

VI - барсук, норка прошлые.  
VII - прыгунья норковая.

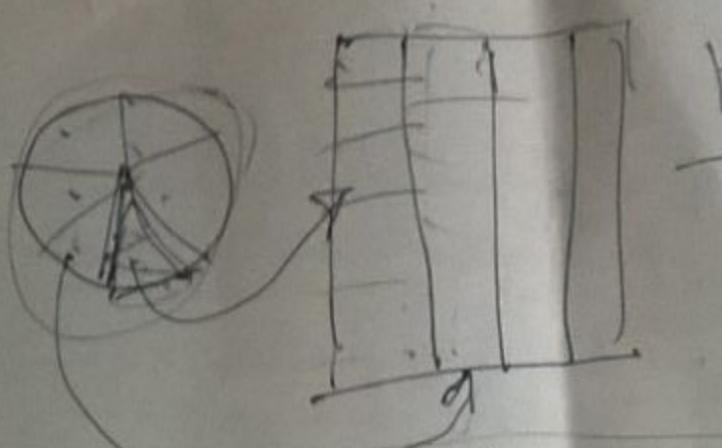
$$\left(\frac{c_i}{c_i}\right)' = 1.$$

Vi - разные программы

V - общее количество операторов на языке

Ui - место введенное для программы в оперативную память

$\frac{Vi}{V}$  - идентичной вариации когда где каждый программы отдаёт все определённые



$\forall i, v_i, n_i$

$$(1) \left\{ \begin{array}{l} \sum_i \frac{v_i}{u_i} n_i \rightarrow \min \\ \sum_i u_i \leq V \\ \forall i, u_i \geq 0. \end{array} \right.$$

$$(2) \left\{ \begin{array}{l} \forall i, \frac{v_i}{u_i} \rightarrow \min \\ \sum_i u_i \leq V_E \\ \forall i, u_i \geq 0. \end{array} \right.$$



$$U_{in} = \{v, v, \dots, v\}$$

решения квадратного уравнения, приведенного на рис. 2.1, при условии, что:

- а) целью оптимизации является минимизация нижней границы времени счета;
- б) допустимое деление на программные единицы определяется топологией подграфов, выделенных на рис. 2.1, б-г;
- в) затраты ресурсов ЭВМ на реализацию приведены на дереве игры, изображенном на рис. 2.2;
- г) объем оперативной памяти, используемой ЭВМ, равен 12.

Можно показать, что оптимальному решению отвечает головная программа, соответствующая на дереве игры  $G_\Gamma(X_\Gamma, U_\Gamma)$  вершине 1', принадлежащей первому ярусу (рис. 2.2). Это позволяет, с учетом следствия 2 теоремы 2.1, осуществлять поиск не на всем дереве, а на поддереве, изображенном на рис. 2.5, б. Расстановка потенциалов вершин в соответствии с алгоритмом 2.2 определяет оптимальную декомпозицию программы пользователя на программные единицы, отвечающие вершинам 1', 7', 3', причем цена игры  $R(\Gamma) = 6$ . Это соответствует завершению подпрограммы, отвечающей вершине 7', вектором состояния переменных, отображаемым вершиной 6. Последняя принадлежит 4-му ярусу поддерева, изображенного на рис. 2.5, б. Следует отметить, что максимальное время счета такой реализации программы связано с завершением работы подпрограммы, отображаемой вершиной 3', принадлежащей 5-му ярусу поддерева. Суммарное время счета в этом случае равно 28, что более чем в два раза превышает цену игры, определяемую алгоритмом 2.1.

Программные реализации рассмотренных выше подходов приведены в 6.2.

#### 2.1.4. ОПТИМАЛЬНАЯ ДЕКОМПОЗИЦИЯ АЛГОРИТМА ПРИ МИНИМИЗАЦИИ ОБЪЕМА ИСПОЛЬЗУЕМОЙ ОПЕРАТИВНОЙ ПАМЯТИ

Сохраняя прежнюю классификацию программистов, рассмотрим стратегию поиска оптимальной реализации конечного ветвящегося алгоритма программистом-пессимистом, целью которого является минимизация объема используемой для хранения текстов программных единиц оперативной памяти при условии, что верхняя граница времени счета не превышает величины  $T$ . При этом полагаем известными верхнюю границу  $V$ , объема оперативной памяти, выделенную для хранения текстов программных единиц, и точность  $\varepsilon$ , с которой следует определить реальные затраты памяти  $V$ .

Приводимое ниже описание алгоритма поиска величины  $V$  представляет собой сочетание дихотомии [11] с алгоритмом 2.1.

##### Алгоритм 2.3

Шаг 1.  $A = 0$ .

Шаг 2.  $B = V$ .

Шаг 3.  $C = (A + B) / 2$ .

Шаг 4. С помощью алгоритма 2.1 ищется реализация алгоритма пользователя, минимизирующая верхнюю границу времени счета при условии, что объем используемой оперативной памяти не превышает величины  $C$ .

Шаг 5. Если цена игры  $R(\Gamma)$ , полученная на предыдущем шаге, не превышает величины  $T$ , то перейти к шагу 6, в противном случае — к шагу 7.

Шаг 6.  $B = C$ , перейти к шагу 8.

Шаг 7.  $A = C$ .

Шаг 8. Если  $|B - A| > \epsilon$ , то перейти к шагу 4, если же  $|B - A| \leq \epsilon$ , то — к шагу 9.

Шаг 9. Оптимальное значение  $V = B$ .

Конец алгоритма.

Решение аналогичной задачи программистом-оптимистом, осуществляющим поиск минимальной величины  $V$  при условии, что нижняя граница времени счета не превышает величины  $T$ , отвечает замене в шаге 4 приведенной выше процедуры алгоритма 2.1 на алгоритм 2.2. В обоих случаях число итераций алгоритма 2.3 пропорционально  $\log_2 V$ .

Пример 2.3. Определить минимальный объем оперативной памяти  $V$ , необходимой для реализации алгоритма, блок-схема которого приведена на рис. 2.1, а, при условии, что:

- верхняя граница времени счета не превышает величины  $T = 20$ ;
- верхняя граница величины  $V = V_s = 20$ ;
- величина  $\epsilon = 1$ .

Решение. На первой итерации величина  $C = 10$ , верхняя граница времени счета  $R(\Gamma)$  в этом случае равна 28, что превышает величину  $T$ . Соответствующее дерево игры изображено на рис. 2.7, а:  $A = 10$ ;  $B = 20$ .

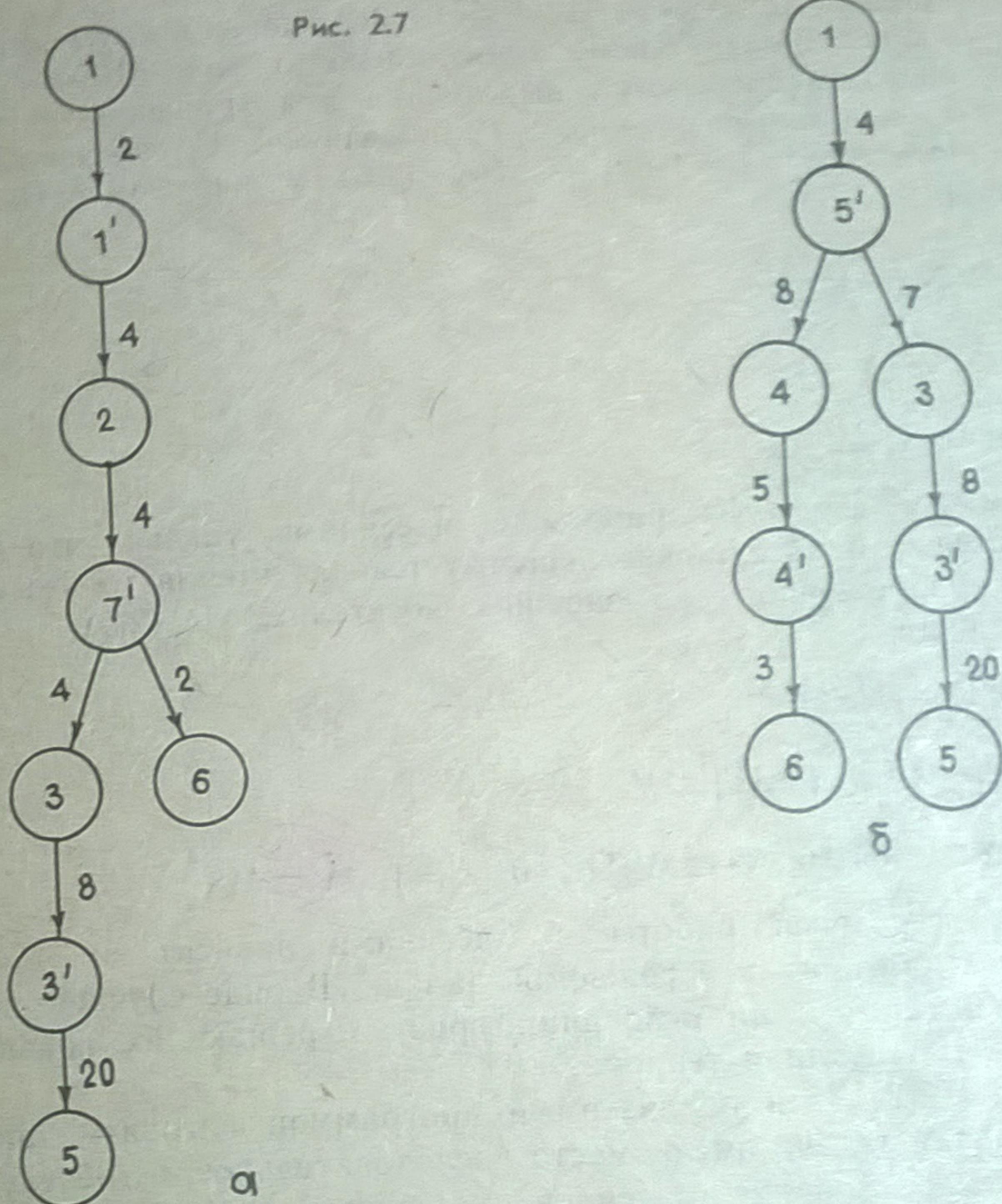
На второй итерации величина  $C = 15$ , соответствующие поддеревья игры изображены на рис. 2.7, б и 2.6, б. Первому дереву отвечает цена игры  $R(\Gamma)$ , равная 27, а второму — 10.  $A = 10$ ;  $B = 15$ .

На третьей итерации величина  $C = 12,5$ , что соответствует поддереву игры, изображенному на рис. 2.5, б. Величина  $R(\Gamma)$  равна 11, т. е. полученное решение допустимо.  $A = 10$ ;  $B = 12,5$ .

На четвертой итерации  $C = 11,25$ , что соответствует цене игры  $R(\Gamma) = 28$ , превышающей допустимое значение  $T$  (см. рис. 2.7, а). Таким образом, границы, в которых заключено оптимальное значение, соответственно равны  $A = 11,25$ ;  $B = 12,5$ .

Так как величина  $V$  целочисленна, а  $\epsilon = 1$ , дальнейший поиск может быть прекращен:  $V_{\text{опт}} = 12$ , соответствующая верхняя граница времени счета  $R(\Gamma) = 11$ .

Рис. 2.7



## 2.2. Оптимальное размещение массивов

Ниже рассматривается задача размещения массивов в памяти ЭВМ, обобщающая постановку, приведенную в 1.3.2.

### 2.2.1. ПОСТАНОВКА ЗАДАЧИ

Требуется разместить массивы, множество индексов которых обозначено  $M$ , во внешней и оперативной памяти ЭВМ таким образом, чтобы минимизировать верхнюю границу числа обращений к внешней памяти. При этом для чтения массивов, размещенных на внешних носителях, используются вспомогательные массивы-окна. Если такой массив один и объем его равен  $U$ , то формальная постановка задачи имеет вид:

$$\begin{cases} \left(1/U\right) \sum_{i \in M} W_i N_i z_i \rightarrow \min; \\ U + \sum_{i \in M} W_i (1 - z_i) \leq V_a; \\ \forall i, z_i = 1, 0; U > 0; V_a = V - V_t, \end{cases} \quad (2.5)$$

где  $W_i$  — объем  $i$ -го массива;  $N_i$  — число обращений к  $i$ -му массиву;  $V$  — объем свободной оперативной памяти используемой ЭВМ;  $V_t$  — часть объема оперативной памяти, выделенная для хранения текстов программных единиц и переменных. Если величина  $U$  фиксирована, то вводя переменную  $y_i = 1 - z_i$  задачу (2.5) можно заменить задачей о рациональности [3]:

$$\begin{cases} \sum_{i \in M} W_i N_i y_i \rightarrow \text{max}; \\ \sum_{i \in M} W_i y_i \leq V - V_t - U; \\ \forall i \in M, y_i = 1, 0. \end{cases} \quad (2.6)$$

Если же стратегия работы с массивами такова, что каждый вспомогательный массив-окно используется для чтения только «своего» массива, размещенного на внешних носителях, то (2.5) заменяется системой вид:

$$\begin{cases} \sum_{i \in M'} W_i N_i z_i / U_i \rightarrow \min; \\ \sum_{i \in M} [(1 - z_i) W_i + z_i U_i] \leq V_a; \\ \forall i \in M, z_i = 1, 0; \forall i \in M'; U_i > 0, z_i = 1; M' \subset M. \end{cases} \quad (2.7)$$

Выбор стратегии работы с массивами зависит от специфики реализуемого алгоритма и решаемой задачи. В ряде случаев он может быть сделан априори, на основании простых решающих правил. Так, справедлива следующая теорема.

**Теорема 2.2.** Если реализуемый программой алгоритм таков, что в ходе решения задачи имеют место последовательное, не чередующееся чтение и использование массивов, то более эффективной является стратегия, определяемая системой (2.5).

**Доказательство.** Пусть  $\vec{Z}$  — вектор, являющийся допустимым решением задачи (2.7), которому соответствует значение целевой функции этой задачи, равное  $S(\vec{Z})$ . Каждому вектору можно поставить в соответствие вектор  $\vec{Z}'$ , являющийся допустимым решением системы (2.5) такой, что:

$$\begin{cases} U = \sum_{i \in M} U_i z_i; \\ \forall i, z'_i = z_i. \end{cases}$$

Значение целевой функции (2.5), отвечающее вектору  $\vec{Z}'$ , обозначим  $S(\vec{Z}')$ . Тогда, обозначая  $\Delta S$  разность  $S(\vec{Z}) - S(\vec{Z}')$ , получим:

$$\begin{aligned} \Delta S &= \sum_i \frac{z_i W_i N_i}{U_i} - (1/U) \sum_i z_i W_i N_i = \\ &= \sum_i z_i W_i N_i [(1/U_i) - (1/U)] = \end{aligned}$$

$$= \sum_{i \in M} \frac{z_i N_i W_i}{U_i U} \sum_{j \neq i} U_j z_j$$

Поскольку все слагаемые неотрицательны, справедливо неравенство  $\Delta S > 0$ . Отсюда следует

$$S(\bar{Z}) > S(\bar{Z}'). \quad (2.8)$$

Так как  $\bar{Z}$  — произвольный допустимый вектор, то неравенство (2.8) остается справедливым и для случая, когда он является оптимальным. Отсюда следует справедливость теоремы.

Нарушение условий теоремы 2.2 исключает гарантию предпочтительности стратегии, определяемой системой (2.5). Так, при перемножении двух векторов, не помещающихся в оперативной памяти, очевидна рациональность использования двух массивов-окон, предназначенных для считывания одноименных компонент «своих» векторов.

## 2.2.2. ОПТИМИЗАЦИЯ РАЗМЕЩЕНИЯ МАССИВОВ В ЭВМ

Существуют различные подходы к решению системы (2.7). Простая идея поиска локально оптимального решения задачи аналогична идее, используемой для решения всей задачи в целом: на первом этапе ищется аналитическое выражение, определяющее зависимость числа обращений к внешней памяти  $Q$  от величины  $U$ , а на втором определяется оптимальное значение  $U = U_{\text{опт}}$  такое, что

$$Q(U_{\text{опт}}) = \min_U Q(U).$$

причем  $0 \leq U \leq V_a$ .

На завершающем этапе ищется оптимальное решение задачи (2.6) при условии, что  $U = U_{\text{опт}}$ . Ниже приводится более подробное описание этой процедуры.

### Алгоритм 2.4

Шаг 1. Ввод числа шагов  $m$ , определяющих точность вычислений.

Шаг 2.  $i = 1$ .

Шаг 3.  $U = (V - V_t)i/m$ .

Шаг 4. Для найденной на предыдущем шаге величины  $U$  решается задача (2.6).

Шаг 5. Определяется величина  $Q(i)$ :

$$Q(i) = \sum_{j \in M} (1 - y_j) W_j N_j$$

Шаг 6. Если  $i = m$ , то перейти к шагу 8, нет — к шагу 7.

Шаг 7. Величину  $i$  увеличить на единицу и перейти к шагу 3.

Шаг 8. Каждому значению  $Q(i)$  ставится в соответствие  $U(i) = V_a i / m$ , после чего ищется аналитическая зависимость  $Q(U)$  на интервале  $[(V_a / m) - (V_a(m-1) / m)]$ .

#### 6.4. Размещение массивов в памяти ЭВМ

Программой ALFAБ, приведенной ниже, осуществляется поиск глобально оптимального решения задачи (2.7): информационные массивы размещаются в оперативной памяти и на внешних носителях таким образом, чтобы минимизировать число обращений к последним. Это достигается созданием в оперативной памяти специальных массивов-окон, каждый из которых предназначен для чтения «своего» файла. Программа реализует метод типа ветвей и границ, осуществляющий поиск с возвратом [3]. Для вычисления оценки используется выражение (1.10). В программе ALFAБ приняты следующие обозначения:

$N$  — число массивов пользователя (величина  $N \leq 100$ ), размещаемых в ЭВМ;

$B$  — объем оперативной памяти, выделенной для массивов;

$i$ -й элемент массива  $A$  содержит величину, равную объему  $i$ -го массива пользователя;

$i$ -й элемент массива  $C$  равен произведению размера  $i$ -го массива на число обращений к нему ( $C_i = W_i \cdot N$ );

$i$ -й элемент массива  $X$  равен единице, если  $i$ -й массив пользователя размещается на внешних носителях, и нулю в противном случае;

$i$ -й элемент массива  $U$  равен нулю, если  $i$ -й элемент массива  $X$  равен нулю, и размеру массива-окна, предназначенного для чтения  $i$ -го массива пользователя, если  $i$ -й элемент массива  $X$  равен единице.

Программа построена таким образом, что всякий раз, когда достигается решение (2.7), лучшее, чем предыдущее, оно выдается на печать. Это позволяет в задачах большой размерности получать локально оптимальные решения по истечении времени счета до завершения анализа всех вариантов обхода дерева ветвлений — последнее полученное решение является наилучшим.

Блок DATA содержит исходные данные, отвечающие условиям примера 2.3.

```
4 REM ПАКЕТ ALFA 5
5 OPEN "LP:" FOR OUTPUT AS FILE #1
10 DIM A(100), C(100), X(100), XI(100), U(100), UI(100)
20 DATA 3,60,20,80,10,50,40,40
30 READ N, B
40 FOR I=1 TO N
50 READ A(I), C(I)
60 NEXT I
70 KI=0
80 R=99999
90 I=1
100 X(I)=1
110 KI=KI+1
120 FI=0
```

```

130 FOR J=1 TO I
140 F1=F1+A(J) * (I - X(J))
150 NEXT J
160 F=0
170 F2=0
180 FOR J=1 TO I
190 F2=F2+SQR(X(J) * C(J))
200 NEXT J
210 IF F2=0 THEN 290
220 FOR J=1 TO I
230 F0=(B - F1) * SQR(C(J) * X(J))/F2
240 U(J)=INT(F0+.5)
250 IF F0=0 THEN 280
260 IF INT(X(J) * C(J)/F0)=X(J) * C(J)/F0 THEN F=F+
+X(J) * C(J)/F0
270 IF INT(X(J) * C(J)/F0)<>X(J) * C(J)/F0 THEN F=F+
+INT(X(J) * C(J)/F0)
280 NEXT J
290 GOSUB 630
300 IF F1>B THEN F=99999
310 IF F>R THEN 470
320 Q=0
330 FOR M=1 TO I
340 Q=Q+A(M) * X(M)
350 NEXT M
360 IF Q>B THEN 470
370 IF I>=N THEN 400
380 I=I+1
390 GOTO 100
400 R=F
410 PRINT #1, "ТЕКУЩЕЕ ЗНАЧЕНИЕ РЕКОРДА РАВНО"; R
420 PRINT #1, "ТЕКУЩИЕ ЗНАЧЕНИЯ ВЕКТОРОВ X И U"
430 FOR K=1 TO N
440 PRINT #1, "X("; K; ")="; X(K); "U("; K; ")="; U(K)
450 X1(K)=X(K)
455 U1(K)=U(K)
460 NEXT K
470 IF X(I)=0 THEN 510
480 X(I)=0
490 K1=K1+1
500 GOTO 120
510 IF I=1 THEN 540
520 I=I-1
530 GOTO 470
540 Q1=K1 * 100/(2 * (2^N - 1))

```

```

550 PRINT #1, "ГЛОБАЛЬНО ОПТИМАЛЬНОЕ ЗНАЧЕНИЕ РЕ-
КОРДА ="; R
560 PRINT #1, "ЧИСЛО ВЫЧИСЛЕННЫХ ОЦЕНОК РАВНО"; K1
570 PRINT #1, "ЧТО СОСТАВЛЯЕТ"; Q1; "ПРОЦЕНТОВ ОТ ИХ
ОБЩЕГО ЧИСЛА"
580 PRINT #1, "ОПТИМАЛЬНЫЕ ВЕКТОРЫ X И U"
590 FOR K=1 TO N
600 PRINT #1, "X1("; K; ") ="; X1(K); "U1("; K; ") ="; U1(K)
610 NEXT K
620 STOP
630 T=0
640 T1=0
650 FOR J=1 TO I
660 T=T+X(J)
670 T1=T1+(1-X(J))*A(J)
680 NEXT J
690 IF T1 < B THEN 710
700 IF T1 > 0 THEN F=99999
710 RETURN
720 CLOSE #1
730 END

```

Ниже приводится распечатка, отображающая поиск решения задачи (27) применительно к условиям примера 2.3 программой ALFA5: получению глобально оптимального решения предшествовал анализ трех планов, причем полученное решение практически не отличается от полученного в гл. 2.

ТЕКУЩЕЕ ЗНАЧЕНИЕ РЕКОРДА РАВНО 14  
ТЕКУЩИЕ ЗНАЧЕНИЯ ВЕКТОРОВ X И U

X ( 1 ) = 1 U ( 1 ) = 11  
X ( 2 ) = 1 U ( 2 ) = 9  
X ( 3 ) = 0 U ( 3 ) = 0

ТЕКУЩЕЕ ЗНАЧЕНИЕ РЕКОРДА РАВНО 5  
ТЕКУЩИЕ ЗНАЧЕНИЯ ВЕКТОРОВ X И U

X ( 1 ) = 1 U ( 1 ) = 29  
X ( 2 ) = 0 U ( 2 ) = 0  
X ( 3 ) = 1 U ( 3 ) = 21

ТЕКУЩЕЕ ЗНАЧЕНИЕ РЕКОРДА РАВНО 2  
ТЕКУЩИЕ ЗНАЧЕНИЯ ВЕКТОРОВ X И U

X ( 1 ) = 0 U ( 1 ) = 0  
X ( 2 ) = 0 U ( 2 ) = 0  
X ( 3 ) = 1 U ( 3 ) = 30

ГЛОБАЛЬНО ОПТИМАЛЬНОЕ ЗНАЧЕНИЕ РЕКОРДА = 2