следующие значения стоимости и массы для 5 предметов:

$$C[1]=5, \ M[1]=4;$$
$$C[2]=7, \ M[2]=5;$$
$$C[3]=4, \ M[3]=3;$$
$$C[4]=9, \ M[4]=7;$$
$$C[5]=8, \ M[5]=6.$$

Таблица значений функции T, которую мы также назовем T, выглядит следующим образом:

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5
2	0	0	0	0	5	7	7	7	7	12	12	12	12	12	12	12	12
3	0	0	0	4	5	7	7	9	11	12	12	12	16	16	16	16	16
4	0	0	0	4	5	7	7	9	11	12	13	14	16	16	18	20	21
5	0	0	0	4	5	7	8	9	11	12	13	15	16	17	19	20	21

Следовательно, решение задачи $T(5, 16)=21$, т. е. можно унести предметов на 21 рубль.

Приведем одну из возможных реализаций.

```
T[0, 0]:=0
нц для j от 1 до 16
| T[0, j]=0
кц
нц для i от 1 до 5
| T[i, 0]=0                                                    (4.9)
кц
нц для i от 1 до 5
| нц для j от 1 до 16
| |если j>=M[i]
| | | то
| | | T[i, j]=max(T[i-1, j], T[i-1, j-M[i]]+C[i])
| | | иначе
| | | T[i, j]=T[i-1, j]
| | все
| кц
кц
```

ЗАДАЧИ ДЛЯ ПОВТОРЕНИЯ

1. К каким подзадачам может сводиться поиск одной фальшивой монеты среди 27 монет, если известно, что она легче других, а все остальные имеют одинаковую массу?

2. К каким подзадачам может сводиться задача вычисления значения $N!$ ($N!=1\cdot 2\cdot 3\cdot \ldots \cdot N$)?

3. К каким подзадачам может сводиться задача поиска суммы положительных элементов таблицы, состоящей из 10 элементов?

4. Являются ли правильными рекуррентные уравнения, где i — натуральное число:

а) $S(i)=S(i-1)-a_i$;

б) $S(i)=S(i-1)+S(i-1)$ для $i\geqslant 2$,
$S(1)=1$;

в) $P(i)=P(i-1)\,i$ для $i\geqslant 2$,
$P(1)=1$;

г) $S(i)=S(i \text{ div } 2)+1$ для $i\geqslant 2$,
$S(0)=1$;

д) $S(i)=S(i-1)+1/i$ для $i\geqslant 1$,
$S(0)=0$;

е) $S(i)=S(i-1)+(-1)^i x^i/i$ для $i\geqslant 1$,
$S(0)=1$;

ж) $F(i)=F((i \text{ div } 2)+1)+1$ для $i\geqslant 2$,
$F(1)=0$, $F(1)=-1$?

5. Указать, при каких размерах таблиц могут быть реализованы рекуррентные соотношения:

а) $S(i)=S(i \text{ div } 2)+S(i-1)$ для $2\leqslant i\leqslant 20$,
$S(1)=1$;

б) $S(i, j)=\min(S(i-1, j), S(i-1, j-1), a_{ij})$ для $2\leqslant i\leqslant N$, $2\leqslant j\leqslant M$,
$S(1, j)=a_{1j}$, $S(i, 1)=a_{i1}$?

6. Пусть $v_1=1{,}5$; $v_i=\dfrac{i+1}{i^2+2}v_{i-1}$. Получить $v_2, v_3, \ldots, v_n$, где n — заданное натуральное число.

7. Пусть $x_0=c$, $x_1=d$; $x_i=x_{i-1}+x_{i-2}+b$. Получить $x_2, x_3, \ldots, x_n$, где n, b, c, d — заданные натуральные числа.

8. Пусть $u_1=0$, $u_2=1$; $u_i=u_{i-1}+u_{i-2}\cdot u_{i-1}-u_{i-2}$. Получить $u_3, u_4, \ldots, u_n$, где n — заданное натуральное число.

9. Пусть $a_0=a_1=1$; $a_i=\dfrac{a_i-1}{2^{i-1}}+a_{i-2}$. Получить $a_2, a_3, \ldots, a_n$, где n — заданное натуральное число.

10. Пусть $a_1=b_1=1$, $a_i=0{,}5\left(\sqrt{b_{i-1}}+0{,}5\sqrt{a_{i-1}}\right)$, $b_i=2a_{i-1}^2+b_{i-1}$. Получить $a_2, a_3, \ldots, a_n, b_2, b_3, \ldots, b_n$, где n — натуральное число.

11. Пусть $a_1=u$, $b_1=v$, $a_i=2b_{i-1}+a_{i-1}$, $b_i=2a_{i-1}^2+b_{i-1}$. Получить $a_2, a_3, \ldots, a_n, b_2, b_3, \ldots, b_n$, где n — натуральное число, а u, v — некоторые действительные числа.

12. Пусть $a_1=1$, $b_1=1$, $a_i=3/4a_{i-1}-b_{i-1}$, $b_i=4/3b_{i-1}-a_{i-1}$. Получить $a_2, a_3, \ldots, a_n, b_2, b_3, \ldots, b_n$, где n — натуральное число.

13. Пусть $a_1=1$, $b_1=1$, $a_i=2^i a_{i-1}+i!b_{i-1}$, $b_i=i!a_{i-1}+3^i b_{i-1}$. Получить $a_2, a_3, \ldots, a_n, b_2, b_3, \ldots, b_n$, где n — натуральное число.

14. Вычислить значение дробей.

а) $\cfrac{1}{1+\cfrac{2}{2+\cfrac{4}{3+\cfrac{8}{\ddots\,+\cfrac{2^n}{n+1}}}}}$

б) $\cfrac{3}{1+\cfrac{3^2}{2+\cfrac{3^3}{3+\cfrac{3^4}{\ddots\,+\cfrac{3^{n+1}}{n+1}}}}}$

в) $\cfrac{1}{1+\cfrac{2}{2+\cfrac{3}{3+\cfrac{4}{\ddots\,+\cfrac{n+1}{n+2}}}}}$

г) $\cfrac{1}{1+\cfrac{-1}{2+\cfrac{1}{3+\cfrac{-1}{\ddots\,+\cfrac{(-1)^n}{n+1}}}}}$

д) $\cfrac{2}{12+\cfrac{4}{12+\cfrac{8}{12+\cfrac{16}{\ddots\,+\cfrac{2^n}{12}}}}}$

е) $\cfrac{1}{2+\cfrac{a}{3+\cfrac{a^2}{4+\cfrac{a^3}{\ddots\,+\cfrac{a^n}{n+2}}}}}+n-1$

ж) $\cfrac{1}{a+\cfrac{2}{a^2+\cfrac{6}{a^3+\cfrac{24}{\ddots\,+\cfrac{n!}{a^{n+1}}}}}}$

з) $\cfrac{a}{2+\cfrac{-2}{3+\cfrac{6}{4+\cfrac{-24}{\ddots\,+\cfrac{(-1)^n n!}{n+1}}}}}$

ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Составить алгоритм определения количества шестизначных «счастливых» трамвайных билетов, у которых сумма первых трех цифр совпадает с суммой трех последних.

2. Составить алгоритм определения количества $2N$-значных «счастливых» билетов, у которых сумма первых N цифр равна сумме последних N цифр; N — произвольное натуральное число.

3. Найти количество n-значных чисел в десятичной системе счисления, у каждого из которых сумма цифр равна k. При этом в качестве n-значного числа мы допускаем и числа, начинающиеся с одного или нескольких нулей. Например, 000102 рассматривается как шестизначное число, сумма цифр которого равна 3.

4. Фишка может двигаться по полю длиной N только вперед. Длина хода фишки не более K. Найти число различных путей, по которым фишка может пройти поле от позиции 1 до позиции N.

Пример. $N=4$, $K=2$.

Возможные длины ходов:

1, 1, 1
1, 2
2, 1

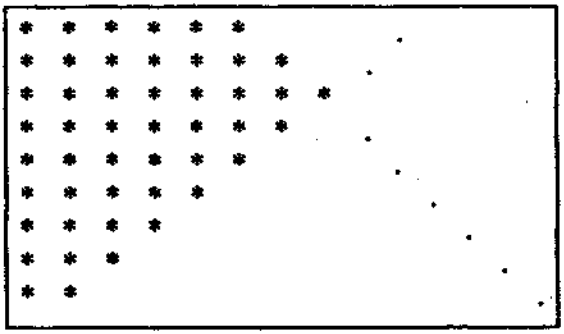
Ответ: 3.

5. Покупатель имеет купюры достоинством $A_1, \ldots, A_n$, а продавец — $B_1, \ldots, B_m$. Найти максимальную стоимость товара P, который покупатель не может купить, потому что нет возможности точно рассчитаться за этот товар с продавцом, хотя денег на покупку его достаточно.

6. У покупателя есть n монет достоинством $H_1, \ldots, H_n$. У продавца есть m монет достоинством $B_1, \ldots, B_m$. Может ли покупатель приобрести вещь стоимостью S так, чтобы у продавца нашлась точная сдача (если она необходима)?

7. По матрице $A[1..N, 1..N]$ построить матрицу $B[1..N, 1..N]$. Элемент $B[i, j]$ равен максимальному из элементов матрицы A, принадлежащему части, ограни-

ченной справа диагоналями, проходящими через $A[i, j]$ (см. таблицу).



8. Задана матрица натуральных чисел $A[1..n, 1..m]$. За каждый проход через клетку (i, j) взимается штраф $A[i, j]$. Необходимо минимизировать штраф и:

а) пройти из какой-либо клетки 1-й строки в n-ю строку, при этом из текущей клетки можно перейти в любую из 3 соседних, стоящих в строке с номером, на 1 большим;

б) реализовать пункт а) для перехода из клетки $(1, 1)$ в клетку (n, m).

9. Выпуклый N-угольник, $N \geqslant 3$, задается координатами своих вершин в порядке обхода по контуру. Разбить его на треугольники $(N-3)$ диагоналями, не пересекающимися, кроме как по концам, таким образом, чтобы:

а) сумма их длин была минимальной;

б) самая длинная из диагоналей имела наименьшую длину.

10. Пусть $x=(a_1, a_2, \ldots, a_m)$ и $y=(b_1, b_2, \ldots, b_n)$ — две заданные строки символов.

Обозначим через $d(x, y)$ минимальное количество вставок, удалений и замен символов, которое необходимо для преобразования x в y.

Например: $d(ptsldd f, tsgldds)=3$

$$ptsldd f \xrightarrow{\text{удаление } p} tsldd f \xrightarrow{\text{вставка } g} tsgldd f \xrightarrow{\text{замена } f} tsgldds$$

Для заданных строк x и y определить $d(x, y)$.

11. Даны две строки x и y. Строка x состоит из нулей и единиц, строка y состоит из символов A и B. Можно ли строку x преобразовать в строку y по следующему правилу: цифра 0 преобразуется в непустую последовательность букв A, а цифра 1 — либо в непустую последовательность букв A, либо в непустую последовательность букв B?

12. Пусть известно, что для перемножения матрицы размера $n \times m$ на матрицу размера $m \times k$ требуется $n \cdot m \cdot k$ операций, и в результате получается матрица размера $n \times k$.

Необходимо определить, какое минимальное число операций потребуется для перемножения s матриц A_1, ..., A_s, заданных своими размерами $n(i) \cdot m(i)$. При этом можно перемножать любые две рядом стоящие матрицы.

Замечание:

$n(i)$ — число строк в матрице A_i,

$m(i)$ — число столбцов в матрице A_i,

$n(i) = m(i+1)$.

13. а) Из заданной числовой последовательности $A[1..N]$ вычеркнуть минимальное количество элементов так, чтобы оставшиеся образовали строго возрастающую последовательность (или, что то же, найти максимальную по длине строго возрастающую подпоследовательность последовательности A).

б) Из заданной числовой последовательности $A[1..N]$ вычеркнуть минимальное число элементов таким образом, чтобы в оставшейся подпоследовательности каждый последующий элемент был больше предыдущего, кроме, быть может, одной пары соседних элементов (одного «разрыва» возрастающей подпоследовательности).

Например: $A = (1, 2, 3, 2, 4, 3, 4, 6)$;

Искомая подпоследовательность (1, 2, 3, 2, 3, 4, 6). Разрыв подчеркнут.

в) Из заданной числовой последовательности $A[1..N]$ вычеркнуть минимальное число элементов так, чтобы в оставшейся подпоследовательности каждый последующий элемент был больше предыдущего, кроме, быть может, m пар соседних элементов (сформировать возрастающую подпоследовательность с m «разрывами»).

14. Из элементов заданной последовательности целых чисел построить такую максимально длинную подпоследовательность чисел, чтобы каждый последующий элемент подпоследовательности делился нацело на предыдущий.

15. Задаются число $n>1$ — размерность пространства и размеры M n-мерных параллелепипедов $(a_{i1}, \ldots, a_{in})$, $i=1, \ldots, m$. Параллелепипед может располагаться в пространстве любым из способов, при которых его ребра параллельны осям координат.

Найти максимальную последовательность вкладываемых друг в друга параллелепипедов.

16. Заданы три числа a, b, c. Можно ли представить число a таким образом, чтобы

$$a=x[1]\cdot x[2]\cdot\ldots\cdot x[k]=\prod_{i=1}^{k} x[i],$$

где $b\leqslant x[i]\leqslant c$, $x[i]$, a, b, c — целые? Лучшим считается алгоритм, находящий такое представление с наименьшим числом множителей. Предусмотреть вариант, когда такого представления не существует.

17. Схема железнодорожной сортировочной станции приведена на рисунке 29.

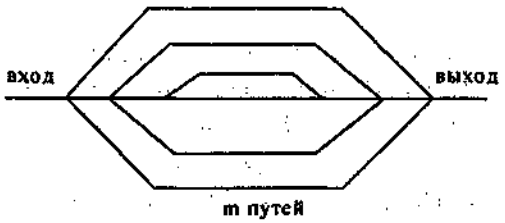


Рис. 29

Пути пронумерованы от 1 до m.

На вход в произвольном порядке подается n вагонов, занумерованных числами от 1 до n. Каждый вагон необходимо направить на один из станционных путей, откуда его затем переводят на выход. На любой путь можно направить произвольное количество вагонов. На выходе необходимо сформировать состав с номерами вагонов в возрастающем порядке.

Описать алгоритм, который по данным n, m и исходной последовательности номеров вагонов отвечает на вопрос, можно ли выполнить требуемую сортировку.

18. Возвести число A в натуральную степень n за как можно меньшее количество умножений.

19. Заданы две последовательности чисел z и y. Можно ли получить последовательность z вычеркиванием элементов из y?

20. Вводятся две последовательности x и y. Найти максимальную по длине последовательность z, которую можно получить вычеркиванием элементов как из x, так и из y.

Например, для x=«*abacs*», y=«*dalas*» последовательность z=«*aas*».

21. Пусть x и y — две бинарных последовательности (т. е. элементы последовательностей — нули и единицы); x и y можно рассматривать как запись в двоичной форме некоторых двух натуральных чисел.

Найти максимальное число z, двоичную запись которого можно получить вычеркиванием цифр как из x, так и из y. Ответ выдать в виде бинарной последовательности.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. Существует игра для одного игрока, которая начинается с задания цепочки с N вершинами. Пример графического представления цепочки показан на рисунке 30, *а*, где N=4. Для каждой вершины цепочки задается значение — целое число, а для каждого ребра —

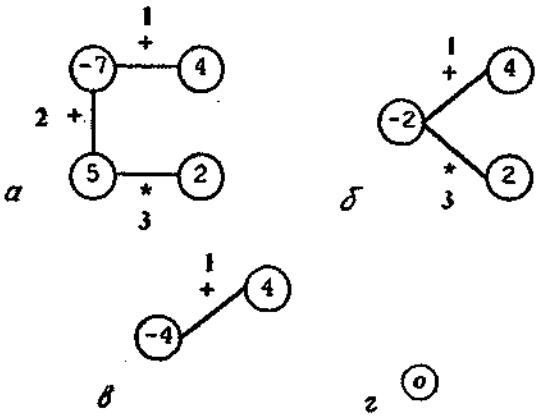


Рис. 30

метка операции + (сложение) либо * (умножение). Ребра цепочки пронумерованы от 1 до $N-1$.

Каждый ход состоит из следующих шагов:

выбирается ребро E и две вершины V_1 и V_2, которые соединены ребром E;

ребро E и вершины V_1 и V_2 заменяются новой вершиной со значением, равным результату выполнения операции, определенной меткой ребра E, над значениями вершин V_1 и V_2.

Игра заканчивается, когда больше нет ни одного ребра. Результат игры — это число, равное значению оставшейся вершины.

Пример игры. Игрок начал игру с удаления ребра *2* (рис. 30, *б*), затем — ребра *3* (рис. 30, *в*), наконец, ребра *1*. Результатом игры будет число 0 (рис. 30, *г*).

Написать программу, которая по заданной цепочке вычисляет максимальное значение оставшейся вершины и выводит список всех тех ребер, удаление которых на первом ходе игры позволяет получить это значение.

Входные данные находятся в файле с именем CHAIN.IN и имеют следующую структуру.

В первой строке находится число N ($N \leqslant 100$) — количество вершин цепочки.

В каждой из последующих N строках находятся значения в вершинах — целые числа.

В каждой из последующих $N-1$ строк находится тип операций (+ или *) для соответствующего ребра. Ребро с номером i соединяет вершины с номерами i и $i+1$.

В выходной файл с именем CHAIN.OUT требуется вывести максимально возможный результат.

2. В связи с эпидемией гриппа в больницу направляется A больных гриппом «А» и B больных гриппом «В». Больных гриппом «А» нельзя помещать в одну палату с больными гриппом «В». Имеется информация об общем количестве палат P в больнице, пронумерованных от 1 до P, и о распределении уже имеющихся там больных.

Написать программу, определяющую максимальное количество больных M, которое больница в состоянии принять. При размещении новых больных не разрешается переселять уже имеющихся больных из палаты в палату.

Входные данные находятся в текстовом файле и имеют следующую структуру:

в первой строке находится число A (целое, $0 \leqslant A \leqslant 100$);

во второй строке — число B (целое, $0 \leqslant B \leqslant 100$);

в третьей строке — число P (натуральное, $P \leqslant 20$);

в каждой из последующих P строк находятся 3 числа n, a, b, разделенных пробелом, где n — вместимость палаты, a — количество уже имеющихся в палате больных гриппом «А», b — количество уже имеющихся в палате больных гриппом «В». Информация о вместимости палат вводится последовательно для палат с номерами 1, 2, ..., P. Числа n, a, b — целые неотрицательные меньше 100.

Выходные данные должны быть записаны в текстовый файл и иметь следующий формат:

в первой строке должно находиться число M;

если все поступившие больные размещены, то во

второй строке должны находиться номера палат, разделенные пробелом, куда помещаются больные гриппом «А».

3. Задается натуральное число N ($N \leqslant 999$). Двое играющих называют по очереди числа, меньше 1000, по следующим правилам. Начиная с числа N, каждое новое число должно увеличивать одну из цифр предыдущего числа (возможно, незначащий нуль) на 1, 2 или 3. Проигравшим считается тот, кто называет число 999.

Для заданного N необходимо определить, может ли выиграть игрок, делающий первый ход, при наилучших последующих ходах противника. Вывести сообщение «Первый выигрывает» или «Первый проигрывает». В случае возможности выигрыша первым игроком требуется напечатать все его возможные первые ходы.

4. Задан числовой треугольник из N строк. Написать программу, которая определяет максимальную сумму чисел, расположенных на пути, который начинается с верхнего числа и заканчивается на каком-нибудь числе в основании треугольника (максимум суммы — среди всех таких путей).

```
      7
    3   8
  8   1   0
2   7   7   4
4   5   2   6   5
```

На каждом шаге можно двигаться к соседнему по диагонали числу влево-вниз или вправо-вниз. Число строк в треугольнике больше 1 и меньше либо равно 100. Все числа в треугольнике — целые в интервале между 0 и 99 включительно.

5. В магазине каждый товар имеет цену. Например, цена одного цветка равна 2, а цена одной вазы равна 5. Чтобы привлечь покупателей, магазин ввел скидки: решил продавать набор одинаковых или разных товаров

по пониженной цене. Например: три цветка за 5 вместо 6 или две вазы вместе с одним цветком за 10 вместо 12.

Написать программу, вычисляющую наименьшую цену, которую покупатель должен заплатить за заданные покупки. Оптимальное решение должно быть получено посредством скидок. Набор товаров, который требуется купить, нельзя дополнять ничем, даже если бы это снизило общую стоимость набора. Для описанных выше цен и скидок наименьшая цена за три цветка и две вазы равна 14: две вазы и один цветок продаются по сниженной цене за 10 и два цветка — по обычной цене за 4.

Входные данные содержатся в двух файлах: INPUT.TXT и OFFER.TXT. Первый файл описывает покупки («корзину с покупками»), второй — скидки. В обоих файлах содержатся только целые числа.

Первая строка файла INPUT.TXT содержит количество b различных видов товара в корзине ($0 \leqslant b \leqslant 5$). Каждая из следующих b строк содержит значения k и p. Значение c — уникальный код товара ($1 \leqslant c \leqslant 999$). Значение p задает, сколько единиц товара находится в корзине ($1 \leqslant k \leqslant 5$). Обратите внимание, что общее количество товаров в корзине может быть не более $5 \cdot 5 = 25$ (единиц).

Первая строка файла OFFER.TXT содержит количество s возможных скидок ($0 \leqslant s \leqslant 99$). Каждая из следующих s строк описывает одну скидку, определяя набор товаров и общую стоимость набора. Первое число n в такой строке определяет количество различных видов товара в наборе ($1 \leqslant n \leqslant 5$). Следующие n пар чисел (c, k) указывают, что k единиц товара с кодом c включены в набор для скидки ($1 \leqslant k \leqslant 5$, $1 \leqslant x \leqslant 999$). Последнее число в строке p определяет уменьшенную стоимость набора ($1 \leqslant p \leqslant 9999$). Стоимость набора меньше суммарной стоимости отдельных единиц товаров в наборе.

Записать в выходной файл OUTPUT.TXT одну строку с наименьшей возможной суммарной стоимостью покупок, заданных во входном файле.

6. В файловой системе настенного персонального компьютера ВС-1 (Висячая Система) файлы организованы в каталоги. В компьютере нет понятия устройства и поэтому полное имя файла является строкой, состоящей из имен каталогов и имени файла, разделенных символом \, причем \ не может быть первым, последним символом, а также идти два раза подряд.

Имя файла (каталога) может быть произвольной длины, но длина полного имени файла не может быть длиннее N символов. В качестве символов, допустимых к употреблению в именах файлов (каталогов), могут использоваться символы из алфавита, состоящего из K букв (символ \ не входит в их число).

Для данных $K(1 \leqslant K \leqslant 13)$ и $N(1 \leqslant N \leqslant 50)$ определить максимальное число файлов, которое можно записать на данный компьютер.

7. Во время трансляции концерта «Песня года» предприниматель K решил сделать бизнес на производстве кассет. Он имеет M кассет с длительностью звучания D каждая и хочет записать на них максимальное число песен. Эти песни (их общее количество N) передаются в порядке 1, 2, ..., N и имеют заранее известные ему длительности звучания $L(1)$, $L(2)$, ..., $L(N)$. Предприниматель может выполнить одно из следующих действий:

записать очередную песню на кассету (если она туда помещается) или пропустить ее;

если песня на кассету не помещается, то может пропустить эту песню или начать записывать ее на новую кассету, при этом старую кассету отложить и туда уже ничего не записывать.

Определить максимальное количество песен, которые предприниматель может записать на кассеты.

8. Учитель информатики живет на N-м этаже девятиэтажного дома с лифтом, который может останавливаться на каждом этаже. Между соседними этажами дома имеется лестница из двух пролетов, разделенных площадкой, по k ступенек в каждом пролете. Сколькими

способами учитель может подняться на свой этаж, если, поднимаясь по лестнице, можно становиться на следующую ступеньку или идти через одну ступеньку?

9. Среди всех N-битных двоичных чисел указать количество тех, у которых в двоичной записи нет подряд идущих k единиц. Сами числа выдавать не надо! N и k — натуральные, $k \leqslant N \leqslant 30$.

10. В связи с открытием олимпиады по информатике N человек ($N \leqslant 10$) решили устроить вечеринку. Для проведения вечеринки достаточно купить MF бутылок фанты, MB бананов и MC тортов. Требуется определить минимальный взнос участника вечеринки.

При покупке определенных наборов товара действуют правила оптовой торговли: стоимость набора товара может отличаться от суммарной стоимости отдельных частей.

Написать программу, которая по входным данным определяет минимальный взнос участника вечеринки.

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. *Вариант* 1. Самое простое — это перебрать все возможные комбинации шести цифр и подсчитать число «счастливых» билетов.

```
Count:=0;        {количество «счастливых» билетов}
for a1:=0 to 9 do
  for a2:=0 to 9 do
    for a3:=0 to 9 do
      for a4:=0 to 9 do
        for a5:=0 to 9 do
          for a6:=0 to 9 do
            if a1+a2+a3=a4+a5+a6
              {«счастливый»?}
            then Count:=Count+1;
```

Под сложностью алгоритма будем понимать количество выполнений операторов наиболее глубоко вло-

женного цикла. Условие *if* во вложенных циклах будет проверяться 10^6 раз, поэтому будем говорить, что сложность этого алгоритма 10^6.

Вариант 2. Обратим внимание на то, что в «счастливом» билете последняя цифра a_6 однозначно определяется первыми пятью:

$$a_6=(a_1+a_2+a_3)-(a_4+a_5).$$

Если $0\leqslant a_6\leqslant 9$, то билет «счастливый», иначе — нет. Таким образом, мы можем убрать шестой вложенный цикл:

```
Count:=0;
for a1:=0 to 9 do
  for a2:=0 to 9 do
    for a3:=0 to 9 do
      for a4:=0 to 9 do
        for a5:=0 to 9 do
          begin
            a6:=(a1+a2+a3)-(a4+a5);
            if (a6>=0) and (a6<=9)
            then Count:=Count+1;
          end;
```

Сложность алгоритма 10^5. Используя зависимость a_6 от первых пяти цифр, мы уменьшили сложность алгоритма и, вообще говоря, время выполнения программы в 10 раз!

Вариант 3. Если комбинаций $a_1a_2a_3$ первых трех цифр с суммой $T=a_1+a_2+a_3$ насчитывается $C[T]$, то всего «счастливых» билетов с суммой половины $T=a_1+a_2+a_3=a_4+a_5+a_6$ будет $C[T]^2$.

Действительно, каждое «счастливое» шестиразрядное число может быть получено «склейкой» двух произвольных трехразрядных чисел с одинаковой суммой цифр. Всего существует 28 всевозможных значений сумм T — от $0=0+0+0$ до $27=9+9+9$. Подсчитаем $C[i]$, $i=$

$=0, \ldots, 27$, затем находим интересующее нас количество «счастливых» билетов: $C[0]^2+C[1]^2+\ldots+C[27]^2$.

Заметим, что «счастливых» билетов с суммой T столько же, сколько и с суммой $27-T$. Действительно, если билет $a_1a_2a_3a_4a_5a_6$ с суммой T — «счастливый», то таковым же является и билет $(999999-a_1a_2a_3a_4a_5a_6)$ с суммой $27-T$. Поэтому число билетов можно вычислять и по формуле $2(C[0]^2+\ldots+C[13]^2)$, т. е. рассматривать только суммы от 0 до 13.

```
  var C: array[0..13] of longint;
{массив C из 14 элементов — по числу рассматриваемых
  сумм}
  ...
  Count:=0;
  for T:=0 to 13 do C[T]:=0;
  for a1:=0 to 9 do                  {перебираем все}
    for a2:=0 to 9 do                {возможные a1 a2 a3}
      for a3:=0 to 9 do
      begin
        T:=a1+a2+a3;
        if T<=13                     {если сумма не пре-
                                      вышает 13, то}

        then C[T]:=C[T]+1            {нашли еще один
                                      билет}
      end;                           {с суммой T}
  for T:=0 to 13 do                  {считаем число би-
                                      летов}
        Count:=Count+C[T]*C[T];
    Count:=Count*2;                  {удваиваем сумму}
```

Сложность этого алгоритма равна 10^3.

Вариант 4. В варианте 3 мы перебирали комбинацию цифр и искали количество комбинаций с суммами $C[T]$. Сейчас мы пойдем от суммы T, и по ней будем определять, какое количество комбинаций $a_1a_2a_3$ ее имеет. Итак, $T=a_1+a_2+a_3$.

Минимальное значение, которое может принимать a_1, — это $\max\{0,\ T-18\}$. Член $T-18$ появляется из следующих соображений: пусть $a_2=a_3=9$, тогда $a_1=T-18$, но a_1 не может быть меньше 0. Максимальное значение $a_1=\min\{9,\ T\}$ (так как a_2 и a_3 неотрицательны, то $a_1\leqslant T$ и одновременно $a_1\leqslant 9$).

Для цифры a_2 аналогично получаем, что она лежит в пределах от $\max\{0,\ T-a_1-9\}$ до $\min\{9,\ T-a_1\}$.

Цифра a_3 по T, a_1 и a_2 определяется однозначно.

Получаем, что комбинаций $a_1a_2a_3$ с суммой T и с первой цифрой a_1 столько же, сколько возможных цифр a_2, а именно

$$\min\{9,\ T-a_1\}-\max\{0,\ T-a_1-9\}+1.$$

Как и в варианте 3, мы можем рассматривать диапазон сумм от 0 до 13.

```
Count:=0;
for T:=0 to 13 do
begin
  CT:=0;
  for a1:=max(0, T-18) to min(9, T) do
     CT:=CT+min(9,  T-a1)-max(0,  T-a1-
     -9)+1;
  Count:=Count+CT * CT
end;
Count:=Count * 2;
```

Сложность этого алгоритма (т. е. количество выполнений операций присваивания внутри двух вложенных циклов) есть 9^5.

2. Задача имеет очевидное решение, которое состоит в генерации всех $2N$-разрядных чисел и проверке их на требуемое свойство. Однако общее количество таких чисел равно 10^{2N} и поэтому при $N>3$ потребуется очень много времени для получения результата даже на мощном компьютере. Следовательно, необходимо разработать алгоритм, который не требует генерации всех чисел.

Обозначим через $S(k, i)$ количество k-разрядных чисел, сумма цифр которых равна i. Например, $S(2, 3)=4$, так как существует 4 двуразрядных числа (03, 12, 21, 30), сумма цифр которых равна 3. Легко заметить, что $S(1, i)=1$ при $i<10$ и $S(1, i)=0$ при $i>9$. Предположим теперь, что мы сумели вычислить значения величин $S(N, i)$ для всех i от 0 до $9N$, т. е. мы знаем, сколько существует N-разрядных чисел с суммой цифр, равной 0, 1, ..., $9N$ ($9N$ — максимальная сумма цифр в N-разрядном числе). Тогда нетрудно убедиться, что общее количество «счастливых» $2N$-разрядных чисел равно

$$P=S(N, 0)^2+S(N, 1)^2+\ldots+S(N, 9N)^2.$$

Действительно, при решении задачи 1 (вариант 3) было показано, что каждое «счастливое» $2N$-разрядное число может быть получено «склейкой» двух произвольных N-разрядных чисел с одинаковой суммой цифр.

Таким образом, необходимо уметь вычислять значения величин $S(k, i)$ для всех $k\leqslant N$, $i\leqslant 9k$. Определим способ вычисления $S(k+1, i)$ через значения величин $S(k, j)$, $j\leqslant i$. Понятно, что любое $(k+1)$-разрядное число может быть получено из k-разрядного добавлением еще одного разряда (цифры). Следовательно,

$$S(k+1, i)=S(k, i-1)+S(k, i-2)+\ldots,$$

где 1, 2, ... — возможные добавленные цифры. Ясно, что это 0, 1, ..., m, где $m=\min(9, i)$. Следовательно,

$$S(k+1, i)=S(k, i-0)+S(k, i-1)+\ldots+S(k, i-m).$$

3. Используем метод решения задачи 2.

Обозначим искомое количество n-значных чисел в десятичной системе счисления, у каждого из которых сумма цифр равна k, через $C(k, n)$. Последняя цифра числа может лежать в промежутке от 0 до 9. В соответствии с этим сумма цифр $(n-1)$-значного числа, получающаяся из n-значного числа отбрасыванием последней цифры,

может принимать одно из значений k, $k-1$, ..., $k-9$. Отсюда получаем, что

$$C(k, n)=C(k, n-1)+\ldots+C(k-9, n-1).$$

Кроме того, $C(k, 1)=1$ при $0\leqslant k\leqslant 9$ и $C(k, 1)=0$ при $k\geqslant 10$.

4. Очевидное решение задачи предполагает разложение числа $N-1$ на всевозможные суммы таким образом, чтобы каждое слагаемое из суммы не превосходило K. Очевидно, что таких разложений очень много, особенно если учитывать, что порядок слагаемых в разложении существенен, так как он соответствует различной последовательности ходов фишки. Но обратим внимание на то, что в условии задачи не требуется выписать все эти разложения, необходимо только указать их общее количество!

Обозначим через $S(i)$ количество различных путей, по которым фишка может пройти поле от начала до позиции с номером i. Предположим теперь, что для любого i от 1 до i известны значения величин $S(j)$. Задача состоит в определении правила вычисления значения $S(i+1)$, используя значения известных величин. Легко заметить, что в позицию с номером $i+1$ фишка может попасть только из позиции i, $i-1$, ..., $i-s$, где $s=\min(K, i-1)$ (мы рассматриваем только позиции с номерами от 1 до N). Следовательно,

$$S(i+1)=S(i)+S(i-1)+\ldots+S(i-K).$$

Таким образом, полагая $S(1)=1$ и вычисляя последовательно значения величин $S(2)$, ..., $S(N)$ по описанному выше правилу, получаем значение $S(N)$, которое и указывает общее количество различных путей, по которым фишка может пройти поле от начала до позиции с номером N.

5. Если покупатель отдаст все свои купюры продавцу, то понятно, что для решения исходной задачи необхо-

димо найти размер минимальной сдачи, которую продавец не может вернуть, используя любые имеющиеся теперь у него купюры C_i (его и покупателя). Для этого удобно отсортировать купюры по их достоинству в порядке неубывания.

Предположим, что продавец может вернуть любую сдачу от 1 до S, используя только первые i купюр. Для следующей $(i+1)$-й купюры достоинства C_{i+1} возможны 2 ситуации.

1) $C_{i+1}<S+2$. Тогда понятно, что продавец может вернуть любую сдачу от 1 до $Ci+1+S$, так как любая из этих сумм представима либо первыми i купюрами, либо $(i+1)$-й купюрой вместе с некоторыми из первых i купюр.

2) $C_{i+1}>S+1$. В этом случае продавец не может вернуть сдачу $S+1$.

Опишем алгоритм вычисления S.

```
S:=0;
i:=1;
нц пока (i<=N) и (C[i]<=S+1)
| S:=S+C[i];
| i:=i+1
кц
```

Если значение S не меньше суммарного количества денег покупателя, то покупатель может купить товар любой доступной ему стоимости, точно рассчитавшись за покупку. Иначе — $P=A_1+\ldots+A_{N-S}$.

6. Если $S>H_1+\ldots+H_n$, то сумму выплатить нельзя.

Если покупатель отдаст все свои купюры продавцу, то понятно, что для решения исходной задачи надо определить, может ли продавец вернуть сумму $H_1+\ldots+H_n+B_1+\ldots+B_m-S$, используя любые имеющиеся теперь у него купюры M_i (его и покупателя). Для этого удобно отсортировать купюры по их достоинству в порядке неубывания.

Пусть $P=M_1+M_2+\ldots+M_{n+m}$. Решим более общую задачу: найдем все непредставимые данными купюрами суммы на промежутке от 0 до P.

Заведем массив $A[0..P]$ натуральных чисел. Элемент $A[i]=1$, если мы можем выплатить сумму i (т. е. существует набор купюр суммарного достоинства i), и $A[i]=0$, если выплатить не можем.

Будем строить всевозможные суммы, используя последовательно 0, 1, 2, ..., N купюр.

Очевидно, что сумма из нуля купюр — это нуль, поэтому сначала $A[0]=1$.

Предположим, что мы нашли всевозможные суммы, которые можно составить, используя не более $(k-1)$ купюры $M_1, \ldots, M_{k-1}$.

Добавим еще одну купюру M_k.

Теперь мы можем выплатить следующие суммы:

1) все суммы, которые можно было составить с помощью купюр $M_1, \ldots, M_{k-1}$;

2) все суммы, которые можно было составить с помощью купюр $M_1, \ldots, M_{k-1}$, увеличенные на M_k.

Расстановка новых пометок в массиве A может выглядеть следующим образом:

```
for i:=P-M[k] downto 0 do
  if A[i]=1
  then A[i+M[k]]:=1;
```

Мы проходим по массиву из конца в начало для того, чтобы не использовать повторно образованные на текущем шаге суммы.

После выполнения $n+m$ шагов алгоритм заканчивает работу.

7. Очевидное решение задачи состоит в использовании процедуры, которая по заданным координатам (номеру строки i и номеру столбца j) элемента определяет максимальное значение элементов, расположенных в нужной части матрицы A.

Однако нетрудно заметить, что для элементов первого столбца матрицы B справедливо соотношение $B[i, 1]=A[i, 1]$, $i=1, \ldots N$. Вычисление же других столбцов можно проводить следующим образом:

$$B[i, j]=\max(A[i, j], \quad B[i-1, j-1], \quad B[i, j-1], \quad B[i+1, j-1]).$$

При этом необходимо учитывать, что индексы элементов должны находиться в пределах границ массива.

8. Для решения пункта а) задачи достаточно воспользоваться тем фактом, что для определения минимальной величины штрафа, взимаемого за проход в клетку i-й строки, достаточно знать минимальные величины штрафа, взимаемого за проход в клетки $(i-1)$-й строки, которые являются соседними рассматриваемой клетке. Поэтому алгоритм решения пункта а) следующий:

```
нц для i от 1 до n
Штраф[i, 1]:=A[i, 1]
кц
кц для i от 2 до n
| нц для j от 1 до m
| | Штраф[i, j]:=Штраф[i-1, j]+A[i, j];
| | если j>1 и Штраф[i, j]<Штраф[i-1, j-1]+A[i, j]
| | | то Штраф[i, j]:=Штраф[i-1, j-1]+A[i, j];
| | все
| | если j<m и Штраф[i, j]<Штраф[i-1, j+1]+
| | +A[i, j]
| | | то Штраф[i, j]:=Штраф[i-1, j+1]+A[i, j];
| | все
| кц
кц
```

9. а) Обозначим вершины N-угольника $x_0, \ldots, x_{N-1}$ в порядке обхода по контуру. В дальнейшем будем считать, что если в выкладках встреча-

ется вершина с индексом k, то это то же, что и вершина с индексом $k \bmod N$ (остаток от деления k на N).

Рассмотрим выпуклый L-угольник, вершинами которого являются L последовательных вершин данного N-угольника, начиная с x_p и заканчивая x_{p+L-1} в порядке обхода по контуру. У этого L-угольника ($L>1$) будем считать, что отрезок $[x_p;\ x_{p+L-1}]$ — его диагональ.

Сумму диагоналей этой фигуры обозначим $S(p,\ p+L-1)$.

Очевидно, что по условию задачи:

$S(p,\ p)=0$; $S(p,\ p+1)=0$ (у точки и отрезка нет диагоналей);

$S(p,\ p+2)=d(p,\ p+2)$ (здесь $d(p,\ p+2)$ — длина отрезка $[x_p;\ x_{p+2}]$).

Предположим, что нам известно $S(p,\ p+L-1)$ для всех $p=0,\ \ldots,\ N-1$ и $L=1,\ \ldots,\ k$.

Найдем $S(p,\ p+k)$.

Мы знаем, что диагонали разбивают $(k+1)$-угольник на треугольники и что $[x_p,\ x_{p+k}]$ — диагональ, т. е. одна из сторон какого-то треугольника. Итак, мы зафиксировали две вершины треугольника — x_p и x_{p+k}. Третьей вершиной может быть либо x_{p+1}, либо x_{p+2}, ..., либо x_{p+k-1}. Если мы считаем, что третья вершина — это x_i, то сумма длин диагоналей будет

$$d(p,\ p+k)+S(p,\ i)+S(i,\ k). \qquad (1)$$

Значения $S(p,\ i)$ и $S(i,\ k)$ были вычислены на предыдущих шагах; $d(p,\ p+k)$ — расстояние между вершинами x_p и x_{p+k}, — тоже можем вычислить.

Так как нас интересует минимальная сумма триангуляции (разбиения на треугольники), то мы ищем выражение (1) с минимальным значением:

$$S(p,\ p+k)=d(p,\ p+k)+\min\left(S(p,\ i)+S(i,\ k)\right) \qquad (2)$$

при $i=p+1,\ p+2,\ \ldots,\ k-1$.

Находим $S(p,\ p+k)$ для каждого $p=0,\ \ldots,\ N-1$.

Минимум $S(p, p+N-2)$ для $p=0, \ldots, N-1$ и даст искомую триангуляцию. Действительно, $S(p, p+N-2)$ есть стоимость разбивки фигуры после проведения $N-3$ диагоналей.

б) Алгоритм аналогичен алгоритму для случая а), только вместо формулы (2) надо использовать следующую:

$$S(p, p+k)=\min_i \max(d(p, p+k), S(p, i), S(i, k)),$$

где $S(p, p+k)$ — длина максимальной диагонали в фигуре с вершинами $x_p, x_{p+1}, \ldots, x_{p+k}$ (отрезок $[x_p, x_{p+k}]$ считается диагональю). Мы берем минимум по всем возможным разбивкам фигуры, а стоимость разбивки определяется как максимум из длины диагонали $d(p, p+k)$ и длин максимальных диагоналей $S(p, i)$ и $S(i, k)$.

10. Для $x=a, \ldots, a_m$ и $y=b_1, \ldots, b_n$, a_i и b_i — символы, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$; $d(x, y)$ можно вычислить, применяя метод динамического программирования.

Определим массив $d[0..m, 0..n]$, элементы которого

$$d[i, j]=d(a_1 \ldots a_i, b_1 \ldots b_j), \quad 0 \leqslant i \leqslant m, \quad 0 \leqslant j \leqslant n.$$

Понятно, что $d[0, j]=j$; $d[i, 0]=i$.

Очевидным образом получаем

$$d[i, j]=\min(d[i-1, j]+1, d[i, j-1]+1, d[i-1, j-1]+P_{ij}),$$

где $P_{ij}=1$, если $a_i \neq b_j$, и $P_{ij}=0$, если $a_i=b_j$. В правой части приведенного выше выражения первому элементу в min соответствует операция удаления из строки $a_1 \ldots a_{i-1}a_i$ последнего элемента a_i, после чего за $d[i-1, j]$ операций строка $a_1 \ldots a_{i-1}$ преобразуется в строку $b_1 \ldots b_j$, второму элементу — операция вставки символа b_j в конец строки $b_1 \ldots b_{j-1}$, полученной за $d[i, j-1]$ операций из строки $a_1 \ldots a_i$; третьему — контекстная замена a_i на b_j, замена осуществляется в случае $a_i \neq b_j$ (тогда $P_{ij}=1$)

и не происходит при совпадении a_i и b_j. Величина $d[m, n]$ соответствует минимальному количеству операций, которые требуются для преобразований строки x в строку y. Алгоритм может быть записан так:

```
for i:=1 to m do
  d[i, 0]:=i;
for j:=1 to n do
  d[0, j]:=j;
for i:=1 to m do
  for j:=1 to n do
    d[i, j]=min(d[i-1, j]+1, d[i, j-1]+1, d[i-1,
      j-1]+Pij);
```

11. Пусть строка x состоит из цифр 0 и 1 и имеет длину N, а строка y (из символов A и B) — длину M.

Заведем матрицу A размера $N \times M$, при этом строки матрицы помечаются i-й цифрой строки x, а столбец — j-м символом строки y.

Возьмем в качестве примера x=«00110», y= =«*AAAABBAA*».

Первая цифра строки x (цифра 0) может быть преобразована в одну из последовательностей букв «*A*», «*AA*», «*AAA*», «*AAAA*», являющихся префиксами строки y. Заносим символ «*» в те столбцы первой строки, буквы-пометки которых соответствуют последним буквам возможных последовательностей.

Таким образом, помечаются элементы $A[1, 1]$, $A[1, 2]$, $A[1, 3]$ и $A[1, 4]$.

На каждом следующем шаге алгоритма будем преобразовывать очередную i-ю цифру строки x, которой соответствует i-я строка матрицы A.

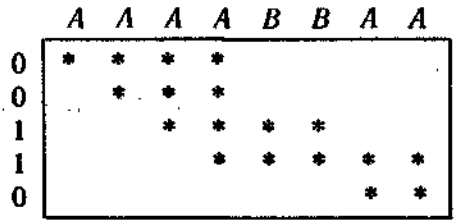
Находим «*» в предыдущей строке при просмотре ее слева направо (этому столбцу соответствует последняя буква в какой-то из непустых последовательностей букв, порожденных на предыдущем шаге). Если текущую цифру можно преобразовать в последовательность букв,

помечающих следующие за найденным столбцы, то в этих столбцах в рассматриваемой строке ставим «*».

Далее от места над последней помеченной ячейкой ищем в предыдущей строке «*» и, когда находим, повторяем указанные выше операции.

Эти действия проводим далее для $i=2, \ldots, N$.

Вид матрицы после N шагов:



Если после N шагов в позиции (N, M) стоит x, то строки можно преобразовать друг в друга.

Замечание. Можно обойтись и одномерным массивом. В самом деле, при заполнении следующей строки мы обращаемся только к элементам предыдущей строки, к каждому — по одному разу.

12. Определим через $F[i, j]$ минимальное число операций, которое требуется для перемножения группы матриц с номерами от i до j включительно.

Ясно, что $F[i, i]=0$.

Перемножение группы матриц с номерами от i до j может производиться различными способами, а именно, для некоторого выбранного k сначала перемножаются наилучшим способом матрицы с номерами от i до k, затем матрицы от $(k+1)$-й до j-й, и, наконец, перемножаются получившиеся матрицы. Понятно, что k может быть величиной от i до $j-1$. Учитывая требование получить наилучший результат, величина $F[i, j]$ определяется как

$$F[i, j]=max(F[i, k]+F[k+1, j]+n[i]\cdot n[k+1]\cdot m[j]),$$

где k может быть величиной от i до $j-1$, а $n[i]$, $n[k+1]$, $m[j]$ определяют размеры матриц, получившихся при перемножении в группах.

```
нц для i от 1 до s выполнять
| F[i, i]:=0;
кц
нц для p от 1 до s—1 выполнять
| нц для i от 1 до s—p выполнять
| | Kol:=бесконечность;
| | j:=i+p;
| | нц для k от i до j—1 выполнять
| | | если Kol>F[i, k]+F[k+1, j]+n[i]*n[k+
| | | +1]*m[j]
| | | то Kol:=F[i, k]+F[k+1, j]+n[i]*n[k+1]*m[j];
| | | все
| | кц
| | F[i, j]:=Kol;
| кц
кц
```

В ячейке $F[1, s]$ после завершения работы алгоритма будет находиться искомое минимальное число операций.

Подумайте, каким образом можно в произведении матриц $A_1, \ldots, A_s$ расставить скобки так, чтобы они определяли оптимальный порядок умножения.

13. а) Рассмотрим сначала наиболее очевидный, но, как это обычно бывает, наименее эффективный (очень медленный) алгоритм. Будем генерировать все подпоследовательности данной N-элементной последовательности и для каждой из подпоследовательностей проверять, является ли она строго возрастающей и максимальной по длине.

Будем генерировать числа от 0 до 2^n-1, находить их двоичное представление и формировать подпоследовательность из элементов массива A с индексами, соответствующими единичным битам в этом представлении.

Всего существует 2^n таких подпоследовательностей, поэтому даже при небольших n результата придется ждать очень долго.

Предположим, что при генерации подпоследовательностей мы нашли k-элементную строго возрастающую подпоследовательность. В дальнейшем имеет смысл рассматривать только подпоследовательности, состоящие из более чем k элементов (подумайте почему).

Рассмотрим исходную n-элементную последовательность. Если она не является искомой, то будем генерировать $(N-1)$-элементные подпоследовательности. Если и среди них не найдено решение, то будем рассматривать подпоследовательности из $(N-2)$ элементов и т. д.

В худшем случае (каком?) придется анализировать порядка 2^n вариантов.

Для более быстрого решения этой задачи можно применить дихотомию по k — количеству элементов в подпоследовательности (как?).

Рассмотрим другое, более эффективное решение этой задачи. Заведем массивы A, B и C длины N: массив $A[1..N]$ используется для хранения чисел исходной последовательности; элемент $B[i]$ — значение длины максимальной возрастающей подпоследовательности, последний элемент которой $A[i]$; величина $C[i]$ есть индекс элемента, предшествующего элементу $A[i]$ в этой максимальной подпоследовательности ($A[i]=0$, если предшествующего элемента нет).

Если $N=1$, то $A[1]$ и есть искомая подпоследовательность. При этом $B[1]=1$, $C[1]=0$. Предположим, что мы заполнили массивы B и C от начала и до элемента $i-1$. Попытаемся получить элементы $B[i]$ и $C[i]$. Для этого будем просматривать массив A от 1 до $(i-1)$-го элемента и искать такой индекс k, для которого одновременно выполняются следующие условия:

1) $A[k]<A[i]$,

2) $B[k]$ максимально.

Очевидно, что максимальную по длине подпоследовательность, заканчивающуюся элементом $A[i]$, можно получить, приписав этот элемент к максимальной подпос-

ледовательности с последним элементом $A[k]$. Следовательно, $B[i]=B[k]+1$ и $C[i]=k$.

Пусть мы обработали все N элементов массива A и нашли максимальный элемент массива B. Пусть это элемент с индексом *IndexMax*. По построению это длина максимальной подпоследовательности.

Получить искомую подпоследовательность можно следующим образом. Пусть j — индекс текущего элемента подпоследовательности, распечатываемой с конца. Сначала полагаем $j:=IndexMax$ и печатаем элемент $A[j]$, который является последним. Предшествующий ему в последовательности элемент имеет индекс $C[j]$, поэтому индекс следующего с конца элемента определяется как $j:=C[j]$. Описанные действия повторяем пока j не станет равным 0 (т. е. пока не дойдем до начала последовательности).

Запись алгоритма на языке Pascal:

```
for i:=2 to N do B[i]:=0;
C[1]:=0; B[1]:=1; Max:=1; IndexMax:=1;
for i:=2 to N do
  for k:=1 to i-1 do
      if (A[k]<A[i]) and (B[i]<B[k]+1) then
      begin
        C[i]:=k;
        B[i]:=B[k]+1;
        if B[i]>Max then
        begin
          Max:=B[i]
          IndexMax:=i
        end;
      end;
j:=IndexMax;
while j<>0 do
begin
  writeln (A[j]);
  j:=C[j]
end;
```

В программе переменная *Max* используется для хранения длины текущей максимальной подпоследовательности.

В этой задаче элемент массива $C[i]$ содержит ссылку на элемент, предшествующий $A[i]$ в подпоследовательности максимальной длины. Такая ссылочная структура данных называется **однонаправленным** списком. Если у элемента есть ссылка как на предыдущий, так и на последующий элемент, то список — **двунаправленный** (его можно реализовать, если использовать не один массив ссылок, а два).

В рассмотренной задаче оптимальные значения хранятся в массиве *B*.

Получить более эффективное решение можно, если на каждом шаге хранить не все полученные ранее оптимальные значения и соответствующие подпоследовательности, а только наиболее перспективные из них.

Пусть $K(L, i)$ обозначает множество возрастающих подпоследовательностей длины L, которые составлены из элементов с номерами от 1 до $i-1$. Из двух подпоследовательностей длины L более перспективной будет та, у которой величина последнего элемента меньше, так как ее может продолжить большее число элементов. Пусть $SP(L, i)$ — самая перспективная подпоследовательность длины L (с минимальным по величине последним элементом), а $S(i)$ — множество всех подпоследовательностей $SP(L, i)$ при всевозможных L. В $S(i)$ содержится не более $i-1$ подпоследовательностей (с длинами $1, \ldots, i-1$).

Пусть мы знаем $S(i)$. Для того чтобы определить, какие подпоследовательности может продолжать i-й элемент последовательности A, достаточно знать последние элементы перспективных подпоследовательностей длины $1, 2, \ldots, N$, индексы которых будут храниться в массиве *Ind*.

Последний элемент перспективной подпоследовательности длины p строго меньше последнего элемента перс-

пективной подпоследовательности длины $p+1$ (объясните почему). Поэтому i-й элемент должен продолжить подпоследовательность максимальной длины, последний элемент которой меньше i-го элемента.

Учитывая упорядоченность последних элементов перспективных подпоследовательностей, поиск можно сделать методом половинного деления (дихотомией), используя массив *Ind*.

При присоединении i-го элемента к такой подпоследовательности длины p ее длина увеличивается на 1, а последним элементом становится $A[i]$. При этом множество $S(i+1)$ совпадает с $S(i)$, за исключением подпоследовательности $SP(p+1, i+1)$, полученной добавлением i-го элемента к подпоследовательности $SP(p, i)$. При хранении подпоследовательности для каждого элемента удобно хранить номер предшествующего ему элемента.

б) Для каждого индекса i найдем подпоследовательность максимальной длины с разрывом в $A[i]$. Будем искать максимальную по длине подпоследовательность, заканчивающуюся в элементе $A[i]$, и максимальную по длине подпоследовательность, начинающуюся в нем (для этого будем просматривать массив A не слева направо, а справа налево).

в) Заведем массив $C[0..m+1, 1..N]$. В нем i-я строка будет хранить информацию о последовательностях с $i-1$ разрывом (нулевая строка — фиктивная); j-й элемент в этой строке есть длина самой длинной подпоследовательности элементов «хвоста» массива A (от j-го элемента до n-го), начинающейся в j-й позиции и имеющей не более $i-1$ разрывов.

Алгоритм:

1. Заполнить нулевую строку нулями (чтобы можно было заполнить первую строку по общему алгоритму).

2. Для каждой строки i от 1 до $m+1$ выполнить следующие действия:

2.1. Для j-го элемента массива A (j изменяется от N до 1) найти максимальную по длине подпоследовательность, которую можно присоединить к этому элементу так, чтобы получить подпоследовательность максимальной длины с не более чем $i-1$ разрывом. Для этого:

2.1.1. найти элемент $A[k]$ последовательности A, больший $A[j]$ и стоящий в массиве A правее j-го элемента и с максимальным $C[i, k]$;

2.1.2. просмотреть элементы $(i-1)$-й строки матрицы C, начиная с $(j+1)$-го и до конца; найти максимальный из них, пусть это $C[i-1, s]$;

2.1.3. сравнить $C[i-1, s]$ с $C[i, k]$, больший из них (обозначим его $C[row, col]$), увеличенный на 1, запомнить в $C[i, j]$; это и будет длина максимальной подпоследовательности, начинающейся в позиции j, с не более чем $i-1$ разрывом;

2.1.4. запомнить индексы row и col элемента массива C, предшествующего $C[i, j]$, как элементы $X[i, j]$ и $Y[i, j]$ соответственно.

После окончания цикла максимальный элемент $(m+1)$-й строки матрицы C и есть максимальная длина возрастающей подпоследовательности с m разрывами. Выписать всю подпоследовательность в обратном порядке можно следующим образом: для каждого элемента подпоследовательности в массивах X и Y хранится информация о предшественнике. Мы, начиная с максимального элемента $(m+1)$-й строки матрицы C, восстанавливаем всю подпоследовательность.

Обоснование алгоритма.

Пусть известны $C[i-1, j]$ для всех j от 1 до N и для некоторого i, а также $C[i, k]$ для k от $j+1$ до N. Мы хотим вычислить $C[i, j]$.

Для j-го элемента массива A существует максималь-

ная по длине подпоследовательность с не более чем $i-1$ разрывом, начинающаяся с $A[j]$. Второй элемент (обозначим его $A[k]$) этой максимальной подпоследовательности (если он, конечно, есть) может быть:

1) больше $A[j]$; тогда находим его среди элементов, обладающих следующими свойствами:

а) $k>j$;

б) $C[i, k]$ максимальный (т. е. мы присоединяем к $A[j]$ максимальную по длине подпоследовательность с не более чем $i-1$ разрывом, формируя подпоследовательность опять не более чем с $i-1$ разрывом);

2) меньше или равный $A[j]$; тогда ищем его среди элементов, обладающих следующими свойствами:

а) $k>j$;

б) $C[i-1, k]$ максимальный (т. е. присоединяем максимальную подпоследовательность с не более чем $i-2$ разрывами, формируя подпоследовательность с не более чем $i-1$ разрывом).

Полученная подпоследовательность имеет максимальную длину, так как длина подпоследовательности, которая начинается с $A[k]$, — максимальна. Упоминавшиеся выше индексы *row* и *col*, которые запоминаются в $X[i, j]$ и $Y[i, j]$ соответственно, обозначают следующее: *col* — индекс следующего за $A[j]$ элемента в максимальной по длине подпоследовательности, начинающейся в позиции j и имеющей не более $i-1$ разрывов; $row-1$ — максимальное количество разрывов в подпоследовательности, начинающейся в $A[col]$.

14. Для решения задачи элементы массива удобно упорядочить по абсолютной величине (в порядке неубывания). Если в массиве есть элементы, равные 0, то один из них и будет последним элементом искомой последовательности, поэтому их можно игнорировать. Пусть K_i обозначает максимальное количество элементов, которое может находиться в некоторой последовательности с тре-

буемым свойством, последним элементом которой является i-й элемент.

Понятно, что наименьшему по абсолютной величине элементу не может предшествовать ни один элемент, поэтому $K_p=0$, где p — индекс первого ненулевого элемента.

Для каждого следующего элемента с номером j, $j=p+1, \ldots, N$, необходимо определить максимальное количество элементов, которое может предшествовать рассматриваемому элементу, с учетом требуемого свойства. Понятно, что это количество есть максимум из величин $K_{p_1}, K_{p_2}, \ldots, K_{p_m}$, где элементы с номерами $p_1, p_2, \ldots, p_m$, $p \leqslant p_1 < p_2 < \ldots < p_m < j$ являются делителями элемента с номером j. Поэтому мы имеем рекуррентную формулу для вычисления K_j:

$$K_j = max(K_{p_1}, K_{p_2}, \ldots, K_{pm}) + 1.$$

Значение K_N, вычисленное по описанному выше правилу, и определяет максимальное количество элементов, которое может находиться в некоторой последовательности с требуемым свойством (без учета возможного нуля в конце последовательности).

Для того чтобы установить, какие элементы образуют максимальную последовательность, достаточно для каждого номера j помнить тот номер из $p_1, p_2, \ldots, p_m$, на котором достигается максимум для чисел $K_{p_1}, K_{p_2}, \ldots, K_{p_m}$. Эти номера можно определять параллельно с вычислением значения $K(j)$ в некотором массиве, например ПРЕДОК. Используя эту информацию, легко определить номера элементов последовательности, проходя от элемента i с максимальным значением K_i к элементу, который ему предшествует (ПРЕДОК (i)), до тех пор, пока не придем к первому элементу f последовательности (ПРЕДОК (f))$=0$.

15. Очевидно, что параллелепипеды можно повернуть так, чтобы размеры ребер каждого параллелепипеда шли в неубывающем порядке. Зафиксируем этот поря-

док. Вложение параллелепипеда B в параллелепипед C возможно только тогда, когда для двух параллелепипедов $B(b(1), \ldots, b(n))$ и $C(c(1), \ldots, c(n))$ выполняются неравенства $b(k) \leqslant c(k)$, $k=1, \ldots, n$.

16. Метод решения этой задачи аналогичен использованному при решении задачи о нахождении максимальной по длине возрастающей подпоследовательности.

Пусть a, b, c — натуральные числа.

Будем рассматривать только случай $a>c$, так как в случае $a<b$ задача неразрешима, а в случае $b \leqslant a \leqslant c$ сразу получаем, что $k=1$ и $x[1]=a$.

Обозначим через L множество делителей числа a, лежащих на отрезке $[b, c]$. Количество всех делителей числа a не превышает $2\sqrt{a}$ (если $a=f \cdot g$, то $f \leqslant \sqrt{a}$, $g \geqslant a$, всего разных f может быть не более a, столько же и разных g). Пусть L_1 — минимальный элемент из L, а L_2 — максимальный. Пусть в массиве $S[1..p]$ первый элемент равен единице, а все остальные — делители числа a, не меньшие b, записанные в порядке возрастания. Из утверждения следует, что $p \leqslant a+2$.

Будем искать минимальную по длине подпоследовательность элементов массива S, которая начинается единицей, заканчивается a, каждый элемент которой делится на предыдущий, причем частное принадлежит множеству L.

В случае, если a, b, c, $x[i]$ — целые, в массив S помещаем в порядке возрастания модуля все делители числа a, начиная с минимального элемента в L. Далее — аналогично.

17. Задачу можно переформулировать следующим образом.

Последовательность из n вагонов, занумерованных от 1 до n, необходимо разбить на не более чем m подпоследовательностей, в каждой из которых номера вагонов возрастают.

Одним из самых простых алгоритмов сортировки вагонов является алгоритм, основывающийся на следующем правиле.

1) Помещаем очередной вагон (номер которого k) на путь с минимально возможным номером, при условии, что последний вагон, стоящий на этом пути, имеет номер меньше k.
2) Если все вагоны состава будут таким образом расположены на станции, то, очевидно, что вагон № 1 на каком-то пути будет самым правым и его можно подать на выход. Затем вагон № 2 также может быть подан на выход, так как не может стоять на пути за вагоном с номером, большим 2, и т. д.
3) Если же какой-то вагон нельзя поместить ни на один путь на станции, то это значит, что все пути «перекрыты» вагонами с большими номерами, причем последние вагоны на этих путях расположены в порядке убывания. Вместе с последним вагоном, не нашедшим себе места, эти m вагонов в исходной последовательности составляют убывающую подпоследовательность длины $m+1$.

Отметим, что если можно отсортировать последовательность вагонов, то можно отсортировать также и любую их подпоследовательность, и наоборот. Поэтому из существования убывающей подпоследовательности длины $m+1$ следует, что исходную последовательность отсортировать нельзя.

18. Можно, конечно, число A умножить само на себя $n-1$ раз, но для этого надо выполнить $n-1$ операцию умножения. Рассмотрим метод, требующий меньшего числа умножений (он, однако, не всегда дает минимальное число умножений).

Если n — четное ($n=2m$), то будем вычислять A^n, используя тождество

$$A^n=(A^m)^2,$$

если же $n=2m+1$, то $A^n=(A^m)^2\cdot A$.

Таким образом, возведение A в 13-ю степень будет выглядеть следующим образом:

$$A^{13}=(A^6)^2\cdot A=((A^3)^2)^2\cdot A=((A\cdot A\cdot A)^2)^2\cdot A,$$

и вычисление требует 5 операций умножения.

Примечание. Используя данный метод, для возведения числа в степень n потребуется порядка $\log_2 n$ операций умножения.

Программа на языке Pascal может выглядеть так:

```
var A, N: integer;
function power (N: integer): integer;
begin
  if N>1 then
       if odd (N) tnen                    {N нечетно?}
          power:=SQR (power (N  div  2)) * A
       else power:=SQR (power (N div 2))
  else power:=A
end;
begin
  read (A, N);
  writeln (power (N));
end;
```

Можно ту же самую идею реализовать и по-другому (далее мы приводим выдержку из книги Д. Кнута «Искусство программирования для ЭВМ», т. 2, с. 482):

«Запишем n в двоичной системе счисления и заменим в этой записи каждую цифру 1 парой букв SX, а каждую цифру 0 — буквой S, после чего вычеркнем крайнюю левую пару букв SX. Результат, читаемый слева направо, превращается в правило вычисления x^n, если букву S интерпретировать как операцию возведения в квадрат, а букву X — как операцию умножения на x.

Например, если $n=23$, то его двоичным представлением будет 10111; строим последовательность $SX\ S\ SX\ SX\ SX$, удаляем из нее начальную пару SX и в итоге получаем следующее правило вычисления: $S\ SX\ SX\ SX$. Согласно этому правилу, мы должны «возвести x в квадрат, затем снова возвести в квадрат, затем умножить на x, возвести в квадрат, умножить на x, возвести в квадрат и, наконец, умножить на x»; при этом мы последовательно вычисляем x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}.

Этот «бинарный метод» легко обосновать, рассмотрев последовательность получаемых в ходе вычисления пока-

зателей: если S интепретировать как операцию умножения на 2, а X — как операцию прибавления 1 и если начать с 1, а не с x, то наше правило дает нам в соответствии со свойствами двоичной системы счисления число n».

Приведенный метод не дает минимального числа операций умножения. Для вычисления x^{23} нам, по изложенному выше методу, потребуется 7 операций умножения. В действительности, их необходимо только 6:

$$x \to x^2 \to x^3 \to x^5 \to x^{10} \to x^{20} \to x^{23}.$$

Алгоритм нахождения минимального числа операций (кроме полного перебора) сейчас неизвестен.

19. Пусть z — массив из N элементов, y — из M. Положим $i=1$ и $j=1$. Берем элемент $z[i]$ и ищем минимальное k, $j \leqslant k \leqslant M$, такое, что $y[k]=z[i]$ (мы находим очередной совпадающий символ в строках z и y). Полагаем $i:=i+1$ и $j:=k+1$. Повторяем поиск элемента $z[i]$ в оставшейся части последовательности y. Условия окончания поиска:

а) если i стало больше N (т. е. все элементы массива z являются подпоследовательностью элементов y), тогда z можно получить вычеркиванием элементов из y;

б) если в оставшейся части последовательности y не найдено элемента, совпадающего с очередным $z[i]$, то z из y получить нельзя.

20. Пусть $x=(x_1, x_2, \ldots, x_m)$, $y=(y_1, y_2, \ldots, y_n)$.

Заведем матрицу $A[0..m, 0..n]$. Элемент $A[i, j]$ будет длиной максимальной общей подпоследовательности $(x_1, \ldots, x_i)$ и $(y_1, \ldots, y_j)$. Сначала $A[i, 0]=A[0, j]=0$, $i=0, \ldots, m$, $j=0, \ldots, n$.

Пусть $x_i=y_j$, тогда требуется увеличить длину максимальной общей подпоследовательности $(x_1, \ldots, x_{i-1})$ и $(y_1, \ldots, y_{j-1})$ на 1:

$$A[i, j]=A[i-1, j-1]+1, \text{ если } x_i=y_j.$$

В случае, если $x_i \neq y_j$, то, очевидно,

$$A[i, j] = max\{A[i-1, j], A[i, j-1], A[i-1, j-1]\},$$

но так как всегда $A[i-1, j-1] \leqslant A[i, j-1]$, то

$$A[i, j] = max\{A[i-1, j], A[i, j-1]\}.$$

Величина $A[m, n]$ и дает длину максимальной общей подпоследовательности. Найдем саму подпоследовательность. Пусть $A[m, n] = d$. Двигаясь по последней строке справа налево, ищем самый левый элемент в этой строке со значением d. Двигаемся от него вверх по столбцу в поиске элемента столбца с минимальным первым индексом и значением d. Пусть это $A[i, j]$. Тогда элемент $A[i-1, j-1]$ равен $d-1$, а x_i и y_j — это последние общие совпадающие элементы в x и y.

Начиная от элемента $A[i-1, j-1]$, повторяем, как было описано выше, движение влево и вверх по матрице, находим предпоследний совпадающий элемент в x и y и т. д.

Программа:

```
for i:=0 to m do A[i, 0]:=0;
for j:=0 to n do A[0, j]:=0;
for i:=1 to m do
  for j:=1 to n do
      if x[i]=y[i] then
        A[i, j]:=A[i-1, j-1]+1
       else A[i, j]:=max(A[i-1, j], A[i, j-1]);
writeln('Длина последовательности=', A[m, n]);
d:=A[m, n]; i:=m; j:=n;
while (d<>0) do
begin
  while A[i, j-1]=d do j:=j-1;
  while A[i-1, j]=d do i:=i-1;
  write('Элемент последовательности номер', d,
  'есть', x[i]);
  i:=i-1; j:=j-1; d:=d-1; {переход к поиску
  предшествую-}
     {щего элемента в последовательности}
end;
```

21. В последовательностях x и y избавляемся от незначащих нулей. Если хоть одна из последовательностей стала пустой, то $z=0$. Если последовательности не пустые, то крайние левые цифры и в x, и в y равны 1. Для полученных последовательностей используем алгоритм задачи 20 (для получения максимального z необходимо, чтобы старшая цифра z была 1 и двоичная запись z имела максимальную длину), но при этом для каждого $A[i, j]$ — длины последовательности, необходимо хранить добавочно и саму последовательность; при присвоении значения $A[i, j]$ одновременно будем запоминать и последовательность максимальной длины. Если таких несколько, то берем из них последовательность с максимальным значением. Поэтому алгоритм задачи 20 запишется следующим образом.

Пусть $S[0..m, 0..n]$ — массив строк. В $S[i, j]$ будет храниться подпоследовательность, длина которой $A[i, j]$.

```
for i:=0 to m do A[i, 0]:=0;
for j:=0 to n do A[j, 0]:=0;
for i:=0 to m do
  for j:=0 to n do
          S[i, j]:='';
for i:=1 to m do
  for j:=1 to n do
  begin
     if x[i]=y[j] then
     begin
       A[i; j]:=A[i-1, j-1]+1;
       S[i, j]:=S[i-1, j-1]+x[i];
     end;
     A[i, j]:=max(A[i, j], A[i-1, j], A[i, j-1]);
     S[i, j]:=max(S[i, j], S[i-1, j], S[i, j-1]);
end;
write(A[m, n], '— длина', S[m, n]);
```

Глава 5. ЗАДАЧИ КОМБИНАТОРИКИ

С задачами, в которых приходится выбирать те или иные предметы, располагать их в определенном порядке и отыскивать среди всевозможных расположений наилучшее, люди сталкиваются постоянно. Например, начальник цеха распределяет несколько видов работ между имеющимися станками, агроном размещает посевы сельскохозяйственных культур на нескольких полях, завуч школы составляет расписание уроков. При решении такого вида задач возникли понятия об упорядочении и группировании объектов. С этими понятиями и работает наука комбинаторика.

Комбинаторика — это область математики, в которой изучаются вопросы о том, сколько различных комбинаций, подчиненных тем или иным условиям, можно составить из заданных объектов.

Иногда количество вариантов, которые надо проанализировать при решении комбинаторной задачи, очень велико. В этом случае единственной возможностью получения результата за допустимое время, а не через столетия, является использование компьютера.

Как и любая другая наука, комбинаторика имеет свою терминологию. Основным понятием комбинаторики является ***комбинаторный объект,*** или ***соединение.*** При выборе m элементов из n различных элементов принято говорить, что они образуют соединение из n элементов по m.

В зависимости от того, имеет ли значение порядок элементов в соединении или нет, а также от того, входят в соединение все n элементов или только часть их, различают три вида соединений. Это — *перестановки, размещения* и *сочетания.*

§ 1. СОЕДИНЕНИЯ

1.1. Перестановки

Соединения, каждое из которых содержит n различных элементов, взятых в определенном порядке, называются ***перестановками*** из n элементов. Следует отметить, что порядок элементов в перестановке существенен и в образовании перестановки участвуют все n элементов ($m=n$).

Пример. Выпишем все перестановки из элементов a, b, c:

$$abc,\ acb,\ bac,\ bca,\ cab,\ cba.$$

Количество всех способов, которыми можно переставить n различных предметов, расположенных на n различных местах, принято обозначать P_n (читается «число перестановок из n»).

Найдем P_n. На первое из имеющихся n мест предмет может быть выбран n способами, на второе место — $(n-1)$ способом, на третье место — $(n-2)$ способами и т. д. На предпоследнее место предмет выбирается из двух оставшихся, а для последнего места выбор предмета единственен. Общее количество способов будет равно произведению $n(n-1)(n-2)\cdot\ldots\cdot 2\cdot 1$. Такое произведение называется ***факториалом*** числа n и обозначается $n!$ Таким образом, $P_n=n!$

Пример. Сколько всего шестизначных четных чисел можно составить из цифр 1, 3, 4, 5, 7 и 9, если в каждом из этих чисел ни одна цифра не повторяется?

Последняя цифра четного числа должна быть четной, поэтому в данном случае последней цифрой числа может быть только цифра 4. Оставшиеся пять цифр могут стоять на оставшихся пяти местах в любом порядке. Поэтому количество способов их расстановки равно $P_5=5!=120$.

Иногда требуется не просто подсчитать количество перестановок из n элементов, но и найти каждую из них.

Существует несколько алгоритмов генерации всех перестановок из n элементов. Они различаются порядком получения перестановок.

Рассмотрим один из хорошо известных алгоритмов, в процессе исполнения которого перестановки n чисел располагаются лексикографически (в словарном порядке). Это значит, что перестановки сравниваются слева направо поэлементно и большей из них является та, у которой раньше встретился элемент, больший соответствующего ему элемента во второй перестановке. (Например, если $S=(3, 5, 4, 6, 7)$, а $L=(3, 5, 6, 4, 7)$, то $S<L$, так как $S_3<L_3$.)

Опишем алгоритм для $n=5$, отчего рассуждения не утратят общности. В дальнейшем, для удобства, будем работать не с самими элементами, а с их номерами (от 1 до n).

Принцип работы алгоритма разъясним на примере. Допустим, необходимо воспроизвести все перестановки чисел 1, 2, 3, 4, 5. Первой перестановкой считаем перестановку (1, 2, 3, 4, 5). Последней перестановкой будет (5, 4, 3, 2, 1). Элементы перестановки будем хранить в массиве.

Предположим, что на некотором шаге работы алгоритма получена перестановка P:

$$P=(3, 4, 5, 2, 1)=(p_1, p_2, p_3, p_4, p_5).$$

Для того чтобы определить непосредственно следующую за ней перестановку, необходимо выполнить шаги:

Шаг 1. Будем просматривать данную перестановку справа налево и следить за тем, чтобы каждый следующий элемент перестановки (элемент массива с большим номером) был больше предыдущего (т. е. элемента массива с меньшим номером), и остановимся сразу же, как только это правило нарушится ($1<2<5$, а $5>4$). Место остановки указано подчеркиванием:

$$(3, \underline{4}, 5, 2, 1).$$

Шаг 2. Затем вновь просматриваем пройденный путь

(справа налево) до тех пор, пока не дойдем до первого числа, которое уже больше отмеченного. Место второй остановки отмечено двойным подчеркиванием.

$$(3,\ \underline{4},\ \underline{\underline{5}},\ 2,\ 1).$$

Шаг 3. Поменяем местами отмеченные числа:

$$(3,\ \underline{\underline{5}},\ \underline{4},\ 2,\ 1).$$

Шаг 4. В части массива, расположенной справа от двойного подчеркивания, отсортируем все числа в порядке возрастания. Так как до сих пор они были упорядочены по убыванию, то это легко сделать, записав в обратном порядке указанную часть массива. Получим новую перестановку, которую обозначим $Q=(q_1, q_2, q_3, q_4, q_5)$:

$$Q=(3,\ 5,\ 1,\ 2,\ 4).$$

Это и есть та перестановка, которая непосредственно следует за P в лексикографическом порядке.

Примечание. Действительно, $P<Q$, так как $p_2=4$, $q_2=5$. Пусть существует такая перестановка R, что $P<R<Q$. Тогда $p_1=r_1=q_1$. По построению q_2 — наименьшее число в множестве $\{1, 2, 4, 5\}$ (это множество содержит элементы перестановки Q без q_1), такое, что $q_2>p_2$. Поэтому для r_2 верно одно из двух равенств: $r_2=p_2$ или $r_2=q_2$. Но так как в P элементы, начиная с p_3, убывают, то из $P<R$ следует, что если $p_2=r_2$, то $P=R$. Аналогично, так как в Q элементы, начиная с q_3, возрастают, то из $R<Q$ следует, что если $r_2=q_2$, то и $R=Q$. Следовательно, перестановки R не существует.

В массиве P будем хранить номера элементов. Вначале в массиве хранятся числа от 1 до n, расположенные в порядке возрастания так, что каждый элемент равен своему номеру. Нулевой элемент фиктивный, он используется для проверки окончания работы. Генерация перестановок будет закончена, когда нулевой элемент станет отличным от нуля.

Запишем алгоритм генерации перестановок:

```
нц для i от 0 до n
| p[i]:=i                    :инициализация массива P
```

```
кц
нц пока p[0]=0
| нц для i от 1 до n
| | p[i]:=p[i]                    :вывод очередной переста-
|                                 :новки на экран
| кц
| j:=N
| нц пока p[j-1]>p[j]             :поиск (справа налево)
|                                 :элемента,
| | j:=j-1                        :большего предшествую-
|                                 :щего ему (ШАГ 1)
| кц
| k:=N                                                  (5.1)
| нц пока p[j-1]>p[k]             :поиск числа, большего
| | k:=k-1                        :отмеченного (ШАГ 2)
| кц
| d:=p[j-1]                       :перестановка двух чисел,
| p[j-1]:=p[k]                    :найденных выше (ШАГ 3)
| p[k]:=d
| нц для i от j до n              :запись части массива в
|                                 :обратном порядке
| | r[i]:=p[N-(i-j)]              :во вспомогательный мас-
|                                 :сив r
| кц
| нц для i от j до n              :перенос элементов из мас-
|                                 :сива r
| | p[i]:=r[i]                    :в исходный массив
|                                 :(ШАГ 4)
| кц
кц
```

Вопросы для повторения

1. Что такое перестановки?
2. Чему равно число перестановок из 4 элементов? из 6 элементов?
3. Как сгенерировать все перестановки из 4 элементов в лексикографическом порядке?
4. Укажите перестановку, лексикографически следующую за (5, 4, 6, 2, 1, 3).

1.2. Сочетания

Соединения, отличающиеся друг от друга, по крайней мере, одним элементом, каждое из которых содержит m элементов, взятых из n различных элементов, называются ***сочетаниями*** (комбинациями или выборками) ***из n элементов по m.*** Порядок следования элементов не учитывается.

Пример. Выпишем все сочетания из элементов a, b, c по два:

$$ab,\ ac,\ bc.$$

Количество способов, которыми можно выбрать m элементов из n, принято обозначать $C(n,\ m)$ или C_n^m (читается «число сочетаний из n по m»). Значение величины $C(n,\ m)$ вычисляется по формуле

$$C(n,\ m)=\frac{n!}{m!\,(n-m)!}. \qquad (1)$$

(Доказательство формулы будет приведено ниже.) Отметим, что

$$C(n,\ m)=\frac{n!}{m!\,(n-m)!}=C(n,\ n-m). \qquad (2)$$

Пример. Сколькими способами можно из 10 человек выбрать команду для игры в городки? Команда состоит из 4 человек.

Количество способов равно числу сочетаний из 10 по 4.

$$C(10,\ 4)=\frac{10!}{4!\,(10-4)!}=\frac{7\cdot 8\cdot 9\cdot 10}{1\cdot 2\cdot 3\cdot 4}=210.$$

Для решения некоторых задач требуется не только подсчитать число сочетаний, но и найти каждое из них. Приведем алгоритм генерации всех сочетаний из n элементов по m. Так же, как и в алгоритме генерации перестановок, будем работать не с самими элементами, а с их номерами 1, 2, ..., n.

Так как порядок элементов в сочетании не учитывается, то получать сочетания удобно в порядке возрастания индексов используемых на данном шаге элементов (общее их число есть m). Текущее сочетание будем хранить в массиве B. В качестве начальной конфигурации возьмем следующую: (1, 2, ..., m), для которой $B[j]=j$, $j=1, \ldots, m$. Сочетания будем получать в возрастающем лексикографическом порядке, поэтому последним сочетанием будет $(n-m+1, n-m+2, \ldots, n-1, n)$. Для каждого элемента последнего сочетания выполняется условие $B[j]=n-m+j$. Для всех остальных сочетаний это равенство будет нарушено хотя бы для одного элемента.

Для генерации очередного сочетания найдем элемент $B[j]$ с максимальным индексом j, такой, что выполняется неравенство

$$B[j]<n-m+j. \qquad (*)$$

При этом будем просматривать текущее сочетание справа налево. Затем увеличим это $B[j]$ на 1, а для всех $k<j$ полагаем $B[k]=B(k-1)+1$. Если такого $B[j]$ не существует, то генерация сочетаний длины m закончена.

```
нц для i от 1 до m
| B[i]:=i                          :инициализация массива
нц
j:=100;
нц пока j<>0
| нц для i от 1 до m
  B[i]:=B[i]                       :вывод очередного полу-
| кц                               :ченного сочетания
| j:=m                                                    (5.2)
| нц пока (j>0) и (B[j]>=n+j-m)    :поиск элемен-
| | j:=j-1                         :та, удовлетво-
| |                                :ряющего усло-
| |                                :вию (*)
```

```
кц
если j<>0                    :проверка окончания генерации сочетаний
| то                         :изменение элемента, удовлетворяющего условию (*)
| B[j]:=B[j]+1
| нц для k от j+1 до m
| | B[k]:=B[k-1]+1           :изменение элементов, стоящих после элемента, удовлетворяющего условию (*)
| кц
| все
кц
```

Вопросы для повторения

1. **Что называется сочетанием из n элементов по m?**
2. **Чему равно число сочетаний из 5 по 3? из 6 по 2?**
3. **Как сгенерировать все сочетания из 5 элементов по 2?**
4. **Какое сочетание будет получено алгоритмом (5.2) вслед за (3, 5, 8), если $n=8$, $m=3$?**

1.3*. Размещения

Соединения, отличающиеся друг от друга составом элементов или их порядком, каждое из которых содержит m ($m<n$) элементов, взятых из n различных элементов, называются ***размещениями из n элементов по m.***

Пример. Выпишем все размещения из элементов a, b, c по два:

$$ab,\ ba,\ ac,\ ca,\ bc,\ cb.$$

Количество способов, которыми можно выбрать и разместить по m различным местам m и n различных элементов, принято обозначать $A(n, m)$ или A_n^m (читается «число размещений из n по m»). Вычислим значение величины A_n^m, т. е. вычислим, сколькими способами можно выбрать и разместить по m различным местам m из n различных предметов.

На первое место можно поместить любой из n предметов, на второе место — любой предмет из $(n-1)$ оставшихся, на третье — один из $(n-2)$ оставшихся и т. д. На предпоследнее место с номером $(m-1)$ — любой из оставшихся $n-(m-2)$ предметов, а на последнее m-е место — один из $n-(m-1)$. Получаем

$$A(n,\ m)=n(n-1)(n-2)\ldots(n-(m-2))(n-(m-1))=$$
$$=\frac{n!}{(n-m)!}.$$

По определению, $0!=1$.

Пример. Сколько всего семизначных телефонных номеров, в каждом из которых ни одна цифра не повторяется?

Эта задача о выборе и размещении по семи различным местам семи из десяти различных цифр, поэтому число указанных телефонных номеров равно

$$A(10,\ 7)=10\cdot 9\cdot 8\cdot 7\cdot 6\cdot 5\cdot 4=604\,800.$$

А теперь докажем формулу (1) п. 1.2 для числа сочетаний.

Выбрать m из n различных предметов можно $C(n, m)$ способами, и в каждом из выбранных сочетаний имеется $m!$ возможностей упорядочить m предметов этого сочетания. Поэтому имеется $m!\,C(n, m)$ возможностей выбрать и разместить по m разным местам m из n разных предметов, т. е.

$$A(n,\ m)=m!\,C(n,\ m).$$

Отсюда следует, что число сочетаний из n различных предметов по m в $m!$ раз меньше, чем числа размещений из n по m, т. е.

$$C(n,\ m)=\frac{A(n,\ m)}{m!}=\frac{n!}{m!\,(n-m)!}.$$

В том случае, если требуется сгенерировать все размещения из n по m, то для каждого сочетания, полученного алгоритмом (5.2), применяют алгоритм генерации перестановок (5.1). (Этот алгоритм разработайте самостоятельно.)

Вопросы для повторения

1. Что называется размещениями из n элементов по m?
2. Чему равно число размещений из 5 по 3? из 6 по 2?
3. Как сгенерировать все размещения из 5 элементов по 2?

§ 2*. СОЕДИНЕНИЯ С ПОВТОРЕНИЯМИ

2.1. Размещения с повторениями

До сих пор рассматривались соединения, в каждое из которых любой из n различных элементов входит один раз. Можно рассматривать соединения с повторениями, т. е. соединения, в каждом из которых любой из n различных элементов может входить более одного раза.

Размещения из n элементов, в каждое из которых входит m элементов, причем один и тот же элемент может повторяться в каждом размещении любое число раз, но не более чем m, называются ***размещениями из n элементов по m с повторениями.***

Пример. Выпишем размещения с повторениями из двух элементов a, b по три:

$$aaa,\ aab,\ aba,\ abb,\ baa,\ bab,\ bba,\ bbb.$$

Количество размещений из n элементов по m с повторениями обозначают $\bar{A}_n^m(n)$. Справедлива формула

$$\bar{A}_n^m(n) = n^m.$$

Действительно, на каждое из m мест мы можем поместить любой из n элементов.

Пример. Каждый телефонный номер состоит из 7 цифр. Сколько всего телефонных номеров, содержащих только цифры 2, 3, 5, 7?

Эта задача о числе размещений в семи позициях семи цифр, каждая из которых может быть 2, 3, 5 или 7. Цифры в телефонном номере могут повторяться.

Число всех указанных номеров есть $\bar{A}_4^7(4) = 4^7 = 16\,384$.

Опишем алгоритм генерации всех размещений из n элементов по m с повторениями. Пронумеруем элементы от 0 до $(n-1)$ и в дальнейшем будем работать не с самими элементами, а с их номерами. Каждому размещению мы можем сопоставить число в n-ичной системе счисления, состоящее из m цифр. И наоборот, каждому из таких чисел соответствует одно-единственное размещение. Минимальным из таких чисел будет число, состоящее из m нулей. Для получения всех чисел, соответствующих размещениям, будем каждый раз прибавлять по 1 к текущему числу, пока не получим число из 1 и m нулей. Данное число не соответствует ни одному из искомых размещений, а является указателем того, что генерация закончена (предыдущим числом было число, состоящее из m цифр, каждая из которых равнялась $n-1$). Приведем фрагмент программы, генерирующей размещения с повторениями.

```
нц для i от 0 до m
| A[i]:=0          :начальное число, состоящее из
                   :всех нулей
кц
нц пока A[m]=0
| i:=0
| A[0]:=A[0]+1     :прибавление 1 к младшему
| |                 :разряду
| нц пока (A[i]=n) и (A[m]=0)
| | A[i]=0
| | A[i+1]:=A[i+1]+1  :перенос 1 в следующий
| |                   :разряд
| | i:=i+1                                    (5.3)
| кц
| нц для i от m-1 до 0 шаг -1
| | A[i]:=A[i]      :вывод очередного размещения с
| |                 :повторениями
| кц
кц
```

2.2. Перестановки с повторениями

Перестановки из n предметов, в каждую из которых входят n_1 одинаковых предметов одного типа, n_2 одинаковых предметов другого типа и т. д. до n_k одинаковых предметов k-го типа, где $n_1+n_2+\ldots+n_k=n$, называются ***перестановками из n элементов с повторениями.***

Пример. Получим перестановки из двух элементов a и b, каждый из которых взят по два раза.

$$aabb,\ abab,\ abba,\ baab,\ baba,\ bbaa.$$

Число всех таких перестановок с повторениями принято обозначать $P_n(n_1, n_2, \ldots, n_k)$. Оно может быть найдено по формуле

$$P_n(n_1, n_2, \ldots, n_k)=\frac{n!}{n_1!\cdot n_2!\cdot\ldots\cdot n_k!}. \qquad (3)$$

Возьмем некоторую перестановку из числа $P_n(n_1, n_2, \ldots, n_k)$ всех перестановок с повторениями. В ней все возможные перестановки элементов первого типа, считая их разными, можно осуществить $n_1!$ способами, затем все возможные перестановки элементов второго типа, считая их разными, можно осуществить $n_2!$ способами и т. д., а затем все возможные перестановки элементов k-го типа, считая их разными, можно осуществить $n_k!$ способами. Осуществляя все возможные перестановки только элементов каждого типа, получим $n_1!n_2!\ldots n_k!$ перестановок, которые бы возникли из взятой перестановки с повторениями, если бы имелась возможность как-то различать входящие в каждый тип одинаковые элементы. Проделав это для каждой перестановки с повторениями, получим $n!$ — число всевозможных перестановок из n различных предметов.

Таким образом, $n_1!n_2!\ldots n_k!\,P_n(n_1, n_2, \ldots, n_k)=n!$, откуда следует формула для числа перестановок с повторениями.

Для получения всех перестановок с повторениями можно воспользоваться алгоритмом (5.1), заменив в нем знаки $>$ на $>=$. (Почему?)

2.3. Сочетания с повторениями

Сочетаниями из n элементов по m с повторениями называются соединения, содержащие m элементов (без учета порядка следования), причем любой элемент может входить в соединение некоторое число раз, не большее m.

Пример. Получим сочетания из элементов a, b по три с повторениями:

$$aaa,\ aab,\ abb,\ bbb.$$

Число всех сочетаний из n элементов по m с повторениями принято обозначать $C_n^m(n)$. Оно может быть найдено по формуле:

$$C_n^m(n)=\frac{(m+n-1)!}{m!\,(n-1)!}=C_{m+n-1}^m. \qquad (4)$$

Для доказательства этой формулы закодируем каждое сочетание с повторением с помощью нулей и единиц. Сначала напишем столько единиц, сколько взято элементов первого типа. Потом, чтобы отделить элементы первого типа от элементов второго типа, запишем нуль, а затем — столько единиц, сколько взято элементов второго типа. Далее снова напишем нуль (если не было взято ни одного элемента второго типа, но в записи появятся два следующих друг за другом нуля). Далее напишем столько единиц, сколько взято элементов третьего типа, снова напишем нуль и т. д., пока не будут выписаны единицы, соответствующие элементам n-го типа.

Для сочетания *aaa* из приведенного выше примера запись будет 1110, а для *abb* — 1011.

Таким образом, число различных сочетаний из n эле-

ментов по m с повторениями равно числу перестановок с повторениями, которые можно составить из m единиц и $(n-1)$ нулей. Для вычисления количества перестановок с повторениями применим формулу (3). Общее количество нулей и единиц равно

$$P_{m+n-1}(m,\ n-1)=\frac{(m+n-1)!}{m!\,(n-1)!}=C_n^m(n).$$

Пример. В кондитерском магазине продавались четыре сорта пирожных. Сколькими способами можно купить семь пирожных?

Эта задача о числе сочетаний из 4 элементов по 7 с повторениями:

$$C_4^7(n)=\frac{10!}{7!3!}=120.$$

Доказательство формулы (4) дает алгоритм генерации всех сочетаний из n элементов по m с повторениями. Для этого применим алгоритм (5.2) для генерации сочетаний из $m+n-1$ элемента по $n-1$ (или по m, что одно и то же, см. формулу (2)).

В массиве B (см. алгоритм (5.2)) хранятся в порядке возрастания индексы используемых на данном шаге элементов. Количество таких индексов в случае генерации сочетаний из $(m+n-1)$ элемента по $(n-1)$ есть $(n-1)$. Каждый из индексов показывает позицию нуля в перестановке с повторениями, составленной из m единиц и $(n-1)$нулей. Количество единиц между соседними нулями (а также количество единиц до первого нуля и после последнего) дает количество элементов каждого типа.

Пример. Пусть на каком-то шаге генерации для $n=4$, $m=5$ ($m+n-1=8$, $n-1=3$) в массиве B получен следующий набор индексов: 1, 3, 6. Этот набор дает конфигурацию 01011011, т. е. элемент первого типа в данную выборку не входит, элемент второго типа входит 1 раз, элемент третьего типа — 2 раза, элемент четвертого типа — 2 раза.

Для генерации сочетаний с повторениями можно воспользоваться и алгоритмом генерации перестановок с повторениями.

§ 3*. ПОДМНОЖЕСТВА

Пусть имеется некий набор из n различных элементов, обладающих некоторым общим свойством. Совокупность этих элементов называют ***множеством.*** Общее свойство элементов множества обычно содержится в самом названии (задании) каждого множества. Множества обычно обозначают заглавными латинскими буквами.

Примеры множеств: множество цифр, множество букв некоторого алфавита, множество точек плоскости, множество звезд во Вселенной.

Если каждый элемент множества A является элементом множества B, то множество A называется ***подмножеством*** множества B. Любая совокупность элементов из данного множества образует подмножество данного множества.

Подмножество, не содержащее ни одного элемента, называют ***пустым.*** Пустое множество является подмножеством всякого множества.

Примеры.

1) Множество всех положительных четных чисел является подмножеством множества всех натуральных чисел.

2) Выпишем все подмножества множества, состоящего из элементов a, b, c. Каждое подмножество будем заключать в фигурные скобки (пустое подмножество обозначают $\{\}$).

$\{\}$, $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, c\}$.

Количество всех подмножеств n-элементного множества равно 2^n ($2^3=8$).

Пусть задано множество из n элементов. Пронумеруем элементы данного множества от 0 до $(n-1)$. В дальнейшем будем работать не с самими элементами, а с их номерами.

Для генерации подмножеств создадим массив $B[0..n]$ из $(n+1)$ элемента (n-й элемент фиктивный и используется для определения окончания работы). $B[i]=0$, если i-й элемент в подмножество не входит, и $B[i]=1$, если входит. Таким образом, пустому подмножеству будет соответствовать набор из n нулей, а n-элементному подмножеству — набор из n единиц. Тут явно заметна связь представления подмножества с двоичным представлением числа.

Будем генерировать числа от 0 до 2^n-1, находить их двоичное представление и формировать подмножество из элементов исходного множества с индексами, соответствующими единицам в этом представлении.

Вначале определим $B[i]=0$ для всех i от 0 до n, что соответствует пустому подмножеству. Будем рассматривать массив B как запись двоичного числа $B[n]...B[0]$ и моделировать операцию сложения этого числа с единицей. При сложении будем просматривать число справа налево, заменяя единицы нулями до тех пор, пока не найдем нуль, в который занесем 1. Генерация подмножеств заканчивается, как только $B[n]=1$ (предыдущая конфигурация была $1...1_2=2^n-1$).

Приведем фрагмент программы генерации всех подмножеств:

```
нц пока B[n]=0
| i:=0
| нц пока B[i]=1
| | B[i]:=0
| | i:=i+1
| кц                                    (5.4)
| B[i]:=1
| нц для i от 0 до n-1
| | B[i]:=B[i]
| кц
кц
```

§ 4. РЕАЛИЗАЦИЯ ПЕРЕБОРА ВАРИАНТОВ. СОКРАЩЕНИЕ ПЕРЕБОРА

Часто встречаются задачи такого рода, когда трудно найти решение, отличное от полного перебора всех возможных вариантов.

Алгоритмы, рассмотренные выше, позволяют организовать основные типы перебора.

Примеры переборных задач.

1. Получить все способы расстановки шести книг разных авторов.

   Для решения задачи применяется алгоритм генерации всех перестановок из шести элементов.
2. Для участия в конкурсе требуется выбрать трех человек из класса в 20 человек.

   Для решения задачи используется алгоритм генерации всех сочетаний из 20 элементов по 3.
3. Получить все четырехзначные числа, у которых все цифры нечетные.

   Для решения задачи применяется алгоритм генерации всех размещений из пяти элементов по четыре с повторениями.
4. Из семи красных и восьми белых роз требуется составить букет из пяти роз. Перечислите все возможные варианты.

   Для решения задачи применяется алгоритм генерации всех сочетаний из двух по пять с повторениями.

Количество рассматриваемых вариантов при переборе может быть очень большим, что влечет за собой большие временные затраты. В некоторых случаях количество рассматриваемых вариантов удается сократить, используя определенные свойства задачи. К сожалению, не существует общих рекомендаций как это сделать. В каждой конкретной задаче приходится искать свой способ.

Проиллюстрируем сокращение перебора на примере следующих задач:

Задача 1. Часы

В матрице размера 3×3 расположены 9 циферблатов с заданным положением стрелок (рис. 31). Требуется установить на всех циферблатах время 12 часов.

Возможно 9 различных способов изменения стрелок на циферблатах. Каждый такой способ задается набором циферблатов (рис. 32), стрелки которых поворачиваются на 90° по часовой стрелке. На рисунке для каждого из способов соответствующие циферблаты выделены серым цветом, каждый способ определяется номером — числом от 1 до 9.

Перевод стрелок из начального состояния в конечное производится последовательностью шагов. За один шаг можно осуществить перевод стрелок на циферблатах одним из указанных способов.

Необходимо найти кратчайшую последовательность шагов, переводящую стрелки всех циферблатов из начальной позиции в позицию 12 часов. Положение стрелки на циферблате задается числом от 0 до 3: 0 — 12 ч, 1 — 3 ч, 2 — 6 ч, 3 — 9 ч. Для решения задачи пронумеруем циферблаты числами от 1 до 9, начиная нумерацию с верхнего левого угла и проводя ее по строкам слева направо. Предположим, что требуемая последовательность шагов найдена. Покажем, что можно ограничиться рассмотрением только таких последовательностей, у которых группа одинаковых преобразований выполняется последовательно друг за другом. Действительно, два

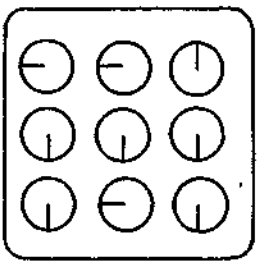


Рис. 31

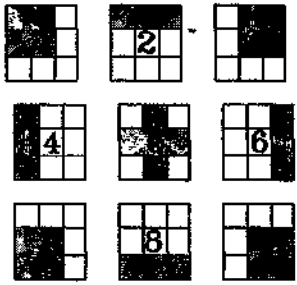


Рис. 32

соседних преобразования в последовательности можно поменять местами, при этом окончательный результат при последовательном выполнении преобразований не изменится. При этом количество одинаковых преобразований в оптимальной (наиболее короткой) последовательности не превосходит 3, так как последовательное применение 4 одинаковых преобразований переводит циферблаты в исходное состояние.

Таким образом, если рассмотреть все возможные допустимые минимальные последовательности преобразований, в которых каждое из преобразований может выполняться от 0 до 3 раз (т. е. рассмотреть все возможные 9-разрядные числа в четверичной системе счисления, когда каждый разряд отражает количество выполнений соответствующего преобразования), то можно найти среди них решение поставленной задачи. Это соответствует генерации всех размещений из 4 элементов по 9 с повторениями. Число таких последовательностей равно 4^9.

Однако можно существенно сократить количество рассматриваемых последовательностей, если проанализировать влияние каждого из преобразований на конкретный циферблат. В следующей таблице приведены множества преобразований, влияющих на каждый из циферблатов.

Номер циферблата	Номера преобразований, влияющих на циферблат
1	1, 2, 4
2	1, 2, 3, 5
3	2, 3, 6
4	1, 4, 5, 7
5	1, 3, 5, 7, 9
6	3, 5, 6, 9
7	4, 7, 8
8	5, 7, 8, 9
9	6, 8, 9

Итак, на циферблат 1 влияют преобразования 1, 2 и 4. Попытаемся установить его в конечное положение, выполняя последовательно все возможные комбинации преобразований 1, 2 и 4. При этом любой произвольный набор комбинаций преобразований 1 и 2 (в количестве a_1 и a_2, соответственно) однозначно определяет количество преобразований, которое необходимо для установления циферблата 1 в конечное положение. Таким образом, a_4 фиксировано для заданных значений a_1 и a_2. Из второй строчки таблицы видно, что теперь достаточно рассмотреть только всевозможные значения a_3 (при уже фиксированных значениях a_1 и a_2), при этом значение a_5 будет фиксировано. Из строчки 3 следует, что a_6 фиксировано, из строчки 4 — что фиксировано a_7, из строчки 5 — a_9, из строчки 7 — a_8.

Таким образом, достаточно осуществить перебор по возможным значениям a_1, a_2 и a_4, вычисляя при этом значения a_3, a_5, a_7, a_9, a_8. При этом полученная последовательность должна переводить циферблаты в конечное состояние.

Итак, нам достаточно перебрать 4^3 комбинаций, что значительно меньше, чем 4^9.

Задача 2. Раздел наследства

Баронесса фон Бирлингхофен оставила после себя двум дочерям ларец с золотыми монетами. Ее завещание гласило, что все золото получит соседний монастырь, если дочерям не удастся разделить содержимое ларца на две равные части. Достоинство каждой золотой монеты является целым числом.

Пример. Если монеты в ларце были достоинством 2, 10, 4, 3, 7, 4, то их можно разделить между наследницами следующим образом: 10, 3, 2 и 7, 4, 4. Если же золотые монеты были достоинством 1, 9, 7, 3, 8, то раздел наследства невозможен, и оно достанется монастырю.

Программа должна вводить количество монет N и их достоинства a_1, a_2, ..., a_N. Результатом работы програм-

мы должно быть сообщение: «раздел наследства невозможен» или «раздел наследства возможен». В последнем случае необходимо указать, монеты какого достоинства получит каждая из наследниц.

Обозначим сумму номиналов всех монет через S. Это число должно быть четным, иначе раздел невозможен.

Для решения данной задачи можно, например, применить алгоритм (5.4) для генерации всех подмножеств множества монет и поиска среди этих подмножеств такого, стоимость которого равна $S/2$. Под стоимостью подмножества будем понимать суммарную стоимость номиналов монет, входящих в данное подмножество. Количество подмножеств равно, как уже отмечалось, 2^N. Попытаемся сократить количество рассматриваемых подмножеств, исходя из следующих соображений:

1) Ни одна из наследниц не может получить более половины наследства. Поэтому нас не интересуют подмножества стоимостью более чем $S/2$.

2) Если уже сгенерировано некоторое подмножество A со стоимостью больше $S/2$, то добавление любых элементов к этому подмножеству не сможет привести к решению и, следовательно, генерировать их не имеет смысла.

3) Максимальная по номинальной стоимости монета (пусть это монета a_1) должна попасть к одной из наследниц, поэтому раздел наследства начнем с этой монеты. Из-за того, что стоимость выбранной монеты максимальна, разность между $S/2$ и стоимостью этой монеты минимальна, что, возможно, приведет к уменьшению количества рассматриваемых вариантов. С этой же целью отсортируем монеты в порядке невозрастания их номиналов.

4) Если одна из монет имеет номинал, равный половине искомой суммы, то разбиение найдено.

5) Если хоть одна монета имеет номинал более половины искомой суммы, то разбиение невозможно.

Пункты 4 и 5 достаточно проверить только для монеты с максимальным номиналом.

В массиве *A* будем хранить номиналы монет. В массиве *st* в ячейках с индексами от 1 до *st*[0] хранятся номера выбранных сейчас монет (в отсортированном массиве). Монеты с этими номерами образуют подмножество стоимости *s*. В массиве *rez* поместим те монеты, которые получит одна из наследниц, тогда другая получит оставшиеся монеты.

Эти рассуждения являются основой следующего алгоритма:

```
summa:=0
для i от 1 до n
нц                                  :находим стоимость
| summa:=summa+a[i]                 :наследства
кц
otvet:=«нет решения»
если mod(summa,2)=0
| то
| summa:=div(summa,2)
| для i от 1 до n-1
| | для j от i+1 до n
| | | если a[i]<a[j]
| | | | то                        :сортировка «пузырьком»
| | | | k:=a[i]                   :массива
| | | | a[i]:=a[j]                :с исходными данными
| | | | a[j]:=k                   :в порядке убывания
| | | |
| | | все
| | кц
| кц
| st[0]:=1      :количество выбранных сейчас монет
| st[1]:=1      :индекс монеты с максимальным
|               :номиналом
| s:=a[1]
| i:=st[1]+1      :номер очередной добавляемой мо-
|                 :неты
```

```
если a[1]>summa
| то
| otvet:=«нет решения»
| st[1]:=0    :полагаем st[1]=0, чтобы пропустить
|             :цикл «пока»
все
если a[1]=summa
| то
| otvet:=«есть решение»                              (5.5)
| rez[1]:=a[1]
все
пока (st[1]=1) и (otvet=«нет решения»)
нц
| пока (i<=n) и (otvet=«нет решения»)
| нц
| | s:=s+a[i]      :пытаемся добавить монету к
| |                :сумме
| | если s=summa
| | | то
| | | otvet:=«есть решение»
| | | x:=st[0]
| | | для j от 1 до x          :найденное решение
| | | нц                       :сохраняем в массиве
| | | | rez [j]:=a[st[j]]       :rez
| | | кц
| | | rez[x+1]:=a[i]
| | | иначе
| | | если s<summa
| | | | то          :добавляем очередную монету к
| | | |             :уже выбранным
| | | | st[0]:=st[0]+1   :увеличиваем количество
| | | |                  :выбранных монет
| | | | x:=st[0]
| | | | st[x]:=i        :индекс выбранной монеты
| | | |                 заносим в массив
| | | | иначе           :монета не подходит
| | | | s:=s-a[i]       :превышена summa
```

```
|  |  | все
|  |  | i:=i+1          :переходим к следующей монете
|  | все
| кц
| :если решение не найдено, то возвращаемся на
| :шаг назад,
| :т. е. выбрасываем последнюю добавленную монету
| x:=st[0]            :удаление последней
| i:=st[x]+1          :добавленной
| s:=s-a[st[x]]       :монеты из выбранных
| st[0]:=x-1
кц
все
```

Рассмотрим работу данного алгоритма на приведенном выше примере. После сортировки массива имеем:

$a=(10, 7, 4, 4, 3, 2)$;

$summa=15$;

$s=10$;

$st=(1, 1, 0, 0, 0, 0, 0)$.

Выпишем значения переменных при каждом прохождении цикла ПОКА.

Значение переменной i	Значение переменной s	Массив st
2	$s=10+a[2]=17>15$; $s=s-a[2]=10$;	$st=(1, 1, 0, 0, 0, 0, 0)$
3	$s=10+a[3]=14<15$;	$st=(2, 1, 3, 0, 0, 0, 0)$
4	$s=14+a[4]=18>15$; $s=s-a[4]=14$;	$st=(2, 1, 3, 0, 0, 0, 0)$
5	$s=14+a[5]=17>15$; $s=s-a[5]=14$;	$st=(2, 1, 3, 0, 0, 0, 0)$
6	$s=14+a[6]=16>15$; $s=s-a[6]=14$; $s=s-a[3]=10$;	$st=(1, 1, 3, 0, 0, 0, 0)$[1]

[1] Элемент $st[2]=3$, хоть и присутствует в массиве, но в дальнейшем рассмотрении не участвует, так как количество элементов равно $st[0]=1$.

Значение переменной i	Значение переменной s	Массив st
4	$s=10+a[4]=14<15$;	$st=(2, 1, 4, 0, 0, 0, 0)$
5	$s=14+a[5]=17>15$; $s=s-a[5]=14$;	$st=(2, 1, 4, 0, 0, 0, 0)$
6	$s=14+a[6]=16>15$; $s=s-a[6]=14$; $s=s-a[4]=10$;	$st=(1, 1, 4, 0, 0, 0, 0)$
5	$s=10+a[5]=13<15$;	$st=(2, 1, 5, 0, 0, 0, 0)$
6	$s=13+a[6]=15$;	

$rez=(10, 3, 2)$.

Метод, использованный для решения данной задачи, называется ***перебором с возвратом*** или ***бектрекингом (Backtracking)***.

Некоторые задачи, являющиеся на первый взгляд переборными, могут быть решены другими методами.

Задача 3. «Счастливые» билеты

Составить алгоритм определения количества шестизначных «счастливых» трамвайных билетов, у которых сумма первых трех цифр совпадает с суммой трех последних. Эта задача рассматривалась среди задач повышенной сложности главы «Рекуррентные уравнения и динамическое программирование» (задача 1). Метод ее решения не является переборным, хотя, на первый взгляд, эта задача сводится к генерации всех сочетаний из 10 по 6 с повторениями.

Если бы в задаче требовалось не только подсчитать количество «счастливых» билетов, но и выдать номера этих билетов, то в этом случае необходимо было бы использовать только переборный метод.

На примере этой задачи видна разница между подсчетом количества вариантов, удовлетворяющих условию задачи, и генерацией этих вариантов.

ЗАДАЧИ ДЛЯ ПОВТОРЕНИЯ

1. Сколькими способами можно раскрасить вершины куба в три цвета (например, красный, синий и зеленый)? Напечатать все возможные способы.

2. Сколькими различными способами можно надеть на нить семь бусин двух цветов — синего и белого? Напечатать все возможные варианты.

3. Сколько различных ожерелий из семи бусин можно составить из бусин двух цветов — синего и белого? (Под ожерельем понимается замкнутая нить с нанизанными бусинами.) Напечатать все возможные варианты.

4. Сколько различных ожерелий можно составить из двух белых, двух синих и двух красных бусин? Напечатать все возможные варианты.

5. Сколькими различными способами можно грани куба раскрасить в четыре цвета? Напечатать все возможные варианты.

6. Грани куба можно раскрасить: а) все в белый цвет; б) все в черный цвет; в) часть в белый, а остальные в черный. Сколько имеется различных способов раскраски? Напечатать все возможные варианты.

7. Напечатать все четырехзначные десятичные числа, у которых все цифры разные. Обобщить на N-значные числа.

8. Дама собирается пригласить семерых своих друзей на несколько званых обедов. Ее стол, однако, невелик, да и обед «в узком кругу» несомненно гораздо приятнее. Поэтому она решает приглашать каждый раз лишь троих гостей. Кроме того, ей хочется, чтобы каждые двое ее друзей непременно встретились за ее столом, причем она предпочла бы, чтобы они встретились лишь однажды. Как хозяйке распределить приглашения по дням?

9. Написать алгоритм выплаты заданной суммы денег всеми возможными способами. В наличии имеются

купюры достоинством 1, 2, 5, 10, 20 и 100 долларов. Количество купюр каждого вида не ограничено.

10. Написать программу, которая из букв A, B и C построит слово длины N, в котором два любых стоящих рядом подслова различны.

Например:

слово *ABCABA* составлено правильно;

слово *CABABC* составлено неправильно, так как сочетания букв *AB* стоят рядом.

11. Существуют числа, обладающие свойствами:

число делится на все свои цифры;

число, полученное из данного записью цифр в обратном порядке, тоже делится на все свои цифры.

Примером такого числа является 216. Составить программу для печати всех трехзначных чисел, обладающих этими свойствами. Числа с одинаковыми первой и последней цифрой не печатать.

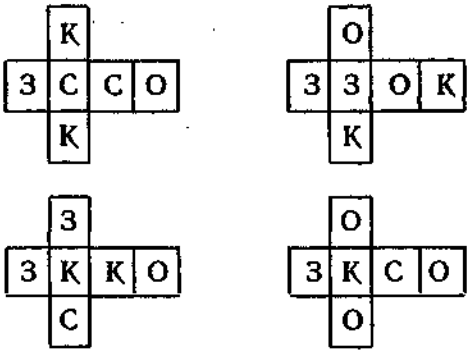
12. Составить алгоритм получения словаря племени тумба-юмба, если известно, что:

алфавит языка этого племени содержит 4 буквы — a, b, c, d;

слова на языке этого племени не содержат двух и более одинаковых букв подряд.

13. Клетчатое поле $n \times n$ охраняется m воинами царя Артаксеркса, занимающими различные клетки этого поля. Царь хочет расположить воинов так, чтобы обеспечить максимальную суммарную защищенность границы своего владения, т. е. $4(n-1)$ клеток непосредственно примыкающих к полю. Каждый воин охраняет те приграничные клетки, которые он «видит» по горизонтали, вертикали и диагоналям. При этом, защищенность клетки пропорциональна количеству воинов, ее охраняющих. Помогите Артаксерксу найти хотя бы одно положение воинов по его желанию.

14. У четырех кубиков грани окрашены в четыре разных цвета: красный, синий, зеленый и оранжевый. Развертки этих четырех кубиков выглядят следующим образом:



Написать программу, которая составляла бы из этих четырех кубиков прямоугольную призму, каждая боковая грань которой раскрашена во все четыре цвета без повторений.

ЗАДАЧИ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Дана строка S и набор A слов $A_1, \ldots, A_k$. Разбить строку S на слова набора всеми возможными способами.

Пример: $S=ABBC$

$A_1=A$, $A_2=AB$, $A_3=BC$, $A_4=BBC$, $A_5=H$, $A_6=B$

$S = A\ B\ BC$

$S = A\ BBC$

$S = AB\ BC$

2. В написанном выражении ((((1?2)?3)?4)?5)?6 вместо каждого знака «?» вставить знак одной из 4 арифметических операций (+, —, *, /) так, чтобы результат вычислений равнялся 35 (при делении дробная часть в частном отбрасывается). Найти все решения.

3. Составить программу, которая печатает все различные представления числа N в виде всевозможных сумм натуральных чисел. Представления числа, отличающиеся только порядком слагаемых, считаются одинаковыми.

4. Написать программу, отвечающую на вопрос, можно ли из N данных прямоугольников $П_i$ размеров (a_i, b_i), $i=1, \ldots, N$, сложить один большой прямоугольник П размера (a, b). Нижний левый угол большого прямоугольника имеет координату (0; 0), стороны параллельны осям

координат; маленькие прямоугольники можно поворачивать, но так, чтобы их стороны после поворота оставались параллельны осям координат. При составлении П прямоугольники $П_i$ могут быть использованы не все. Прямоугольники $П_i$ не перекрываются.

Ограничения: $N<8$; a, b, a_i, b_i — целые числа.

Ответ выдать в виде «да» — «нет», и в случае «да» необходимо выдать для каждого прямоугольника $П_i$, образующего П, координату его левого нижнего и правого верхнего углов и его номер i.

5. Игровой автомат состоит из нескольких изогнутых трубок, по форме похожих на перевернутую букву Y. Сверху у трубы находится входное отверстие, а два выходных отверстия — снизу. Труба может находиться в одном из двух возможных состояний. Состояние 1 — это состояние, в котором закрыт левый выход, Состояние 0 — закрыт правый выход.

Шарики кладутся внутрь один за другим через входное отверстие. В Состоянии 1 (рис. 33, *а*) шарик покидает трубу через незаблокированный правый выход, при этом Состояние 1 автоматически изменяется на Состояние 0 ((рис. 33, *б*), правый вход заблокирован, левый — нет). В Состоянии 0 все происходит наоборот.

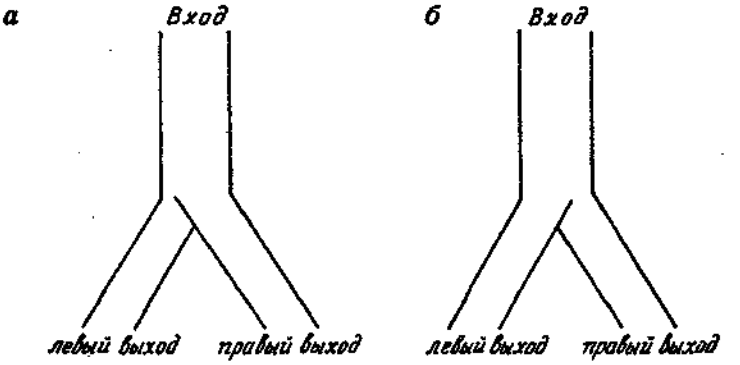


Рис. 33

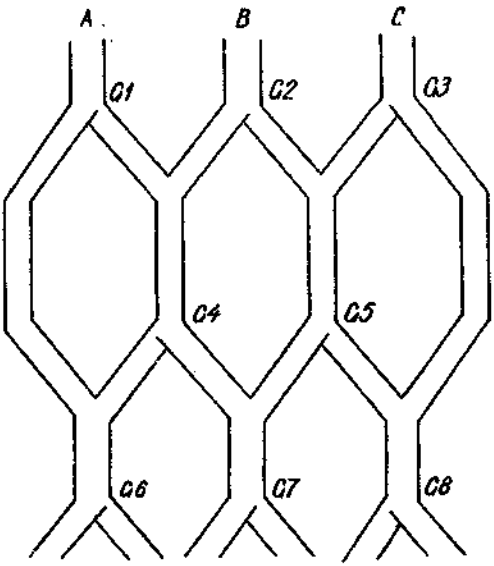


Рис. 34

Пусть игральный автомат состоит из 8 Y-образных трубок, их взаимное расположение показано на рисунке 34. Вы можете забрасывать шарики через Вход *A*, Вход *B* или Вход *C*. Пусть, например, первый шарик опускается через Вход *A*. Он покидает трубу G_1 через левый выход, изменяя ее состояние с 0 на 1, поступает в G_6 и покидает автомат через левый выход G_6 (изменяя Состояние G_6 с 0 на 1). Эту процедуру запишем следующим образом:

A

Затем забросим шарик через Вход *A* (он находится в Состоянии 1). Шарик покидает G_1 через правый выход, изменяя Состояние 1 в Состояние 0, попадает в G_4, покидает G_4 через правый выход, изменяя Состояние 1 в Состояние 0, попадает в G_7, покидает автомат через левый выход (изменяя G_7 из Состояния 0 в Состояние 1). Все вышеизложенные действия мы запишем в виде:

AA

Если мы опустим третий шарик через Вход B, четвертый — через Вход C, то все действия запишутся следующим образом:

$$AABC$$

Необходимо:

1) Ввести с клавиатуры последовательность из 8 двоичных цифр (бит), показывающих соответственно начальное Состояние $G_1 - G_8$:

Бит	7	6	5	4	3	2	1	0
Вход	G_8	G_7	G_6	G_5	G_4	G_3	G_2	G_1

Например, последовательность 11001010 означает, что G_8, G_7, G_4 и G_2 находятся в Состоянии 1 (левый вход заблокирован), тогда как G_6, G_5, G_3 и G_1 находятся в Состоянии 0 (правый вход заблокирован).

2) Ввести с клавиатуры вторую последовательность из 8 бит, показывающих конечные Состояния $G_1' - G_8'$.

3) Напечатать последовательность ходов A, B или C, указывающую, каким образом можно получить конечную позицию, забрасывая шарики через Входы A, B и C.

6. Пусть слово — это последовательность от 1 до 8 заглавных букв латинского алфавита.

Задается множество слов $A = \{a[1], a[2], \ldots, a[n]\}$, $n < 10$. Из слов множества A составляется текст — последовательность слов, записанных друг за другом без пробелов. Слова могут встречаться в тексте произвольное число раз.

Дешифровка текста — это разбивка текста на слова множества A. В дешифрованном тексте слова разделяются пробелами.

Необходимо:

1) Определить, существует ли для заданного множества A такой текст, который дешифруется не единственным образом (сам текст приводить не надо).

Примеры:

а) $A=\{B, C\}$

Любой текст дешифруется однозначно.

б) $A=\{B, BC, C\}$

Существует текст, который дешифруется двумя способами:

Текст	→	Дешифровка
BBC	→	$B\ B\ C$
BBC	→	$B\ BC$

2) Если такой текст существует, то исключить из множества A минимальное число слов так, чтобы после этого любой текст, составленный из слов полученного множества A, дешифровался однозначно. Напечатать эти исключенные слова. Если такой набор не единственный, то напечатать все наборы.

3) Для введенного текста произвести его дешифровку и, если дешифровка не единственная, вывести все варианты.

Примечание. Порядок выполнения пунктов строго фиксирован.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

1. Во время поездки на поезде девочка заменила в названии поезда каждую букву ее номером в русском алфавите и получила запись из единиц и двоек «211221—21221». Определить, откуда и куда идет поезд.

2. Данные N косточек домино по правилам игры выкладываются в прямую цепочку, начиная с косточки, выбранной произвольно, в оба конца до тех пор, пока это возможно. Построить алгоритм, позволяющий определить такой вариант выкладывания заданных косточек, при котором к моменту, когда цепочка не может быть продолжена, «на руках» останется максимальное число очков.

3. В клетках таблицы расставлены числа. Расставить в этих клетках K ферзей так, чтобы они друг друга не

били и чтобы сумма чисел, ими закрываемых, была максимальной.

4. Вводится строка не более чем из 6 цифр и некоторое целое число R. Расставить знаки арифметических операций $+$, $-$, $*$, $/$ (деление есть деление нацело, т. е. $11/3=3$) и открывающие и закрывающие круглые скобки так, чтобы получить в результате вычисления число R. Лишние круглые скобки ошибкой не являются.

Например: Строка 502597, $R=120$:

$$((5+0)*(25-(9/7)))=120.$$

5. Перечислить все расстановки скобок в произведении n сомножителей. Порядок сомножителей не меняется, скобки полностью определяют порядок действий. (Например, для $n=4$ есть 5 расстановок:

$$((ab)c)d,\ (a(bc))d,\ (ab)(cd),\ a((bc)d),\ a(b(cd)).)$$

6. Составить программу, которая печатает все различные представления числа N в виде всевозможных сумм K натуральных чисел ($1<K<N$). Представления числа, отличающиеся только порядком слагаемых, считаются одинаковыми.

7. Составить программу, которая печатает все различные представления числа N в виде всевозможных произведений K натуральных чисел (N, K — вводятся, $1<K<N$). Если $K=0$, то выдать все возможные произведения. Представления числа, отличающиеся только порядком сомножителей, считаются одинаковыми.

8. В некоей детской книге страницы разделены на три части, причем каждую часть можно перелистывать отдельно. На каждой странице трехсложные имена зверей расположены так, что каждый слог находится на одной части страницы; например, слоги стоят так: E-le-fant, kro-ko-dil, kan-gu-ruh. После перелистывания одной части страницы возможны совмещения: Kro-ko-fant, kan-le-dil.

Написать программу:

а) с помощью которой трехсложные имена зверей можно будет добавлять в книгу;

б) которая находит и протоколирует, как можно перелистывать книгу, чтобы найти путь от одного данного имени животного к другому;

в) которая, решив пункт б), находит несколько способов перелистывания.

9. Если кто-то захочет развести в аквариуме экзотических рыб, то он не сможет обойтись без совета эксперта. При покупке следует иметь в виду, что не каждый вид рыб уживается с другим видом и, конечно, рыбы несовместимых видов не должны жить вместе в аквариуме и бороться или даже поедать друг друга.

Необходимо написать программу, которая запрашивает:

а) сумму, которой распоряжаются при покупке рыб;

б) имеющиеся виды рыб с ценами за штуку;

в) виды рыб, не переносящие друг друга.

Покупателю должен быть выписан чек на максимально возможную сумму, с перечислением видов рыб, которые могут уживаться друг с другом.

10. Дырокол пробивает на длинной полосе бумаги по два одинаковых отверстия за одно нажатие его рычага и после нескольких нажатий получается следующее состояние бумаги.

o	o	o	oo	o	oo	oo	o	o
Л	П	Л	ПЛ	П	ЛЛ	ПП	Л	П

Буквы Л и П под лентой обозначают соответственно левое и правое отверстия дырокола.

По известным координатам отверстий на полосе определить возможную ширину между отверстиями дырокола.

11. В билете пассажира оказалось пробито отверстий больше, чем штырей в компостере. Пассажир утверждал, что пользовался только одним компостером, но случайно нажал на него несколько раз. Контролеру

требуется определить, могло ли быть получено заданное расположение отверстий одним и тем же компостером, если билет можно пробивать с обеих сторон неограниченное число раз и произвольно перемещать и поворачивать относительно компостера. Пробитые отверстия не выходят за пределы билета. В билете было пробито N ($N<10$) отверстий.

1) Для компостера с двумя штырями ($S=2$) составить программу, которая:

а) определяет, можно ли получить заданным компостером требуемое расположение отверстий в билете (если это возможно, то изображает вид билета после каждого нажатия компостера, в противном случае выводит соответствующее сообщение);

б) определяет количество K различных компостеров, каждым из которых можно пробить заданную конфигурацию;

в) при $K=0$ находит компостер, с помощью которого можно пробить наибольшее количество из заданных отверстий;

г) находит минимальное число нажатий, требуемое для пробивки заданной конфигурации отверстий, для каждого компостера из п. б).

2) Решить задачу 1) для компостеров с числом штырей S ($S>2$).

Примечания.

1. Все исходные данные — натуральные числа.

2. Компостеры, дающие при однократном нажатии совпадающие конфигурации отверстий, считаются одинаковыми.

3. Относительное расположение отверстий в билете и штырей в компостере вводится либо с клавиатуры, либо из файла с именем COMP.DAT. Структура вводимой информации:

$$x[1],\ y[1],\ \ldots,\ x[N],\ y[N],\ S,\ u[1],\ v[1],\ \ldots,\ u[S],\ v[S],$$

где $x[i]$, $y[i]$ — координаты отверстий в билете, $u[i]$, $v[i]$ — координаты штырей в компостере.

4. Нажатие компостера моделировать клавишей «Пробел».

5. При выводе конфигурации на экран изображать координатную сетку. При этом программа должна осуществлять подходящее масштабирование.

12. Одним из стандартных методов кодирования информации когда-то был следующий: текст (длины l) записывался по строкам слева направо сверху вниз в прямоугольнике размера $m \times n$ ($m \cdot n \geqslant l$), дополняясь, по необходимости, пробелами; после этого текст из прямоугольника выписывался по столбцам и получалось шифрованное сообщение.

Например, исходный текст «It is a nice weather today» ($m=5$, $n=6$):

	1	2	3	4	5	6
1	I	t	—	i	s	—
2	a	—	n	i	c	e
3	—	w	e	a	t	h
4	e	r	—	t	o	d
5	a	y	—	—	—	—

Шифрованный текст: «Ia eat wry ne iiat scto ehd».

Дается шифрованный текст длины s, состоящий из букв и пробелов, и известно, что в исходном тексте есть, по крайней мере, одно из слов набора $a_0, \ldots, a_k$, $k \leqslant 10$. Под словом в тексте понимается последовательность символов, не содержащая пробелов, и окруженная пробелами, за исключением случаев, когда слово стоит в начале либо в конце текста (тогда ему может не предшествовать или за ним может не следовать) один или несколько пробелов. Дешифровать текст.

13. «Царевна». В одной из клеток поля размера $n \times n$ ($n > 1$) Кощей Бессмертный спрятал Марью Царевну, создав еще неизвестное число m ($1 < m < n^2$) ее двойников в различных свободных клетках. И царевна, и ее двойники одинаково надежно укрыты и невидимы.

Отправившийся на поиски царевны Иванушка-дурачок попросил у благоволящей к нему щуки датчик

биосигналов. Известно, что и Марья Царевна и ее двойники испускают незатухающие направленные биолучи, распространяющиеся параллельно сторонам и диагоналям поля.

Иванушка-дурачок также знает, что интенсивность биолуча Марьи Царевны в m раз выше интенсивности биолучей двойников. Иванушка может установить свой датчик в любую клетку поля и получить величину суммарной интенсивности биолучей, приходящих в клетку.

Помогите Иванушке определить местонахождение настоящей царевны.

14. К Штирлицу попала закодированная записка Бормана:

15, 16, 16, 16, 16, 4, 5, 8, 31, 25, 20, 2, 19, 18.

Штирлиц знал, что Борман пишет по-русски, используя обычную нумерацию букв в русском алфавите от 1 до 33. Пробел между словами он обозначает номером 0. Также он знал, что Борман кодирует свои сообщения, добавляя к номеру каждой буквы число $x=na+b$, где n — порядковый номер этой буквы в сообщении, a и b — константы, известные только Борману. Если результат оказывается больше 33, то из него вычитается 34. Кроме того, Борман ни в одном сообщении не обходится без местоимения «Я». Расшифруйте записку.

15. На доске размера 3×3 произвольным образом расставлены фишки, занумерованные от 1 до 7. За один ход можно передвинуть фишку на свободное соседнее поле по горизонтали или по вертикали. Написать программу, приводящую фишки в конечную позицию:

1	2	3
4	5	6
7		

16. На оси Ox были заданы N точек с целочисленными координатами. Некоторые точки могут иметь одинаковые координаты. Были измерены и записаны всевозможные расстояния между этими точками. Расстояние между двумя точками мы храним только один раз, расстояние от точки до нее самой (равное 0) не хранится, поэтому расстояний всего $N(N-1)/2$.

Необходимо по введенной последовательности из $N(N-1)/2$ расстояний найти одно из возможных расположений точек на прямой или указать, что такого не существует.

17. Даны 4 слова. Длина каждого слова не более 12 символов. Написать программу, проверяющую, можно ли из данных слов составить кроссворд при условии, что каждое слово пересекается с двумя другими и располагается сверху вниз или слева направо. Сетка не обязательно симметрична. Результат вывести на экран в виде кроссворда.

Ввод: вводятся 4 слова, по одному в строке.

Вывод: на экран выводится один из вариантов кроссворда или слово "Нельзя".

Пример:

Ввод:	Вывод:
АЛЬФА	АЛЬФА
АСТРА	С А
ФАКИР	Т К
АВАРИЯ	Р И
	АВАРИЯ

18. Напечатать все последовательности из k положительных целых чисел, у которых i-й член не превосходит i.

19. Построить все отображения множества $\{1, \ldots, k\}$ в $\{1, \ldots, n\}$ (предполагается, что $k \leqslant n$), такие, что ни один элемент не переходит сам в себя. Порождение очередного элемента должно требовать порядка k действий.

20. Напечатать все перестановки чисел 1, 2, ..., n так, чтобы каждая следующая получалась из предыдущей перестановкой (транспозицией) двух соседних чисел. Например, при $n=3$ допустим такой порядок:

$$3.2\,1 \to 2\,3.1 \to 2.1\,3 \to 1\,2.3 \to 1.3\,2 \to 3\,1\,2$$

(между переставляемыми числами вставлены точки).

21. Элементами последовательности длины $2n$ могут быть числа 1 или -1. Сумма всех элементов последовательности равна нулю. Перечислить все такие последовательности, у которых сумма любого начального отрезка положительна (т. е. число минус единиц в нем не превосходит числа единиц).

22. На окружности задано $2n$ точек, пронумерованных от 1 до $2n$. Перечислить все способы проведения n непересекающихся хорд с вершинами в этих точках.

23. Перечислить все последовательности из n нулей, единиц и двоек, в которых никакая группа цифр не повторяется два раза подряд (нет куска вида XX).

24. Есть N карточек. На каждой из них черными чернилами написан ее уникальный номер — число от 1 до N. Также на каждой карточке красными чернилами написано еще одно целое число, лежащее в промежутке от 1 до N (некоторыми одинаковыми «красными» числами могут помечаться несколько карточек).

Например, $N=5$; 5 карточек помечены следующим образом:

«Черное» число	1	2	3	4	5
«Красное» число	3	3	2	4	2

Необходимо выбрать из данных N карточек максимальное число карточек таким образом, чтобы множества «красных» и «черных» чисел на них совпадали.

Для рассмотренного примера это будут карточки с «черными» номерами 2, 3, 4 (множество красных номеров, как и требуется в задаче, то же — {2, 3, 4}).

ВВОД

<N=>N, N⩽50

<"Черный" номер 1, "красный"—>"красное"_число_1

.....

<"Черный" номер N, "красный"—>"красное"_число_N

ВЫВОД:

<В выбранном множестве элементов>количество_элементов S

<"Черные" номера выбранных карточек>a1, ..., aS

25. «Тетрамино». Дано поле размера 6×6 клеток. Клетки пронумерованы от 1 до 36 слева направо и сверху вниз (как при письме).

Ровно четыре клетки из них закрашены. Написать программу, которая для введенных номеров закрашенных клеток выясняет, образуют ли они какую-либо фигуру тетрамино (эти фигуры могут быть повернуты):

Если введенный номер меньше 1 или больше 36, или какой-то номер введен повторно, то вывести сообщение "Ошибка".

Если закрашенные клетки образуют фигуру тетрамино, вывести номер этого тетрамино, в противном случае выдать сообщение: "Не тетрамино".

Пусть закрашены N клеток. Число N и номера этих клеток вводятся. Определить, можно ли разбить закрашенную область на фигуры тетрамино. Фигуры не могут пересекаться, и объединение этих фигур полностью совпадает с закрашенной областью.

Выдать сообщение "Можно разбить" или "Нельзя

разбить". В случае, если разбивка возможна, выдать какую-нибудь разбивку в виде:

Номер фигуры. Номера образующих ее клеток.

...

Номер фигуры. Номера образующих ее клеток.

26. Имеется N ($N<7$) дисков одинаковой толщины с радиусами $r_1, \ldots, r_n$. Эти диски упаковываются в коробку таким образом, что каждый из них стоит ребром на дне коробки и все диски находятся в одной плоскости.

Найти минимальную длину коробки, в которую все они могут быть упакованы, и указать порядок одной из возможных упаковок.

Вход:

<1-я строка>N

<2-я строка>$r_1 r_2 \ldots r_N$

Выход:

Минимальная длина: число

Возможный порядок: порядок

Пример:

3

2.0 2.0 1.0

Минимальная длина: 9.65685

Возможный порядок: 1 3 2

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ ПОВЫШЕННОЙ СЛОЖНОСТИ

1. Эта задача реализуется следующим рекурсивным алгоритмом поиска с возвращением (все слова набора A упорядочены по номерам).

а) Если строка пустая, то одна из возможных дешифровок найдена, иначе — при разборе текста мы проверяем A_i (при изменении i от 1 до n) на вхождение в начало дешифруемой в данный момент строки.

б) Если какое-то A_i входит в строку как префикс, то запоминаем номер i этого слова, затем выделяем слово из строки, а с остатком текста производим операцию разбора по пункту а).

Если ни одно из A_i не входит в качестве приставки в дешифруемую сейчас строку, то осуществляем возврат на пункт а), предварительно добавляя в начало строки последнее удаленное оттуда слово, и пытаемся выделить из текста слово с большим номеров в A. Если возврат осуществить невозможно (так как мы находимся в начале исходной строки), то алгоритм заканчивает свою работу. Все возможные дешифровки найдены.

2. Всего может быть 4 арифметических операции (+, —, *, /). Занумеруем их от 0 до 3. Вместо каждого из пяти знаков «?» может стоять один из знаков операции. Заведем массив A из 5 целых чисел, в i-м элементе массива будет храниться код соответствующей i-му знаку «?» операции, т. е. число от 0 до 3. Начальная конфигурация — все элементы массива нулевые, конечная — все они равны 3. Генерация очередной конфигурации знаков равносильна прибавлению единицы в четверичной системе счисления, в которой разрешается пользоваться только цифрами от 0 до 3. Такая генерация описана в алгоритме (5.3).

3. Предложим простой способ построения всех разбиений числа на слагаемые. Разбиения будут строиться в порядке, обратном лексикографическому. Очевидно, что первым разбиением в таком порядке будет разбиение, содержащее одно слагаемое, равное N, а последним — разбиение из N слагаемых, равных 1.

Как выглядит разбиение, следующее непосредственно за разбиением

$$n = c_1 + \ldots + c_k?$$

Будем искать разбиение, которое имеет самое большое число начальных слагаемых, равных начальным слагаемым данного разбиения (обозначим эти слагаемые $a_1, \ldots, a_{t-1}$) и оставшиеся слагаемые которого определяются разбиением, непосредственно следующим за разбиением

$$s = a_t + a_{t+1} + \ldots + a_k.$$

Легко видеть, что эти условия однозначно определяют значение t:

$$t=\max\{i: a_i>1\}.$$

Таким образом, задача свелась к нахождению разбиения, непосредственно следующего за разбиением

$$s=a_{t+1}+\ldots+1,$$

где $a_i>1$, а количество единиц равно $k-t$. Таким разбиением является разбиение

$$s_1=p+p+\ldots+p+(s \bmod p),$$

где $p=a_t-1$.

4. Каждый Π_i можно положить либо горизонтально, либо вертикально. Будем заполнять П от нижнего левого угла.

На каждом шаге:

1) искать в уже заполненной фигуре "нишу" с минимальными координатами левого нижнего угла;
2) для каждого из еще не использованных Π_i повторять
   брать очередной Π_i и пытаться вставить в "нишу"
   если удалось, и мы не вышли за пределы П, то
   пометить Π_i как использованный и на шаг 1
   scratch:
   пометить Π_i как неиспользованный
   конец то

конец для
если использовали все Π_i, то печать результата.
Стоп
если не использован ни один Π_i,
то решения нет
иначе возврат на scratch.

5. Во всех узлах (трубках) схемы мы должны получить состояние G_t'. Первоначально установим G_1, G_2 и G_3

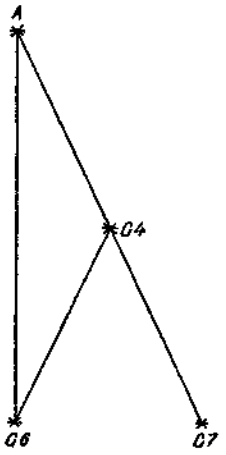


Рис. 35

в G_1', G_2' и G_3', вбрасывая, по необходимости, шарики через входы A, B и C соответственно. Для того чтобы G_1, G_2 и G_3 оставались в состоянии G_1', G_2' и G_3', необходимо, чтобы через каждый из входов забрасывалось лишь четное число шариков.

Если проследить за шариками, заброшенными в игровой автомат, то мы можем заметить, что результат хода, например, ABC аналогичен результату хода BCA, т. е. от порядка забрасывания результат не зависит.

Рассмотрим узлы, состояние которых может меняться при забрасывании шариков через вход A, так как при выбранной методике забрасывания только четного числа шариков (рис. 35) через входы всегда $G_1 = G_1'$, то состояний $G_6G_4G_7$ может быть лишь $2^3 = 8$, от 000 до 111. Если мы будем забрасывать через A пары шариков, то при забрасывании серии максимум из 8 пар шариков какое-то состояние $G_6G_4G_7$ должно обязательно повториться на протяжении серии. Далее состояния будут циклически повторяться, и случай с забрасыванием более 8 пар шариков сводится к случаю с количеством пар, не превосходящим 8.

Аналогично для забрасывания шариков через B мы получаем серию из не более чем $2^5 = 32$ пар шариков, на протяжении которой будут установлены все возможные состояния G_4, G_5, G_6, G_7, G_8, а для входа C серия, как и для A, будет иметь длину не более 8 пар.

Перебирая все возможные комбинации A_{2x}, B_{2y}, C_{2z} забрасывания шариков (тут A_{2x} обозначает серию из $2x$ забрасываний шариков через вход A, $0 \leqslant x$, $y \leqslant 8$, $0 \leqslant z \leqslant 32$), получаем либо комбинацию — решение задачи, либо определяем отсутствие решения.

6. Один из возможных алгоритмов решения задачи такой:

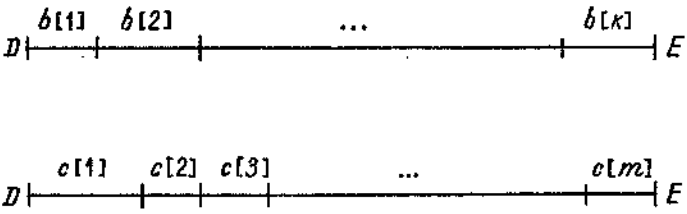


Рис. 36

1) Предположим, что существует текст, дешифровка которого неоднозначна, следовательно, существует и текст минимальной длины, для которого возможны по меньшей мере две разбивки

$$b[1]\,b[2]\ldots b[k]=c[1]\,c[2]\ldots c[m]$$

на слова множества A. Представляя текст в виде отрезка DE, эти разбивки мы можем изобразить, как на рисунке 36.

Из того, что текст минимальной длины, следует, что концы слов в разных разбивках не могут лежать на одной вертикали (кроме конца текста). Так как $b[1]\neq c[1]$, то одно из кодовых слов (например, $b[1]$) должно входить в другое в качестве префикса и представляться в виде $c[1]=b[1]\,p[1]$, где $p[1]$ — оставшаяся часть слова (суффикс). Далее, либо $p[1]$ входит в $b[2]$ ($b[2]=p[1]\,p[2]$), либо $b[2]$ входит в $p[1]$ ($p[1]=b[2]\,p[1]$) в качестве префикса. Определяем новый суффикс $p[2]$. Продолжая выделять префиксы и суффиксы, получаем, что на каком-то шаге $p[j]$ совпадает с одним из кодовых слов.

Эти рассуждения являются основой следующего алгоритма.

На нулевом шаге возьмем все пары $(a[i];\ a[j],\ i\neq j$ таких кодовых слов, что одно из них есть префикс другого, и найдем все суффиксы $p[0,\ k]$.

На j-м шаге для всех пар $(p[j-1,\ k];\ a[i])$, где одно из слов является префиксом другого, опять находим все суффиксы, и те из них, которые не появлялись на предыдущих шагах алгоритма, обозначим $p[j,\ k]$.

Эти шаги повторяем либо до тех пор, пока какой-либо суффикс не совпадает с одним из кодовых слов (и тогда существует неоднозначно декодируемый текст), либо пока на очередном шаге не появится ни одного нового суффикса (и тогда для любого текста существует единственная дешифровка).

Так как количество кодовых слов ограничено, то и суффиксов также конечное число, и на каком-то шаге алгоритм обязательно остановится.

Этот алгоритм в первоначальном варианте принадлежит А. Сардинасу и Дж. Паттерсону (Кибернетический сборник. Вып. 3. С. 93—102).

2) Если существует текст, который при использовании A дешифруется неоднозначно, то начинаем выбрасывать из A слова и их комбинации, пытаясь получить множество A' с максимальным числом слов такое, что все тексты, составленные из слов A', дешифруются однозначно.

Сначала из A удаляем i-е слово, $i=1, 2, \ldots, n$, и делаем проверку по пункту 1). Если удалением одного слова из A мы не получаем искомого множества A', то тогда генерируем все сочетания по $n-2$ слова из множества A и для каждого полученного множества делаем проверку по пункту 1) и т. д.

Воспользуемся алгоритмом (5.2) генерации сочетаний по i слов из множества A.

3) Этот пункт реализуется следующим рекурсивным алгоритмом поиска с возвращением (все слова массива A упорядочены по номерам):

а) если строка пустая, то одна из возможных дешифровок найдена, иначе при разборе текста мы проверяем $a[i]$ (при изменении i от 1 до n) на вхождение в начало дешифруемой в данный момент строки.

б) если какое-то $a[i]$ входит в строку как префикс, то запоминаем номер i этого слова, затем выделяем слово из строки, а с остатком текста производим операцию разбора по пункту а).

Если ни одно из $a[i]$ не входит в качестве префикса в дешифруемую сейчас строку, то осуществляем возврат на пункт а), предварительно добавляя в начало строки последнее удаленное оттуда слово, и пытаемся выделить из текста слово с большим номером в A. Если возврат осуществить невозможно (так как мы находимся в начале исходной строки), то алгоритм заканчивает свою работу. Все возможные дешифровки найдены.

Задача 1. Докажите следующую теорему (Марков А. А.):

Для однозначности декодировки текста достаточно выполнения одного из двух следующих условий:

1) не существует ни одной пары кодовых слов ($a[i]$, $a[j]$), $i<>j$, такой, что одно из этих слов есть префикс другого.
2) не существует ни одной пары кодовых слов ($a[i]$, $a[j]$), $i<>j$, такой, что одно из них есть суффикс другого.

Задача 2. Приведите пример такого множества кодовых слов, что для него не выполняется ни условие 1, ни условие 2 из задачи 1, а декодировка любого текста, составленного при помощи этих слов,— однозначная.

```
A:= y2-y1;
B:= x1-x2;                                          (1.1)
C:= -x1*(y2-y1)+y1*(x2-x1);
```

```
P:=false;
if (x3-x1)*(y2-y1)-(y3-y1)*(x2-x1)=0 then
   P:=true;                                         (1.2)
```

```
L:="по одну";
Z1:=(x3-x1)*(y2-y1)-(y3-y1)*(x2-x1);
Z2:=(x4-x1)*(y2-y1)-(y4-y1)*(x2-x1);
if Z1*Z2<0 then
   L:="по разные";                                  (1.3)
```

```
P:=true;
Z1:=(x3-x1)*(y2-y1)-(y3-y1)*(x2-x1);
Z2:=(x4-x1)*(y2-y1)-(y4-y1)*(x2-x1);
if Z1*Z2>0 then
   P:=false;                                        (1.4)
Z3:=(x1-x3)*(y4-y3)-(y1-y3)*(x4-x3);
Z4:=(x2-x3)*(y4-y3)-(y2-y3)*(x4-x3);
if Z3*Z4>0 then
   P:=false;
```

```
A:=y2-y1;
B:=x1-x2;
C:=-x1*(y2-y1)+y1*(x2-x1);                          (1.5)
T:=SQRT(A*A+B*B);
D:=ABS((A*x3+B*y3+C)/T);
```

```
L:="Выпуклый"
for i:=1 to n do
begin
   j:=i mod n+1;
   k:=j mod n+1;
   m:=i-1;
   if i=1 then
```

```
    m:=n;
  Z1:=(x[m]-x[i])*(y[j]-y[i])-(y[m]-y[i])*(x[j]-x[i]);
  Z2:=(x[k]-x[i])*(y[j]-y[i])-(y[k]-y[i])*(x[j]-x[i]);
  if Z1*Z2<0 then
    L:="Невыпуклый";
end;
```

(1.6)

```
p:=(a+b+c)/2
S:=SQRT(p*(p-a)*(p-b)*(p-c));
```

(1.7)

```
ymin:=y[1];
for k:=2 to n do
  if ymin > y[k] then
    ymin:=y[k];
for k:=1 to n+1 do
  y1[k]:=y[k]-ymin;
S:=0;
for к:=1 to n do
  S:=S+(x[i+1]-x[i])*(y1[i+1]+y1[i]);
S:=ABS(S)/2;
```

(1.8)

```
xmin:=x[1];
for k:=2 to n do
  if xmin > x[k] then
    xmin:=x[k];
x:=xmin-1;
y:=y[0];
S:=0;
for i:=1 to n do
begin
  Z1:=(x[i]-x[0])*(y-y[0])-(y[i]-y[0])*(x-x[0])
  Z2:=(x[i+1]-x[0])*(y-y[0])-(y[i+1]-y[0])*(x-x[0])
  if Z1*Z2<0 then
    S:=S+1;
  end;
  L:="внутри";
  if S mod 2 = 0 then
    L:="вне";
```

(1.9)

```
P:=false;
for i:=1 to N do
   if A[i]=X then                                    (2.1)
      P:=true;
```

```
K:=0;
for i:=1 to N do
   if A[i]=X then                                    (2.2)
      K:=i
```

```
i:=1;
while (i<=n) and (A[i]<>X)                           (2.3)
   i:=i+1
```

```
A[N+1]:=X; i:=1;
while (A[i]<>X) do                                   (2.4)
   i:=i+1
```

```
K:=0;
for i:=1 to N do
   if A[i]=X then
   begin
      K:=K+1;
      B[K]:=i;                                       (2.5)
   end
```

```
max:=A[1];
for i:=2 to N do
   if A[i]>max then                                  (2.6)
      max:=A[i];
```

```
max:=A[1]; K:=1;
for i:=2 to N do
   if A[i]>max then                                  (2.7)
   begin
      max:=A[i];
      K:=i;
   end;
```

```
for i:=1 to N-1 do
begin
```

```
   K:=i; max:=A[i];
   for j:=i+1 to N do
      if A[j]>max then                                    (2.8)
      begin
         max:=A[j];
         K:=j;
      end;
   A[K]:=A[i]; A[i]:=max;
end;
```

```
for i:=1 to N –1 do
   for j:=1 to N – i do
      if A[j]<A[j+1] then
      begin
         x:=A[j];                                         (2.9)
         A[j]:=A[j+1];
         A[j+1]:=x;
      end;
```

```
P:=true; K:=N-1;
while P=true do
begin
   P:=false; R:=K;
   for j:=1 to R do
      if A[j]<A[j+1] then
      begin
         x:=A[j];                                         (2.10)
         A[j]:=A[j+1];
         A[j+1]:=x;
         P:=true;
         K:=j;
      end;
end;
```

```
L=1; R:=N; P:=false;
while (L<=R) and (P=false) do
begin
   m:=(R+L) div 2;
   if A[m]=X then
      P:=true
```

```
   else
      if A[m]>X then                                    (2.11)
         L:=m+1
      else
         R:=m-1;
end;
```

```
L=1; R:=N;
while (L<R)  do
begin
   m:=(R+L) div 2;
   if A[m]>X then
      L:=m+1                                            (2.12)
    else
      R:=m;
end;
if a[R]=X then
   p:=true
else
   p:=false
```

```
for i:=2 to n do
begin
   r:=i;  l:=1;
   while (l<r) do
   begin
      m:=(l+r) div 2;
      if a[m]>a[i] then
         l:=m+1                                         (2.13)
      else
         r:=m;
   end;
   k:=r;  x:=a[i];
   for j:=i downto k+1 do
      a[j]:=a[j-1];
   a[k]:=x;
end;
```

```
Ai: = 1; Bi: = 1; Ci: = 1;
while Ci< = N + M do
```

```
begin
   if A[Ai]>B[Bi] then
   begin
      C[Ci]: = A[Ai];
      Ai: = Ai + 1
   end
   else
   begin
      C[Ci]: = B[Bi];
      Bi: = Bi + 1;
   end                                          (2.14)
   Ci: = Ci + 1;
   if (Ai>N) and (Ci<>N+M) then
      for i: = Bi to M do
      begin
         C[Ci]: = B[i];
         Ci: = Ci + 1;
      end;
   if (Bi>M) and (Ci<>N+M) then
      for i: = Ai to N do
      begin
         C[Ci]: = A[i];
         Ci: = Ci + 1;
      end;
end; { while }
```

```
if a mod b=0 then
   p:='Делится'                                 (3.1)
else
   p:='Не делится';
```

```
p:='Простое';
B[1]:=1; B[2]:=a;
K:=2;
for i:=2 to a-1 do
   if a mod i=0 then                            (3.2)
   begin
      K:=K+1;
      B[K]:=i;
```

```
      p:='составное';
   end;
```

```
p:='простое';
for i:=2 to trunc(sqrt(a))+1 do
   if (a mod i=0) and (i<a) then                    (3.3)
      p:='составное';
```

```
p:='простое';
i:=2;
while (i<=trunc(sqrt(a))+1) and (a mod i<>0) do
   i:=i+1;                                          (3.4)
if (a mod i=0) and (i<a) then
   p:='составное';
```

```
n:=(n+1) div 2;
B[1]:=2;
for i:=2 to n do
   B[i]:= 2*i-1;
j:=2;
while j<=n do
begin
   i:=j+B[j];                                       (3.5)
   while i<=n do
   begin
      B[i]:=0;
      i:=i+B[j];
        end;
        j:=j+1;
        while (B[j]=0) and (j<=n ) do
      j:=j+1;
 end;
```

```
b[1]:=2; j:=2; i:=3;
while i<=n do
begin                                               (3.6)
   k:=1;
   d:=trunc(sqrt(i))+1;
   while (b[k]<=d) and (i mod b[k]<>0) and (k<j) do
      k:=k+1;
```

```
   if b[k]>d then
   begin
      b[j]:=i;
      j:=j+1;
   end;
   i:=i+2;
end;
```

```
k:=0; d:=2;
while a>1 do
   if a mod d=0 then
   begin                                          (3.7)
      k:=k+1;
   B[k]:=d;
   a:=a div d;
end
else
   if d=2 then
      d:=d+1
   else
      d:=d+2;
```

```
while a<>b do
   if a>b then
      a:=a-b                                      (3.8)
 else
      b:=b-a;
nod:=a;
```

```
while (a<>0) and (b<>0) do
   if a>b then
      a:=a mod b
   else
      b:=b mod a;                                 (3.9)
if a=0 then
   nod:=b
else
   nod:=a;
```

```
k:=0;
while N>=1 do
begin
   k:=k+1;
   B[k]:=N mod 10;
   N:= N div 10;                                    (3.10)
end;
```

```
k:=0; L:=N;
while L>=1 do
begin
   L:=L div 10;
   k:=k+1;
end;
m:=k;                                               (3.11)
while N>=1 do
begin
   B[k]:=N mod 10;
   N:=N div 10;
   k:=k-1;
end;
```

```
a:=0;
for i:=1 to N do                                    (3.12)
   a:=a*10+B[i];
```

```
a:=0; r:=1;
for i:=1 to N do
begin                                               (3.13)
   a:=a+B[i]*r;
   r:=r*10;
end;
```

```
k:=0;
while N>=1 do
begin
   k:=k+1;                                          (3.14)
   B[k]:=N mod p;
   N:=N div p;
end;
```

```
s:=0; r:=1;
for i:=1 to N do
begin                                    (3.15)
  s:=(s+B[i]*r) mod m;
  r:=r*10 mod m
end;
```

```
if N>M then
   K:=N
else
  K:=M;
K:=K+1;
for i:=1 to K do
  C[i]:=0;
for i:=1 to K do
begin
  C[i]:=A[i]+B[i]+C[i];
  if C[i]>=10 then                       (3.16)
  begin
     C[i+1]:=C[i+1]+1;
     C[i]:=C[i] mod 10;
  end;
end;
if C[K]=0 then
  K:=K-1;
```

```
for i:=1 to N do
   C[i]:=0;
for i:=1 to N do
begin
   C[i]:=A[i]-B[i];
   if C[i]<0 then
   begin                                 (3.17)
      C[i]:=C[i]+10;
      C[i+1]:=C[i+1]-1;
   end;
end;
while C[N]=0 do
   N:=N-1;
```

```
K:=M+N;
for i:=1 to K do
   C[i]:=0;
for i:=1 to M do
   for j:=1 to N do
      C[i+j-1]:=C[j]*B[i]+C[i+j-1];
for i:=1 to k do                                    (3.18)
begin
   C[i+1]:=C[i+1]+C[i] div 10;
   C[i]:=C[i] mod 10;
end;
while C[k]=0 do
   K:=K-1;
```

```
S[0]:=0;
for i:=1 to N do                                    (4. 1)
   S[i]:=S[i-1]+a[i];
```

```
S[0]:=1;
a[0]:=1;
for i от 1 to N do                                  (4. 2)
begin
   a[i]:=a[i-1]/x;
   S[i]:=S[i-1]+a[i];
end;
```

```
F[0]:=1;
F[1]:=1;
for i:=2 to N do                                    (4. 3)
   F[i]:=F[i-1]+F[i-2];
```

```
a:=1;
b:=1;
for i:=2 to N do                                    (4. 4)
begin
   c:=b+a;
   a:=b;
   b:=c;
end;
```

```
P[0]:=1;
for i:=1 to N do                                    (4. 5)
   P[i]:=P[i-1]*a[i];
```

```
B[1,1]:=A[1,1];
for j:=2 to 6 do
   B[1,j]=max(B[1,j-1], A[1,j]);
for i:=2 to 5 do                                    (4. 6)
   B[i,1]=max(B[i-1,1], A[i,1]);
for i:=2 to 5 do
   for j:=2 to 6 do
   begin
      B[i,j]=max(B[i,j-1],B[i-1,j]);
      B[i,j]=max(B[i,j],A[i,j]);
   end;
```

```
K[1]:=1;
K[2]:=2;
for i:=3 to 10 do                                   (4. 7)
   K[i]:=K[i-1]+K[i-2];
```

```
B[1,1]:=a[1,1];
for j:=2 to 6 do
   B[1,j]=A[1,j];
for i:=2 to 5 do                                    (4. 8)
   B[i,1]=A[i,1];
for i:=2 to 5 do
   for j:=2 to 6 do
   if A[i,j]=1 then
   begin
      B[i,j]=min(B[i,j-1],B[i-1,j]);
      B[i,j]=min(B[i,j],B[i-1,j-1])+1;
   end
   else
      B[i,j]:=0;
```

```
T[0,0]:=0;
for j:=1 to 16 do
   T[0,j]=0;
for i:=1 to 5 do                                    (4. 9)
   T[i,0]=0;
```

```
for i:=1 to 5 do
   for j:=1 to 16 do
      if j>=M[i] then
         T[i,j]=max(T[i-1,j],T[i-1,j-M[i]]+C[i])
      else
         T[i,j]=T[i-1,j];
```

```
for i:=0 to n do
   p[i]:=i;
while p[0]=0 do
begin
   for i:=1 to n do
      write(p[i]:3);
   writeln;
   j:=N;
   while p[j-1]>p[j] do
      j:=j- 1;
  k:=N;
  while p[j-1]>p[k] do
     k:=k-1;                                    (5.1)
  d:=p[j-1];
  p[j-1]:=p[k];
  p[k]:=d;
  for i:=j to n do
     r[i]:=p[N-(i-j)];
  for i:=j to n do
        p[i]:=r[i];
   end;
```

```
for i:=1 to m do
   b[i]:=i;
repeat
   for i:=1 to m do
   write(b[i]:3);
writeln;
j:=m;
while (j>0) and (b[j]>=n+j-m) do
   j:=j-1;
if j<>0 then                                    (5.2)
begin
```

```
      b[j]:=b[j]+1;
      for k:=j+1 to m do
         b[k]:=b[k-1]+1;
   end;
until j=0;
```

```
for i:=0 to m do
   A[i]:=0;
while A[m]=0 do
begin
   i:=0;
   A[0]:=A[0]+1;
   while (A[i]=n) and (A[m] = 0)  do
   begin
      A[i]:=0;
      A[i+1]:=A[i+1]+1;
      i:=i+1;                                    (5.3)
   end;
   for i:=m-1 downto 0 do
      write(A[i],' ');
   writeln;
end;
```

```
while B[n]=0 do
begin
   i:=0;
   while (B[i]=1) do
   begin
      B[i]:=0;
      i:=i+1;                                    (5.4)
      end;
      B[i]:=1;
      for i:=0 to n-1 do
         write(B[i],' ');
      writeln;
   end;
```

```
   summa:=0;
   for i:=1 to n do
      summa:=summa+a[i]
```

```
otvet:='нет решения'
if summa mod 2 = 1 then
  writeln('нет решения')
else
begin
  summa:=summa div 2;
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if a[i]<a[j] then
      begin
        k:=a[i];
        a[i]:=a[j];
        a[j]:=k;
      end;
  st[0]:=1;
  st[1]:=1;
  s:=a[1];
  i:=st[1]+1;
  if a[1]>summa then
  begin
    otvet:='нет решения';
    st[1]:=0;
  end;
  if a[1]=summa then
  begin
    otvet:='есть решения';
  writeln(a[1]);
end;
while (st[1]=1) and (otvet='нет решения') do
begin
  while (i<=n) and (otvet='нет решения') do
  begin  {while}
    s:=s+a[i];
    if s=summa then
    begin
      writeln;
      writeln('решение:');
      for j:=1 to stack[0] do
      write(a[stack[j]]:8);
      writeln(a[i]:8);
      writeln(ss,' ',s);
```

```
            halt;
          end
          else
          begin                                          (5.5)
            if s<summa then
            begin
              st[0]:=st[0]+1;
              st[st[0]]:=i;
            end
            else
              s:=s-a[i];
            i:=i+1;
          end;
        end; {while)
        i:=st[st[0]]+1;
        s:=s-a[st[st[0]]];
        st[0]:=st[0]-1;
      end; { stack[1] }
      writeln;
      writeln('нет решения. ');
end; {else}
```

СОДЕРЖАНИЕ

Учебное издание

Котов Владимир Михайлович
Волков Игорь Анатольевич
Лапо Анжелика Ивановна

ИНФОРМАТИКА
Методы алгоритмизации

Учебное пособие для 8—9 классов
общеобразовательной школы
с углубленным изучением информатики
с русским языком обучения

Редактор *Н. Г. Левчук.* Художественный редактор *Л. В. Павленко.* Технический редактор *С. И. Лицкевич.* Корректоры *З. Н. Гришели, С. А. Янович.*

Сдано в набор 29.08.2000. Подписано в печать 16.10.2000. Бумага офсетная № 1. Формат 84×108$^{1}/_{32}$. Гарнитура литературная. Высокая печать с ФПФ. Усл.-печ. л. 15,96. Усл. кр.-отт. 16,38. Уч.-изд. л. 12,12. Тираж 28 100 экз. Заказ 1566.

Налоговая льгота — Общегосударственный классификатор Республики Беларусь ОКРБ-98, ч. 1; 22.11.20.100.

Издательское республиканское унитарное предприятие «Народная асвета» Государственного комитета Республики Беларусь по печати. Лицензия ЛВ № 4 от 08.09.2000. 220600, Минск, проспект Машерова, 11.

Республиканское унитарное предприятие «Полиграфический комбинат имени Я. Коласа». 220600, Минск, Красная, 23.

Котов В. М. и др.

К95 Информатика. Методы алгоритмизации: Учеб. пособие для 8—9-х кл. общеобразоват. шк. с углубл. изучением информатики с рус. языком обучения / В. М. Котов, И. А. Волков, А. И. Лапо.— Мн.: Нар. асвета, 2000.—300 с.: ил.

ISBN 985-12-0259-2

УДК [002.6+681.3](075.3=82)
ББК 32.81я721.6