И. В. РОМАНОВСКИЙ

АЛГОРИТМЫ РЕШЕНИЯ ЭКСТРЕМАЛЬНЫХ ЗАДАЧ

И. В. РОМАНОВСКИЙ

АЛГОРИТМЫ РЕШЕНИЯ ЭКСТРЕМАЛЬНЫХ ЗАДАЧ

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
Москва 1977

518
Р 69
УДК 519.9

Алгоритмы решения экстремальных задач. И. В. Романовский. Главная редакция физико-математической литературы изд-ва «Наука», М., 1977.

В книге излагаются теория и численные методы решения важных классов экстремальных задач: общей задачи линейного программирования, транспортной задачи и задач, ей родственных, комбинаторных задач на графах, ряда дискретных задач динамического программирования.

Изложение численных методов сопровождается разбором алгоритмов, записанных на алгоритмических языках алгол-60 и алгол-68; при этом особое внимание уделено вопросам представления данных при эффективной организации вычислительного процесса.

Книга рассчитана на студентов-старшекурсников и аспирантов математических факультетов, специализирующихся в приложениях, а также на сотрудников научно-исследовательских учреждений и вычислительных центров, связанных с разработкой численных методов решения экстремальных задач.

Илл. 73, табл. 37, библ. 233 назв.

Иосиф Владимирович Романовский

АЛГОРИТМЫ РЕШЕНИЯ ЭКСТРЕМАЛЬНЫХ ЗАДАЧ

М., 1977 г., 352 стр. с илл.

Редакторы *Н. П. Рябенькая, Г. Я. Пирогова*
Техн. редактор *Н. В. Кошелева*
Корректор *Е. В. Сидоркина*

Сдано в набор 5. 11. 1976 г. Подписано к печати 20. 04. 1977 г. Формат 84×108$^{1}/_{32}$, тип. № 3. Физ. печ. л. 11. Услови. печ. л. 18,48. Уч.-изд. л. 19,05. Тираж 16 000 экз. Т-08427. Цена книги 1 р. 44 к. Заказ № 919.

Издательство «Наука»
Главная редакция физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

Ордена Трудового Красного Знамени Ленинградское производственно-техническое объединение «Печатный Двор» имени А. М. Горького Союзполиграфпрома при Государственном комитете Совета Министров СССР по делам издательств, полиграфии и книжной торговли, 197136, Ленинград, П-136, Гатчинская ул., 26.

Р $\frac{20204—069}{053\ (02-)77}$ 48-77

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

В этой книге излагаются в различных аспектах — от теоретического исследования до программы — методы решения экстремальных задач, главным образом дискретных.

Серьезные экстремальные задачи решаются на вычислительных машинах, и это обстоятельство, несомненно, должно влиять на разработку, изучение и изложение методов решения экстремальных задач. Речь должна идти об эффективной организации вычислительного процесса, при разработке которой следует учитывать многие «технические» и кажущиеся несущественными детали — структуру вычислительной машины, формы представления данных, особенности используемых алгоритмических языков и соответствие их грамматических конструкций машинным возможностям.

Имеется широкий круг практически важных задач, в которых «вычислительная» в традиционном понимании часть действий оказывается очень скромной, и основные сложности лежат в логической структуре алгоритма и в вопросах представления исходной и вспомогательной информации. В таких задачах (а размеры и время счета в них часто бывают значительными) обычно особенно существенно правильно организовать вычисления.

Цель этой книги — показать в действии некоторые способы организации вычислений, используемые в современных методах решения экстремальных задач. Однако нельзя достичь хорошего понимания этих вопросов, если формализация задачи отделена от выбора метода решения, метод решения — от возможностей алгоритмического языка, запись алгоритма — от понимания особенностей вычислительной машины и класса решаемых задач.

Очевидно, что задача влияет на метод, метод на алгоритм, алгоритм на ход вычислений. Важно отметить, что хороший алгоритм должен учитывать специфику вычислительного процесса, хороший метод вычислений — конструкции алгоритмического языка, хорошая постановка задачи — понятия, характерные для предполагаемого численного метода.

Таким образом, деление изложения на «теоретическую», «алгоритмическую» и «чисто программистскую» части представляется не только неестественным, но даже и вредным, и было сделано все возможное, чтобы соединить эти части в единое целое и по возможности облегчить переходы. С этой целью была выбрана система обозначений, в которой такие переходы не вызывают затруднений. Для демонстрации готовых алгоритмов и отдельных конструкций используются два языка: алгол-60 и алгол-68, и такой выбор требует некоторых пояснений.

Было бы достаточно алгола-68, который очень хорошо подходит для всех приводимых в книге алгоритмов. Этот язык лаконичен и нагляден, имеет гибкие способы работы со структурами, четкую и осмысленную грамматику. Поэтому в качестве языка публикации он предпочтительнее, чем, скажем, ПЛ/1. Опыт показывает, что переписать программу с алгола-68 на ПЛ/1 совсем несложно, и такая программа может гораздо лучше использовать особенности вычислительной машины (например, Единой системы ЭВМ) чем программа на фортране или алголе-60.

Языка алгол-60 для поставленных целей недостаточно. Тем не менее записи на языке в книге оставлены с учетом его широкой известности, наличием трансляторов на большинстве вычислительных машин, а также для сопоставления различных форм описания алгоритмов и представления информации.

Некоторые вводные сведения о характерных приемах представления данных и описания алгоритмов излагаются в гл. 1. Здесь же вводится упомянутая система обозначений и, поскольку алгол-68 распространен еще не очень широко, разъясняются некоторые характерные его конструкции.

Хотя основной нашей целью являются дискретные экстремальные задачи, линейное программирование может

рассматриваться как естественное начало. Поэтому в гл. 2 в лаконичной форме приведены основные факты теории линейного программирования, постановки задач, метод последовательного улучшения допустимого базисного решения и различные подходы к его вычислительной реализации.

В гл. 3 вводятся определения теории графов, которые затем интенсивно используются во всем дальнейшем изложении, после чего начинается изучение задач, связанных с транспортными потоками. Эти задачи интересны не только своими непосредственными практическими приложениями, но и неожиданными и интересными связями с различными разделами математики, в частности с комбинаторикой. Эти комбинаторные следствия из теории потоков и родственные им задачи собраны в гл. 4. Далее, в гл. 5 появляются более сложные комбинаторные задачи, для которых, как правило, нет хороших методов решения и в качестве основного вычислительного средства рассматривается метод улучшенного перебора.

Последние две главы посвящены специфическим задачам (задачам динамического программирования), которые можно решать рекуррентно, сводя задачу к такой же задаче меньших размеров. В гл. 6 рассматриваются комбинаторные задачи этого типа, а в гл. 7 — близкие по духу к материалу остальной книги вероятностные задачи, связанные с управляемыми марковскими цепями (марковские процессы решения).

Читателю, знакомому с литературой, бросится в глаза отсутствие многих традиционных разделов математического программирования. Это естественно — задача предлагаемой книги в том, чтобы пополнить литературу изложением вопросов, которые слабее в ней представлены, а нуждаются в не меньшем внимании. В библиографических указаниях в конце книги приводятся ссылки на «традиционную» литературу.

К сожалению, даже в тех вопросах, которые включены в книгу, изложение может показаться односторонним, и для такого впечатления есть основания. Единство стиля как в теоретических, так и в вычислительных вопросах требовало серьезной переработки каждой используемой работы, часто существенно изменяющей первоначальные идеи ее. По возможности неполнота текста компенсируется библиографическими указаниями.

Эта книга появилась в результате моей преподавательской деятельности на математико-механическом факультете Ленинградского государственного университета. Ее можно использовать как пособие для спецкурсов по методам дискретной оптимизации, читаемых на математических факультетах университетов и технических вузов.

При работе над книгой большую помощь мне оказали заведующий кафедрой исследования операций проф. М. К. Гавурин и сотрудники кафедры и лаборатории исследования операций ЛГУ. Приношу им искреннюю благодарность.

Автор

ГЛАВА 1

ПОДГОТОВИТЕЛЬНЫЕ СВЕДЕНИЯ

§ 1. Принятые обозначения

Так как основная часть этой книги посвящена вычислительным аспектам оптимального программирования, нам пришлось во многих случаях углубляться в такие *детали организации* вычислений, которые могут показаться чисто *техническими* и лишними для понимания *существа* проблем.

Между тем в том круге задач, который мы здесь рассматриваем, часто *именно эти детали являются наиболее существенными*. Поэтому мы вынуждены приводить алгоритмы и структуру данных в них и ожидаем, что читатель не будет делить текст на теоремы и алгоритмы.

Для того чтобы в этих алгоритмах было проще разбираться, были предприняты некоторые специальные усилия. Прежде всего, все алгоритмы написаны так, чтобы у читателя вырабатывались некоторые полезные стереотипы их восприятия, — одни и те же (или сходные) обозначения используются для индексов, похожих переменных, процедур и массивов. Например, число итераций, как правило, обозначается *iter*, малое число *eps* (от «епсилон»), очень большое число мы привыкли обозначать *gz* (от немецкого groβe Zahl) и т. д. Названия меток, как правило, начинаются с *mk*.

Некоторые приемы программирования описываются в следующих параграфах этой главы на простых задачах. Не все из этих задач встретятся в дальнейшем тексте буквально, — некоторые нужны лишь для описания новых конструкций и для выработки соответствующих стереотипов. В § 2 мы дадим рекомендации по чтению алгоритмов, в § 3 опишем некоторые распространенные структуры данных, а в последнем параграфе специально более подробно остановимся на операциях со списками.

Наконец, для того чтобы алгоритмы не выделялись из общего текста, нам пришлось несколько изменить систему векторно-матричных обозначений, сближая ее с той, которая представляется естественной в описаниях алгоритмов. Используемая нами система обозначений описана в этом параграфе.

Нам будет удобно в обозначении вектора иметь множество индексов его компонент. Таким образом, вектор будет рассматриваться как функция, заданная на конечном множестве, которое не обязательно будет множеством целых чисел от 1 до какого-то n. Вектор $x=\{x_i\}$, где i пробегает множество M, будет обозначаться через $x[M]$. Аналогично матрица будет рассматриваться как функция, заданная на прямом произведении двух конечных множеств. Разумеется можно само это произведение считать множеством — при этом наша матрица будет рассматриваться как вектор на этом множестве. Матрицу $Z=\|z_{ij}\|$, $i\in M$, $j\in N$, мы обозначим через $z[M, N]$.

Преимущество этой системы обозначений видно в тех случаях, когда требуется выделить какую-то часть («вырезку») вектора или матрицы, соответствующую некоторому подмножеству множества индексов. Если $K\subset M$, то соответствующий K «кусок» вектора x естественно обозначить через $x[K]$. Компонента вектора, имеющая индекс i, естественно обозначается через $x[i]$, — различие между компонентой вектора и однокомпонентным вектором определяется обычным различием между элементом множества и одноэлементным множеством: $x[i]$ — компонента, $x[\{i\}]$ — однокомпонентный вектор. Для матриц мы получаем больше возможностей:

$z[i, j]$ — элемент z_{ij},

$z[i, N]$ — i-я строка матрицы $z[M, N]$ (вектор),

$z[M, j]$ — j-й столбец той же матрицы (тоже вектор),

$z[K, L]$ — подматрица матрицы $z[M, N]$, в которой множеством строк является $K\subset M$, а множеством столбцов $L\subset N$.

Мы не различаем векторов-строк и векторов-столбцов. Для превращения вектора $y[K]$ в матрицу необходимо ввести еще одно множество индексов — какое-либо одноэлементное множество R. Матрица $y[K, R]$ будет соответствующей $y[K]$ одностолбцовой матрицей, а $y[R, K]$ — однострочной.

Существенно, что однотипность векторов и матриц определяется не совпадением числа компонент, а совпадением множества индексов. Этой условности должны подчиняться и действия над ними. Так, сложить или скалярно перемножить два вектора $x[M]$ и $y[N]$ можно лишь в том случае, если $M=N$. Для скалярного произведения векторов и для произведений матрицы на вектор и матрицы на матрицу мы будем всегда использовать знак $\times$ (при умножении скаляров мы также будем ставить знак умножения, чтобы иметь возможность вводить для переменных многобуквенные обозначения). Естественно, что матрицу $a[M, N]$ можно умножить справа на матрицу $b[K, L]$ только в том случае, если $N=K$. Их произведение будет обозначаться

$$c[M, L]=a[M, N]\times b[N, L].$$

Особого упоминания заслуживают нулевой вектор $\mathbf{0}[N]$ (*векторный нуль*), все компоненты которого равны 0, и *векторная единица* $\mathbf{1}[N]$ — вектор, все компоненты которого равны 1. Аналогично определяются *матричный нуль* и *матричная единица*. Для векторов (и матриц), в отличие от вещественных чисел, имеется три вида неравенств[1]):

$x[N]>y[N]$ ($x[N]$ строго больше $y[N]$),

если $x[i]>y[i]$, $i\in N$,

$x[N]\geqq y[N]$ ($x[N]$ не меньше $y[N]$),

если $x[i]\geqq y[i]$, $i\in N$,

$x[N]\geqq y[N]$ ($x[N]$ больше $y[N]$),

если $x[N]\geqq y[N]$, $x[N]\neq y[N]$.

Разность $x[N]-y[N]$ называется в этих случаях соответственно *строго положительной*, *неотрицательной* или *положительной*.

Векторную единицу удобно использовать для суммирования компонент вектора:

$$\Sigma_{i\in K}\, y[i]=\mathbf{1}[K]\times y[K].$$

---

[1]) Часто используется знак $\geqq$ для отношения «не меньше» и $\geq$ для «больше». Принятая нами здесь система обозначений введена польскими математиками. Она удобнее тем, что в ней сохраняется существующая условность неравенств для скаляров, а отношение «больше» для векторов используется чрезвычайно редко.

Неотрицательный вектор $x[N]$, сумма компонент которого равна 1, т. е.

$$1[N]\times x[N]=1, \quad x[N]\geqslant 0[N],$$

называется *вероятностным вектором*. Компоненты этого вектора могут рассматриваться как вероятности событий в некоторой вероятностной схеме с множеством исходов N. Совокупность всех вероятностных векторов с множеством индексов N называется *основным симплексом* и будет обозначаться через S_N.

Для задания матриц или их подматриц в памяти вычислительной машины в качестве множества индексов должны использоваться отрезки множества целых чисел. Множество целых чисел от k до l будет обозначаться через $k:l$. Если нам понадобится записать матрицу или вектор с каким-либо иным множеством индексов, это множество придется отобразить в отрезок множества целых чисел. Таким образом, нам необходимо предусмотреть возможности задания отображений одного множества в другое. В связи с этим нам понадобятся еще несколько обозначений.

Введем процедуру-функцию δ от условия (логической переменной) a, принимающую значение 1, когда a истинно, и 0, когда a ложно. На алголе-60 эта процедура описывается так:

```
integer procedure δ(a); value a; boolean a;
    δ:= if a then 1 else 0
```

Ее можно использовать, например, вместо символа Кронекера, так как δ_{ij} в обычном понимании здесь запишется $\delta(i=j)$. Функция Хевисайда с помощью нашей процедуры примет вид $\delta(x>0)$ [1]).

Пусть задано отображение $T: M \to N$. Матрица $\Delta[M, N]$, где $\Delta[i, j]=\delta(j=Ti)$ называется *матрицей отображения* T. Если отображение T взаимно однозначное, Δ называется *назначающей* матрицей, а если, кроме того,

---

[1]) В языке алгол-68, который мы также будем иногда использовать, эта процедура реализована одной из операций, описанных в так называемом стандартном вступлении вместе с другими стандартными средствами, именно операцией **abs**. Таким образом, мы можем написать соответственно **abs** a, **abs** $(i=j)$, **abs** $(x>0)$.

$M.=N$ — *переставляющей* матрицей. Переставляющая матрица $\Delta[M, M]$, соответствующая тождественному отображению $Ii=i$, называется *единичной* и обозначается через $E[M, M]$.

Теперь предположим, что в памяти машины требуется записать, например, матрицу $b[M, 1:n]$. В этом случае нужно задать отображение T: $M \to 1: m$, где $m=|M|$ — число элементов множества M, после чего можно будет работать с матрицей $b'[1:m, 1:n]$, в которой

$$b'[Ti, 1:n]=b[i, 1:n].$$

Связь матриц b' и b можно описать с помощью матрицы Δ отображения T

$$b[M, 1:n]=\Delta[M, 1:m]\times b'[1:m, 1:n].$$

Отображение T: $M \to N$ может быть задано также и непосредственно вектором $t[M]$, значения компонент которого суть элементы из N. Используя такое обозначение, мы можем описать связь b и b' еще проще:

$$b[M, 1:n]=b'[t[M], 1:n].$$

Векторное задание отображений удобно и при рассмотрении суперпозиций отображений: если $t[M]$ задает отображение $T: M \to N$, а $s[N]$ — отображение $S: N \to R$, то отображение $ST: M \to R$ будет задаваться вектором $s[t[M]]$.

В заключение этого параграфа введем несколько обозначений, связанных с понятиями линейной алгебры. Пусть $|M|=|N|$. Матрица $b^-[N, M]$ называется *обратной* к матрице $b[M, N]$, если

$$b^-[N, M]\times b[M, N]=E[N, N].$$

Известно, что для существования обратной матрицы необходимо и достаточно, чтобы строки матрицы $b[M, N]$ были линейно независимы, т. е. из равенства

$$y[M]\times b[M, N]=0[N]$$

следовало, что $y[M]=0[M]$. При этом

$$b[M, N]\times b^-[N, M]=E[M, M],$$

и если $y[N]$ есть решение системы

$$b[M,\ N]\times y[N]=d[M],$$

то

$$y[N]=b^{-}[N,\ M]\times d[M].$$

Ранг матрицы $a[M,\ N]$ мы будем обозначать через $\mathrm{rank}\,(a[M,\ N])$, а определитель — через $\det(a[M,\ N])$. Для вычисления определителя необходимо, чтобы $M=N$ или чтобы было задано взаимно однозначное отображение $T\colon M\to N$. От задания этого отображения зависит знак определителя. Мы будем пренебрегать этим требованием, когда речь будет идти просто о невырожденности матрицы, т. е. об условии $\det(a[M,\ N])\neq 0$.

Наконец, часто нам придется изменять множество, добавляя к нему один элемент или удаляя из него один элемент. Вместо громоздких обозначений

$$N\cup\{j\}\quad\text{и}\quad N\setminus\{j\}$$

мы будем в этом случае писать

$$N+j\quad\text{и}\quad N-j.$$

Надеемся, у читателя не вызовет затруднений и смысл выражения

$$N+j0-j1.$$

§ 2. Описания алгоритмов

Как уже отмечалось, нам придется описывать, анализировать и конструировать алгоритмы. По сути дела этому посвящена основная часть текста, так как в рассматриваемых нами комбинаторных задачах формулы встречаются очень редко и играют подчиненную роль — они служат для подготовки и обоснования вычислительного процесса.

В связи с этим можно вспомнить о происхождении формул. Они появились в математике как способ описания вычислительного процесса. В рассматриваемых нами задачах формульное описание вычислений становится мало удобным по нескольким причинам. Прежде всего, те процессы, о которых будет идти речь, имеют очень сложную логическую структуру и с помощью формул

описываются плохо. Далее, эти процессы, как правило, итеративны, и для их описания требуется индукция в переходе от комплекта информации одного шага к комплекту другого шага. Наконец, если мы описываем вычислительный процесс, предназначенный для вычислительной машины, то очень важной частью этого процесса является способ представления вычислительной информации.

Поэтому сейчас принято описывать вычислительный процесс именно как процесс. Чтобы задать процесс, нужно сказать, что представляет собой состояние процесса — комплект информации, каково начальное состояние — значение этой информации в начале процесса — и каковы правила перехода от текущего состояния к следующему. Эти правила описываются в виде последовательности пунктов с указанием условий перехода от одного пункта к другому. (По традиции, если не сказано, к какому пункту следует переходить дальше, то нужно перейти просто к следующему пункту.)

Рассмотрим небольшой пример, в котором используется такое описание вычислительного процесса. Пусть требуется перебрать все подмножества $1:n$, т. е. множества целых чисел от 1 до n. Каждое такое множество A можно задать его *характеристическим вектором* $chi[1:n]$, где $chi[i]=\delta(i \in A)$. В свою очередь каждый такой вектор можно рассматривать как двоичную запись некоторого натурального числа, и перебрать все такие векторы можно, перебрав все числа от 0 до 2^n-1 (т. е. все элементы множества $0:2^n-1$). Организация перебора чисел из отрезка натурального ряда труда не представляет — следует взять первое из чисел и последовательно прибавлять к нему по единице нужное число раз.

Теперь можно описать наш алгоритм. Состоянием вычислительного процесса будет вектор $chi[1:n]$. Начальное его значение $chi[1:n]=0[1:n]$.

Правила преобразования состояния таковы (это правила прибавления единицы к числу, представленному в двоичной системе счисления):

1. Использовать имеющийся вектор *chi* (перебор характеристических векторов ведется с какой-то целью, поэтому в вычислительном процессе перебора должно быть зарезервировано место для реализации этой цели).

2. Найти минимальное $i \in 1:n$, для которого $chi[i]=0$. Если такого i нет, т. е. если $chi\,[1:n]=1\,[1:n]$, перейти к п. 4.

3. Положить $chi[k]=0$ для $k<i$, положить $chi[i]=1$, остальные компоненты вектора оставить без изменений и перейти к п. 1.

4. Завершить вычисления.

Кое-какие детали в этом вычислительном процессе описаны, однако, неудовлетворительно. Пункты 2 и 3 нашего алгоритма на самом деле выполняются вместе, и более точное описание алгоритма выглядит так:

1. Использовать имеющийся вектор *chi*.
2. Положить целое число k равным 1.
3. Если $k>n$, то перейти к п. 6.
4. Если $chi[k]=0$, то положить $chi\,[k]$ равным 1 и перейти к п. 1.
5. Положить $chi\,[k]$ равным 0, увеличить k на 1 и перейти к п. 3.
6. Завершить вычисления.

Здесь появляется и часто используется действие «положить равным». В алгоритмических языках типа алгола для этого действия имеется специальное обозначение «: =», которым мы будем широко пользоваться в описаниях алгоритмов. Но часто проще прямо описывать алгоритм на алгоритмическом языке. Мы будем пользоваться для этой цели языком алгол-60, а в дальнейшем, когда нам потребуется язык с большими возможностями, — языком алгол-68. Читателю следует разбирать приводимые описания алгоритмов и выполнять на небольших примерах действия, которые в них описываются.

Например, разобранный только что алгоритм описывается на алголе-60 следующей процедурой:

```
procedure subsets (n, use); value n;
    integer n; procedure use;
begin integer i; integer array chi[1:n];
   for i:=1 step 1 until n do chi[i] :=0;
mk: use(chi);
   for i:=1 step 1 until n do
     if chi[i]=1 then chi[i]:=0
     else begin chi[i]:=1; go to mk end
end
```

Посмотрим, как будет выполняться оператор *subsets* (4, *печать*), т. е. как будет действовать процедура в случае $n=4$, когда действием, выполняемым над вектором *chi*, является его печать.

Прежде всего будет отведено место для параметра n и на это место будет записано число 4. Дальше будет отведено место для числа i и массива (вектора) *chi* [1:4]

n	*i*	*chi*			
4					

После первого выхода на метку *mk* получаем

n	*i*	*chi*			
4		0	0	0	0

Дальше действие передается процедуре *печать*, которая выводит (выписывает отдельно) вектор $chi=(0, 0, 0, 0)$, после чего начинает выполняться оператор цикла. При $i=1$, поскольку $chi[1]=0$, полагаем $chi[1]:=1$ и возвращаемся к *mk*. Следует вывод вектора (1, 0, 0, 0) (нумерация компонент вектора идет слева направо, а разрядов в позиционной записи числа — справа налево, поэтому запись двоичного представления числа 1 выглядит необычно). При $i=1$ заменяем $chi[1]$ на 0, а $chi[2]$ полагаем равным 1. Таким образом, после третьего выхода к *mk* процесс вычислений приходит в следующее состояние:

Рабочая информация

n	*i*	*chi*			
4	2	0	1	0	0

Результат печати

0 0 0 0 (0)
1 0 0 0 (1)

После девятого выхода к *mk* состояние вычислений таково:

Рабочая информация

n	*i*	*chi*			
4	1	1	0	0	1

Вывод

0 0 0 0 (0)
1 0 0 0 (1)
0 1 0 0 (2)
1 1 0 0 (3)
0 0 1 0 (4)
1 0 1 0 (5)
0 1 1 0 (6)
1 1 1 0 (7)
0 0 0 1 (8)

И т. д.

Простые алгоритмы достаточно понятны и по алгольской записи без всяких пояснений. Приведем в качестве примера несколько простых действий над теоретико-множественными объектами. Попутно будет введено еще несколько важных определений.

По-видимому, никаких трудностей в понимании не должно вызывать у читателя вычисление пересечения двух подмножеств множества $1{:}n$, представляемых их характеристическими векторами $a[1{:}n]$ и $b[1{:}n]$,

for $i := 1$ **step** 1 **until** n **do** $c[i] := a[i] \times b[i]$

Аналогично записывается и вычисление объединения

for $i := 1$ **step** 1 **until** n **do** $c[i] := a[i] + b[i] - a[i] \times b[i]$

Вот чуть более сложный пример: вектор $t[1{:}n]$ задает взаимно однозначное отображение T: $1{:}\ n \to 1{:}n$, требуется построить обратное отображение S, также задав его вектором $s[1{:}n]$. Этот вектор строится одним просмотром множества индексов $1{:}n$

for $i := 1$ **step** 1 **until** n **do** $s[t[i]] := i$

Если T: $1{:}n \to 1{:}k$ и S: $1{:}k \to 1{:}l$, то суперпозиция ST этих отображений, заданных соответственно векторами $t[1{:}n]$ и $s[1{:}k]$, получается оператором цикла

for $i := 1$ **step** 1 **until** n **do** $st[i] := s[t[i]]$

Несколько сложнее описываются действия, в которых участвуют разбиения множеств. Напомним, что набор подмножеств N_k, $k \in K$, множества N называется *разбиением* этого множества, если $N = \bigcup_{k \in K} N_k$ и множества N_k попарно дизъюнктны. Разбиение можно задать характеристическим вектором $\varkappa[N]$, задающим отображение $N \to K$, где $\varkappa[i] = k$ для $i \in N_k$.

Нам часто придется иметь дело с разбиениями конечных множеств. Например, в некоторых практически важных задачах появляются матрицы так называемой «блочной структуры». Блочную матрицу $a[M, N]$, изображенную на рис. 1, можно так описать с помощью разбиений: задано разбиение $\mathfrak{M} : M = \bigcup_{k \in K} M_k$ и разбиение $\mathfrak{N} : N = \bigcup_{k \in K} N_k$, заданы матрицы $a[M_k, N_k]$, $k \in K$, все остальные элементы матрицы $a[M, N]$ равны 0.

Пусть $\mathfrak{M}:N=\bigcup_{k\in K}M_k$ и $\mathfrak{N}:N=\bigcup_{l\in L}N_l$ — два разбиения одного и того же множества N. Разбиение $\mathfrak{R}:N=\bigcup_{s\in S}R_s$ называется *произведением разбиений* $\mathfrak{M}$ и $\mathfrak{N}$, если для каждого $s\in S$ найдутся такие $k\in K$ и $l\in L$, что $R_s=M_k\cap N_l$. Формально просто построить произведение двух разбиений. Для этого достаточно построить отображение σ произведения $K\times L$ в новое множество S и в качестве характеристического вектора нового разбиения принять $c[N]$, где $c[i]=\sigma(a[i], b[i])$, а $a[N]$ и $b[N]$ — соответственно характеристические векторы первого и второго разбиений. В частности, если K и L — подмножества отрезков натурального ряда, имеется простой и достаточно стандартный прием построения отображения σ: выбрав $d>\max\{k\mid k\in K\}$, следует положить

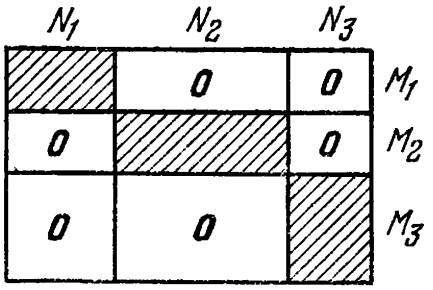

Рис. 1. Блочная матрица

$$\sigma(k,\ l)=l\times d+k. \tag{1}$$

Такая нумерация, хотя она и очень проста, неудобна, так как приводит к появлению лакун в нумерации и быстрому росту максимального номера.

Если мы хотим, чтобы элементы разбиения были занумерованы подряд, то целесообразно завести «справочник» номеров, в котором будут регистрироваться соответствия между появляющимися номерами, вычисленными по формуле (1), и порядковыми номерами справочника.

Для каждого очередного i будет вычисляться $r=\sigma(a[i], b[i])$ по (1), будут просматриваться значения r, уже встретившиеся ранее и занесенные в справочник, и если вновь вычисленное r там найдется, то его порядковый номер в справочнике и будет занесен в $c[i]$. Если же r не найдется, то справочник будет расширен на один элемент (r будет приписано в конце), после чего опять-таки в $c[i]$ будет записан номер, под которым значится r.

Опишем этот алгоритм на алголе-60, заведя для справочника целочисленный массив *ref* и храня в *kr* длину

справочника (d принято равным n):

```
kr: = 0;
for i: = 1 step 1 until n do
    begin r: = n × b[i] + a[i];
      for j: = 1 step 1 until kr do
        if ref[j] = r then begin c[i]: = j;
                                 go to mkf end;
      kr: = kr + 1; ref[kr]: = r; c[i]: = kr;
mkf: end
```

Нумерация элементов разбиения, получающаяся в результате работы этого алгоритма, обладает важной особенностью: элемент разбиения N_k получает номер меньший, чем у элемента N_l, в том и только в том случае, если наименьшее из чисел, содержащихся в N_k, меньше наименьшего из чисел, содержащихся в N_l. Это условие хорошо записывается в виде формулы

$$k < l \equiv \min\{i \mid i \in N_k\} < \min\{i \mid i \in N_l\}. \qquad (2)$$

Такая монотонная нумерация устанавливается по разбиению однозначно. В дальнейшем под характеристическим вектором разбиения мы всегда будем понимать характеристический вектор с монотонной нумерацией.

Представляет интерес задача перебора всех разбиений множества $1{:}n$. Легко описать те целочисленные векторы, которые являются характеристическими векторами разбиений. Пусть $chi[1{:}n]$ — неотрицательный целочисленный вектор, $chi[1] = 1$ и вектор $psi[1{:}n]$ определен равенством

$$psi[k] = \max\{chi[i] \mid i \in 1{:}k\}, \qquad (3)$$

т. е. $psi[k]$ — число элементов разбиения, пересекающихся с множеством $1{:}k$. Вектор psi может вычисляться так:

```
k: = psi[1]: = 1;
for i: = 2 step 1 until n do
  begin if chi[i] > k then k: = chi[i]; psi[i]: = k end
```

Для того чтобы $chi[1{:}n]$ был характеристическим вектором разбиения, необходимо и достаточно [1]), чтобы для любого $k \in 1{:}n - 1$

$$psi[k+1] \leqslant psi[k] + 1.$$

[1]) Проверку этого условия мы представляем читателю.

Будем говорить, что вектор $a[1:n]$ *лексикографически предшествует* вектору $b[1:n]$, если для наименьшего $k \in 1:n$, для которого $a[k] \neq b[k]$, имеет место $a[k] < b[k]$. В дальнейшем мы вернемся к этому определению еще раз в ситуации, где термин окажется более естественным, сейчас же нам достаточно такого определения.

Оказывается удобным перебирать характеристические векторы разбиений в порядке их лексикографического возрастания. Отметим, что каждый начальный отрезок $chi[1:k]$, $k \in 1:n$, вектора $chi[1:n]$ является, очевидно, характеристическим вектором некоторого разбиения множества $1:k$ и при фиксированном $chi[1:k]$ компонента $chi[1:k+1]$ может принимать любые значения из множества $1:psi[k]+1$. Лексикографическая упорядоченность перебора заставляет нас сохранять неизменным возможно больший начальный отрезок вектора $chi[1:n]$. Поэтому переход от имеющегося вектора $chi[1:n]$ к следующему заключается в том, что разыскивается наибольшее $k \in 1:n$, для которого $chi[k]$ может быть увеличено, а при $i > k$ значение $chi[i]$ полагается равным 1. Вычисления сильно упрощаются, если вместе с вектором *chi* хранить и перерабатывать вектор $psi[1:n]$, определенный формулой (3).

Перебор оформлен в виде процедуры, в которой обращение к процедуре *use* имеет тот же смысл, что и раньше в процедуре *subsets*:

```
procedure partitions (n, use);
    value n; integer n; procedure use;
begin integer i, j, k; integer array chi, psi[1:n];
    for i:=1 step 1 until n do chi[i]:=psi[i]:=1;
mk: use (chi);
    for i:=n step -1 until 2 do
      if psi[i]=psi[i-1] then
       begin k:=chi[i]:=chi[i]+1;
         if k>psi[i] then psi[i]:=k;
         for j:=i+1 step 1 until n do
                                    psi[j]:=psi[i];
         go to mk end
      else chi[i]:=1
end
```

В дальнейшем нам встретятся экстремальные задачи, в которых требуется выбрать разбиение некоторого множества, удовлетворяющее дополнительным ограничениям и минимизирующее некоторый функционал на множестве разбиений.

§ 3. Представление данных в вычислительной машине

При разработке алгоритма, который должен быть реализован на вычислительной машине, одной из важнейших составных частей этой работы является выбор способа представления информации, нужной для вычислений. Здесь переплетаются вопросы различного типа: прежде всего, вопрос о том, какие именно данные будут нужны для вычислений, решается неоднозначно — может оказаться удобным, кроме минимально необходимой информации, хранить некоторую вспомогательную информацию, очевидным образом восстанавливаемую по необходимой информации, но удобную для пересчета и упрощающую вычисления. Такой удобной дополнительной информацией был в предыдущем параграфе вектор *psi* в процедуре *partitions*.

Нужные для вычислений данные могут быть представлены в различной форме — как целые числа, как вещественные числа, как специальные шкалы. Наконец, очень важным является вопрос о том, как расположить в памяти машины (эта память линейна — ее можно рассматривать как последовательность ячеек, которую требуется разделить между массивами информации) необходимую информацию. Рассмотрим некоторые из возникающих здесь проблем.

Начнем с простейшего. Часто нам приходится иметь дело с векторами, в которых число ненулевых компонент значительно меньше размерности вектора. В таких случаях становится очень невыгодным представлять этот вектор в виде обычного массива, а хочется выписать лишь нужные нам компоненты. Так, если в векторе $x[1{:}n]$ имеется k ненулевых компонент, то $x[1{:}n]$ можно задать с помощью вектора $x1[1{:}k]$ того же типа, что и x, и целочисленного вектора $ind[1{:}k]$, указывающего на места ненулевых элементов.

Часто k заранее неизвестно. В этом случае приходится резервировать для ind и для $x1$ массивы достаточного размера (какой размер считается достаточным, зависит разумеется, от конкретных условий):

Легко перейти от сокращенной записи к полной:

```
for i:=1 step 1 until n do x[i]:=0;
for i:=1 step 1 until k do x[ind[i]]:=x1[i];
```

и от полной записи к сокращенной:

```
k:=0;
for i:=1 step 1 until n do if x[i]≠0 then
  begin k:=k+1; ind[k]:=i; x1[k]:=x[i] end
```

Нетрудно записать и вычисление скалярного произведения для таких «дырявых» векторов, первый из которых задан векторами $x1$ и $ind1$ и имеет $k1$ ненулевых компонент, а второй — соответственно $x2$, $ind2$, $k2$,

```
scal:=0;
for i:=1 step 1 until k1 do
for j:=1 step 1 until k2 do
  if ind1[i]=ind2[j] then scal:=scal+x1[i]×x2[j]
```

В случае, если компоненты векторов $ind1$ и $ind2$ расположены в порядке возрастания, можно обойтись одним просмотром каждого из векторов:

```
scal:=0; j:=0;
for i:=1 step 1 until kt do
begin for j:=j+1 while j≤k2 do
   if ind2[j]≥ind1[i] then
     begin if ind2[j]=ind1[i] then
             scal:=scal+x1[i]×x2[j]
           else j:=j-1; go to mkf end;
mkf: end
```

Аналогично можно записать и сложение двух векторов. Здесь, однако, приходится вести последовательную

запись всех ненулевых элементов, которые могут быть трех типов — суммы ненулевых компонент слагаемых (причем только ненулевые суммы) и ненулевые компоненты из первого или второго вектора, не имеющие «пары» из другого вектора. Слагаемые задаются, как раньше, а результат задан числом компонент $k3$ и массивами $x3$ и $ind3$:

```
j:=0; k3:=0;
for i:=1 step 1 until k1 do begin y:=x1[i];
   for j:=j+1 while j≤k2 do
     if ind2[j]=ind1[i] then begin y:=y+x2[j];
      if y=0 then go to mkf else go to mkr end
     else if ind2[j]<ind1[i] then
      begin k3:=k3+1; x3[k3]:=x2[j];
                      ind3[k3]:=ind2[j] end
     else begin j:=j-1; go to mkr end j;
mkr: k3:=k3+1; x3[k3]:=y; ind3[k3]:=ind1[i];
mkf: end i
```

Обратите внимание на механизм накопления информации о новом векторе. Такой принцип пополнения — приписывать в конец — встречался нам и раньше, при заполнении справочника индексов разбиения. Его можно расширить, добавив возможность чтения информации с конца с забыванием прочитанной информации. В этом случае говорят о *магазинном* или *стековом* хранении информации.

Рассмотрим в качестве примера использования магазина уже встречавшуюся нам задачу перебора всех подмножеств множества $1{:}n$. Теперь будем задавать подмножество списком его элементов, упорядоченным в порядке возрастания номеров. При переходе к следующему подмножеству будем пытаться прежде всего увеличить список. Если это не будет получаться, то сократим список на один элемент и увеличим на единицу последний элемент получившегося списка.

Процедура *uselist* имеет тот же смысл, что и процедура *use*, но использует не характеристический вектор, а список элементов множества:

```
procedure subsetstack (n, uselist);
    value n; integer n; procedure uselist;
begin integer k; integer array list [1:n];
    k:=0;
mk: uselist (k, list);
    if k=0 then k:=list[1]:=1
    else if list[k]<n then
      begin k:=k+1; list[k]:=list[k-1]+1 end
    else begin k:=k-1;
      if k>0 then list[k]:=list[k]+1
      else go to mkf
        end; go to mk;
mkf: end sss
```

Здесь k — количество элементов в списке, вначале $k=0$ и массив *list* фактически не используется.

Техника работы с магазинами будет для нас особенно полезна в главе 5.

Часто приходится иметь дело и с редко заполненными матрицами. Есть много способов задания таких матриц. Один из способов заключается в том, чтобы для каждого ненулевого элемента матрицы хранить два индекса — номер строки и номер столбца. Таким образом, мы получаем три массива, задающих матрицу

array *a*1 [1:*k*]; **integer array** *row*, *col* [1:*k*]

Чаще, однако, используется другой способ — массивы *a*1 и *row* вводятся так, как было сказано, а вместо того чтобы хранить массив индексов столбцов, упорядочивают информацию так, чтобы элементы шли в порядке неубывания номеров столбцов и вводят отдельный вектор *lst*[1:*n*] разметки вектора *a*1 [1:*k*]. Здесь *lst*[*j*] — номер последнего элемента, относящегося к столбцу *j*. Заметим, что работа с этим массивом оказывается более удобной, если добавить к нему еще один элемент *lst*[0]=0.

Например, матрица

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

будет представлена следующим образом:

al:	5	3	2	1	6	4	2	1	7	6	3	2	9
row:	1	3	1	2	3	4	2	2	4	3	4	1	4
lst:	0	2	4	6	7	9	11	13					

Читателю рекомендуется в качестве упражнения разобраться в следующих двух процедурах, реализующих умножение матрицы на вектор соответственно слева и справа с записью результата в виде полного вектора:

```
procedure a times x(x, ind, k, a, row, lst, k1, m, y);
   value k, k1, m; integer k, k1, m;
   array x, a, y; integer array ind, row, lst;
begin integer i, j; real x1;
   for i:=1 step 1 until m do y[i]:=0;
   for j:=1 step 1 until k do
     begin x1:=x[j];
        for i:=lst[ind[j]-1]+1 step 1 until
                                   lst[ind[j]] do
           y[row[i]]:=y[row[i]]+x1×a[i]
end end;
procedure x times a(x, ind, k, a, row, lst, k1, m, n, y);
   value k, k1, m, n; integer k, k1, m, n;
   array x, a, y; integer array ind, row, lst;
begin integer i, i1, j; real x1; i1:=0;
   for j:=1 step 1 until n do
     begin x1:=0;
        for i:=1 step 1 until k do
          begin for i1:=i1+1 while i1≤lst[j] do
             if row[i1]≥ind[j] then
               begin if row[i1]=ind[i]
                                   then x1:=x1+x[i]×a[i]
                     else i1:=i1-1;    go to mkf end;
mkf:     end;
                  y[j]:=x1 end
end
```

Во многих алгоритмах, о которых пойдет речь ниже, оказывается удобной именно запись матрицы по столбцам. Разумеется, в тех случаях, когда эта форма представления неудобна, следует предпочесть что-либо другое.

В действиях над такими объектами, как редко заполненные матрицы и векторы в рамках алгола-60 заметно некоторое неудобство, проистекающее из неопределенной заранее величины массивов. Другая трудность, более существенная и совсем плохо преодолимая в рамках алгола-60, — это трудность задания множественной информации.

Часто элементы редко заполненной матрицы соответствуют некоторым реальным объектам, и имеется не одна, а целый набор матриц различных типов, в каждой из которых требуется задавать коэффициенты, стоящие на одних и тех же местах. В алголе-60 мы можем сопоставить каждой такой матрице *a* вектор *a*1 так, как это мы только что делали, а вектор *row* имеет один и тот же для всех матриц. В этом отношении «коллективное» задание «дырявых» матриц выглядит даже проще, однако следует иметь в виду, что здесь речь идет о простейшей форме множественной информации.

Серьезные проблемы возникают в том случае, если эти разнородные массивы должны храниться во внешней памяти, из которой вся информация, относящаяся к данному объекту, должна поступать одновременно. В этом случае было бы удобно иметь нечто вроде картотеки, в которой для каждого объекта определенного типа заведена соответствующая «карточка», в которой хранятся все необходимые параметры. Во многих современных алгоритмических языках имеются возможности [1]) для такого представления информации. В частности, есть такие возможности и в языке алгол-68, к которому мы и будем прибегать для описания алгоритмов с «картотечным» хранением информации.

В алголе-68 мы можем описать вещественный вектор [2], в котором хранятся лишь ненулевые компоненты, следующим образом. Вначале мы введем новый вид данных— структуру «пара», составленную из индекса и значения

---

[1]) Впервые такие возможности появились в коболе, затем в языке симула, предназначенном для машинного моделирования функционирования сложных систем. Из более современных языков следует назвать ПЛ/1, паскаль.

[2]) В алголе-68 в сравнении с алголом-60 приняты сокращенные варианты написания служебных слов: **int** вместо **integer**, **bool** вместо **boolean** и т. д.

коэффициента,

mode pair = **struct** (**int** *ind*, **real** *a*)

а затем опишем массив, составленный из таких пар,

[1 : *k*1] **pair** *x*1; [1 : *k*2] **pair** *x*2

и т. п.

Каждый такой массив заменяет два соответствующих массива в алголе-60, причем информация об i-м элементе (индекс и значение) расположена в нем вся вместе. Вместо *ind*1 [*i*] мы должны теперь писать *ind* **of** *x*1 [*i*] и *a* **of** *x*1 [*i*] вместо *x*1 [*i*].

Повторим на алголе-68 описанные выше алгоритмы скалярного произведения и сложения векторов. Сначала сделаем «буквальный перевод» с алгола-60 алгоритма скалярного произведения:

```
scal := 0; j := 1;
for i to k1 do
   while j ≤ k2 do
      if ind of x2[j] ≥ ind of x1[i] then
       if ind of x2[j] = ind of x1[i] then
         scal +:= a of x2[j] × a of x1[i]
       else j —:= 1 fi
      else go to mkf fi; j +:= 1 od;
mkf: skip od
```

С нашей стороны было бы слишком смело пытаться мимоходом заменить учебник по алголу-68. Пояснения, которые мы здесь дадим, следует рассматривать лишь как комментарий к написанному тексту. Оператор $c +:= d$ эквивалентен оператору $c := c + d$, и аналогично трактуется $c —:= d$. Оператор **skip** — это в данном контексте пустой оператор. Читателю, привыкшему к алголу-60, сразу заметно отсутствие операторных скобок **begin** и **end**. В этом языке в качестве открывающих операторных скобок фигурируют сами **do** и **if**, закрывающими скобками для них являются соответственно **od** и **fi**.

Оператор цикла в алголе-68 имеет полную форму

for *k* **from** *a* **by** *b* **to** *c* **while** *d* **do**

Тривиальные части этой формулы (**by** 1, **while true**, **to** ∞, **from** 1 и даже **for** k, если k не используется в теле цикла) могут быть опущены.

Так же, как в алголе-60, мы могли бы описать вычисление скалярного произведения в виде процедуры. Процедуры в алголе-68 оформляются несколько иначе; эти отличия мы еще будем обсуждать ниже. Здесь мы предпочтем использовать другую возможность языка — описание операции. Мы введем скалярное произведение как операцию над векторами (массивами пар) **scpr** [1]) и будем использовать ее в записях типа $x1$ **scpr** $x2$.

Итак,

```
op scpr = (ref [ ] pair x1, x2) real:
begin int k1 := upb x1, k2 := upb x2,
                              j := 1; real scal := .0;
   for i to k1 do while j ≤ k2 do
     if ind of x2[j] ≥ ind of x1[i] then
       if ind of x2[j] = ind of x1[i] then
         scal +:= a of x2[j] × a of x1[i]
       else j —:= 1 fi;
     else go to mkf fi; j +:= 1 od;
mkf: skip od; scal
end
```

Несколько новинок языка: при описании переменных можно присваивать им начальные значения. Описывать переменную цикла нельзя (точнее, бесполезно). **upb** x — верхняя граница массива x. *scal* перед **end** — то вещественное значение, которое вырабатывается операцией.

Вид вырабатываемого операцией значения ставится перед телом операции и отделяется двоеточием, а перед указанием вида в скобках ставятся параметры операции — в нашем случае два массива. Что значит в этих параметрах **ref**, мы обсудим несколько ниже.

Теперь уже в виде операции опишем сложение двух векторов. Знаком операции будет + (мы доопределяем операцию сложения для нового типа объектов):

---

[1]) **scpr** при этом становится знаком операции.

```
op+ = (ref [ ] pair x1, x2) ref [ ] pair:
begin int j := 1, k3 := 0, k1 := upb x1, k2 := upb x2;
   real y; [1 : k1 + k2] pair x3;
   for i to k1 do while j ≤ k2 do
     if ind of x2 [j] = ind of x1 [i] then
       y := a of x1 [i] + a of x2 [j];
       if y ≠ 0 then x3 [k3 + := 1] := (ind of x1 [i], y);
                                          go to mkf fi
     elif ind of x2 [j] < ind of x1 [i]
                                  then x3 [k3 + := 1] := x2 [j]
     else x3 [k3 + := 1] := x1 [i]; j - := 1;
                                          go to mkf fi;
     j + := 1 od;
mkf: skip od;
   heap [1 : k3] pair := x3 [1 : k3]
end
```

Здесь нам встретилось **elif** — слово, которое заменяет сочетание **else if**, причем не открывает лишней скобки условия. Кроме того, впервые появилось увеличение индекса в левой части оператора присваивания, присваивание для структур ($x3[k3] := x1[i]$ либо $x3[k3] := (i, a)$) и для массивов. Однако самая существенная новинка — генерирование новых объектов из «кучи». В алголе-60 вся память, требуемая в программе, определяется описаниями и в границах данного блока не зависит от хода вычислений в этом блоке. Часто трансляторы распределяют память по магазинному принципу — каждый следующий блок получает память, начиная со свободного места и освобождает ее по окончании. В алголе-68 можно увеличить память, заказывая ее в ходе вычислений, причем наряду с «магазинной памятью» используется еще один механизм распределения памяти — «куча», из которой память заказывается с помощью **heap** (дополнительная память из магазина заказывается с помощью **loc**, при выходе из блока эта память теряется).

Для того чтобы работать с появляющимися новыми объектами, нужно иметь возможность к ним обращаться. В алголе-60 мы сопоставляем каждому объекту имя — идентификатор. Объекты, появляющиеся в ходе вычислений, снабдить идентификаторами мы уже не сможем. Однако существует другая возможность — ввести такие имена,

которые сами рассматривались бы как одна из разновидностей вычислительной информации и записывались бы в предназначенные для этого места памяти. В алголе-68 и во многих других современных языках программирования имеется возможность вводить новые виды переменных (и, следовательно, полей в записях) — *ссылки*, указывающие на конкретные объекты того или иного вида [1]. Понятие ссылки очень важно, и на нем нам следует остановиться.

Естественность этого понятия для языков программирования определяется особенностями устройства вычислительных машин и организации работы машинных программ. Почти каждая элементарная машинная операция— команда имеет в качестве аргументов (операндов) именно ссылки — адреса реальных объектов вычисления. Например, команда сложения в трехадресной машине выглядит так:

Сл A B C

(сложить числа, записанные в поля памяти с адресами A и B, и поместить результат в поле с адресом C.) Вместе с тем в некоторых случаях операнды указываются в команде непосредственно. Скажем, имеется команда сдвига содержимого ячейки (слова), в которой величина сдвига k указывается непосредственно первым аргументом. Например,

СдСЛ 7 A B

сдвигает содержимое ячейки с адресом A на 7 разрядов влево и помещает результат в ячейку с адресом B.

В алголе-60 мы оперируем фактически только со ссылками на объекты, которые в тексте программы задаются идентификаторами или непосредственно заданными величинами, а в машинной программе ссылки заменяются адресами. Оператор $c := a + b$ эквивалентен машинной команде сложения и будет переведен транслятором в такую команду. Однако есть некоторая разница между этим

---

[1]) Используемое при описании ссылок **ref** (от английского **reference** — ссылка) в книгах по алголу-68 обычно переводится как **имя**. Однако в других языках распространен термин «ссылка», и нам он представляется более отвечающим существу дела.

оператором и оператором $c := a + 5$. В обоих случаях есть места в памяти, отведенные для каждого из слагаемых. Но в первом случае мы располагаем полем, отведенным под b, и можем записать туда то, что требуется. Во втором случае записать что-либо иное в поле, где стоит 5, языком не разрешается. Таким образом, в алголе-60 мы располагаем лишь ссылками на введенные объекты, их и подразумевая под переменными.

В алголе-68 при описании переменных мы также порождаем ссылки и работаем со ссылками. Однако этот язык позволяет при описании алгоритма точнее приблизиться к понятиям машинной программы, но требует более отчетливой системы представлений и описаний. Здесь можно предусмотреть действия, требующие не адресов, а непосредственно величин (как в команде сдвига). Поэтому при описании операций и процедур требуется задавать аргументы с точным указанием вида и, следовательно, добавить к описаниям тех объектов, которые мы задаем их адресами, «лишний» **ref**.

Таким образом, если мы захотим описать действия, выполняемые двумя упомянутыми машинными командами, в виде операций, то заголовок операции сложения должен выглядеть так:

op + = (**ref real** *a*, *b*) **ref real**: ...

а операции сдвига влево так:

op lshift = (**ref int** *a*, **int** *k*) **ref int**: ...

Если в программе введен (для примера) вид **object**, то переменная такого вида имеет вид **ref object**, но описать ее можно так: **object** *a*, и мы будем говорить в таких случаях, что *a* — переменная вида **object**. Возможность описания объектов вида **ref object**, **ref ref object** и т. д. появляется одновременно с возможностью описания объектов вида **object**, и можно вводить ссылки как переменные вида **ref object**. Для всех видов определено особое значение ссылок **nil** — ссылка *ни на что*. Имеются две операции сравнения ссылок одного вида, вырабатывающие логические значения: *a* **is** *b* — ссылки совпадают и *a* **isnt** *b* — они не совпадают. Время от времени мы будем пользоваться еще одной операцией, вырабатывающей

логическое значение «*a* на что-то ссылается»

op lin = (ref object *a*) **bool**: *a* **isnt nil**

Не описывая эту операцию, в дальнейшем мы будем в случае надобности лишь указывать вид ссылок, для которых она будет определена.

Отсылая читателя за дальнейшими подробностями к руководствам по алголу-68, мы будем считать, что высокие требования этого языка к точности в описаниях обойдутся нам главным образом в лишние **ref** при описаниях процедур и операций, и лишь в случае необходимости будем комментировать особо такие места встречающихся алгоритмов. Однако есть еще одно трудное место, о котором следует сказать сразу.

Рассмотрим простейший оператор присваивания $a := b$, где a — переменная вида **object** или **ref object**, а b — величина или переменная вида **object** или **ref object**. В отличие от алгола-60, здесь наиболее естественной формой оператора присваивания считается такая, в которой в левой части стоит переменная какого-то вида, а в правой — величина того же вида. Переменная указывает на какой-то объект, и в этот объект и будет скопировано значение величины из правой части. Если a — переменная вида **object**, а b — величина вида **object**, то скопируется **object**. Если a — переменная вида **ref object**, а b — переменная вида **object** (и, значит, величина вида **ref object**), скопируется величина вида **ref object**. При несоответствии уровней ссылок в левой и правой частях автоматически производится их согласование (снятие лишних **ref** в правой части) до «естественного» сочетания уровней. Так, если a и b — переменные виды **object**, в поле, адресуемое ссылкой a, запишется содержимое объекта, адресуемого ссылкой b. Если a и b — переменные вида **ref object**, действие пересылки будет произведено над ссылками, и после его выполнения содержимое a будет ссылаться на тот же объект вида **object**, что и b.

Наконец, если a — переменная **object**, а b — переменная **ref object**, то b указывает на поле, в которое записана ссылка на некоторый объект вида **object**, который и станет значением правой части оператора присваивания.

В некоторых случаях, когда автоматическое согласование уровней дает не тот результат, который требуется,

приходится вмешиваться в этот процесс (называемый «разыменованием»). Вот пример такого вмешательства. Пусть мы имеем две переменные вида **ref object** и хотим переписать в объект **object**, на который указывает *a*, содержимое объекта **object**, на который указывает *b*. Для этого достаточно указать вид требуемого объекта в левой части оператора присваивания, причем подобно параметрам в процедурах здесь требуется точное указание вида нужного нам объекта, и значит, появляется «лишний» **ref**. В нашем случае оператор присваивания выглядит так:

ref object (*a*) : = *b*

Попробуем теперь, используя технику ссылок, представить вектор в виде набора элементов, каждый из которых ссылается на следующий. Исключение будет составлять последний элемент вектора, у которого эта ссылка будет не заполнена. Итак, каждой ненулевой компоненте вектора будет сопоставляться элемент вида

mode elem = **struct** (**ref elem** *next*, **int** *ind*, **real** *a*)

С помощью ссылки *next* элемент будет указывать на следующую ненулевую компоненту вектора. Сам вектор может теперь задаваться ссылкой на начальный элемент. Опишем для таких векторов операцию сложения:

```
op + = (ref ref elem x1, x2) ref elem:
begin ref elem i := x1, j := x2,
      k := loc elem := (nil, skip, skip),
   l := k;
   while lin i do
      while lin j do
         if ind of j = ind of i then y := a of i + a of j;
          if y ≠ 0 then l := next of l :=
                                heap elem := (nil, ind of i, y);
            j := next of j; go to mkf fi
          elif ind of j < ind of i then
            l := next of l := heap elem := i
          else l := next of l := heap elem := j; go to mkf fi;
          j := next of i od;
mkf : i := next of i od;
    next of l := nil; next of k
end
```

Мы ввели начальный элемент, на который ссылаются k и l для удобства, **skip** в двух полях этого элемента означает, что значения полей не определяются, **loc** означает, что элемент вводится локально — только в самой операции, a **of** i — это поле a в той структуре, на которую ссылается i.

Построение наборов элементов одной и той же или различной структуры, связанных между собой ссылками,— вычислительный прием, имеющий много полезных применений. Некоторые стандартные действия и понятия, связанные с построением и переработкой линейных последовательностей (списков), описаны в следующем параграфе.

§ 4. Списки

Пусть задано некоторое множество N, каждому элементу которого сопоставлен набор целочисленных, вещественных, логических и текстовых (а возможно, и ссылочных) параметров. По этим параметрам мы можем сравни-

Рис. 2. Физическая очередь.

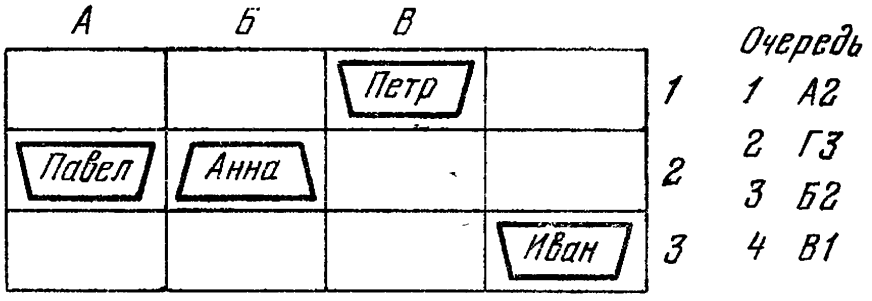

Рис. 3. Именная очередь.

вать эти элементы и вводить для них упорядочение, т. е. можем для любых двух элементов i, $j \in N$ сказать, какой из них должен предшествовать другому.

Имеется несколько способов фиксации этого упорядочения в вычислительной машине. Эти способы, впрочем,

давно знакомы нам по обыденной жизни, где такие упорядочения встречаются в виде очередей.

Самый простой и естественный способ построения упорядочения — это способ *физической очереди*: объекты, составляющие очередь, выстраиваются в порядке, соответствующем упорядочению, и элемент i предшествует элементу j в том и только в том случае, если он в очереди стоит раньше (рис. 2).

Однако в случае, если объекты уже расположены в памяти, занимают там достаточно много места, а упорядочение достаточно часто может изменяться, выстраивать их в физическую очередь представляется столь же неестественным, как делать физическую очередь из домов, отобранных для капитального ремонта. Выход заключается в том, чтобы сделать физическую очередь из ссылок на эти объекты — перечислить ссылки на объекты в порядке, соответствующем упорядочению (рис. 3), — назовем это способом *именной очереди*.

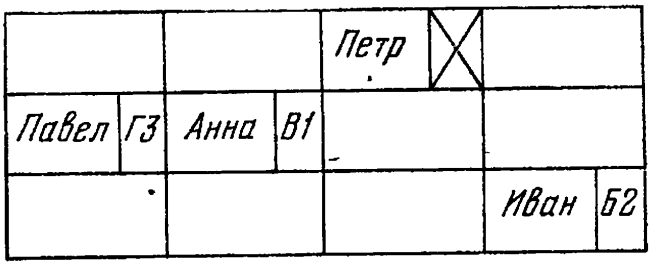

Рис. 4. Односторонние локальные ссылки назад. Начало очереди А2.

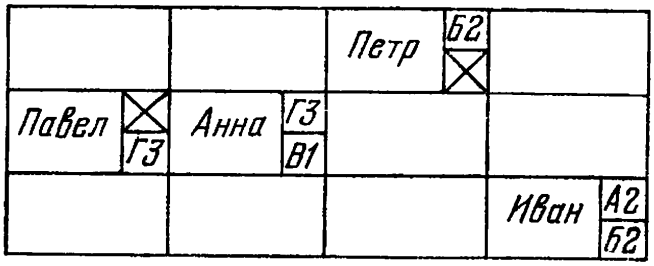

Рис. 5. Двусторонние локальные ссылки. Начало очереди А2, конец В1.

Этот метод очень прост и достаточно удобен в вычислениях, особенно в тех случаях, когда упорядочение достаточно стабильно, новых объектов не возникает,

имеющиеся объекты из очереди не выбывают. Однако для мобильной очереди более удобной оказывается третья форма организации очереди — способ *локальных ссылок*, в котором каждый из объектов очереди с помощью ссылок запоминает, за кем он стоит, перед кем он стоит или и то и другое вместе (рис. 4 и 5).

В подавляющем большинстве случаев достаточно при организации очереди иметь односторонние ссылки (например, только назад, на следующего, такой была ссылка *next* в последнем варианте операции сложения векторов), однако бывают случаи, когда требуется перемещаться по очереди и вперед и назад, и в этом случае лучше иметь по две ссылки у каждого объекта.

Список с односторонними ссылками задается ссылкой на его начало, список с двусторонними ссылками — двумя ссылками: на начало и на конец или на любой из них (в зависимости от условий). Рассмотрим несколько простейших действий над списками. Так как речь идет о ссылках, нам естественно прибегнуть к алголу-68. В описаниях этих действий в качестве элементов списков будут использоваться для примера структуры вида

mode elem = struct (ref elem *next*, *pre*, **real** *a*, *b*, **int** *i*)

состоящие для определенности из двух вещественных чисел, одного целого и двух ссылок на такие же структуры.

Самыми простыми являются операции магазинного списка — включение элемента в магазин и исключение элемента из непустого магазина:

```
op cons = (ref ref elem a, ref elem b) ref ref elem:
  begin next of b := a; a := b end;
op head = (ref ref elem a) ref elem:
  begin ref elem b := a; a := next of a; b end
```

Использование этих операций выглядит так: *a* **cons** *b* для присоединения *b* в качестве первого элемента к списку и *b* := **head** *a* для отделения первого элемента списка. При переносе элемента из одного списка в другой требуется последовательное применение этих двух операций, соответствующая объединяющая их операция может быть записана так:

op rob = (ref ref elem *a*, *b*) **ref ref elem:** *a* **cons head** *b*

Примером использования списков и введенных операций может служить решение следующей простой задачи.

Задано множество $N = 1 : n$, каждому элементу i которого сопоставлены два числа — *оценка* $c[i]$ и положительный *вес* $d[i]$. Требуется найти такой элемент $i0$ и такое разбиение $N - i0$ на два подмножества N_1 и N_2, чтобы выполнялись неравенства

$$c[i] \geqslant c[i0], \quad i \in N_1, \quad c[i] \leqslant c[i0], \quad i \in N_2,$$
$$\Sigma_{i \in N_1} d[i] \leqslant w1, \quad \Sigma_{i \in N_2} d[i] \leqslant w2,$$

где $w1$, $w2$ — заданные положительные числа и

$$w1 + w2 = 1[N] \times d[N].$$

Эта задача (она снова встретится нам в главе 5) решается очень просто, если элементы упорядочены по убыванию оценок. Тогда, двигаясь от $i = 1$, мы сможем набрать нужное число элементов в N_1, ближайший «не помещающийся» элемент будет $i0$, а остальные элементы войдут в N_2. Однако упорядочение иногда дорого в вычислительном отношении, и мы здесь обойдемся без него.

Выберем какой-либо элемент i' и испробуем его в качестве $i0$. Для этого будем просматривать элементы N и относить в N_1' те элементы, для которых $c[i] > c[i']$, а в N_2' те, для которых $c[i] < c[i']$ (будем для простоты считать все $c[i]$ различными). Попутно мы будем накапливать суммы весов $s1$ и $s2$ и сравнивать их с границами $w1$ и $w2$. Если мы благополучно доберемся до конца, N_1' и N_2' и будут искомыми N_1 и N_2. Если же увеличиваемый вес, для определенности $s1$, превзойдет свою границу $w1$, это будет значить, что i' — слишком низкий разграничитель и следует попытаться взять элемент с большим $c[i]$. Однако сам элемент i' и накопленное множество N_2' уже бесспорно должны окончательно войти в N_2, так как кандидатов в N_1 достаточно и без них. Вот основная идея нашего алгоритма, а дальнейшие детали будут видны из описания.

Мы опишем структурный вид

mode el = struct (ref el *next*, **int** *c*, *d*)

где *next* будет использовано для цепных списков, а c, d — введенные выше параметры элемента, затем введем муль-

тизначение (массив)

[1:*n*] el *el*

и сделаем начальное присваивание

```
for i to n do
  el[i]:=(if i=n then nil else el[i+1] fi,
          c[i], d[i]) od
```

Дальше алгоритм описывается прямо на алголе-68 с поясняющими комментариями, которые, как полагается, заключаются в одинаковые скобки **co** (это тоже сокращение).

Введем ссылки на некоторые элементы:

ref el *bou* **co** это наш граничный элемент i' **co**,

cur **co** текущий элемент списка **co**,

fp:=*el*[1] **co** начало списка нерассмотренных элементов **co**,

*b*1, *e*1 **co** начало и конец списка, описывающего N_1' **co**,

*b*2, *e*2 **co** аналогично для N_2' **co**,

*r*1:=**nil**, *r*2:=**nil** **co** начала списков для элементов, окончательно включенных в множества.

Кроме ссылок, нам понадобятся и некоторые целые числа **co**;

int *s*1, *s*2 **co** о них уже говорилось **co**,

cb, *db* **co** параметры элемента *bou* **co**,

cc, *dc* **co** параметры элемента *cur* **co**,

*wr*1:=0, *wr*2:=0 **co** накопленные веса окончательно включенных элементов **co**,

*cr*1:=*maxint*, *cr*2:=—*maxint* **co** границы оценок окончательно включаемых элементов **co**;

mk: **co** отсюда начинается попытка построить два списка. Выделяем граничный элемент **co**

bou:=**head** *fp*; *cb*:=*c* **of** *bou*; *db*:=*d* **of** *bou*;

co — готовим начальную информацию о рабочих списках **co**

*b*1:=*e*1:=*b*2:=*e*2:=**nil**; *s*1:=*wr*1; *s*2:=*wr*2;

while lin *fp* **do** *cur*:=**head** *fp*; *cc*:=*c* **of** *cur*;
dc:=*d* **of** *cur*;

co выделили из списка очередной элемент *cur*, сейчас будет решаться его судьба, и это решение начнется сравнением *cur* с *bou* **co**

```
if cc>cb then
  if (s1+:=dc)>w1 then
```

со переполнилось N_1', нужно пробовать заново, но сначала увеличим окончательную часть N_2 со

```
r2 cons bou; if lin b2
then next of e2:=r2; r2:=b2 fi;
cr2:=cb; dr2:=db+s2;
```

со элементы N_1' отправим обратно в нерассмотренные элементы со

```
if lin b1 then next of e1:=fp;
                       fp:=b1 cons cur fi;
go to mk
elif cc≥cr1 then
```

со случай бесспорной кандидатуры из N_1 со

```
dr1+:=dc; r1 cons cur

elif lin b1 then b1 cons cur else
```

со случай пустого списка требует особого рассмотрения со

```
b1:=e1:=cur; next of cur:=nil fi

elif со симметрично разбирается вторая возможность со
    (s2+:=dc)>w2 then r1 cons bou;
    if lin b1 then next of e1:=r1; r1:=b1 fi;
    cr1:=cb; dr1:=db+s1;
    if lin b2 then next of e2:=fp; fp:=b2 cons cur fi;
    go to mk
elif cc<cr2 со естественное изменение
                              знака неравенства со
    then dr2+:=dc; r2 cons cur
    elif lin b2 then b2 cons cur
    else b2:=e2:=cur; next of cur:=nil fi
```

Кроме «магазинных» операций, иногда используются операции включения в конец списка и в подходящее место списка, выбираемое в соответствии с некоторым упорядочением элементов.

Присоединение элемента к концу списка пишется так:

```
op join=(ref ref elem a, ref elem b) ref ref elem:
    a:=next of a:=b
```

(в ссылочное поле элемента *a* заносится ссылка на *b*, а затем *a* начинает адресовать этот элемент). Полезно еще действие присоединения элемента к концу списка с проверкой его непустоты. Такое действие оформляется в виде процедуры. В приводимой процедуре *a* — ссылка на начало списка, *b* — на конец, *c* — присоединяемый элемент:

```
proc join = (ref ref elem a, b, ref elem c) ref ref elem:
  if lin a then b join c else b:=a:=c fi
```

Пусть теперь имеется операция сравнения объектов: *a* **prec** *b* эквивалентно «*a* предшествует *b*». Рассмотрим процедуру включения элемента *b* в соответствии с упорядочением **prec** в непустой упорядоченный список, начинающийся с *a*. Последовательность действий в такой процедуре ясна, — нужно найти в списке место, на которое требуется поставить новый элемент, т. е. такой элемент *c*, что

$$c \ \textbf{prec} \ b \wedge b \ \textbf{prec} \ \textit{next} \ \textbf{of} \ c$$

(*c* предшествует *b* и *b* предшествует элементу, следующему за *c*) и вставить новый элемент на это место, т. е. сделать в нем ссылку на следующий элемент такую, как

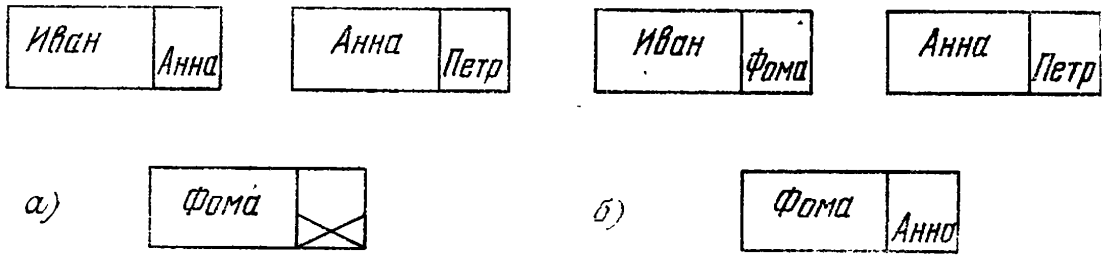

Рис. 6. Включение в очередь нового элемента.

у *c*, а в *next* **of** *c* вписать ссылку на *b* (рис. 6). Движение по списку осуществляется очень просто с помощью одной из уже встречавшихся нам разновидностей оператора цикла.

Мы опишем это действие в виде процедуры (здесь процедура называется **proc**). Описание процедуры похоже на описание операции — в скобках указаны параметры процедуры, а после скобок — вид вырабатываемого процеду-

рой значения. Наша процедура вырабатывает *никакой* вид. Операция **lin** описана для вида **ref elem**:

```
proc inclel = (ref ref elem a, ref elem b) void:
begin ref elem c, d; c := a;
    if b prec a then a cons b
    else while lin (d := next of c) do
            if d prec b then c := d else go to mkf fi od:
mkf: next of b := d; next of c := b fi
end
```

Если b предшествует a и, следовательно, должно быть помещено в начале списка, то мы пользуемся магазинной операцией. В противном случае отыскивается место для b (между c и d) и после выхода на метку *mkf* запись фиксируется на своем месте двумя ссылками.

Для списка с двусторонними ссылками важной элементарной операцией является исключение элемента из списка. Грубо говоря, нужно заставить соседей этого элемента ссылаться друг на друга:

next **of** *pre* **of** a := *next* **of** a; *pre* **of** *next* **of** a := *pre* **of** a

Некоторые усложнения связаны с эффектами края списка:

```
proc excel = (ref elem a) void:
begin ref elem b := pre of a, c := next of a;
    if lin b then next of b := c;
     if lin c then pre of c := b fi
    else pre of c := nil fi
end
```

Включение элемента в список естественно требует для двусторонних ссылок больших усилий: нужно изменить две ссылки в месте вставки и заново образовать две ссылки у вставляемого элемента. В остальном процедура не изменяется:

```
proc inclel2 = (ref ref elem a, b) void:
begin ref elem c, d; c := a;
    if b prec a then pre of a := b; a cons b
    else while lin (d := next of c) do
            if d prec b then c := d
            else go to mkf fi od;
mkf: next of b := d; pre of b := c;
                                next of c := pre of d := b fi
end
```

Включение одного упорядоченного списка в другой может проводиться с использованием описанных процедур. При этом можно иметь в виду, что элемент *b*, включенный в список, уже указывает на начало оставшейся части списка, в которую должны включаться следующие за ним элементы. Поэтому соответствующая процедура выглядит очень просто:

```
proc inclist = (ref ref elem a, b) ref ref elem:
begin ref elem c := a, d := b;
   while lin d do ref elem e := next of d;
      inclel (c, d); c := d; d := e od;
   if a prec b then a else b
end
```

(в случае двусторонних ссылок нужно, разумеется, заменить *inclel* на *inclel2*).

Для односторонних списков полезной оказывается операция обращения списка — бывает, что изредка требуется перебрать элементы списка в противоположном порядке, но при этом такая потребность возникает не настолько часто, чтобы заводить специально для нее двусторонние ссылки. Соответствующая процедура выглядит так:

```
proc listinvert = (ref ref elem a) ref ref elem:
begin ref elem c := a, d := nil;
   while lin c do d rob c od;
   d co начало нового списка co end
```

Понятие списка, рассматриваемого как конечная последовательность элементов некоторого множества (кортеж), возникает в различных областях математики (в математической лингвистике, теории статистических решений, динамическом программировании, о котором у нас еще пойдет речь) в качестве основного изучаемого объекта. Для дальнейшего окажутся полезными еще несколько понятий, связанных с действиями над списками и функциями от списков.

Прежде всего отметим, что если отделить от непустого списка его первый элемент (*голову*), то оставшаяся часть списка (*хвост* списка) в свою очередь будет списком.

Пусть $\varphi(l)$ — функция, заданная на множестве списков, элементы которых принадлежат множеству A. Обозначим множество таких списков через $\mathbf{List}(A)$.

Обозначим через $head(l)$ начальный элемент списка l, а через $tail(l)$ хвост списка.

Назовем функцию $\varphi(l)$ *аддитивной*, если найдется такая функция $\alpha(a)$ на A, что для любого непустого $l \in \mathbf{List}(A)$ имеет место

$$\varphi(l) = \alpha(head(l)) + \varphi(tail(l)). \tag{4}$$

Функция $\varphi(l)$ называется *мультипликативной*, если найдется такая функция $\beta(a)$ на A, что для любого непустого $l \in \mathbf{List}(A)$ имеет место

$$\varphi(l) = \beta(head(l)) \times \varphi(tail(l)). \tag{5}$$

Наконец, функция $\varphi(l)$ называется *рекуррентной*, если найдутся такие функции $\alpha(a)$ и $\beta(a)$ на A, что для любого непустого $l \in \mathbf{List}(A)$ имеет место

$$\varphi(l) = \alpha(head(l)) + \beta(head(l)) \times \varphi(tail(l)). \tag{6}$$

Хороший пример рекуррентной функции — значение целого числа, представленного в обычной позиционной записи. Если принять в качестве A множество $0:9$ (точнее множество десятичных цифр), считать начальным элементом позиционной записи числа последнюю цифру, а хвостом списка оставшийся список, то значение $\varphi(l)$ числа, представленного списком l, будет, очевидно, равно

$$\varphi(l) = head(l) + 10 \times \varphi(tail(l)).$$

Так, число 1975 представляется последовательностью цифр 5, 7, 9, 1. И мы имеем

$$\begin{aligned}
\varphi(5,\ 7,\ 9,\ 1) &= 5 + 10 \times \varphi(7,\ 9,\ 1),\\
\varphi(7,\ 9,\ 1) &= 7 + 10 \times \varphi(9,\ 1),\\
\varphi(9,\ 1) &= 9 + 10 \times \varphi(1),\\
\varphi(1) &= 1 + 10 \times \varphi(\mathbf{o}),
\end{aligned}$$

где $\mathbf{o}$ — пустой список. Приняв $\varphi(\mathbf{o}) = 0$, мы получаем

$$\varphi(5,\ 7,\ 9,\ 1) = 5 + 7 \times 10 + 9 \times 10^2 + 1 \times 10^3.$$

Важной операцией над списками является их упорядочение. Среди возможных методов упорядочения особое

место занимает уже упоминавшееся лексикографическое упорядочение. Пусть на множестве A задано некоторое упорядочение. Будем говорить, что список $l1$ *лексикографически предшествует* списку $l2$, если $l1 = \mathbf{o}$ или $l1$, $l2 \neq \mathbf{o}$и, либо *head* $(l1) \neq$ *head* $(l2)$ и *head* $(l1)$ предшествует *head* $(l2)$, либо *head* $(l1) =$ *head* $(l2)$ и список *tail* $(l1)$ лексикографически предшествует списку *tail* $(l2)$ [1]).

Иными словами, $l1$ лексикографически предшествует $l2$ в том случае, если $l1$ является началом $l2$, либо первые несовпадающие элементы этих списков стоят на k-м месте и k-й элемент списка $l1$ предшествует k-му элементу списка $l2$.

Название «лексикографическое упорядочение» происходит от известного всем упорядочения слов в словаре. Слово может рассматриваться как список из букв. На множестве букв (алфавите) задано упорядочение. Слово «чад» предшествует слову «чадра», так как является его началом, слова «чадра» предшествует слову «чардаш», так как первые две буквы у этих слов совпадают, а третья буква в слове «чадра» предшествует третьей букве в «чардаш».

Процедура *subsetstack* вырабатывает лексикографически упорядоченные списки элементов множества $1 : n$.

Вещественный или целочисленный вектор заданной размерности (с заданным упорядочением множества индексов) может также рассматриваться как список фиксированного размера. Легко видеть, что для таких списков определение лексикографической упорядоченности совпадает с введенным ранее.

Мы не будем в дальнейшем злоупотреблять алголом-68. Алгол-60 будет единственным языком программирования в следующей главе и основным языком в остальных главах. Однако некоторые приемы программирования, встречавшиеся в этой главе, могут быть использованы и будут использоваться и в текстах алгола-60.

---

[1]) Возвращаясь к алголу-68, мы можем описать лексикографическое предшествование цепных списков операцией **lexprec** по операции сравнения элементов **prec** следующим образом:

```
op lexprec=(ref ref elem a, b) bool:
if lin a then if lin b then if a prec b then true
elif b prec a then false else next of a lexprec next of b fi
else false fi else true fi
```

ГЛАВА 2

НЕКОТОРЫЕ ОБЩИЕ СВЕДЕНИЯ О ЛИНЕЙНОМ ПРОГРАММИРОВАНИИ

§ 1. Введение

В задачу этой книги не входит систематическое изложение общей теории и методов решения задач линейного программирования. На эту тему написано уже много подробных руководств, в которых систематически излагается вычислительный, геометрический и прикладной аспект моделей и методов линейного программирования.

Нам все это нужно постольку, поскольку часть интересующей нас проблематики может рассматриваться как некоторый специальный класс задач линейного программирования. Для того чтобы облегчить ссылки на встречающиеся нам факты из области линейного программирования и иметь нужные формулировки в том виде, который для нас наиболее удобен, мы и предлагаем читателю настоящую главу.

В ней несколько нестандартен § 2, в котором рассматривается вопрос о расширениях экстремальных задач и об эквивалентных задачах. Хотя в основном речь идет там о задачах линейного программирования, сами определения и подход важны и полезны во всех задачах на экстремум. Теория двойственности изложена в § 3 нарочито формально, и читателю, который заинтересуется геометрическими или экономическими трактовками этих красивых теорем, рекомендуется обратиться к библиографическим указаниям к этой главе.

Для дальнейшего изложения очень важны §§ 4—6. Метод последовательных улучшений допустимого базисного решения, который в них излагается сначала в виде общей принципиальной схемы, а затем в виде нескольких вычислительных реализаций, будет широко использоваться в последующих главах. Однако нам представляется, что для читателя, интересующегося вычислительным аспектом линейного программирования, эти параграфы должны

представить и самостоятельный интерес, так как только в очень немногих монографиях по линейному программированию вопросы реализации методов линейного программирования на ЭВМ обсуждаются достаточно полно и современно.

§ 2. Постановка задачи линейного программирования

Задачей линейного программирования называется задача нахождения максимума (или минимума) линейной функции от переменных, подчиненных линейным ограничениям. Традиционно выделяют условия неотрицательности переменных, которые часто встречаются в задачах экономического характера и для соблюдения которых стандартные алгоритмы линейного программирования специальным образом подготовлены.

В конечном счете каждая задача линейного программирования может быть приведена к следующему виду:

Основная задача линейного программирования

Пусть заданы два конечных множества M и N, векторы $b[M]$ и $c[N]$, матрица $a[M, N]$. Пусть заданы также разбиения $M=M_1+M_2$ и $N=N_1+N_2$. Требуется найти вектор $x[N]$, удовлетворяющий условиям

$$x[N_1] \geqslant \mathbf{0}[N_1], \tag{1}$$

$$a[M_1, N] \times x[N] \geqslant b[M_1], \tag{2}$$

$$a[M_2, N] \times x[N] = b[M_2] \tag{3}$$

и минимизирующий линейную функцию

$$c[N] \times x[N]. \tag{4}$$

Случай, когда $M=M_2$ и $N=N_1$, нас будет интересовать особо. Такая задача будет называться *стандартной задачей линейного программирования.*

В различных ситуациях оказывается целесообразным менять формулировку экстремальной задачи, переходя к той или иной эквивалентной задаче или даже несколько меняя задачу по существу. Мы дадим формальное определение эквивалентных задач и расширения задач, а затем

посмотрим, как эти определения могут быть использованы в случае задач линейного программирования.

Рассмотрим две экстремальные задачи.

Задача А

| Минимизировать $f(x)$ по $x \in A$

и

Задача Б

| Минимизировать $g(y)$ по $y \in B$.

Все экстремальные задачи имеют стандартные особенности: задается некоторое множество, среди элементов которого можно выбирать. Это множество называется множеством *допустимых решений*. Задается также функция, минимум или максимум которой требуется найти. Эта функция называется *целевой функцией* задачи. Не представляет труда перейти от задачи минимизации к задаче максимизации — для этого достаточно изменить знак у целевой функции, поэтому мы ограничимся задачами минимизации.

Допустимое решение, на котором достигается минимум целевой функции, называется *оптимальным решением* задачи.

Будем говорить, что функции $\varphi(x)$ и $\psi(x)$ *задают одинаковое упорядочение* на множестве A, если для любых x, $y \in A$

$$\operatorname{sign}(\varphi(x) - \varphi(y)) = \operatorname{sign}(\psi(x) - \psi(y)). \qquad (5)$$

Легко видеть, что для целевых функций, задающих одинаковое упорядочение, множества оптимальных решений совпадают.

Будем говорить, что задача Б есть *расширение* задачи А, если существует отображение T: $A \to B$ такое, что функции $f(x)$ и $g(Tx)$ задают одинаковое упорядочение на множестве A.

Задачи А и Б называются *эквивалентными*, если существуют такие отображения T: $A \to B$ и S: $B \to A$, что $f(x)$ и $g(Tx)$ задают одинаковое упорядочение на A, $g(y)$ и $f(Sy)$ задают одинаковое упорядочение на B и $f(STx) = f(x)$.

Отображения, с помощью которых устанавливаются эти связи между задачами, назовем *связующими* отображениями.

Полезность введенных определений видна из следующих простых утверждений.

Теорема 1. *Пусть задачи* А *и* Б *эквивалентны и* T: $A \to B$, S: $B \to A$ — *связующие отображения. Если* x^* — *оптимальное решение задачи* А, *то* $y^* = Tx^*$ — *оптимальное решение задачи* Б.

Доказательство. Действительно, если бы в B существовало $\bar{y}$, для которого $g(\bar{y}) < g(y^*)$, то для него мы имели бы

$$f(S\bar{y}) < f(Sy^*) = f(STx^*) = f(x^*),$$

что противоречит оптимальности x^*. ▲ [1])

Теорема 2. *Пусть задача* Б *является расширением задачи* А *и* T: $A \to B$ — *связующее отображение. Если* x^* — *оптимальное решение задачи* А, *то* $\min\{g(y) \mid y \in B\} \leqslant g(Tx^*)$. *Если* y^* — *оптимальное решение задачи* Б *и существует такой элемент* $x^* \in A$, *что* $Tx^* = y^*$, *то* x^* — *оптимальное решение задачи* А.

Доказательство. Первая часть теоремы очевидна. Предположим, что выполнено условие второй части, но существует $\bar{x} \in A$, $f(\bar{x}) < f(x^*)$. Тогда $g(T\bar{x}) < g(Tx^*)$, что противоречит оптимальности y^*. ▲

Приведем несколько преобразований, оставляющих эквивалентными задачи линейного программирования (связующие отображения при этом будут, как правило, тривиально простыми):

1. Линейное преобразование целевой функции с положительным коэффициентом:

$$c[N] \times x[N] \to \alpha + \beta \times (c[N] \times x[N]), \quad \beta > 0.$$

2. Умножение равенств на невырожденную матрицу. Если $\operatorname{rank}(r[M', M_2]) = |M_2|$, то следующая задача эквивалентна задаче (1)—(4):

> Найти вектор $x[N]$, удовлетворяющий (1), (2) и
> $$(r[M', M_2] \times a[M_2, N]) \times x[N] = r[M', M_2] \times b[M_2] \quad (6)$$
> и минимизирующий (4).

[1]) Этот знак будет ставиться в конце доказательства.

3. Замена целевой функции (4) любой функцией вида

$$(c[N]+\lambda[M_2]\times a[M_2,\ N])\times x[N].$$

4. Замена ограничения (2) ограничением вида

$$(a[M_1,\ N]+\mu[M_1,\ M_2]\times a[M_2,\ N]\times x[N])\geqslant \\ \geqslant b[M_1]+\mu[M_1,\ M_2]\times b[M_2].$$

5. Введение новых переменных $y[M^*]$, $M^*\subset M_1$ и замена ограничения (2) ограничениями

$$y[M^*]\geqslant 0[M^*], \tag{7}$$
$$a[M^*,\ N]\times x[N]-y[M^*]=b[M^*], \tag{8}$$
$$a[M_1\setminus M^*,\ N]\times x[N]\geqslant b[M_1\setminus M^*]. \tag{9}$$

Переменные $y[M^*]$ называются *дополнительными*, так как $y[i]$ дополняет i-е неравенство до равенства.

6. Замена группы уравнений (3), соответствующих множеству $M^*\subset M_2$, двумя группами неравенств

$$a[M^*,\ N]\times x[N]\geqslant b[M^*],$$
$$-a[M^*,\ N]\times x[N]\geqslant -b[M^*].$$

7. Замена переменной $x[j]$, $j\in N_2$, разностью двух новых неотрицательных переменных $x'[j]$ и $x''[j]$:

$$x[j]=x'[j]-x''[j]. \tag{10}$$

Связь новой задачи со старой в первых шести преобразованиях достаточно ясна. Покажем, как она устанавливается в последнем, седьмом преобразовании. Положим[1]) $x'[j]=(x[j])^+$, $x''[j]=(x[j])^-$. Для так определенных $x'[j]$, $x''[j]$ выполнение (10) очевидно. Напротив, если задано допустимое решение измененной задачи $x[N-j]$, $x'[j]$, $x''[j]$, то, полагая $x[j]=x'[j]-x''[j]$, получим решение исходной задачи. В целевую функцию и в ограничения (2) и (3) для новой задачи входит только разность $x'[j]-x''[j]$, поэтому справедливость этих ограничений при таких заменах не нарушится.

С помощью описанных эквивалентных преобразований любая основная задача линейного программирования

---

[1]) Как обычно, $(\alpha)^+$ — положительная часть α, т. е. $\max\{0,\ \alpha\}$, $(\alpha)^-$ — отрицательная часть α, т. е. $\max\{0,\ -\alpha\}$.

может быть приведена к стандартной. Для такого сведения нужно воспользоваться пятым и седьмым преобразованиями. Именно, положим $N_{21} = N_2 \times \{1\}$, $N_{22} = N_2 \times \{2\}$, $\bar{N} = N_1 \cup N_{21} \cup N_{22} \cup M_1$ и зададим матрицу $\bar{a}[M, \bar{N}]$ и вектор $\bar{c}[\bar{N}]$, как показано на рис. 7.

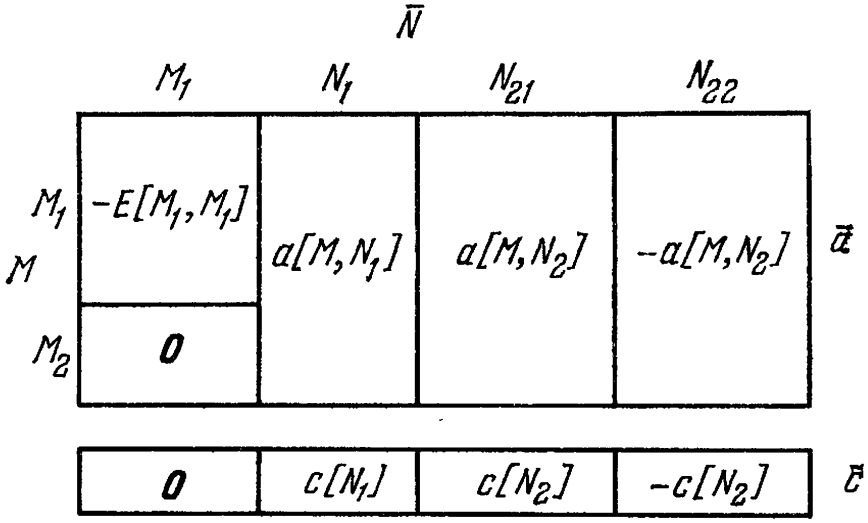

Рис. 7. Матрица стандартной задачи, построенной по общей.

Введя эти обозначения, мы уже можем выписать стандартную задачу:

Найти неотрицательный вектор $y[\bar{N}]$, удовлетворяющий условиям

$$\bar{a}[M, \bar{N}] \times y[\bar{N}] = b[M]$$

и минимизирующий значение

$$\bar{c}[\bar{N}] \times y[\bar{N}].$$

Эта задача эквивалентна основной задаче (1)—(4).

Эквивалентные преобразования будут широко использоваться при изложении методов решения задач линейного программирования.

Среди преобразований, *расширяющих* задачу линейного программирования, отметим следующие:

1. Ослабление ограничений. Под этим общим названием объединены в частности:

а) снятие какого-либо ограничения (любого вида);

б) замена уравнения неравенством;

в) уменьшение компонент вектора $b[M_1]$.

2. Замена ограничения (2) ограничением

$$(r[M', M_1]\times a[M_1, N])\times x[N]\geqslant r[M', M_1]\times b[M_1],$$

где $r[M', M_1]$ — неотрицательная матрица (часто используется даже умножение на неотрицательный вектор).

3. Умножение ограничений (3) на произвольную матрицу $r[M', M_2]$ (без предположений о полноте ранга).

4. Расширение вектора $x[N]$ добавлением новых переменных с любыми коэффициентами в целевой функции и в матрице ограничений.

Связь между исходными и расширенными задачами в каждом из этих преобразований очевидна, и мы на ней останавливаться не будем.

Отметим, что расширение задачи часто используется в целочисленном программировании для построения приближенных методов и для оценки минимума целевой функции. В линейном программировании преобразование 4 используется для приведения задачи к виду, более удобному для расчетов.

§ 3. Теория двойственности

Теория двойственности представляет собой самую красивую часть теории экстремума. Она объединяет многочисленные отдельные факты и методы, накопленные при анализе различных экстремальных задач, — метод неопределенных множителей Лагранжа, уравнение Эйлера из вариационного исчисления, геометрическую теорию двойственности выпуклых множеств Минковского.

К сожалению, здесь мы не имеем возможности вдаваться в детали, объясняющие смысл и происхождения двойственности. Формально введя некоторую задачу линейного программирования, которая базируется на том же наборе исходных данных, что и уже имеющаяся задача, покажем, что между этими двумя задачами имеется тесная связь.

В теоремах этого параграфа мы установим те связи между задачами, которые понадобятся для дальнейшего. В следующих главах при рассмотрении конкретных задач линейного программирования будет видно, какой смысл можно придать переменным возникающих двойственных задач.

Итак, задачей, *двойственной* к задаче (1)—(4), называется задача:

Максимизировать

$$v[M]\times b[M] \tag{11}$$

при условиях

$$v[M_1]\geqslant 0[M_1], \tag{12}$$

$$v[M]\times a[M,\ N_1]\leqslant c[N_1], \tag{13}$$

$$v[M]\times a[M,\ N_2]=c[N_2]. \tag{14}$$

Отметим, что если выписать задачу, двойственную к (11)—(14), мы снова получим задачу (1)—(4). Поэтому говорят обычно о *паре двойственных задач*, и только когда в этой паре выделена основная задача, ее называют *прямой* задачей, а вторую задачу — *двойственной*.

Если $x[N]$ — допустимое решение прямой задачи, а $v[M]$ — допустимое решение двойственной, то

$$c[N]\times x[N]\geqslant v[M]\times b[M]. \tag{15}$$

Действительно, так как

$$a[M_2,\ N]\times x[N]=b[M_2],$$

то

$$v[M_2]\times a[M_2,\ N]\times x[N]=v[M_2]\times b[M_2].$$

Так как $v[M_1]\geqslant 0[M_1]$ и $a[M_1,\ N]\times x[N]\geqslant b[M_1]$, то

$$v[M_1]\times a[M_1,\ N]\times x[N]\geqslant v[M_1]\times b[M_1]$$

и, значит,

$$v[M]\times a[M,\ N]\times x[N]\geqslant v[M]\times b[M]. \tag{16}$$

Аналогично получаем

$$c[N]\times x[N]\geqslant v[M]\times a[M,\ N]\times x[N], \tag{16'}$$

что вместе с (16) и дает (15).

Из (15) следует, что

$$\min c[N]\times x[N]\geqslant \max v[N]\times b[N], \tag{17}$$

где минимум и максимум берутся по множествам допустимых решений соответственно прямой и двойственной задач.

Оказывается, что в случае, когда и прямая и двойственная задачи имеют допустимые решения, в неравенстве (17) достигается знак равенства. Чтобы это доказать, нам потребуется один весьма глубокий геометрический факт, получивший название «теорема отделимости». Следующие ниже определения и рассуждения, как правило, могут легко быть истолкованы геометрически.

Вектор $y[N]$ называется *выпуклой комбинацией* векторов $x0[N]$ и $x1[N]$, если для некоторого $\lambda \in [0, 1]$

$$y[N] = \lambda \times x0[N] + (1 - \lambda) \times x1[N]. \tag{18}$$

Множество S называется *выпуклым*, если оно содержит все выпуклые комбинации из точек S. В частности, выпукло множество допустимых решений задач линейного программирования. Действительно, если $x0[N]$ и $x1[N]$ — два допустимых решения, т. е. если они удовлетворяют условиям

$$\begin{gathered} x[N_1] \geqslant 0[N_1], \\ a[M_1,\ N] \times x[N] \geqslant b[M_1], \\ a[M_2,\ N] \times x[N] = b[M_2], \end{gathered}$$

то для любого $\lambda \in [0, 1]$ и $y[N]$, определяемого формулой (18), все эти условия также выполняются.

Обозначим через R_N пространство всех векторов $x[N]$.

Теорема 3 (отделимости). *Пусть S — замкнутое выпуклое множество в R_N и $d[N] \notin S$. Существуют такие вектор $y[N]$ и число z, что*

$$d[N] \times y[N] < z$$

и

$$x[N] \times y[N] > z$$

для всех $x[N] \in S$.

Доказательство. Пусть

$$\rho(x[N]) = x[N] \times x[N]. \tag{19}$$

Из замкнутости S следует, что строго выпуклая функция $\rho_1(x[N]) = \rho(x[N] - d[N])$ достигает минимума на S в однозначно определяемой точке $x'[N]$.

В качестве вектора $y[N]$ примем $x'[N] - d[N]$. Пусть $z_0 = d[N] \times y[N]$. Тогда

$$z_1 = x'[N] \times y[N] = z_0 + \rho(y[N]) > z_0.$$

Пусть $x[N]$ — произвольная точка из S. Нам достаточно показать, что

$$x[N]\times y[N]\geqslant x'[N]\times y[N]. \tag{20}$$

Пусть $x_\lambda[N]=x'[N]+\lambda\times(x[N]-x'[N])$. При $\lambda\in(0, 1]$ вследствие выпуклости S имеем $x_\lambda[N]\in S$. Так как $x'[N]$ — единственный минимум функции ρ_1, то для всех $\lambda\in(0, 1]$

$$\rho_1(x'[N])<\rho_1(x_\lambda[N])=\rho_1(x'[N])+$$
$$+2\lambda\times(x'[N]-d[N])\times(x[N]-x'[N])+\lambda^2\times\rho(x[N]-x'[N]).$$

После очевидных преобразований получаем

$$2\times(x'[N]-d[N])\times(x[N]-x'[N])+$$
$$+\lambda\times\rho(x[N]-x'[N])>0,$$

и, следовательно,

$$(x'[N]-d[N])\times(x[N]-x'[N])\geqslant 0,$$

что эквивалентно (20). В качестве z можно взять любое число из (z_0, z_1). ▲

Теперь мы перейдем к доказательству равенства в (17).

Теорема 4 (двойственности). *Если и прямая и двойственная задачи имеют допустимые решения, то*

$$\min c[N]\times x[N]=\max v[M]\times b[M]. \tag{21}$$

Доказательство. Мы докажем эту теорему сначала для более простого случая стандартной задачи. Пусть $M_1=\emptyset$ и $N_2=\emptyset$. Предположим, что теорема неверна и существует такое α, что

$$\min c[N]\times x[N]>\alpha>\max v[M]\times b[M]. \tag{22}$$

Пополним множество M новым элементом i_0. Пусть $\bar{M}=M+i_0$, $b[i_0]=\alpha$, $a[i_0, N]=c[N]$. Рассмотрим систему уравнений

$$a[\bar{M}, N]\times x[N]=b[\bar{M}]. \tag{23}$$

По предположению эта система не имеет неотрицательных решений (такое неотрицательное решение было бы допустимым решением прямой задачи линейного программирования со значением целевой функции, равным α).

Иными словами, множество W векторов, представимых в виде $w[\bar{M}]=a[\bar{M}, N]\times x[N]$, где $x[N]\geqslant 0[N]$, не

содержит вектора $b[\bar{M}]$. Так как это множество выпукло и замкнуто, то в силу теоремы 3 найдутся такие $y[\bar{M}]$ и z, что $y[\bar{M}]\times b[\bar{M}]>z$ и $y[\bar{M}]\times w[\bar{M}]<z$ для всех $w[\bar{M}]\in W$, т. е. для всех $x[N]\geqslant 0[N]$ имеет место неравенство

$$y[\bar{M}]\times a[\bar{M},\ N]\times x[N]<z.$$

Так как при $x[N]=0[N]$ левая часть неравенства равна 0, то $z>0$. Далее, поскольку неравенство верно для векторов вида $\lambda\times E[N,\ j]$, $\lambda>0$, то

$$y[\bar{M}]\times a[\bar{M},\ j]<z/\lambda$$

для сколь угодно большого λ, и, следовательно,

$$y[\bar{M}]\times a[\bar{M},\ N]\leqslant 0[N]$$

и

$$y[\bar{M}]\times b[\bar{M}]>0.$$

Далее доказательство идет различными путями в зависимости от предположений о значении $y[i_0]$. Если $y[i_0]=0$, то $y[\bar{M}]\times a[M,\ N]\leqslant 0[N]$, $y[\bar{M}]\times b[M]=\varkappa>0$. Прибавляя к допустимому решению двойственной задачи $v[M]$ вектор $y[\bar{M}]$ с любым положительным множителем λ, мы снова получим допустимое решение. При этом значение целевой функции будет равно $v[M]\times b[M]+\lambda\times\varkappa$ и за счет выбора λ может быть сделано больше α, что противоречит выбору α.

При $y[i_0]>0$ рассмотрим вектор $y_1[M]=y[M]/y[i_0]$. Он удовлетворяет неравенствам

$$\begin{gathered}y_1[M]\times a[M,\ N]+c[N]\leqslant 0[N],\\ y_1[M]\times b[M]>-\alpha.\end{gathered}\qquad(24)$$

Пусть $x[N]$ — допустимое решение прямой задачи, т. е. $a[M,\ N]\times x[N]=b[M]$, $x[N]>0[N]$. Тогда $c[N]\times x[N]>\alpha$. Вместе с тем, умножая неравенства в (24) на $x[N]$, мы получим $y_1[M]\times b[M]+c[N]\times x[N]\leqslant 0$ и $c[N]\times x[N]<\alpha$, что невозможно.

Нам осталось рассмотреть случай $y[i_0]<0$. Положим $y_1[M]=-y[M]/y[i_0]$. Тогда $y_1[M]\times a[M,\ N]\leqslant c[N]$, т. е. $y_1[M]$ является допустимым решением двойственной задачи, и $y_1[M]\times b[M]>\alpha$, что противоречит предположению о выборе α. Следовательно, предположение (22)

не может иметь места, и для стандартной задачи равенство (21) доказано.

Как мы показали в § 2, основная задача линейного программирования может быть приведена к эквивалентной стандартной задаче (11)—(14). Задача, двойственная к ней, выглядит так:

Найти вектор $v[M]$, удовлетворяющий условиям

$$v[M]\times\bar{a}[M,\ \bar{N}]\leqslant c[\bar{N}], \tag{25}$$

и максимизирующий величину

$$v[M]\times b[M]. \tag{26}$$

Условие (25) можно переписать в виде

$$\begin{gathered} v[M]\times a[M,\ N_1]\leqslant c[N_1],\\ v[M]\times a[M,\ N_2]\leqslant c[N_2],\\ v[M]\times(-a[M,\ N_2])\leqslant -c[N_2],\\ v[M_1]\times(-E[M_1,\ M_1])\leqslant 0[M_1], \end{gathered}$$

что равносильно условиям

$$v[M_1]\geqslant 0[M_1], \tag{27}$$

$$v[M]\times a[M,\ N_1]\leqslant c[N_1] \tag{28}$$

и

$$v[M]\times a[M,\ N_2]=c[N_2]. \tag{29}$$

Эти условия, которые эквивалентны условию (25), в точности совпадают с условиями задачи, двойственной к основной.

Значения целевых функций на соответствующих друг другу решениях эквивалентных задач в данном случае совпадают. Поэтому из доказанного равенства экстремальных значений целевых функций в задаче (25), (26) и стандартной задаче следует равенство для основной задачи. ▲

Для оптимальных решений в неравенствах (16) и (16′), полученных при выводе неравенства (17), должен достигаться знак равенства. Поэтому справедливы следующие соотношения, получившие название *соотношений двойственности.*

Если для какого-либо оптимального $x^*[N]$ и для некоторого $j\in N_1$ имеет место $x^*[j]>0$, то для любого

оптимального $v^*[M]$

$$v^*[M]\times a[M,\ j]=c[j].$$

Если для какого-либо $i\in M_1$ и какого-либо оптимального $x^*[N]$ справедливо неравенство $a[i,\ N]\times x^*[N]>b[i]$, то $v^*[i]=0$ в любом оптимальном $v^*[M]$.

Докажем первое из этих утверждений. Пусть для некоторых оптимальных $x^*[N]$ и $v^*[M]$ нашлось такое $j\in N_1$, что одновременно $x^*[j]>0$ и $v^*[M]\times a[M,\ j]<c[j]$. Тогда $v^*[M]\times a[M,\ j]\times x^*[j]<c[j]\times x^*[j]$. Поскольку

$$v^*[M]\times a[M,\ N-j]\times x^*[N-j]\leqslant c[N-j]\times x^*[N-j],$$

то в (16) имеет место строгое неравенство, что невозможно.

Теорема 5 (критерий оптимальности). *Для того чтобы допустимое решение $x[N]$ прямой задачи было оптимальным, необходимо и достаточно, чтобы нашлось такое допустимое решение $v[M]$ двойственной задачи, что*

$$c[N]\times x[N]=v[M]\times b[M]. \tag{30}$$

Доказательство. Достаточность очевидна. Необходимость мы будем доказывать только для стандартной задачи.

Пусть $x[N]$ — оптимальное решение. При $\alpha<c[N]\times\times x[N]$ система уравнений

$$\begin{gathered}c[N]\times x[N]=\alpha,\\ a[M,\ N]\times x[N]=b[M]\end{gathered}$$

не имеет неотрицательных решений. Поэтому найдутся (ср. доказательство теоремы 4) такие y_0 и $y[M]$, что

$$y[M]\times a[M,\ N]+y_0\times c[N]\leqslant \mathbf{0}[N] \tag{31}$$

и

$$y[M]\times b[M]+y_0\times\alpha>0.$$

Умножая обе части (31) на $x[N]$ и производя необходимые преобразования, получим

$$y_0\times c[N]\times x[N]<\alpha\times y_0,$$

откуда следует, что $y_0 < 0$. Разделив на y_0 в (31) и положив $v[M] = -y[M]/y_0$, мы получим

$$v[M] \times a[M, N] \leqslant c[N],$$

т. е. двойственная задача имеет допустимое решение, а в этих условиях, согласно теореме 4, для оптимальности $x[N]$ необходимо выполнение равенства (30). ▲

В основе многих численных методов линейного программирования лежит критерий оптимальности допустимого решения, описанный в этой теореме. Имея некоторое допустимое решение, мы пытаемся построить решение двойственной задачи, уравнения для которого получаются из соотношений двойственности. Если нам удается построить такое решение, то решение прямой задачи оптимально, если же нет, то нужно искать другое решение. В следующем параграфе будет показано, как решения двойственной задачи могут помочь и в поиске улучшений.

§ 4. Базисные решения. Метод последовательных улучшений

В изучении выпуклых множеств особую роль играют так называемые крайние точки. Точка $x[N]$ из выпуклого множества S называется его *крайней точкой*, если из того, что

$$x[N] = \frac{1}{2} \times x'[N] + \frac{1}{2} \times x''[N] \text{ и } x'[N],\ x''[N] \in S$$

следует, что $x[N] = x'[N] = x''[N]$.

Иными словами, не существует лежащего в S отрезка положительной длины, серединой которого являлась бы крайняя точка $x[N]$.

Посмотрим, что представляют собой крайние точки множества допустимых решений стандартной задачи линейного программирования.

Найти $x[N]$, удовлетворяющий условиям

$$x[N] \geqslant 0[N], \tag{32}$$

$$a[M, N] \times x[N] = b[M] \tag{33}$$

и минимизирующий

$$c[N] \times x[N]. \tag{34}$$

Пусть $r = \operatorname{rank}(a[M, N])$. Будем считать (это необходимо для разрешимости условий (33)), что добавление к матрице $a[M, N]$ вектора $b[M]$ в качестве еще одного столбца не увеличивает ранга матрицы.

Пусть множество $N' \subset N$ такого, что $\operatorname{rank}(a[M, N']) = r$ и $|N'| = r$. Вектор $x[N]$, удовлетворяющий равенствам

$$a[M, N'] \times x[N'] = b[M], \tag{35}$$

$$x[N \setminus N'] = 0[N \setminus N'], \tag{36}$$

называется *базисным решением*. Множество N' называется *базисом*. Напомним, что $x[N]$ находится из (35), (36) однозначно. Действительно, достаточно выбрать $M' \subset M$ таким образом, чтобы $|M'| = r$, $\operatorname{rank}(a[M', N']) = r$, и положить

$$x[N'] = a^{-}[N', M'] \times b[M'],$$

где $a^{-}[N', M']$ — матрица, обратная к $a[M', N']$.

Одно и то же базисное решение может соответствовать различным базисам. Например, в задаче с условиями

$$3 \times x_1 + 4 \times x_2 + x_3 + 6 \times x_4 = 6,$$
$$x_1 - x_2 + 2 \times x_3 + 2 \times x_4 = 2$$

базисное решение $x_1 = 2$, $x_2 = x_3 = x_4 = 0$ соответствует любому из базисов $\{1, 2\}$ и $\{1, 3\}$.

Если $x[N'] \geqslant 0[N']$, базисное решение называется *допустимым базисным решением*.

Для простоты мы ограничимся сейчас случаем, когда $\operatorname{rank}(a[M, N]) = |M|$.

Введенные только что допустимые базисные решения и являются крайними точками множества допустимых решений. Именно, справедлива следующая теорема.

Теорема 6. *Для того чтобы вектор $x[N]$ был крайней точкой множества допустимых решений, необходимо и достаточно, чтобы он был допустимым базисным решением.*

Доказательство. Достаточность. Пусть $x[N]$ — допустимое базисное решение, соответствующее базису N'. Предположим, что нашлись такие допустимые решения $x'[N]$ и $x''[N]$, что $x[N] = \frac{1}{2} \times x'[N] + \frac{1}{2} \times x''[N]$. Обозначим через N'' множество положительных компонент вектора $x''[N]$. Очевидно, $N'' \subset N'$ и, по предположению,

$N' \setminus N'' \neq \emptyset$, откуда следует, что $|N''| < r = |M|$. Таким образом, система

$$a[M, N'] \times x[N'] = b[M]$$

имеет, кроме решения $x[N'] = a^-[N', M] \times b[M]$, еще и решение $x''[N']$, отличное от $x[N']$, что невозможно, так как ранг матрицы $a[M, N']$ равен r.

Необходимость. Пусть допустимое решение $x[N]$ не является базисным. Обозначим через N^+ множество положительных компонент $x[N]$. Тогда $|N^+| > k = \operatorname{rank}(a[M, N^+])$. Действительно, если $|N^+| = k$, то, расширяя множество N^+ до множества N', для которого $|N'| = r$, $\operatorname{rank}(a[M, N']) = r$, мы, вопреки предположению, установили бы, что N' является базисом, порождающим базисное решение $x[N]$.

Выберем $M'' \subset M$ и $N'' \subset N$ так, чтобы $|M''| = |N''| = k$, $\det(a[M'', N'']) \neq 0$. Положим $y[N''] = a^-[N'', M''] \times b[M'']$, где $a^-[N'', M'']$ — матрица, обратная к $a[M'', N'']$,

$$y[N \setminus N''] = 0[N \setminus N''].$$

Для достаточно малого ε векторы

$$x_1[N] = (1 + \varepsilon) \times x[N] - \varepsilon \times y[N],$$
$$x_2[N] = (1 - \varepsilon) \times x[N] + \varepsilon \times y[N]$$

будут неотрицательными (и допустимыми), и $x[N]$ является их полусуммой, что завершает доказательство для случая $y[N''] \neq 0[N'']$. Добиться того, чтобы $y[N'] \neq 0[N']$, очень просто — достаточно включить в M'' элемент i, для которого $b[i] \neq 0$. Если же $b[M] = 0[M]$, то

$$x[N] = \frac{1}{2} \times \left(\frac{1}{2} \times x[N]\right) + \frac{1}{2} \times \left(\frac{3}{2} \times x[N]\right). \blacktriangle$$

Вектор $x[N]$ называется *выпуклой комбинацией* векторов $y_i[M]$, $i \in M$, если найдется такой вероятностный вектор $\lambda[M]$, что

$$x[N] = \lambda[M] \times y_M[N] = \sum_{i \in M} \lambda[i] \times y_i[N].$$

Множество всех выпуклых комбинаций векторов $y_i[N]$ называется *выпуклой оболочкой* этих векторов.

Теорема 7. *Если множество допустимых решений задачи линейного программирования ограничено, оно является выпуклой оболочкой допустимых базисных решений.*

Доказательство. Мы покажем, что любое допустимое решение является выпуклой комбинацией допустимых базисных решений. Для допустимого решения $x[N]$ положим $N^+ = \{j \mid x[j] > 0\}$. Рассмотрим случай, когда $|N^+| > \operatorname{rank}(a[M, N^+])$. Это неравенство означает, что найдется такой вектор $\xi[N^+] \neq 0[N^+]$, что $a[M, N^+] \times \xi[N^+] = 0[M]$. Можно считать, что по крайней мере одна из компонент вектора $\xi[N^+]$ строго положительна. Вектор $\xi[N^+]$ не может быть положительным, так как в этом случае любой вектор вида $x_\lambda^+[N] = x[N] + \lambda \times \xi[N]$, где $\lambda > 0$, $\xi[N \setminus N^+] = 0[N \setminus N^+]$, являлся бы допустимым решением и множество допустимых решений, вопреки предположению, было бы не ограничено (к этому случаю мы позднее еще вернемся).

Таким образом, некоторые компоненты вектора $x_\lambda^+[N]$ уменьшаются с ростом λ, и при некотором критическом значении $\lambda = \lambda^*$ впервые одна из компонент становится равной 0. Пусть $j \in N^+$ и $\xi[j] < 0$. Компонента $x_\lambda^+[j]$ обращается в нуль при $\lambda = \lambda[j] = -x[j]/\xi[j] > 0$, а λ^* есть минимум из таких $\lambda[j]$.

Рассмотрим вектор $x_\lambda^-[N] = x[N] - \lambda \times \xi[N]$, аналогично найдем λ^{**}, при котором впервые одна из ненулевых компонент вектора $x_\lambda^-[N]$ становится равной 0.

$$\lambda^{**} = \min \{x[j]/\xi[j] \mid j \in N^+, \quad \xi[j] > 0\}.$$

Вектор $x[N]$ можно теперь представить в виде выпуклой комбинации $x_{\lambda^*}^+[N]$ и $x_{\lambda^{**}}^-[N]$:

$$x[N] = \frac{\lambda^{**}}{\lambda^* + \lambda^{**}} \times x_{\lambda^*}^+[N] + \frac{\lambda^*}{\lambda^* + \lambda^{**}} \times x_{\lambda^{**}}^-[N].$$

Множество положительных элементов каждого из этих двух векторов меньше чем N^+. Если число элементов в них меньше ранга соответствующих матриц, то к $x_{\lambda^*}^+$ и $x_{\lambda^{**}}^-$ можно применить ту же процедуру и представить x в виде выпуклой комбинации векторов с еще меньшими множествами положительных элементов. Так как в матрице $a[M, N]$ не может быть нулевых столбцов (соответствующее $x[j]$ могло бы принимать любые положительные значения вопреки ограниченности множества допустимых решений) и, следовательно, каждая одностолбцовая матрица имеет ранг 1, процесс закончится представлением вектора $x[N]$ в виде

$$x[N] = \mu[R] \times x_R[N],$$

где $\mu[R]$ — вероятностный вектор, а $x_r[N]$, $r \in R$ — допустимые решения, обладающие тем свойством, что если

$$N_r^+ = \{j \in N,\ x_r[j] > 0\},$$

то $|N_r^+| = \operatorname{rank}(a[M,\ N_r^+])$.

Покажем теперь, что допустимое решение $x[N]$, для которого $|N^+| = \operatorname{rank}(a[M,\ N^+])$, является базисным решением. Как известно, $\operatorname{rank}(a[M,\ N^+]) \leqslant |M|$. Так как, по предположению, $\operatorname{rank}(a[M,\ N]) = |M|$, то найдется такое множество $N' \supset N^+$, $|N'| = |M|$, что $\operatorname{rank}(a[M,\ N']) = |M|$. Так как

$$a[M,\ N^+] \times x[N^+] = b[M],$$
$$x[N \setminus N^+] = \mathbf{0}[N \setminus N^+],$$

то $a[M,\ N'] \times x[N'] = b[M]$, и, следовательно, $x[N]$ является базисным решением. ▲

В случае, если существует такой положительный вектор $\xi[N]$, что

$$a[M,\ N] \times \xi[N] = \mathbf{0}[M],$$

множество таких векторов $\mathscr{C}$ образует *выпуклый однородный конус*. Это значит, что оно удовлетворяет условию: если $\xi_1[N]$, $\xi_2[N] \in \mathscr{C}$, то для любых неотрицательных λ_1 и λ_2 вектор $\lambda_1 \times \xi_1[N] + \lambda_2 \times \xi_2[N] \in \mathscr{C}$.

Теорема 8. *Любое допустимое решение задачи линейного программирования представимо в виде суммы*

$$x[N] = x_0[N] + y[N],$$

где $y[N]$ — элемент конуса неотрицательных решений однородной системы линейных ограничений, а $x_0[N]$ — выпуклая комбинация базисных решений.

Доказательство повторяет доказательство теоремы 7 с той лишь разницей, что при выделении из $x[N]$ слагаемых с меньшим количеством положительных элементов может встретиться какой-либо элемент из конуса. Все такие элементы в сумме и составляют вектор $y[N]$. ▲

Теорема 9. *Если в задаче линейного программирования имеется оптимальное решение, то в ней имеется и оптимальное базисное решение.*

Доказательство. Пусть $x[N]$ — оптимальное решение. Из теоремы 8 следует, что его можно представить

в виде

$$x[N]=\sum_{i\in R}\lambda[i]\times x_i[N]+y[N],$$

где $x_i[N]$ — допустимые базисные решения, $\lambda[R]$ — вероятностный вектор, $y[N]$ — вектор из конуса однородных решений.

Так как $x[N]$ — оптимальное решение и при любом $\lambda\geqslant 0$ вектор $x[N]+\lambda\times y[N]$ является допустимым решением, то

$$c[N]\times y[N]\geqslant 0$$

(иначе значение целевой функции не ограничено), и допустимое решение

$$\sum_{i\in R}\lambda[i]\times x_i[N]$$

также является оптимальным. Так как $x_i[N]$ — допустимое решение, то

$$\gamma[i]=c[N]\times x_i[N]\geqslant\gamma_0=c[N]\times x[N],$$

а поскольку $\lambda[R]\times\gamma[R]=\gamma_0$, то $\gamma[i]=\gamma_0$ при $\lambda[i]>0$. Выберем такое i_0, чтобы $\lambda[i_0]>0$. Допустимое базисное решение $x_{i_0}[N]$ и будет оптимальным базисным решением. ▲

Геометрический смысл этой теоремы прозрачен: минимум линейной функции на выпуклом многограннике всегда достигается на одной из вершин. Из этого не следует, разумеется, что он достигается только на вершинах, и не во всех случаях нахождение оптимальной *вершины* соответствует постановке задачи.

Однако в подавляющем большинстве методов решения задач линейного программирования отыскиваются именно оптимальные базисные решения. Наибольшее распространение получил *метод последовательных улучшений*. В основе этого метода лежит способ, по которому, имея некоторое допустимое базисное решение, можно проверить его оптимальность и в случае его неоптимальности построить новое допустимое базисное решение с меньшим значением целевой функции. К рассмотрению этого метода мы сейчас и перейдем.

Упомянутая нами проверка оптимальности основывается на критерии оптимальности. Пусть задано допустимое базисное решение $x[N]$, соответствующее базису N'.

Таким образом, $|N'|=|M|$, $\det(a[M, N'])\neq 0$ и вектор $x[N']$, являющийся решением системы

$$a[M, N'] \times x[N'] = b[M], \tag{37}$$

неотрицателен, т. е.

$$a^{-}[N', M] \times b[M] \geqslant 0[N'],$$

где $a^{-}[N', M]$ — матрица, обратная к $a[M, N']$. Рассмотрим сначала случай $x[N'] > 0[N']$. В силу соотношений двойственности, если $x[N]$ — оптимальное решение, то каждое оптимальное решение двойственной задачи должно удовлетворять условию

$$v[M] \times a[M, N'] = c[N']. \tag{38}$$

Из этих уравнений находятся значения переменных $v[M]$:

$$v[M] = c[N'] \times a^{-}[N', M],$$

и нужно только проверить, дают ли они допустимое решение двойственной задачи, т. е. выполняются ли неравенства

$$v[M] \times a[M, N] \leqslant c[N].$$

Если неравенства выполняются, то решение $x[N]$ является оптимальным.

Предположим теперь, что для некоторого $j_0 \in N$

$$v[M] \times a[M, j_0] > c[j_0]. \tag{39}$$

Покажем, что при увеличении компоненты $x[j_0]$ (в базисном решении она равна 0, так как j_0 не входит в базис, это легко видеть, сравнивая (38) и (39)) и соответствующем изменении $x[N']$ линейная форма убывает. Действительно, пусть $x[j_0]=\varepsilon$. Условия (33), отбрасывая переменные, которые остаются равными нулю, можно представить в виде

$$a[M, N'] \times x[N'] = b[M] - \varepsilon \times a[M, j_0]. \tag{40}$$

Найдем $x[N']$ из этой системы, обозначив его через $x_\varepsilon[N']$ и полагая $x_\varepsilon[j_0]=\varepsilon$, $x_\varepsilon[N \setminus (N' + j_0)] = 0[N \setminus (N' + j_0)]$. Мы получим

$$x_\varepsilon[N'] = x[N'] - \varepsilon \times a^{-}[N', M] \times a[M, j_0]$$

и

$$c[N]\times x_\varepsilon[N]=c[N']\times x_\varepsilon[N']+c[j_0]\times\varepsilon=$$
$$=c[N']\times x[N']+\varepsilon\times(c[j_0]-$$
$$-c[N']\times a^-[N',\ M]\times a[M,\ j_0])=c[N']\times x[N']+$$
$$+\varepsilon\times(c[j_0]-v[M]\times a[M,\ j_0])<c[N']\times x[N'].$$

Очевидно, что чем больше ε, тем меньше значение целевой функции. Однако его увеличение ограничивается требованием допустимости решения $x_\varepsilon[N]$. Уравнения (40) гарантируют выполнение равенств (33). Неотрицательность переменных $x_\varepsilon[N\setminus N']$ также достигается автоматически (переменная $x_\varepsilon[j_0]$ неотрицательна, а остальные равны 0). Таким образом, остается следить за неотрицательностью вектора

$$x_\varepsilon[N']=x[N']-\varepsilon\times\alpha[N'],\qquad(41)$$

где

$$\alpha[N']=a^-[N',\ M]\times a[M,\ j_0].$$

При $\alpha[j]\leqslant 0$ величина $x[j]-\varepsilon\times\alpha[j]$ не убывает с ростом ε, поэтому нам достаточно ограничиться множеством $N_\alpha=\{j\in N',\ \alpha[j]>0\}$. Пусть N_α непусто. Положим

$$\lambda=\min\{x[j]/\alpha[j]\mid j\in N_\alpha\}.\qquad(42)$$

Обозначим через j_1 индекс, на котором в (42) достигается минимум. При $\varepsilon\leqslant\lambda$ вектор (41) неотрицателен, а при $\varepsilon>\lambda$ имеет место $x[j_1]-\varepsilon\times\alpha[j_1]<0$. Таким образом, в качестве максимального ε следует взять λ. (Если N_α пусто, то λ, а с ним и целевая функция, неограничены.)

Покажем, что допустимый вектор $x_\lambda[N]$ является базисным решением, соответствующим базису $N''=N+j_0-j_1$. Так как $|N''|=|N'|$, нам достаточно убедиться в том, что $\det(a[M,\ N''])\neq 0$. Покажем, что существует матрица $\bar a[N'',\ M]$, обратная к $a[M,\ N'']$.

Будем искать эту матрицу в виде

$$\bar a[N'',\ M]=d[N'',\ N']\times a^-[N',\ M].$$

Тогда матрица $d[N'',\ N']$ должна удовлетворять условиям

$$d[N'',\ N']\times a1[N',\ N'']=E[N'',\ N''],$$

где $a1[N', N''] = a^{-}[N', M] \times a[M, N'']$. Положим $N^0 = N'' - j_0 = N' - j_1$. Матрица $a1[N', N'']$ имеет вид

$$\begin{aligned} a1[N', N^0] &= E[N', N^0], \\ a1[N', j_0] &= \alpha[N']. \end{aligned} \tag{43}$$

Зададим матрицу $d[N'', N']$ следующим образом:

$$\begin{aligned} d[N'', N^0] = E[N'', N^0],\ d[N^0, j_0] &= -\alpha[N^0]/\alpha[j_1], \\ d[j_1, j_0] &= 1/\alpha[j_1]. \end{aligned} \tag{44}$$

Обозначим произведение $d[N'', N'] \times a1[N', N'']$ через $r[N'', N'']$. Нам нужно показать, что $r[N'', N'']$ — единичная матрица. Вычислим отдельно:

$$\begin{aligned} r[N'', N^0] &= d[N'', N^0] \times a1[N^0, N^0] + d[N'', j_1] \times a1[j_1, N^0], \\ r[N^0, j_1] &= d[N^0, N^0] \times a1[N^0, j_0] + d[N^0, j_1] \times a1[j_1, j_0], \\ r[j_0, j_0] &= d[j_0, N^0] \times a1[N^0, j_0] + d[j_0, j_1] \times a1[j_1, j_0]. \end{aligned}$$

Подставляя выражения из (43) и (44), получим

$$\begin{aligned} r[N'', N^0] &= E[N'', N^0] \times E[N^0, N^0] + \\ &\quad + d[N'', j_1] \times 0[j_1, N^0] = E[N'', N^0], \\ r[N^0, j_0] &= E[N^0, N^0] \times \alpha[N^0] - \alpha[j_1] \times \alpha[N^0]/\alpha[j_1] = \\ &\quad = 0[N^0] = E[N^0, j_0], \\ r[j_0, j_0] &= E[j_0, N^0] \times \alpha[N^0] + \alpha[j_1]/\alpha[j_1] = 1 = E[j_0, j_0]. \end{aligned}$$

Таким образом, матрица $d[N'', N']$ является обратной к $a1[N', N'']$ и матрица $d[N'', N'] \times a^{-}[N', M]$ является обратной к $a[M, N'']$. Очевидно, что с точностью до знака

$$\det(a[M, N'']) = \det(a[M, N']) \times \alpha[j_0] \neq 0.$$

Итак, по каждому неоптимальному допустимому базисному решению со строго положительной базисной частью $x[N']$ можно построить другое допустимое базисное решение с меньшим значением целевой функции. Убывание значений целевой функции гарантирует, что базис, встретившийся однажды, уже не появится при итерациях этой процедуры улучшений.

Метод построения улучшений сам по себе не требует строгой положительности $x[N']$. Однако при $x[N'] \geqslant 0[N']$ может не произойти уменьшения значения целевой функции и вычислительный процесс может зациклиться: найдется конечный набор допустимых базисных решений, который

будет повторяться в циклическом порядке. Это вырождение процесса может быть устранено введением более сложной процедуры выбора индекса j_1, на котором достигается минимум в (42).

Такую процедуру и доказательство ее работоспособности мы рассмотрим в следующем параграфе. Отметим, однако, что в большинстве практически используемых программ опасность зацикливания игнорируется, так как ни в одной из огромного количества решенных практических задач таких зацикливаний не встретилось.

В настоящее время разработано довольно много различных реализаций метода последовательных улучшений. Они отличаются тем, как осуществляется решение систем (37), (38) и системы

$$a[M,\ N'] \times \alpha[N'] = a[M,\ j_0]. \qquad (45)$$

Разумеется, возможно непосредственное решение этих систем заново на каждом шаге процесса. Однако этот путь не рационален с точки зрения экономии вычислений. Ниже, в § 6 и далее, мы рассмотрим несколько достаточно экономных реализаций метода последовательных улучшений.

Мы рассмотрим один шаг улучшений, который может повторяться многократно. Однако должно быть какое-то допустимое базисное решение, с которого можно было бы начать — начальное допустимое базисное решение. Проще всего такое решение можно получить с помощью так называемого *метода искусственных переменных*.

Перепишем равенства (33) так, чтобы получить $b[M] \geqslant \geqslant 0[M]$, и включим в каждое равенство по одной новой переменной с коэффициентом 1. Считая для простоты, что $N \cap M = \emptyset$, мы можем новые переменные обозначить через $x[M]$. Добавлением этих переменных мы расширили нашу исходную задачу. Цель этого расширения — упрощение поиска начального базисного решения. Однако нельзя позволять, чтобы эти новые, искусственные, переменные оставались в окончательном решении с положительными значениями.

Чтобы предотвратить это, нужно сделать их использование невыгодным. Для этого в целевую функцию их включают с «очень большими» коэффициентами. Вводится число gz, которое считается больше любого встречающегося

в вычислениях числа. Фактически это означает, что каждое встречающееся нам число представляется в виде $\alpha \times gz + \beta$, и число $\alpha_1 \times gz + \beta_1$ больше числа $\alpha_2 \times gz + \beta_2$, если $\alpha_1 > \alpha_2$ или $\alpha_1 = \alpha_2$ и $\beta_1 > \beta_2$ (т. е. если вектор (α_1, β_1) лексикографически больше вектора (α_2, β_2)).

Полагая $a[M, M] = E[M, M]$, $c[M] = gz \times \mathbf{1}[M]$, $\bar{N} = N \cup M$, мы получим следующую задачу.

Задача с искусственным базисом.

Найти $x[\bar{N}]$, удовлетворяющий условиям

$$x[\bar{N}] \geqslant \mathbf{0}[\bar{N}],$$

$$a[M, \bar{N}] \times x[\bar{N}] = b[M]$$

и минимизирующий

$$c[\bar{N}] \times x[\bar{N}].$$

В этой задаче начальное допустимое базисное решение всегда имеется и легко находится. Это решение $x[N] = \mathbf{0}[N]$, $x[M] = b[M]$. Дальнейшие вычисления можно проводить двумя путями: сохраняя искусственные переменные до конца (вводя строку коэффициентов при gz или задавая для конкретных задач достаточно большое реальное gz) или первоначально решив вспомогательную задачу минимизации суммы искусственных переменных.

В обоих случаях получение в оптимальном решении положительной искусственной переменной означает, что исходная задача не имеет допустимых решений. Действительно, пусть $x^*[\bar{N}]$ — оптимальное решение задачи с искусственным базисом, а $x[N]$ — какое-либо допустимое решение исходной задачи. Пусть $x^*[i] > 0$ для некоторого $i \in M$. Значение целевой функции на $x^*[\bar{N}]$ равно $\alpha \times gz + \beta$, где $\alpha \geqslant x^*[i] > 0$, а значение целевой функции на $x[N]$ равно $c[N] \times x[N] = 0 \times gz + \beta 1$, что по условию меньше.

Может случиться, что в базис оптимального решения входит искусственная переменная с нулевым значением. Это означает, что $\operatorname{rank}(a[M, N]) < |M|$.

Метод искусственного базиса достаточно прост и универсален. Однако не следует пренебрегать особенностями конкретных задач, позволяющими получить более качественное начальное допустимое базисное решение.

Например, если имеем ограничения в форме неравенств

$$a[M_1, N] \times x[N] \leqslant b[M_1] \qquad (46)$$

и при этом $b[M_1] \geqslant 0[M_1]$, то при сведении этих неравенств к равенствам

$$a[M_1,\ N] \times x[N] + y[M_1] = b[M_1] \qquad (47)$$

переменные $y[M_1]$ можно ввести в базис, и для этих ограничений искусственных переменных уже не требуется.

Случай $b[i] < 0$ несколько сложнее. Кроме таких же дополнительных переменных, как в (47), нужно во все такие ограничения добавить с коэффициентом -1 одну и ту же искусственную переменную $x[j^-]$ так, чтобы ограничения приняли вид

$$a[i,\ M] \times x[M] + y[i] - x[j^-] = b[i]. \qquad (48)$$

Пусть i^- — индекс, на котором достигается $\min b[i]$. В качестве базисных переменных, соответствующих ограничениям (48), можно принять $y[i]$ при $i \neq i^-$ и $x[j^-]$ для i^-.

Значениями этих базисных переменных будут

$$x[j^-] = -\min b[i], \qquad y[i] = b[i] + x[j^-], \qquad i \neq i^-.$$

Начальная матрица будет лишь одним столбцом (соответствующих $x[j^-]$) отличаться от единичной, и построить для нее обратную большого труда не составит.

§ 5. Устранение зацикливания и сходимость метода последовательных улучшений

Идея устранения зацикливания очень проста: нужно так видоизменить задачу, чтобы на каждой итерации целевая функция строго убывала. Для этого к правым частям ограничений (33) добавляются пренебрежимо малые добавки различного порядка так, чтобы устранить возможность одновременного достижения минимума в (42) на нескольких $j \in N'$.

Пусть для простоты $M = 1 : m$. Пусть eps — «очень малое число» в том же смысле, как «очень большое число» в предыдущем параграфе. Положим [1])

$$b_{eps}[i] = b[i] + eps^i, \qquad i \in 1 : m.$$

Все переменные $x[N']$ и значение целевой функции для каждого допустимого базисного решения будут полиномами

[1]) Редкий для нас случай: eps^i обозначает eps в степени i.

от eps степени m, причем их свободные члены будут равны значениям соответствующих величин в «невозмущенной» задаче. Полиномы, соответствующие в данном базисном решении различным базисным переменным, оказываются при этом различными. Действительно, для того чтобы получить набор коэффициентов такого полинома, как вектор, нужно матрицу $a^-[N', M]$ (обратную базисной матрице $a[M, N']$) умножить на матрицу, составленную из вектора $b[M]$ и единичной матрицы, что дает матрицу, составленную из вектора $\beta[N']$ и матрицы $a^-[N', M]$. Равенство (и даже пропорциональность) двух полиномов означало бы равенство (пропорциональность) двух строк этой матрицы, что невозможно ввиду ее невырожденности.

При нахождении минимума в (42) мы должны найти «наименьший» из полиномов $\beta_j(eps)/\alpha[j]$. При этом мы считаем, что полином $f_0+f_1\times eps+\ldots+f_m\times eps^m$ меньше, чем полином $g_0+g_1\times eps+\ldots+g_m\times eps^m$, если вектор $(f_0, f_1, \ldots, f_m)$ лексикографически меньше вектора $(g_0, g_1, \ldots, g_m)$.

Отметим, что из лексикографической неотрицательности вектора коэффициентов полинома следует неотрицательность самого полинома при достаточно малых значениях аргумента. Таким образом, метод последовательных улучшений выглядит теперь так: начальное допустимое базисное решение должно быть выбрано таким образом, чтобы все переменные были строго лексикографически положительны. Если мы начинаем с единичной матрицы и неотрицательных значений базисных переменных, то это условие выполняется автоматически.

На каждом шаге метода последовательных улучшений мы также будем иметь набор строго лексикографически положительных значений базисных переменных. Выбрав переменную j_0, которую нужно ввести в базис, мы будет искать переменную, выводимую из базиса, и коэффициент λ_{eps} таким образом, чтобы λ_{eps} (который также является полиномом по eps) был лексикографически максимальным, а все значения $x_{eps}[N']$ были лексикографически неотрицательны. Для этого в множестве $N_0^+=\{j\in N',\ \alpha[j]>0\}$ выбирается множество N_1^+ таких $j\in N_0^+$, для которых достигается минимум отношения $x[j]/\alpha[j]$. Если в N_1^+ больше одного элемента, то выбирается подмножество N_2^+

тех $j \in N_1^+$, на которых достигается минимум $a^-[j, 1]/\alpha[j]$, затем определяется N_3^+ как множество тех $j \in N_2^+$, на которых достигается минимум $a^-[j, 2]/\alpha[j]$ и т. д. Так как среди полиномов нет пропорциональных, на каком-то k-м шаге множество N_k^+ будет состоять из одного элемента. Его и следует принять за j_1. В качестве λ_{eps} примем

$$\lambda_0 + \lambda_1 \times eps + \ldots + \lambda_m \times eps^m,$$

где

$$\lambda_0 = \min \{x[j]/\alpha[j] \mid j \in N_0^+\}$$

и

$$\lambda_i = \min \{a^-[j, i]/\alpha[j] \mid j \in N_i^+\},$$
$$N_i^+ = N_k^+ \text{ при } i \geqslant k.$$

Легко видеть, что λ_{eps} лексикографически неотрицателен и что новое значение целевой функции будет равно

$$c[N] \times x_{eps}[N] + \lambda_{eps} \times \gamma,$$

т. е. будет лексикографически строго меньше предыдущего.

Поскольку переход от одного базиса к другому сопровождается уменьшением (хотя бы только лексикографически) целевой функции, возвращение к уже встречавшемуся базису становится невозможным и ввиду ограниченности множества возможных базисов инеративный процесс сходится за конечное число шагов.

§ 6. Некоторые реализации метода последовательных улучшений

В этом параграфе будут изложены в виде процедур на языке алгол-60 несколько наиболее простых реализаций метода последовательных улучшений. Общая схема этого метода описывается следующей процедурой:

```
procedure LP opt basic solution (start, new, work, fin);
    procedure start, new, work, fin;
begin boolean optimum;
    start;
mk1: new (optimum); if optimum then go to mk2;
    work; go to mk1;
mk2: fin
end LP obs
```

Здесь *start* — процедура подготовки данных для вычислений (начальное заполнение массивов, построение начального допустимого базисного решения и др.), *new* — процедура проверки решения на оптимальность (*optimum* ≡ 'решение оптимально') и выработки вводимой в базис переменной при *optimum* ≡ **false**, *work* — процедура нахождения выводимой из базиса переменной и изменения записи в соответствии с новым базисом, *fin* — процедура оформления в нужном виде окончательного решения.

а) Симплекс-метод. Симплекс-метод — наиболее простая из процедур метода последовательных улучшений. В нем информация о текущем состоянии вычислительного процесса задается перечнем базисных переменных и их значений, матрицей

$$a^-[N', M]\times a[M, N],$$

вектором

$$v[M]\times a[M, N]-c[N], \tag{49}$$

где

$$v[M]=c[N']-a^-[N', M],$$

и значением $c[N']\times x[N']$. Как правило, в качестве M принимается $1:m$, а в качестве N — множество $1:n$.

Таким образом, информацию о вычислительном процессе (запись) мы можем представить в виде целочисленного вектора $base[1:m]$, задающего одно-однозначное отображение T: $1:m \to N'$, и матрицы $as[0:m, 0:n]$, где

$$\begin{aligned}
as[1:m,\ 1:n] &= a^-[T(1:m),\ M]\times a[M,\ 1:n],\\
as[0,\ 1:n] &= c[1:n]-v[M]\times a[M,\ 1:n],\\
as[1:m,\ 0] &= x[T(1:m)]=a^-[T(1:m),\ M]\times b[M],\\
as[0,\ 0] &= c[N']\times x[N'].
\end{aligned}$$

Для искусственных переменных используется множество индексов $n+1:n+m$. Соответственно доопределяется и матрица as.

Следующая ниже процедура *start*1 формирует искусственный базис с заданным «большим числом» *gz*. Считается, что $b[M] \geqslant 0[M]$:

```
procedure start1;
 begin integer i, j; as[0, 0]:=0; iter:=0;
   for j:=1 step 1 until n do as [0, j]:=—c[j];
   for i:=1 step 1 until m do
     begin as[i, 0]:=b[i];
       for j:=1 step 1 until n do
         begin as[i, j]:=a[i, j];
                  as[0, j]:=as[0, j]+gz×a[i, j] end;
       for j:=n+1 step 1 until n+m do as[i, j]:=0;
       as[i, n+i]:=1; base[i]:=n+i;
                  as[0, 0]:=as[0, 0]+a[i, 0]×gz
     end i;
 for j:=n+1 step 1 until n+m do as[0, j]:=0
end start
```

Поясним написанное: в основную часть матрицы $as[1:m, 1:n]$ переписана матрица $a[1:m, 1:n]$, в $as \times$ $\times[1:m, 0]$ — вектор $b[1:m]$. В $base[1:m]$ записаны номера базисных переменных (искусственных). Матрица $as[N', N']$ формируется и должна оставаться единичной. $as[0, 1:m]$ формируется по формуле (49).

Переменная *iter*, которой присваивается значение 0, является целочисленной переменной, описанной вне процедуры. Она предназначена для счета числа итераций.

Перейдем теперь к описаниям других процедур. Пояснения будут даваться после описаний:

```
procedure new1 (opt); boolean opt;
begin integer j; real max; j0:=0; max:=0;
   for j:=1 step 1 until n do
     if as[0, j]>max then
      begin max:=as[0,j]; j0:=j end;
   opt:=max=0
end new
```

Здесь производится перебор всех $j \in 1:n$ с запоминанием максимального значения $as[0, j]$ в max, а j, на котором достигается этот максимум, в $j0$:

```
procedure work1;
begin integer i, j, i0; real min, a1; boolean un;
  iter:=iter+1; un:=true; min:=gz;
  for i:=1 step 1 until m do if as[i, j0]>0 then
    begin a1:=as[i, 0]/as[i, j0];
      if un ∨ min>a1 then
       begin min:=a1; i0:=i; un:=false end
    end;
  if un then go to unbo;
```

comment $un \equiv$ **true**, если целевая функция задачи не ограничена. Условием такой неограниченности является $as[1:m, j0] \leqslant 0\ [1:m]$. Это условие и проверяется циклом по i. Как только в столбце $as[1:m, j0]$ найдется положительная компонента, un полагается равным **false** и больше не меняется. Если оказалось, что линейная форма не ограничена, обычно предусматривается какая-либо специальная печать;

```
  base[i0]:=j0; a1:=1/as[i0, j0];
  for j:=0 step 1 until n+m do
                  as[i0, j]:=as[i0, j]×a1:
  for i:=0 step 1 until i0-1, i0+1
                                  step 1 until m do
    begin a1:=as[i, j0];
      for j:=0 step 1 until n+m do
        as[i, j]:=as[i, j]-a1×as[i0, j]
    end
end work
```

После нахождения $i0$ — места выводимой из базиса переменной — происходит пересчет матрицы — умножение ее на матрицу $d[N'', N']$. Для этого строка $as[i0, N]$ делится на $as[i0, j0]$, а из оставшихся строк вычитается строка $as[i0, N]$ с соответствующим весом. Отметим, что после выполнения этих вычислений в столбце $as[M, j0]$ стоит 1 на пересечении со строкой $i0$ и 0 в остальных строчках.

Наконец, процедура *fin* может быть описана так:

```
procedure fin1;
begin integer j; array res[1 : n];
  for j := 1 step 1 until n do x[j] := 0;
  for j := 1 step 1 until m do x[base[j]] := as[j, 0];
  print (x, iter)
end fin
```

Эта процедура выдает на печать (процедура выдачи обозначена *print*) оптимальное решение $x[1:n]$ и число итераций *iter*. В реально используемых программах при решении задач с большим n такая выдача информации будет нецелесообразной. Обычно на печать выдается список базисных переменных с их значениями. Кроме того, целесообразно вычислять для полученных переменных значение $a[M, N] \times x[N]$ для контрольного сопоставления его с $b[M]$.

В реальных алгоритмах используется также метод порога для выбора вводимой в базис переменной. При большом числе переменных нет смысла перебирать все переменные в поисках максимума $as[0, j]$. Вместе с тем для уменьшения числа итераций все же желательно, чтобы вводимые в базис переменные имели достаточно высокий показатель $as[0, j]$. Для этого показатели сравниваются с некоторым порогом. Когда находится переменная, у которой $as[0, j]$ больше этого порога, она вводится в базис. Если такой переменной не находится, то после полного просмотра в базис вводится переменная с максимальным значением $as[0, j]$, а порог уменьшается. В качестве нового значения порога можно принять, например, $as[0, j0]/2$. Просмотр переменных на следующей итерации продолжается при этом с того места, на котором он оборвался.

Условие $as[0, N] \leqslant 0[N]$ для оптимального решения на практике также претерпевает небольшое изменение. Мы прекратим вычисления, когда $\max as[0, j]$ будет меньше наперед заданного малого числа *eps*. Нужно только во избежание неправильной остановки процесса следить за тем, чтобы порог не становился меньше *eps*.

С учетом всех этих изменений общую программу симплекс-метода для решения задач линейного программирования можно записать следующим образом:

```
procedure LP sol1 (m, n, gz, eps, a, b, c);
   integer m, n; real gz, eps; array a, b, c;
begin integer iter, j0, j1; real lev; boolean opt;
   array as[0 : m, 0 : m+n]; integer array base[1 : m];
procedure start 1;
begin integer i, j; as[0, 0]:=0; iter:=0;
   for j:=1 step 1 until n do as[0, j]:=—c[j];
   for i:=1 step 1 until m do
     begin as[i, 0]:=b[i];
        for j:=1 step 1 until n do
          begin as[i, j]:=a[i, j];
                as[0, j]:=as[0, j]+gz×a[i, j] end;
        for j:=n+1 step 1 until n+m do as[i, j]:=0;
        as[i, n+i]:=1; base[i]:=n+i;
                as[0, 0]:=as[0, 0]+as[i, 0]×gz
        end i;
   for j:=n+1 step 1 until n+m do as[0, j]:=0;
end start;
procedure work1;
begin integer i, j, i0; real min, a1; boolean un;
   iter:=iter+1; un:=true; min:=gz;
   for i:=1 step 1 until m do if as[i, j0]>0 then
     begin a1:=as[i, 0]/as[i, j0];
        if un \/ min>a1 then
          begin min:=a1; i0:=i; un:=false end
     end;
   if un then go to unbo;
   base[i0]:=j0; a1:=1/as[i0, j0];
   for j:=0 step 1 until n+m do as[i0, j]:=as[i0, j]×a1;
   for i:=0 step 1 until i0—1, i0+1 step 1 until m do
     begin a1:=as[i, j0];
        for j:=0 step 1 until n+m do
          as[i, j]:=as[i, j]—a1×as[i0, j]
     end
end work;
procedure new2 (opt); boolean opt;
begin integer j: real max; j0:=j1; max:=0;
   for j:=j1+1 step 1 until n, 1 step 1 until j1—1 do
   if as[0, j]>max then
    begin max:=as[0, j]; j0:=j;
          if max>lev then go to mk end;
```

```
    lev:=max/2; if lev<eps then lev:=eps;
mk: opt:=max<eps; j1:=j0
  end new;
  procedure fin2;
  begin integer i, j; real al;
    for i:=1 step 1 until m do begin a1:=0;
      for j:=1 step 1 until m do
        if base[j]≤n then a1:=a1+a[i, base[j]]×as[j, 0];
      print (base[i], as[i, 0], a1, b[i], as[0, n+i]+gz)
  end end fin;
    lev:=gz; j1:=1;
    start1;
mk1: new2 (opt); if opt then go to mk2;
    work1; go to mk1;
mk2: fin2; go to mk3;
unbo: print (j0, as[0, j0]); fin2;
mk3: end LPsol
```

Решим с помощью этой процедуры небольшой численный пример.

Пример. Максимизировать

$$3\times x_1+2\times x_2+x_3$$

(или, что то же, минимизировать $-3\times x_1-2\times x_2-x_3$) при условиях

$$\begin{aligned} x_1+2\times x_2+x_3 &= 5,\\ -x_1 \quad -2\times x_3+x_4 &= 3,\\ x_j &\geqslant 0. \end{aligned}$$

На рис. 8 изображена форма для информации о состоянии вычислительного процесса в *LP sol* 1 при $m=2$ и $n=4$, а на рис. 9 выписаны в табличном виде исходные данные примера. Примем $gz=1000$ и $eps=0{,}01$. После перехода к *LP opt basic solution* и действия процедуры *start* 1 состояние процесса примет вид рис. 10. *new2* (*opt*) дает $j0=2$, $max=2002$, $opt\equiv$ ***false***. Поэтому переходим к процедуре *work* 1, в которой получаем $i0=1$ и после необходимых преобразований приходим к записи, изображенной на рис. 11 (не изменяющиеся в ходе вычислений величины *m*, *n*, *gz*, *eps* в дальнейшем опускаются).

Возвращаемся к метке *mk* 1 и, выполняя оператор процедуры *new2*, получаем $j0=4$ и $opt\equiv$ ***false***, процедура

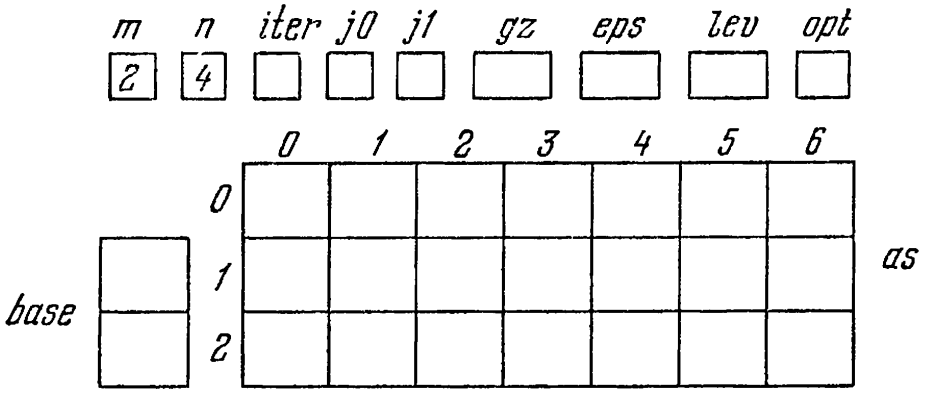

Рис. 8 Рабочая память в симплекс-методе.

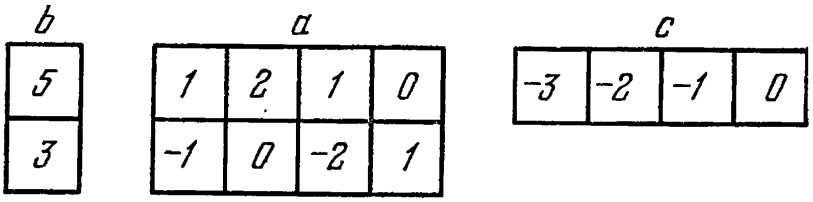

Рис. 9. Исходные данные.

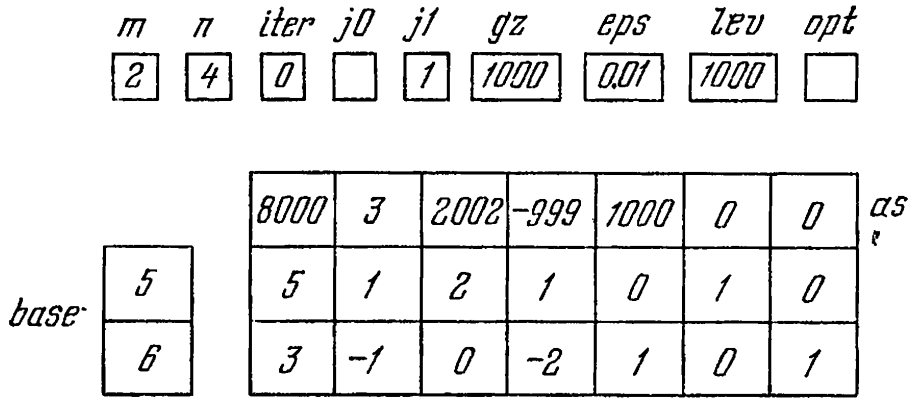

Рис. 10. Заполнение рабочей памяти после процедуры *start*.

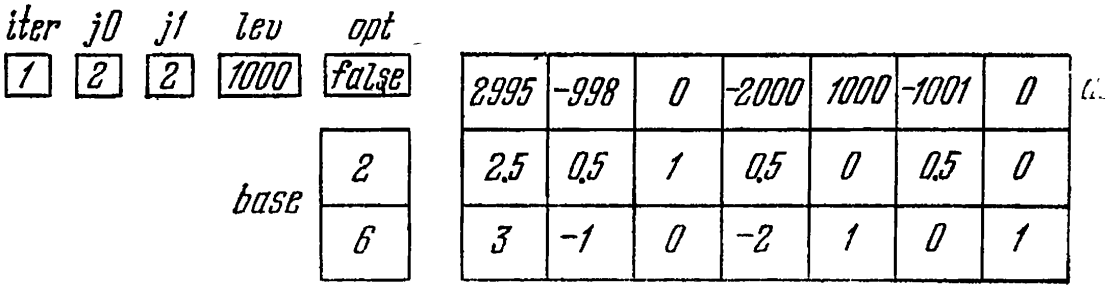

Рис. 11. Результат первой итерации.

*work*1 дает $i0 = 2$ и запись рис. 12. Рис. 13 изображает состояние процесса после третьей итерации.

iter	*j0*	*j1*	*lev*	*opt*
2	4	4	1000	false

base								
	2	-5	2	0	0	0	-1007	-1000	*as*
	4	2.5	0.5	1	0.5	0	0.5	0	
		3	-1	0	-2	1	0	1	

Рис. 12. Результат второй итерации.

iter	*j0*	*j1*	*lev*	*opt*
3	1	1	1	false

base								
	1	-15	0	-4	-2	0	-1003	-1000	*as*
	4	5	1	2	1	0	1	0	
		8	0	2	-1	1	1	0	

Рис. 13. Результат третьей итерации.

При следующем обращении к *new*2 мы получим *opt* $\equiv$ $\equiv$ **true** и в действие вступает процедура *fin*2, которая выдает следующую информацию:

```
 1  5  5  5  3
 4  8  3  3  0
15  3
```

т. е. $x[1] = 5$, $v[1] = 3$, $x[4] = 8$, $v[2] = 0$, значение максимизируемой функции равно 15, и решение получено за 3 итерации.

При решении серьезных практических задач симплекс-метод используется крайне редко; как правило, он фигурирует лишь в руководствах по линейному программированию.

Обычно в программах используются другие методы представления рабочей информации, два из которых мы рассмотрим ниже.

б) Метод обратной матрицы (модифицированный симплекс-метод). В этом методе запись состоит из неизменного задания исходных данных, задания базиса (например, как раньше, с помощью массива *base*), вектора

значений базисных переменных $x[N']$, вектора значений двойственных переменных $v[1:m]$, обратной матрицы $a^-[N', M]$ и текущего значения целевой функции.

Все эти данные удобно хранить в виде одной матрицы $as[0:m, 0:m]$, где

$$\begin{aligned} as[0, 0] &= c[N'] \times x[N'], \\ as[1:m, 0] &= x[T(1:m)], \\ as[0, 1:m] &= v[1:m], \\ as[1:m, 1:m] &= a^-[T(1:m), 1:m]. \end{aligned}$$

Мы приведем здесь алгоритм, гарантированный от зацикливания. Для того чтобы избавиться от фиксированной системы задания исходных данных, будем считать, что определена процедура *spos* (j, p), которая по номеру j формирует $p[0] := -c[j]$ и $p[1:m] := a[1:m, j]$.

Например, если заданы массивы $a[1:m, 1:n]$ и $c[1:n]$ (как это было раньше), такой процедурой может быть

```
procedure spos (j, p); value j; integer j; array p;
begin integer i; p[0]:=-c[j];
   for i:=1 step 1 until m do p[i]:=a[i, j]
end spos
```

В дальнейшем мы рассмотрим и другие варианты этой процедуры. Итак, выпишем метод обратной матрицы целиком:

```
procedure LPsol2 (m, n, gz, eps, spos, b); value m, n;
   integer m, n; real eps, gz; procedure spos; array b;
begin integer iter, j0, j1; array as[0:m, 0:m], p, q[0:m];
   integer array base[1:m]; real lev; boolean opt;
 procedure start2;
 begin integer i, j; as[0, 0]:=0; j1:=iter:=0;
   for i:=1 step 1 until m do
     begin as[i, 0]:=b[i]; as[0, 0]:=as[0, 0]+gz×b[i];
       for j:=1 step 1 until m do as[i, j]:=0;
       as[i, i]:=1; as[0, i]:=-gz; base[i]:=n+i
 end i end start;
 procedure new3 (opt); boolean opt;
 begin integer i, j; real max, a1;
   j0:=j1; max:=0;
   for j:=j1+1 step 1 until n, 1 step 1 until j1-1 do
     begin spos (j, p); a1:=p[0];
```

```
        for i:=1 step 1 until m do a1:=a1-p[i]×as[0, i];
        if a1>max then
         begin max:=a1; j0:=j;
             if max>lev then go to mk end;
      end j;
      lev:=max/2; if lev<eps then lev:=eps;
mk: opt:=max<eps; j1:=j0
 end new;
 procedure fin3;
 begin integer i, j; real a1;
   for i:=1 step 1 until m do p[i]:=0;
   for j:=1 step 1 until m do begin
     if base[j]≤n then begin spos(base[j], q);
      for i:=1 step 1 until m do
        p[i]:=p[i]+q[i]×as[j, 0] end end;
   for i:=1 step 1 until m do
      print(base[i], as[i, 0], p[i], b[i], as[0, i])
end fin;
procedure work2;
begin integer i, j, i0; real a1; boolean un;
  iter:=iter+1; un:=true; spos(j0, p);
   for i:=0 step 1 until m do
     begin q[i]:=0;
       for j:=1 step 1 until m do
                                  q[i]:=q[i]+as[i, j]×p[j]
     end; q[0]:=q[0]-p[0];
    for i:=1 step 1 until m do if q[i]>eps then
     begin if un then begin i0:=i; un:=false end
       else for j:=0 step 1 until m do
            if as[i, j]/q[i]<as[i0, j]/q[i0] then
              begin i0:=i; go to mki end
            else if as[i, j]/q[i]>as[i0, j]/q[i0]
                                          then go to mki;
mki: end i;
   if un then go to unbo;
   base[i0]:=j0; a1:=1/q[i0];
   for j:=0 step 1 until m do as[i0, j]:=as[i0, j]×a1;
   for i:=0 step 1 until i0-1, i0+1 step 1 until m do
    begin a1:=q[i];
      for j:=0 step 1 until m do
        as[i, j]:=as[i, j]-a1×as[i0, j]
    end
```

```
end work;
   lev:=gz; j1:=1;
   start2;
mk1: new3(opt); if opt then go to mk2;
   work2; go to mk1;
mk2: fin3; go to mk3;
unbo: print(j0, as[0, j0]); fin3;
mk3: end LPsol
```

Для дополнительных и искусственных переменных не нужно при использовании процедуры *spos* расширять матрицу a, соответствующие столбцы могут легко вырабатываться алгоритмически. Вот, например, как выглядит эта процедура для случая, когда все m ограничений являются неравенствами и вводится m дополнительных переменных:

```
procedure spos2(j, p); value j; integer j; array p;
begin integer i;
   if j≤n then spos(j, p) else
    begin for i:=0 step 1 until m do p[i]:=0;
      p[j-n]:=1 end end spos
```

Посмотрим теперь, как будет выглядеть решение примера с помощью процедуры *LP sol2*.

Постоянная часть записи изображена на рис. 14, изменяющаяся часть — на рис. 15.

Мы внесли в запись данные, получающиеся после действий процедуры *start3*. После действия процедуры *new3*, которая полностью аналогична *new2*, с той лишь разницей, что столбцы матрицы a последовательно выписываются в массив p и оценки, которые в *LP sol1* были записаны в $as[0, N]$, теперь вычисляются умножением $as[0, M]\times p[M]$, мы получаем $j0=2$ и $opt\equiv$ **false**. В процедуре *work2* снова восстанавливается в массиве p столбец $a[M, j0]$, который затем умножается на матрицу $as[1:m, 1:m]$ с записью результата в $q[0:m]$. Получаем

p	2	2	0	q	2002	2	0

После перехода к метке *mk1* получаем состояние, изображенное на рис. 16, и т. д.

Читателю рекомендуется довести до конца вычисления и сравнить получающиеся таблицы с записями в решении этого примера с процедурой *LP sol*1.

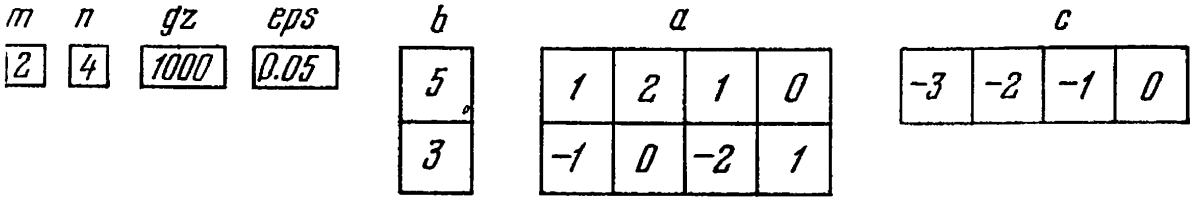

Рис. 14. Исходные данные для метода обратной матрицы.

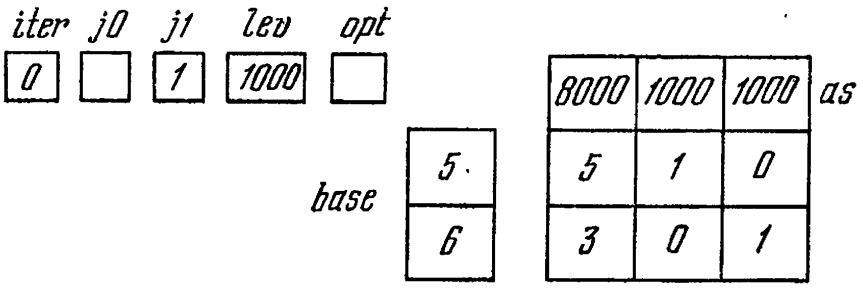

Рис. 15. Заполнение рабочей памяти в методе обратной матрицы после процедуры *start*.

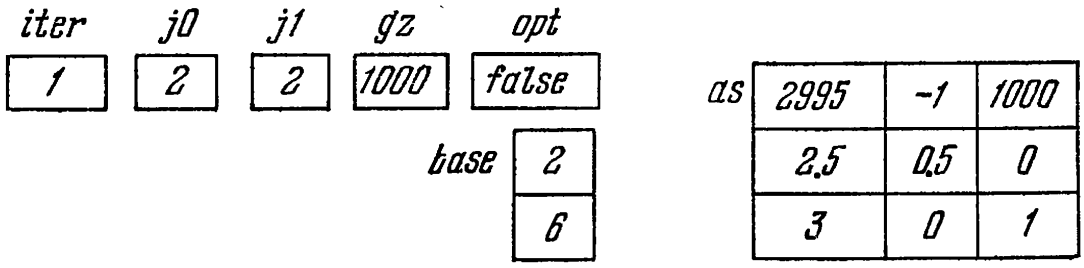

Рис. 16. Рабочая информация метода обратной матрицы.

Достоинства модифицированного симплекс-метода становятся яснее в тех случаях, когда удается более компактно представить исходные данные задачи. Одним из наиболее характерных способов компактной записи является запись ненулевых элементов матрицы, которых в реальных задачах часто оказывается гораздо меньше, чем нулей. Такую запись мы уже встречали в § 3 гл. 1.

При таком задании данных процедура *spos* может быть написана следующим образом:

```
procedure spos3(j, p); value j; integer j; array p;
begin integer i;
   for i:=0 step 1 until m do p[i]:=0;
   for i:=lst[j-1]+1 step 1 until lst[j] do
     p[ind[i]]:=ad[i];
   p[0]:=-p[0]
end spos
```

в) Мультипликативный метод. Правильное название этого метода — метод последовательных улучшений с мультипликативным представлением обратной матрицы. В нем матрица $a^-[N'_k, M]$ на k-й итерации представляется в виде произведения матриц $d_i[N'_i, N'_{i-1}]$ (здесь N'_i — базис i-й итерации)

$$a^-[N'_k, M] = d_k[N'_k, N'_{k-1}] \times \ldots \times d_1[N'_1, M]. \quad (50)$$

Для того чтобы вычислить произведение $a^-[N'_k, M] \times b[M]$, нужно последовательно умножить $b[M]$ на d_1, d_2 и т. д. Матрица d_i задается номером столбца r_i, отличного от единичного, и самим этим столбцом $dm[1:m]$. Для того чтобы умножить вектор $x[1:m]$ на d_i, нужно просто выполнить оператор

$$x[1:m] := x[1:m] + x[r_i] \times dm[1:m]. \quad (51)$$

Аналогично, для того чтобы вычислить произведение $c[N'_k] \times a^-[N'_k, M]$, нужно последовательно умножить $c[N'_k]$ на d_k, d_{k-1} и т. д. А для того чтобы умножить $x[1:m]$ на d_i, нужно изменить лишь компоненту $x[r_i]$

$$x[r_i] := x[1:m] \times dm[1:m]. \quad (52)$$

Для изменения информации об обратной матрице тоже не требуется тяжелых операций — достаточно к списку «мультипликаторов» d_i добавить в конце $(k+1)$-й.

Итак, в простейшем варианте мультипликативного метода запись состоит из задания базиса (*base*), вектора значений базисных переменных $x[N']$, вектора значений

двойственных переменных $v[1:m]$, текущего значения линейной формы γ, числа итераций *iter* и, наконец, информации об обратной матрице — числа мультипликаторов $nm1$ и списка из $nm1$ мультипликаторов. Каждый элемент этого списка состоит из номера столбца r_i и из информации о столбце. Обычно в мультипликаторах оказывается сравнительно немного ненулевых элементов, поэтому столбцы мультипликаторов хранятся в сжатой форме. Представление мультипликаторов, потребность в дополнительной информации, возможность хранения пар из индекса и элемента столбца — все это существенно зависит от вычислительной машины, выбранного алгоритмического языка и от транслятора, поэтому в детали использования внешней памяти мы здесь входить не будем. Для полноты описания в приводимой ниже процедуре мультипликаторы будут храниться в двух массивах в оперативной памяти

array *va*[1 : *n max*]; **integer array** *nom*[1 : *n max*],

где *n max* — задаваемый при обращении к процедуре размер.

Описанный простейший вариант алгоритма недостаточно эффективен при большом числе интераций, а при хранении информации в оперативной помяти даже и нереализуем, так как число записей в массивы *va* и *nom* (равное $nm-1$) может превысить *n max*. Чтобы уменьшить информацию и упростить вычисления, используют так называемое «повторение» — в процедуре *work* после нахождения переменной, удаляемой из базиса, выполняется оператор

if $nm < nmax - m$ **then** *inclmult* (*nm*, *i*0, *q*)
else *repetition*

Повторение (*repetition*) заключается в том, что базис текущей итерации получается заново из первоначального базиса по кратчайшему пути (т. е. не больше чем за m последовательных замен). При этом не производится промежуточный контроль неотрицательности $x[N']$, а очередной вектор, вводимый в базис, умножается на обратную матрицу, из него делается мультипликатор, определяется переменная, выводимая из базиса, и мультипликатор записывается в список:

```
procedure repetition;
begin integer i, k, bk; real max; integer array bs [1 : m];
    nm1 := 0; nm := 1; max = 0;
    for k := 1 step 1 until m do
      begin bs [k] := base [k]; base [k] := 0 end;
    for k := 1 step 1 until m do
      begin spos (bs [k], p); lhmult (p, nm - 1);
        for i := k step 1 until m do
          if base [i] = 0 ∧ abs (p [i]) > max then
                          begin max := abs (p [i]); bk := i end;
        base [bk] := bs [k]; inclmult (nm, i, p); v [k] := p [0];
    end k;
    for i := 1 step 1 until m do x [i] := b [i]; lhmult (x, nm - 1)
end rep
```

Мультипликативный метод наиболее распространен в эффективных программах по линейному программированию. Имеются различные варианты процедуры повторения, обеспечивающие более компактное и экономное задание информации об обратной матрице после повторения. Литературу по этому вопросу читатель может найти в библиографических указаниях.

Приведем здесь простейший вариант мультипликативного метода, в котором двойственные переменные и значение базисных переменных находятся последовательным пересчетом и используется простейшая процедура повтония. Кроме процедуры *spos*, исходные данные использует процедура-функция *scpr* (*j*, *p*), вырабатывающая скалярное произведение $a[0:m, j]$ на вектор $p[0:m]$, где $p[0]=1$.

```
procedure LP sol3 (m, n, gz, eps1, eps, nmax, b, spos, scpr);
    value n, m, nmax; real gz, eps1, eps; procedure spos;
    integer n, m, nmax; array b; real procedure scpr;
begin integer inter, j0, j1, nm, nm1; real lev, cost, max;
    integer array base [1 : m], nom [1 : nmax];
    array p, x, v [0 : m], va [1 : nmax];
    procedure start3;
    begin integer i; inter := 0; nm := 1; cost := 0;
      for i := 1 step 1 until m do
        begin x [i] := b [i]; v [i] := gz;
          cost := cost + b [i] × gz; baze [i] := n + i end;
    end start;
```

```
procedure inclmult (k, r, p); value r; integer k, r; array p;
begin integer i; real x;
      va [k]: = r;  nml: = nml + 1:
      x: = p [r];  p [r]: = — 1;  r: = k + 1;
      for i: = 1 step 1 until m do if abs (p [i]) > eps then
        begin nom [r]: = i;  va [r]: = — p [i]/x;  r: = r + 1 end;
      nom [k]: = — r;  k: = r end;
procedure lhmult (p, k); integer k; array p;
begin integer i, j, s, s1; real x; s1: = 1;
   for i: = 1 step 1 until k do if nom [i] < 0 then
            begin x: = p [va [i]];  p [va [i]]: = 0 end
            else p [nom [i]]: = p [nom [i]] + x × va [i]
end;
procedure rhmult (p, k); integer k; array p;
begin integer i, j, s; real x;  s: = — k — 1;  x: = 0;
  for i: = k step —1 until 1 do
    if s = nom [i] then
     begin p [va [i]]: = x;  x: = 0;  s: = — i end
    else x: = x + p [nom [i]] × va [i]
 end;
boolean procedure new4;
 begin integer i, j; real a1; j0: = j1;  max: = 0;
  for j: = 1 step 1 until m do p [j]: = v [j];
  rhmult (p, nm — 1);
  for j: = j1 + 1 step 1 until n, 1 step 1 until j1 — 1 do
    begin a1: = — scpr (j, p);
     if a1 > max then begin max: = a1;  j0: = j;
        if max ≥ lev then go to mk end
    end j;
  lev: = max/2;  if lev < eps1 then lev: = eps:
mk: new3: = max ≥ eps1;  j1: = i0
 end new;
 procedure work3;
 begin integer i, i0; real a1, a2; boolean un;
  spos (j0, p);  un: = true;  lhmult (p, nm — 1);
  for i: = 1 step 1 until m do if p [i] > esp then
   begin if un then begin i0: = i;  un: = false end
         else if x [i]/p [i] < x [i0]/p [i0] then i0: = i;
mki: end i;
  v [i0]: = p [0];
  if un then go to unbo;
  base [i0]: = j0;  a1: = x [i0]/p [i0];  a2: = 0;
```

```
  for i:=1 step 1 until m do x[i]:=x[i]-a1*p[i];
  x[i0]:=a1;
  if nm<nmax then
    begin inclmult(nm, i0, p) end else repetition
end work;
procedure fin3;
begin integer i, j; real a1; array q, p[0:m];
  for i:=1 step 1 until m do q[i]:=0;
  a1:=0;
  for i:=1 step 1 until m do begin
    if base[i]>n then p[0]:=gz
    else spos (base [i], p);
    a1:=a1-p[0]×x[i];
    for j:=1 step 1 until m do
                          q[j]:=q[j]+x[i]×p[j] end i;
  for i:=1 step 1 until m do
    print('?', i, base[i], x[i], q[i], b[i], v[i]);
  print(iter, a1)
end fin;
      lev:=gz; j1:=1; nm1:=0;
      start3;
      for iter:=iter+1 while new4 do work3;
      comment вместо меток цикл while!;
      go to mk3;
  unbo: print('unbounded solution');
  mk3: fin3 end LP sol
```

В настоящее время начали распространяться другие методы представления информации о базисной матрице, которые в чистом виде или в комбинации с другими методами дают весьма эффективные разновидности метода последовательных улучшений. Так, большое распространение получает так называемое *LU-представление*. Известно, что любая неособенная матрица $a[1:m, 1:m]$ может быть представлена в виде произведения

$$a[1:m, 1:m]=L[1:m, 1:m]\times U[1:m, 1:m], \quad (53)$$

где L — нижнетреугольная, а U — верхнетреугольная матрица. Вместо того чтобы решать систему

$$a[1:m, 1:m]\times x[1:m]=b[1:m], \quad (54)$$

можно последовательно решить систему

$$L[1:m, 1:m]\times z[1:m]=b[1:m] \quad (55)$$

и систему

$$U[1:m,\ 1:m]\times x[1:m]=z[1:m]. \qquad (56)$$

Решение систем с треугольной матрицей труда не представляет — такие системы решаются последовательным вычислением и подстановкой переменных. При изменениях базисной матрицы может использоваться алгоритм поддержания LU-представления, в детали которого мы здесь входить не будем, или может использоваться сочетание LU-представления с мультипликативным накоплением поправок.

Отметим, что при LU-представлении обе треугольные матрицы оказываются редкозаполненными и для их хранения используется техника компактной записи матриц, уже встречавшаяся нам выше.

Предлагалось также в качестве одного из способов так называемое LP-представление базисной матрицы

$$a[1:m,\ 1:m]=L[1:m,\ 1:m]\times P[1:m,\ 1:m], \qquad (57)$$

где L — треугольная матрица, а P — ортонормированная. В библиографических указаниях приведены соответствующие ссылки на подробные изложения этого подхода.

Значительного повышения эффективности вычислительных процедур можно добиться, ограничиваясь матрицами какой-либо определенной структуры. Так, например, предположим, что в задаче (32) — (34) заданы разбиения множеств индексов

$$M=M_0+M_1+\ldots+M_k, \qquad N=N_0+N_1+\ldots+N_k \qquad (58)$$

и при этом

$$a[M_j,\ N_j]=0[M_i,\ N_j] \quad \text{для} \quad j\neq 0,\ j\neq i, \qquad (59)$$

т. е. матрицу ограничений можно схематически представить рис. 17. Такая задача линейного программирования обычно называется *блочной*.

Блочная задача линейного программирования.

Минимизировать $c[N]\times x[N]$ при условиях

$$x[N]\geqslant 0[N],$$

$$a[M_0,\ N]\times x[N]=b[M_0],$$

$$a[M_i,\ N_i]\times x[N_i]=b[M_i], \qquad i\in 1:k.$$

Легко видеть, что в этой задаче базисную матрицу можно представить в виде рис. 18, где

$$|N'_i| = |M_i|, \quad N'_i \subset N_i, \quad i \in 1:k. \tag{60}$$

Для решения систем с такой матрицей достаточно иметь обратные матрицы «блоков» $a_i^-[N'_i, M_i]$ и некоторую матрицу $a_0^-[\bar{N}', M_0]$.

Особенно просто выглядит решение такой задачи, когда каждый из блоков состоит из одного ограничения («узкоблочная задача»).

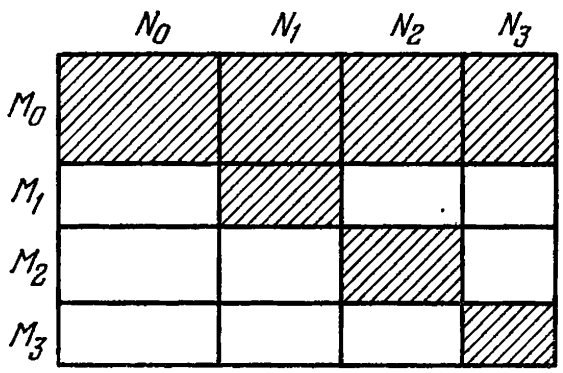

Рис. 17. Матрица ограничений блочной задачи.

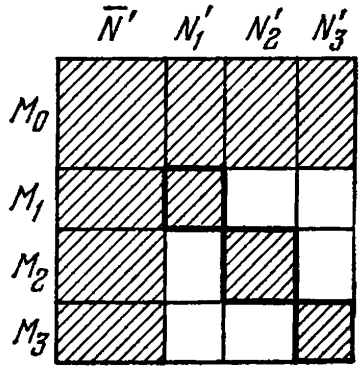

Рис. 18. Базисная матрица блочной задачи.

Другие специализированные задачи линейного программирования мы рассмотрим в следующей главе.

г) Генерирование столбцов в методе последовательных улучшений. Как мы уже видели, в методе обратной матрицы и в его более мощных вычислительных вариантах не имеет значения, как задана матрица ограничений задачи. Важно лишь, чтобы по номеру столбца можно было этот столбец восстановить.

Оказывается, что от задания матрицы ограничений можно потребовать еще меньшего. Действительно, если посмотреть внимательно, где в методе последовательного улучшения используется матрица ограничений, то легко увидеть, что она появляется только при проверке допустимости двойственных переменных $v[M]$, полученных из уравнений (38),

$$v[M] \times a[M, N] \leqslant c[N] \tag{61}$$

или

$$c[j] - v[M] \times a[M, j] \geqslant 0, \quad j \in N \tag{62}$$

можно, очевидно, заменить решением экстремальной задачи.

Вспомогательная экстремальная задача.

Найти максимум $v[M]\times a[M, j]-c[j]$ по $j\in N$.

Если этот максимум оказывается больше 0, то $j0$, на котором достигается этот максимум, и должно вводиться в базис (причем тут важен не сам индекс, а соответствующий ему столбец $a[M, j]$). Если же максимум равен 0, то имеющееся базисное решение оптимально.

Таким образом, если мы умеем решать вспомогательную экстремальную задачу, не используя матрицу $a[M, N]$, то больше эта матрица нам нигде не понадобится.

Рассмотрим, например, следующую задачу.

Задача линейного ассортиментного раскроя.

Задано положительное вещественное число l_0 и положительный вектор $[1:m]$. Пусть N — множество целочисленных векторов $r[1:m]$, удовлетворяющих условиям

$$l[1:m]\times r[1:m]\leqslant l0,\quad r[1:m]\geqslant \mathbf{0}[1:m]. \qquad (63)$$

Требуется найти вектор $x[N]\geqslant \mathbf{0}[N]$, удовлетворяющий условию

$$\Sigma_{r\in N}\, r[1:m]\times x[r]=b[1:m]$$

и минимизирующий

$$\Sigma_{r\in N}\, x[r].$$

Вспомогательная экстремальная задача, служащая в данном случае для проверки оптимальности текущего базисного решения, выглядит в данном случае следующим образом.

Задача о наивыгоднейшем раскрое.

Найти целочисленный неотрицательный вектор $r[1:m]$, удовлетворяющий условию (63) и максимизирующий

$$v[1:m]\times r[1:m]-1.$$

Методы решения этой задачи будут рассматриваться в пятой и шестой главах.

В качестве вспомогательной экстремальной задачи может фигурировать снова задача линейного программирования. Например, если рассматривается задача блочной структуры (59), то для нее можно предложить вариант алгоритма, при котором размеры задачи будут значительно

меньше. Действительно, предположим для простоты, что каждое из множеств решений систем

$$a[M_i, N_i]\times x[N_i]=b[M_i], \quad x[N_i]\geqslant \mathbf{0}[N_i], \quad i\in 1:k, \tag{64}$$

ограничено. По теореме 7 каждое решение (64) может быть представлено в виде

$$x[N_i]=d[N_i, R_i]\times\lambda[R_i], \tag{65}$$

где $\lambda[R_i]$ — вероятностный вектор, а $d[N_i, r]$ — крайние точки множества допустимых решений. Условия нашей задачи линейного программирования можно заменить эквивалентной задачей.

Преобразованная блочная задача.

Минимизировать

$$c[N_0]\times x[N_0]+\Sigma_{i\in 1:k}(c[N_i]\times d[N_i, R_i])\times\lambda[R_i] \tag{66}$$

при условиях

$$\lambda[R_i]\geqslant \mathbf{0}[R_i], \quad i\in 1:k, \tag{67}$$

$$\mathbf{1}[R_i]\times\lambda[R_i]=1, \quad i\in 1:k, \tag{68}$$

$$a[M_0, N_0]\times x[N_0]+ +\Sigma_{i\in 1:k}(a[M_0, N_i]\times d[N_i, R_i])\times\lambda[R_i]=b[M_0]. \tag{69}$$

Двойственной к ней является задача:

Найти векторы $v[M_0]$ и $v1[1:k]$, удовлетворяющие условиям

$$v[M_0]\times a[M_0, N_0]\leqslant c[N_0], \tag{70}$$

$$v[M_0]\times(a[M_0, N_i]\times d[N_i, r])+v1[i]\leqslant \leqslant(c[N_i]\times d[N_i, r]), \quad r\in R_i, \quad i\in 1:k \tag{71}$$

и максимизирующие

$$v[M_0]\times b[M_0]\times v1[1:m]\times\mathbf{1}[1:m]. \tag{72}$$

Для проверки оптимальности базисного решения преобразованной блочной задачи, кроме проверки условий (70), которая проводится обычным образом, нужно решать k экстремальных задач вида

Задача i-го блока.

Найти $r\in R_i$, на котором достигается минимум

$$(c[N_i]-v[M_0]\times a[M_0, N_i])\times d[N_i, r]+v1[i]. \tag{73}$$

Так как метод последовательных улучшений дает наилучшую из крайних точек, то для решения этой задачи может использоваться решение этим методом следующей задачи линейного программирования.

Локальная линейная задача.

Найти вектор $x[N_i]$, удовлетворяющий условию (64) и минимизирующий

$$(c[N_i] - v[M_0] \times a[M_0, N_i]) \times x[N_i] + v1[i]. \quad (74)$$

В этом случае, если в какой-либо из этих задач минимум оказывается положительным, то из соответствующего решения $x^*[N_i]$ формируется вектор $g[M_0 \cup 1:k]$, где

$$g[1:k] = E[1:k, i] \text{ и } g[M_0] = a[M_0, N_i] \times x^*[N_i],$$

который и вводится в базис.

Такой метод решения блочной задачи называется методом разложения Данцига — Вулфа.

Метод генерирования столбцов имеет много чрезвычайно разнообразных применений. Именно благодаря идее генерирования в настоящее время практически размеры задач линейного программирования ограничиваются лишь числом ограничений задачи, а число переменных большой роли не играет.

ГЛАВА 3
ТРАНСПОРТНАЯ ЗАДАЧА

§ 1. Основные определения теории графов

В этой главе мы приступаем к важнейшей проблематике книги — к теории оптимальных потоков в сетях. Для того чтобы определить сами сети и потоки, воспользуемся широко распространившейся сейчас терминологией теории графов, которая в основном будет введена (с необходимыми фактами теоретического и вычислительного характера) в первых трех параграфах.

Только после этого мы приступим к описанию потоков на графах, изучению их свойств, постановке и решению экстремальных задач для потоков.

Транспортные задачи могут рассматриваться как задачи линейного программирования со специальными матрицами ограничений (как раз так мы их и рассматриваем). Построенные для них специальные методы допускают обобщение на несколько более широкие задачи. В конце главы будет рассмотрена одна из таких задач.

Но ближайшая наша цель — определения.

Графом мы будем называть тройку $\langle M, N, T\rangle$, где M и N конечные множества, а T — отображение $T: N \to M \times M$. Элементы M называются *вершинами* графа, элементы N — *дугами* графа.

Отображение T сопоставляет каждой дуге $u \in N$ упорядоченную пару вершин $(i(u), j(u))$. Первая называется *началом* дуги u, а вторая — *концом* дуги u. Иногда для обеих вершин вместе используется термин «граничные вершины [1]».

---

[1]) Используется также название «конечный ориентированный мультиграф» — первое слово добавляется, чтобы подчеркнуть конечность множеств M и N, второе, чтобы указать на различие, которое делается между граничными вершинами графа, «мульти-» отражает возможность нескольких дуг с одинаковыми началом и концом. В последнее время распространился также неудачный с нашей точки зрения термин «орограф».

Граф может быть изображен рисунком (от этого и произошло его название), на котором вершинам соответствуют точки, а дугам — линии со стрелками, идущие от начал к концам.

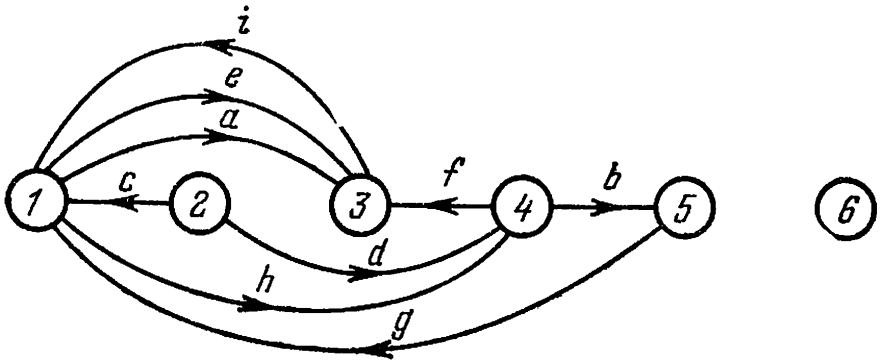

Рис. 19.

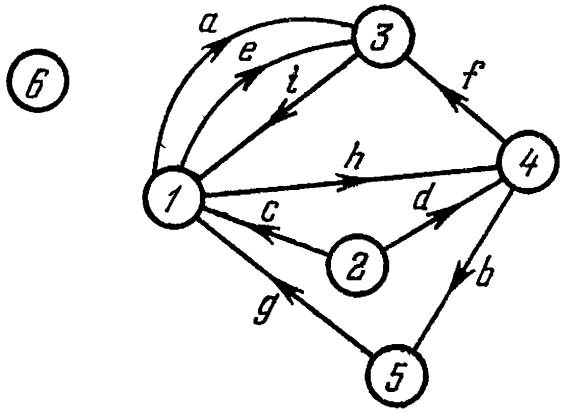

Рис. 20.

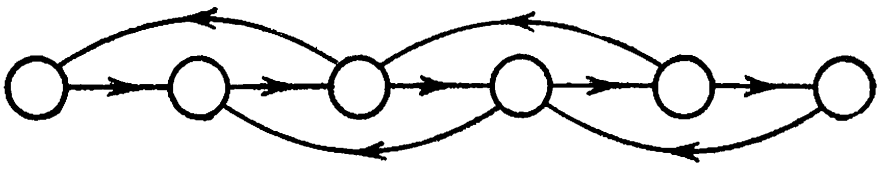

Рис. 21.

Примеры. 1. Пусть

$$M = 1:6,\quad N = \{a, b, c, d, e, f, g, h, i\},$$
$$Ta = (1, 3),\quad Tb = (4, 5),\quad Tc = (2, 1),$$
$$Td = (2, 4),\quad Te = (1, 3),\quad Tf = (4, 3),$$
$$Tg = (5, 1),\quad Th = (1, 4),\quad Ti = (3, 1).$$

Этот граф может быть изображен рис. 19 или эквивалентным ему рис. 20.

2. $M = 1 : m$, $N = 1 : 2 \times m - 3$, $Tu = (u, u + 1)$ для $u \in 1 : m - 1$ и $Tu = (u - m + 3, u - m + 1)$ для $u \in m : 2 \times m - 3$. При $m = 6$ этот граф изображается на рис. 21.

3. $M = K \cup L$ $(K \cap L = \emptyset)$, $N = K \times L$, $T(i, j) = (i, j)$. Вершины этого графа разбиты на два непересекающихся подмножества, и дуги идут из любой вершины первого подмножества K в любую вершину второго подмножества L. Нарисовать этот граф предоставляется читателю.

Часто приходится рассматривать графы, в которых в сравнении с исходным уменьшены множества дуг и вершин.

Граф $\langle M, N', T\rangle$, где N' — подмножество N, называется *частичным графом* графа $\langle M, N, T\rangle$. Таким образом, частичный граф — это граф, получающийся из исходного после удаления некоторых дуг.

Граф $\langle M', N(M'), T\rangle$, где M' — подмножество M, а $N(M') = \{u \in N, i(u), j(u) \in M'\}$, называется *подграфом* графа $\langle M, N, T\rangle$ Подграф получается из исходного графа удалением некоторых вершин и всех дуг, для которых удаленные вершины являются граничными.

Частичный подграф определяется как частичный граф некоторого подграфа.

Если при использовании этих определений нам потребуется подчеркнуть, что имеются в виду собственные подмножества, мы будем к соответствующему определению добавлять слово «собственный».

Множество дуг, выходящих из вершины i, мы будем обозначать через N_i^-; множество дуг, входящих в i, — через N_i^+. Очевидно, что каждый из наборов $\{N_i^-\}_{i \in M}$ и $\{N_i^+\}_{i \in M}$ образует разбиение N.

Если i является граничной вершиной u, говорят, что i и u инцидентны (incident — свойственный, присущий).

§ 2. Способы задания графа

Очень редко граф можно задать правилом формирования дуг и граничных вершин данной дуги, как это делалось в примере 2.

Основным способом задания графа следует считать списковый, в котором составляется список дуг, и запись о каж-

дой дуге состоит из названия дуги, названия ее начала и названия ее конца.

Так, в примере 1 мы задавали граф именно таким списком: (a,1, 3) (b,4, 5) и т. д. Остальные методы являются лишь более удобными с тех или иных точек зрения вариантами этого основного метода.

Во многих случаях для теоретических рассмотрений граф удобно задавать матрицей. Матрица $a\,[M, N]$, в которой строки соответствуют вершинам графа, а столбцы дугам, называется *матрицей инциденций* [1]). Элементы матрицы инциденций задаются правилом: $a\,[i\,(u), u] = -1$, $a\,[j\,(u), u] = 1$, остальные элементы равны 0.

Для примера 1 получаем табл. 1.

Таблица 1

	a	b	c	d	e	f	g	h	i
1	—1	0	1	0	—1	0	1	—1	1
2	0	0	—1	—1	0	0	0	0	0
3	1	0	0	0	1	1	0	0	—1
4	0	—1	0	1	0	—1	0	1	0
5	0	1	0	0	0	0	—1	0	0
6	0	0	0	0	0	0	0	0	0

Легко видеть, насколько неэкономно использовать для задания графа матрицу инциденций. Однако она полезна при рассмотрении систем уравнений (и неравенств), определяемых графом. Отметим, что ранг этой матрицы всегда меньше $|M|$ — числа элементов M. (Одна линейная зависимость между строками матрицы очевидна — сумма строк равна 0.) От чего зависит ранг матрицы инциденций и чему он равен, мы установим ниже.

Имеется другой способ матричного задания графа, используемый в тех случаях, когда не требуется различать дуги с одними и теми же началами и концами, а достаточно только знать их общее количество. Реализующая этот способ *матрица смежностей* $r\,[M, M]$ — это неотрицательная целочисленная матрица, в которой

$$r\,[i, j] = |\,N_i^- \cap N_j^+\,|.$$

[1]) При рассмотрении матрицы инциденций предполагается, что граф не имеет *петель*, т. е. дуг, у которых начало совпадает с концом.

В примере 1 матрица $r[1:6, 1:6]$ такова (табл. 2).

Таблица 2

	1	2	3	4	5	6
1	0	0	2	1	0	0
2	1	0	0	1	0	0
3	1	0	0	0	0	0
4	0	0	1	0	1	0
5	1	0	0	0	0	0
6	0	0	0	0	0	0

Матрица смежностей используется сравнительно редко даже в теоретических рассмотрениях.

Чаще всего используется то или иное списковое задание графа. Упоминавшееся уже задание графа тройками фактически является разновидностью сжатого задания матрицы инциденций. Эта форма представления графа удобна тем, что позволяет быстро изменять информацию о графе, добавлять в информации о дугах нужное число числовых характеристик, организовывать вычислительный процесс, используя последовательный просмотр дуг (что во многих случаях оказывается очень удобным). Однако это представление имеет два недостатка, которые в некоторых задачах оказываются существенными: во-первых, оно не содержит всей информации о графе, например, потому, что в нем не фигурируют изолированные вершины, т. е. вершины, которым не инцидентна ни одна дуга, во-вторых, информация может быть задана проще, если рассматриваемый в задаче граф не придется часто изменять.

Свободный от этих недостатков способ может рассматриваться как вариант сжатого задания матрицы смежностей. Мы можем задать граф в виде списка вершин, причем в записи о каждой вершине будет присутствовать как информация о самой вершине, так и информация об исходящих из этой вершины дугах. Информация о дугах представляет собой также список, в котором каждой дуге соответствует запись, включающая информацию о конце дуги и необходимые числовые характеристики самой дуги

Реализация этих способов в вычислительной машине существенно зависит от используемых вычислительных возможностей. В случае, если используется достаточно простой алгоритмический язык, типа алгол-60, второй способ задания графа можно реализовать с помощью тех же методов, которыми задаются в машине редко заполненные матрицы. Дуги нумеруются числами от 1 до $n = |N|$, а вершины — числами от 1 до $m = |M|$ таким образом, чтобы дуги, выходящие из i-й вершины, предшествовали в этой нумерации дугам, выходящим из j-й вершины при

$i < j$. Далее вводится целочисленный массив $j\,[1 : n]$, играющий ту же роль, что и массив *ind*. Наконец, с помощью массива $lst\,[0 : m]$ производится разметка массива $j\,[1 : n]$, именно, $lst\,[i]$ указывает место в $j\,[1 : n]$, где кончается информация о N_i^-. Необходимая числовая информация о дугах и вершинах может при этом задаваться с помощью отдельных массивов необходимой размерности.

В примере 1 нумерация вершин уже имеется. Если перенумеровать дуги в следующем порядке: $a, h, i, c, d, e, f, b, g$, то граф этого примера будет задаваться массивами

$$j: 3, 4, 3, 1, 4, 1, 3, 5, 1,$$
$$lst: 0, 3, 5, 6, 8, 9, 9$$

Отметим, что так как $N_6^- = \emptyset$, то $lst\,[6] = lst\,[5]$.

При использовании алгола-68 можно ввести специальные структурные виды, описывающие дуги и вершины, и необходимые связи между дугами и вершинами задавать с помощью ссылок. Например, для первого способа задания графа мы можем ввести виды

mode vert = struct (...)

для вершин и для дуг

mode arc = struct (ref vert *i*, *j*, ...)

где i и j будут ссылками на начало и конец данной дуги. Для второго способа задания можно ввести виды

mode vert = struct (ref arc *list*, ...)

для вершин, а для дуг

mode arc = struct (ref arc *next*, **ref vert** *j*, ...)

причем для каждой вершины ссылка *list* будет указывать на начало цепного списка, перечисляющего N_i^-. Этот список будет организовываться с помощью ссылок *next*.

Многоточие во всех этих описаниях обозначает место, оставляемое для числовых характеристик объекта.

§ 3. Связность

Продолжим рассмотрение основных определений и фактов теории графов. В тех случаях, когда отображение T в задании графа $\langle M, N, T\rangle$ остается одним и тем же, будем для краткости обозначать граф просто через $\langle M, N\rangle$.

Пусть заданы какая-то последовательность вершин $i_1, i_2, \ldots, i_{k+1}$ графа $\langle M, N\rangle$ и последовательность $u_1, u_2, \ldots, u_k$ его дуг.

Будем называть эту пару последовательностей *путем*, если для любого $l \in 1 : k$ выполняются равенства

$$i_l = i(u_l), \quad i_{l+1} = j(u_l). \tag{1}$$

Вершина i_1 называется *началом пути*, вершина i_{k+1} — *концом пути*, число k называется *звенностью* пути. Путь, у которого начало и конец совпадают, называется *замкнутым*.

Рассмотрим замкнутый путь, у которого все вершины различны (здесь начало и конец мы считаем одной вершиной). Пусть M' — множество его вершин, N' — множество его дуг. Мы будем называть частичные подграфы $\langle M', N'\rangle$ такого типа *контурами*. Целесообразно, однако, определять контур по-другому. Частичный подграф $\langle M', N'\rangle$ называется *контуром*, если:

1° $|M'| = |N'|$.

2° Для каждого $i \in M'$ справедливо $|N'^-_i| = |N'^+_i| = 1$, т. е. в каждую вершину одна дуга входит и одна выходит.

3° Никакой собственный подграф $\langle M', N'\rangle$ этими двумя свойствами не обладает.

Предложение 1. *Вершины и дуги контура можно перенумеровать так, чтобы они образовывали замкнутый путь.*

Доказательство. Выберем произвольно вершину из M', назовем ее i_1. В N' (по свойству 2°) имеется единственная дуга, выходящая из i_1, назовем ее u_1, а ее конец i_2. Либо i_2 совпадает с началом пути, либо имеется дуга, выходящая из i_2. Назовем ее u_2, а ее конец i_3 и т. д. Так как на каждом шаге этого процесса появляется новая вершина (конец ранее не встречавшейся дуги), на некотором l-м шаге конец пути совпадает с началом. При этом множества встретившихся вершин и дуг будут, очевидно, обладать свойствами 1° и 2° из определения контура, а значит, в силу 3° совпадают с M' и N'. ▲

В духе определения контура можно ввести еще несколько полезных для дальнейшего понятий.

Частичный подграф $\langle M', N'\rangle$ называется *циклом*, если:

1° $|M'| = |N'|$.

2° Для каждого $i \in M'$ справедливо $|N'^{-}_i| + |N'^{+}_i| = 2$, т. е. каждой вершине инцидентно ровно две дуги.

3° Никакой собственный подграф $\langle M', N' \rangle$ этими двумя свойствами не обладает.

Контур — это цикл, в котором можно обойти все дуги, двигаясь по каждой от ее начала к ее концу. В произвольном цикле также можно обойти все дуги, но по некоторым дугам придется идти от конца к началу. Дуги первого типа мы будем называть *положительно ориентированными*, дуги второго типа — *отрицательно ориентированными.*

Предложение 2. *Вершины и дуги цикла можно перенумеровать так, чтобы для любой дуги u_l выполнялось*

$$(i_l = i(u_l) \wedge i_{l+1} = j(u_l)) \vee (i_l = j(u_l) \wedge i_{l+1} = i(u_l)) \quad (2)$$

(*при выполнении первой альтернативы дуга положительно ориентирована, а при выполнении второй — отрицательно ориентирована*).

Доказательство является простым вариантом предыдущего.

Частичный граф $\langle M', N' \rangle$ называется *цепью*, если:

1° $|M'| = |N'| + 1$.

2° Существуют такие $\bar{i}, \bar{j} \in M'$, что

$$|N'^{-}_i| + |N'^{+}_i| = \begin{cases} 1 \text{ при } i = \bar{i} \text{ и } i = \bar{j}, \\ 2 \text{ для остальных } i \in M'. \end{cases}$$

3° Никакой собственный подграф $\langle M', N' \rangle$ не является циклом.

Такую цепь называют также цепью, соединяющей $\bar{i}$ и $\bar{j}$.

Предложение 3. *Вершины и дуги цепи можно перенумеровать так, чтобы они обладали свойством* (2), *и при этом $\bar{i}$ было первой вершиной, а $\bar{j}$ — последней.*

С точки зрения этих определений цикл — это замкнутая цепь. На рис. 22 приводятся примеры введенных объектов.

Граф $\langle M, N \rangle$ называется *связным*, если любые две его вершины можно соединить цепью.

Естественно, что граф, сам являющийся цепью, связен. Однако, как это видно на примере графа, являющегося циклом, иногда можно обеспечить связность графа за счет меньшего числа дуг. В цепи без нарушения связности нельзя удалить ни одной дуги, и в этом смысле цепь минимальна.

Представляет интерес нахождение такого рода минимальных связных частичных графов в произвольных графах.

Теорема 1. *Пусть* $\langle M, N\rangle$ *— связный граф. Найдется такое множество дуг* $N' \subset N$, *что* $|N'| = |M| - 1$

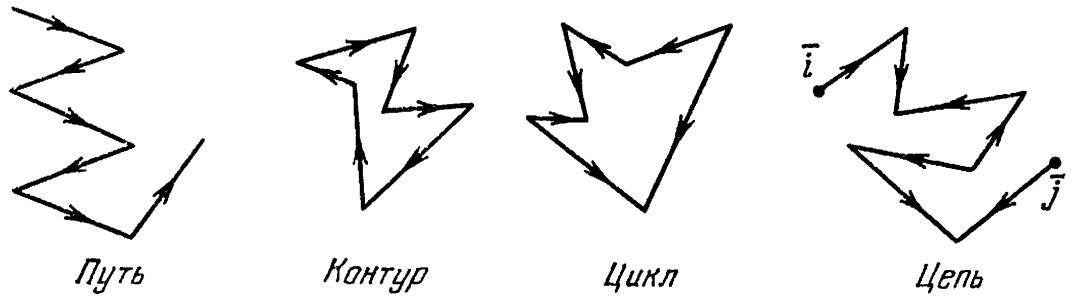

Рис. 22.

и граф $\langle M, N'\rangle$ *связен. При этом каждой вершине* $i \in M$ *и каждой дуге* $u \in N'$ *можно присвоить номера* $m(i)$ *и* $n(u)$ *таким образом, что для любой дуги* u

$$n(u) = \max\{m(i(u)),\ m(j(u))\} \tag{3}$$

и различным вершинам (дугам) будут соответствовать различные номера.

Доказательство. Мы будем строить последовательность связных частичных подграфов $\langle M', N'\rangle$, в которых $|M'| = |N'| + 1$. В каждом таком графе все вершины и дуги будут перенумерованы в соответствии с требованиями теоремы.

В качестве исходного возьмем граф $\langle\{i_0\}, \emptyset\rangle$, где i_0 — любая вершина. Присвоим ей номер 0.

Пусть теперь мы имеем некий связный собственный частичный подграф $\langle M', N'\rangle$ и выполняются все требуемые условия. Пусть $j \in M \setminus M'$. Ввиду связности $\langle M, N\rangle$ существует цепь, соединяющая i_0 и j. Перенумеруем ее дуги и вершины, начиная от i_0:

$$i_0,\ i_1,\ \ldots,\ i_s,\ i_{s+1} = j$$

(эта временная нумерация не имеет ничего общего с той, о которой говорится в формулировке теоремы). Так как $i_0 \in M'$, $j \notin M'$, то найдется наименьшее k, для которого $i_k \in M'$, $i_{k+1} \notin M'$.

Положим

$$M' := M' + i_{k+1}; \qquad N' := N' + u_k;$$
$$m(i_{k+1}) := n(u_k) := |N'|.$$

Получившийся граф связен, так как i_{k+1} соединяется с любой вершиной цепью, состоящей из цепи от этой вершины до i_k и из дуги u_k, а прочие вершины и без того связаны. Так как добавлены одна вершина и одна дуга, количественное соотношение между $|M'|$ и $|N'|$ сохранено. Процесс повторяется до совпадения M' с M. ▲

Следствие 1. *В связном графе число дуг не меньше числа вершин без единицы.*

Следствие 2. *Ранг матрицы инциденций связного графа* $\langle M, N\rangle$ *равен* $|M| - 1$.

Доказательство. Построим множество N', о котором говорится в теореме. Пусть $k = |M| - 1$. Рассмотрим матрицу $a[M - i_0, N']$ Построенная в теореме нумерация переводит множества индексов этой матрицы в множество $1 : k$, и матрицу можно рассматривать как матрицу $a[1 : k, 1 : k]$. По свойствам нумерации $|a[i, i]| = 1$, $a[i, j] = 0$ при $i > j$. Следовательно, ранг $a[1 : k, 1 : k]$ (и $a[M, N]$) не меньше k. А как уже отмечалось выше, больше чем k он быть не может. ▲

Отметим, что каждый минор матрицы инциденций равен 0, 1 или —1 (если в каждом столбце минора два ненулевых элемента, он равен 0, а в остальных случаях его можно разложить по элементам какого-либо столбца и воспользоваться индукцией).

Следствие 2 интересно тем, что связывается чисто алгебраическое свойство матрицы с наглядным геометрическим понятием.

В произвольном графе все вершины и дуги разбиваются на *компоненты связности* — максимальные связные подграфы (т. е. на такие подграфы, в которых множество вершин не может быть увеличено без нарушения связности).

Предложение 4. *Пусть* $i \in M$, M_i — *множество вершин, которые можно соединить с* i *цепью. Подграф* $\langle M_i, N(M_i)\rangle$ *является компонентой связности.*

Доказательство. Нам нужно доказать, что этот подграф связен и что добавление к нему какой-либо вершины с сохранением связности невозможно Пусть $l, j \in M_i$ и отличны от i. Существуют цепи, соединяющие i с l и с j. Перенумеровав их, получим (выписывая только вершины)

$$i_1 = i, i_2, i_3, \ldots, i_s = l,$$
$$j_1 = i, j_2, j_3, \ldots, j_t = j.$$

Пусть r — наибольший номер вершины из второй цепи, содержащейся в первой цепи. Пусть p — номер этой вершины в первой цепи. Цепь

$$\bar{\imath} = i_s,\ i_{s-1},\ \ldots,\ i_p = j_r,\ j_{r+1},\ \ldots,\ j_t = \bar{\jmath}$$

соединяет вершины $\bar{\imath}$ и $\bar{\jmath}$ (рис. 23).

Остается показать, что $\langle M_i,\ N(M_i)\rangle$ — максимальный связный подграф. Но это следует из того, что M_i содержит все вершины, которые можно связать с i, — добавление новых вершин нарушило бы связность из-за отсутствия цепей, соединяющих их с i. ▲

Легко видеть, что если $\langle M_i,\ N_i\rangle$ — компонента связности, то

$$a[M \setminus M_i,\ N_i] = 0[M \setminus M_i,\ N_i],$$
$$a[M_i,\ N \setminus N_i] = 0[M_i,\ N \setminus N_i].$$

Отсюда и из следствия 2 без труда получаем следующее утверждение.

Предложение 5. *Ранг матрицы инциденций равен* $|M| - p$, *где* p — *число компонент связности графа.*

Предложение 6. *Пусть* p — *число компонент связности графа* $\langle M, N\rangle$. *Если* $|N| > |M| - p$, *то в графе содержится цикл.*

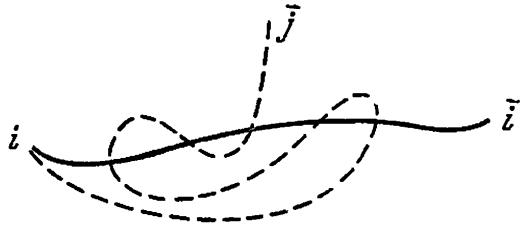

Рис. 23.

Доказательство. Найдется компонента связности $\langle M_\alpha,\ N_\alpha\rangle$, для которой $|N_\alpha| > |M_\alpha| - 1$. По теореме 1 можно построить связный частичный граф $\langle M_\alpha,\ N'\rangle$ графа $\langle M_\alpha,\ N_\alpha\rangle$, для которого $|N'| = |M_\alpha| - 1$. Следовательно, существует $u \in N_\alpha \setminus N'$. Вершины $i(u)$ и $j(u)$ можно соединить цепью из дуг, лежащих в N'. Дополнив эту цепь дугой u, мы получим цикл. ▲

В дальнейшем мы, как правило, будем заниматься связными графами. В некоторых случаях используются более сильные определения связности, в которых требуется существование путей, соединяющих вершины. Одно такое определение понадобится в следующем параграфе.

Граф называется *вполне связным*, если для любых двух его вершин i и j найдется путь, ведущий из i в j.

Пример 4. Пусть M — множество утверждений, N — множество заключений о том, что одно из утверждений

следует из другого. Для того чтобы установить эквивалентность всех утверждений из M, необходимо, чтобы граф $\langle M, N\rangle$ был связным.

Теорема 2, которую нам предстоит доказать в § 4, как раз и утверждает эквивалентность шести различных утверждений.

В гл. 6 и 7 потребуется следующий факт, по уровню используемых определений больше подходящий для этого параграфа. Назовем *длиной* контура число дуг в нем.

Предложение 7. *Если граф* $\langle M, N\rangle$ *вполне связен и общий наибольший делитель длин его контуров равен* 1, *то для любого достаточно большого* k *и любых* $i, j \in M$ *найдется путь из* i *в* j *звенности* k.

Доказательство. Из того, что граф вполне связен, следует существование замкнутого пути p, содержащего все вершины из M. Этот путь может быть добавлен к пути от i к j (который существует, так как граф вполне связен), а к p может быть добавлен любой контур.

Таким образом, осталось показать, что любое достаточно большое число может быть представлено в виде линейной комбинации длин контуров с целыми неотрицательными коэффициентами. Это, однако, известный факт теории чисел. ▲

§ 4. Деревья

Деревом называется связный граф, в котором число дуг на единицу меньше числа вершин. Таким образом, в теореме 1 утверждалось, что в каждом связном графе можно выделить частичный граф — дерево. Это понятие понадобится многократно, поэтому свойства деревьев очень важны. Многие из этих свойств выясняются в следующей теореме.

Теорема 2. *Следующие определения дерева эквивалентны:*

а) *связный граф без циклов*;

б) *граф без циклов, у которого число дуг на единицу меньше числа вершин*;

в) *связный граф, у которого число дуг на единицу меньше числа вершин*;

г) *граф без циклов, в котором добавление дуги приводит к появлению цикла*;

д) *связный граф, который перестает быть связным после удаления любой дуги;*

е) *граф, у которого всякие две вершины соединены цепью и притом только одной.*

Доказательство. Будем вести доказательство по контуру (следовательно, по вполне связному графу), изображенному на рис. 24.

1. *б* → *в*. По предложению 6 этот граф не может иметь больше одной компоненты связности.

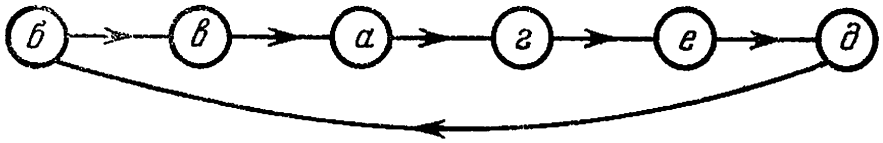

Рис. 24.

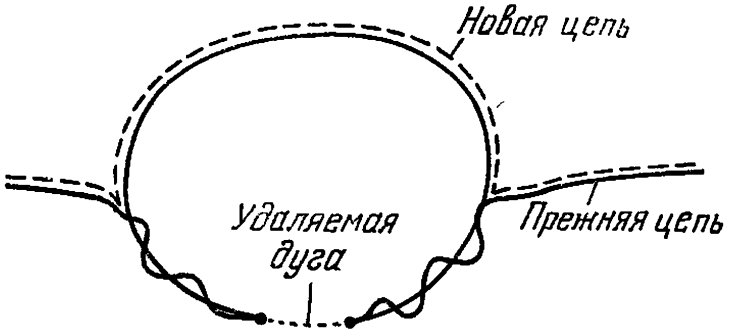

Рис. 25.

2. *в* → *а*. Если связный граф имеет цикл, то одну из дуг этого цикла можно удалить, не нарушая связности графа (рис. 25). Действительно, все связи, не использовавшие удалений дуги, сохраняются, а цепи, в которые она входила, могут быть заменены цепями с «обходом» по оставшейся части цикла. Так как в нашем графе минимальное для связного графа число дуг, он циклов содержать не может.

3. *а* → *г*. Так как граф связен, то $|N| \geqslant |M| - 1$. После добавления еще одной дуги по предложению 6 в графе должен найтись цикл.

4. *г* → *е*. Поскольку добавление дуги приводит к появлению цикла, то цикл содержит эту дугу. Остальная часть цикла представляет собой цепь, соединяющую граничные вершины дуги. Если бы какие-либо две вершины были соединены двумя различными цепями, из этих цепей (или частей) можно было бы составить цикл (рис. 26).

5. *е* → *д*. Рассмотрим граничные вершины какой-либо дуги. Единственная соединяющая их цепь состоит из этой дуги. Ее удаление нарушает, следовательно, связность графа.

6 *д* → *б*. То, что этот граф не имеет циклов, очевидно. Мы уже знаем, что удаление дуги из цикла связности не нарушает. Теперь равенство $|N| = |M| - 1$ следует из предложения 6. ▲

Дерево $\langle M, N' \rangle$ называется *иерархическим*, если в нем найдется вершина $i_0 \in M$, из которой существуют пути во все остальные вершины. Вершина i_0, если она существует, определяется этим условием однозначно. Она называется *корнем* дерева (рис. 27).

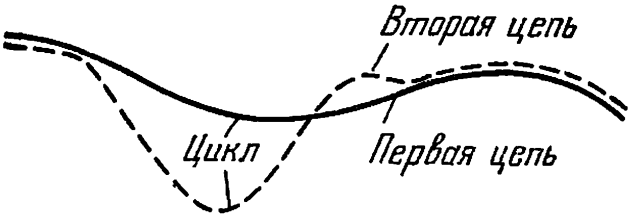

Рис. 26.

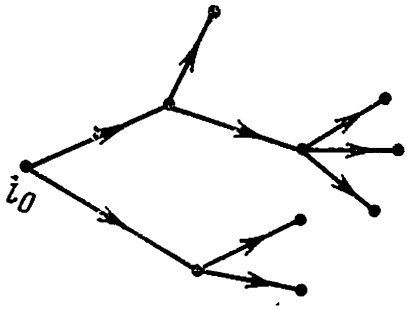

Рис. 27. Иерархическое дерево.

Предложение 8. *Если граф* $\langle M, N \rangle$ *вполне связен, для любой его вершины* i_1 *найдется такое* $N' \subset N$, *что граф* $\langle M, N' \rangle$ *будет иерархическим деревом с корнем в* i_1.

Доказательство проводится точно так же, как доказательство теоремы 1, только в качестве i_0 следует выбрать корень i_1 и строить для присоединения новых вершин не цепи, а пути. ▲

Отметим, что для задания дерева удобно пользоваться нумерацией, о которой шла речь в теореме 1, и каждой вершине сопоставлять номер вершины, к которой она прикрепляется (т е другой граничной вершины дуги, имеющей тот же номер, что данная).

При использовании языков типа алгол-68 для задания дерева удобно в структуре, описывающей вершину, предусматривать ссылку на прикрепляющую дугу и на вершину, к которой данная вершина прикрепляется.

§ 5. Постановка транспортной задачи

Наиболее распространена следующая «классическая» постановка транспортной задачи.

Имеется множество K пунктов производства и множество L пунктов потребления однородного продукта Для каждого пункта производства $i \in K$ задан объем производства $\alpha[i] > 0$, а для каждого пункта потребления $j \in L$ — объем потребления $\beta[j] > 0$. Стоимость перевозки количества продукта x из пункта производства i в пункт потребления j равна $c[i, j] \times x$. Матрица $c[K, L]$ задана. Требуется составить план перевозки продукта, в котором все пункты потребления были бы обеспечены необходимыми количествами продукта, ни из какого пункта производства не вывозилось бы продуктов больше, чем там производится, а стоимость перевозки была бы минимальной.

Если обозначить через $x[i, j]$ количество продукта, перевозимое из i в j, мы получим следующую задачу линейного программирования.

Матричная транспортная задача.

Найти матрицу $x[K, L]$, минимизирующую

$$\sum_{i \in K, j \in L} c[i, j] \times x[i, j] \tag{4}$$

при условиях

$$x[K, L] \geqslant 0[K, L], \tag{5}$$

$$\sum_{j \in L} x[i, j] \leqslant \alpha[i], \quad i \in K, \tag{6}$$

$$\sum_{i \in K} x[i, j] = \beta[j], \quad j \in L. \tag{7}$$

Для того чтобы система ограничений (6), (7) имела решение, необходимо, чтобы $\sum_{i \in K} \alpha[i] \geqslant \sum_{j \in L} \beta[j]$. Мы будем для простоты рассматривать так называемую *закрытую* задачу, в которой эти суммы равны. В этом случае неравенства (6) можно заменить равенствами. Для удобства дальнейшего рассмотрения положим $b[K] = -\alpha[K]$, $b[L] = \beta[L]$ и запишем (6) в виде

$$-\sum_{j \in L} x[i, j] = b[i], \quad i \in K. \tag{8}$$

И для решения этой задачи, и для возможности более простого ее обобщения целесообразно переформулировать задачу в терминах теории графов.

Рассмотрим граф $\langle M, N, T \rangle$, в котором $M = K \cup L$, $N = K \times L$, так что каждое $u \in N$ представимо в виде

(i, j), где $i \in K$, $j \in L$ Естественно считать, что $i((i, j)) = i$, $j((i, j)) = j$. Тогда вместо матриц c и x можно рассматривать векторы $c[N]$ и $x[N]$. Далее заметим, что

$$N_i^- = \{(i, j) \mid j \in L\}, \quad N_i^+ = \emptyset \quad \text{для } i \in K,$$
$$N_i^- = \emptyset, \quad N_i^+ = \{(j, i) \mid j \in K\} \quad \text{для } i \in L$$

Таким образом, (7) можно записать в виде

$$\Sigma_{N_i^+} x[u] = b[i], \quad i \in L, \tag{9}$$

а (8) — в виде

$$-\Sigma_{N_i^-} x[u] = b[i], \quad i \in K. \tag{10}$$

Теперь наша задача может быть переписана в таком виде:

Задан граф $\langle M, N, T\rangle$, вектор $b[M]$, сумма компонент которого равна 0, и вектор $c[N]$. Требуется найти вектор $x[N]$, минимизирующий величину

$$c[N] \times x[N] \tag{11}$$

при условиях

$$x[N] \geqslant 0[N], \tag{12}$$

$$\Sigma_{N_i^-} x[u] - \Sigma_{N_i^+} x[u] = b[i], \quad i \in M. \tag{13}$$

В (13) мы пользуемся тем, что $N_i^- = \emptyset$ для $i \in L$ и $N_i^+ = \emptyset$ для $i \in K$, и поэтому (13) служит единой записью условий (9) и (10). В этой записи уже полностью устранены особенности конкретного вида графа, — такая задача может быть поставлена для произвольного графа и интерпретирована следующим образом.

Граф $\langle M, N, T\rangle$ представляет собой транспортную сеть, узлами которой являются вершины M, а магистралями — дуги N (параметры, характеризующие затраты и возможности движения, могут быть различными для различных направлений). В каждом узле задан объем потребления продукта (отрицательное потребление означает производство). Требуется составить план перевозки с минимальными транспортными издержками.

Здесь мы считаем, что издержки складываются из издержек на отдельных дугах, причем на каждой дуге издержки пропорциональны грузопотоку Это позволяет упростить задание плана перевозки и вместо плана, указываю-

щего «откуда — куда — сколько везти», задавать *сбалансированный поток грузов*. Именно такой сбалансированный поток и разыскивается в задаче (11)—(13). Условием баланса является условие (13), которое может быть в терминах потоков сформулировано так: *для каждой вершины втекающий поток равен сумме вытекающего потока и потребления*

В дальнейшем мы будем заниматься задачей (11)—(13), которая называется *сетевой транспортной задачей* Условие (13) можно переписать короче, если вспомнить, что для матрицы инциденций

$$a[i,\ u]=\begin{cases} 1 \text{ для } u\in N_i^+, \\ -1 \text{ для } u\in N_i^-, \\ 0 \text{ в остальных случаях,} \end{cases}$$

или

$$a[i,\ u] = \delta(u \in N_i^+) - \delta(u \in N_i^-).$$

Теперь (13) можно записать так:

$$a[i,\ N] \times x[N] = b[i] \quad \text{для} \quad i \in M. \tag{13'}$$

или

$$a[M,\ N] \times x[N] = b[M]. \tag{13''}$$

В таком виде легче будет написать двойственную задачу:

Найти вектор $v[M]$, максимизирующий

$$v[M] \times b[M] \tag{14}$$

при условиях

$$v[M] \times a[M,\ N] \leqslant c[N]. \tag{15}$$

Неравенство (15), в свою очередь, можно выписать иначе, вспомнив структуру матрицы инциденций:

$$-v[i(u)] + v[j(u)] \leqslant c[u], \quad u \in N,$$

или

$$v[j(u)] \leqslant c[u] + v[i(u)], \quad u \in N. \tag{16}$$

Переменные этой двойственной задачи принято называть *потенциалами* Далее этот термин будет встречаться неоднократно, а в начале гл 7 излагаются некоторые понятия и факты теории потенциалов, позволяющие установить связи между потенциалами в различном понимании.

Сформулируем в терминах потенциалов критерий оптимальности решения транспортной задачи, который дается теоремой 5 гл. 2 и соотношениями двойственности:

Критерий оптимальности. Пусть $x[N]$ — поток, удовлетворяющий условиям (12), (13) Для того чтобы этот поток был оптимальным решением сетевой транспортной задачи, необходимо и достаточно, чтобы нашелся вектор потенциалов $v[M]$, удовлетворяющий условию (16) и такой, что

$$v[j(u)] = c[u] + v[i(u)] \quad \text{при} \quad x[u] > 0. \tag{17}$$

§ 6. Решение систем с матрицей инциденций

В наши ближайшие цели входит описание метода последовательного улучшения плана для сетевой транспортной задачи. Существенной частью этого метода является изменение имеющегося базисного решения, которое можно описать в следующих геометрических терминах: имеется точка, принадлежащая выпуклому многограннику (даже вершина). Рассматриваются все направления движения из этой точки, в которых целевая функция убывает и которые не выводят движущуюся точку из множества. Выбирается одно из этих направлений, а на нем точка из множества, в которой значение целевой функции наименьшее, после чего процесс повторяется. В том варианте метода последовательного улучшения, который мы рассматривали в гл. 2, в качестве такого направления выбиралось одно из ребер многогранника, поэтому процесс переходил из одной вершины многогранника в другую. Для нас сейчас важно, что, имея решение $x_0[N]$ системы

$$a[M, N] \times x_0[N] = b[M],$$

мы представляли новое решение в виде

$$x_\lambda[N] = x_0[N] + \lambda \times z[N], \quad \lambda \geqslant 0, \tag{18}$$

где $z[N]$ — решение однородной системы

$$a[M, N] \times z[N] = 0[M]. \tag{19}$$

Это направление допустимо, если

$$z[N_0] \geqslant 0[N_0], \quad \text{где} \quad N_0 = \{u \mid x_0[u] = 0\}.$$

В этом параграфе мы получим представление множества решений и множества неотрицательных решений системы (19) для случая, когда $a[M, N]$ — матрица инциденций.

Пусть $\langle M', N' \rangle$ — контур. Характеристический вектор $w[N]$ подмножества $N' \subset N$ мы будем называть *контурным* вектором. Для цикла $\langle M', N' \rangle$ зададим в соответствии с предложением 2 ориентацию (направление обхода) и определим вектор $w[N]$, полагая $w[u] = 1$ для положительно ориентированных дуг цикла, $w[u] = -1$ для отрицательных дуг и $w[u] = 0$ для дуг, не принадлежащих циклу. Такой вектор назовём *циклическим*. Знак компонент циклического вектора зависит от выбранного направления обхода. Циклический вектор, соответствующий контуру, при одном из направлений обхода совпадает с контурным вектором этого контура.

Теорема 3 (Пуанкаре — Веблена — Александера). *Каждое решение* $z[N]$ *системы* (19) *представимо в виде линейной комбинации циклических векторов.*

Доказательство. Пусть

$$N_z = \{u \mid z[u] \neq 0\}.$$

Мы будем доказывать теорему редукцией множества N_z. Прежде всего заметим, что циклические векторы удовлетворяют (19). Действительно, пусть $w[N]$ — циклический вектор для цикла $\langle M', N' \rangle$. Для $i \notin M'$, очевидно,

$$w[N_i'^{+} \cup N_i'^{-}] = 0\,[N_i'^{+} \cup N_i'^{-}],$$

и поэтому $a[i, N] \times w[N] = 0$. Вершине $i \in M'$ в N' инцидентны две дуги. Назовем их u_1 и u_2. Если одна из них принадлежит $N_i'^{+}$, а другая $N_i'^{-}$, то обе эти дуги одинаково ориентированы, и так как $w[u_1] = w[u_2]$, выполняется равенство

$$\sum_{u \in N_i'^{+}} w[u] = \sum_{u \in N_i'^{-}} w[u].$$

Если же они принадлежат одному и тому же множеству, то имеют различную ориентацию, $w[u_1] = -w[u_2]$ и равенство также выполняется.

Рассмотрим теперь граф $\langle M_z, N_z \rangle$, где M_z — множество граничных вершин дуг N_z. Каждой вершине $i \in M_z$ инцидентно не менее двух дуг из N_z, так как иначе из равенства

$$\sum_{u \in N_i^{+}} z[u] = \sum_{u \in N_i^{-}} z[u]$$

следовало бы, что все слагаемые равны 0, что противоречит определению N_z и M_z. Выберем произвольно $i_1 \in M_z$.

Найдется дуга $u_1 \in N_z$, инцидентная этой вершине. Обозначим вторую граничную вершину этой дуги через i_2. Найдется отличная от u_1 дуга $u_2 \in N_z$, инцидентная i_2. Будем продолжать этот процесс построения цепи до тех пор, пока какая-либо из вершин i' цепи не встретится второй раз. Часть цепи между этими двумя появлениями i' составляет цикл. Пусть $w[N]$ — соответствующий ему циклический вектор. Представим $z[N]$ в виде

$$z[N] = \lambda \times w[N] + z'[N].$$

Вектор $z'[N]$, очевидно, удовлетворяет системе (19). Выбрав λ так, чтобы одна из компонент $z'[N]$ обратилась в нуль, мы получим новый вектор с меньшим множеством N_z. Утверждение теперь следует из возможности повторения описанной процедуры до полного исчерпания N_z. ▲

Напомним, что конической комбинацией называется линейная комбинация с неотрицательными коэффициентами.

Теорема 4 (Бержа). *Каждое неотрицательное решение системы* (19) *представимо в виде конической комбинации контурных векторов.*

Теорема доказывается совершенно аналогично теореме 3, выяснение различий в доказательстве предоставляется читателю.

Нам будет полезно одно специальное представление замкнутого потока циклическими векторами. Пусть граф $\langle M, N\rangle$ связен, $\langle M, N'\rangle$ — дерево в этом графе. Назовем *основным циклом* такой, в котором только одна дуга не содержится в N'. Соответствующий основному циклу циклический вектор будем называть *основным циклическим вектором.*

Теорема 5 (Кирхгофа). *Решение* $z[N]$ *системы* (19) *представимо в виде линейной комбинации основных циклических векторов.*

Доказательство. Отличие от теоремы 3 здесь лишь в способе выбора цикла для редукции. Пусть, как раньше,

$$N_z = \{u \mid z[u] \neq 0\}.$$

Так как в $\langle M_z, N_z\rangle$ содержатся циклы, то найдется $u \in N_z \setminus N'$. Добавление дуги u к $\langle M, N'\rangle$ порождает однозначно определенный цикл, содержащий u. Пусть $w[N]$ — соответствующий ему циклический вектор. Выбе-

рем ориентацию цикла так, чтобы $w[u] = 1$. Представим $z[N]$ в виде

$$z[N] = z[u] \times w[N] + z'[N].$$

Очевидно, что $z'[u] = 0$; дуга u исключается из N_z и множество $N_z \setminus N'$ уменьшается на один элемент. При исчерпании множества $N_z \setminus N'$ автоматически исчерпывается и множество N_z. ▲

Следствие. *Любое решение системы* (19), *неотрицательное вне* N', *представимо в виде конической комбинации основных циклических векторов.*

Таким образом, если план перевозки положителен на дугах, составляющих дерево, то конус возможных направлений изменения плана является конической оболочкой основных циклических векторов.

§ 7. Метод потенциалов

Основной частью метода последовательного улучшения плана является «улучшение» — проверка оптимальности базисного решения, выбор небазисной переменной, вводимой в базис и изменение базиса.

Рассмотрим, как выглядят все эти действия в случае транспортной задачи. Для простоты будем предполагать граф задачи $\langle M, N\rangle$ связным

Базис в задаче линейного программирования определялся как набор переменных, которому отвечает максимальный набор линейно независимых столбцов матрицы ограничений. Из доказанных теорем следует, что набор N' будет базисом в том и только в том случае, если $\langle M, N'\rangle$ — дерево.

Базисный план находится из условий

$$x[N \setminus N'] = 0[N \setminus N'],$$
$$a[M, N'] \times x[N'] = b[M].$$

Решению этой системы существенно помогает нумерация вершин и дуг, о которой шла речь в теореме 1. Пусть $k = |M| - 1$ Уравнение, соответствующее i_k, имеет вид

$$a[i_k, u_k] \times x[u_k] = b[i_k]$$

Найдя $x[u_k]$ из этого уравнения и подставив его значение в уравнение, соответствующее второй граничной вершине

дуги u_k, мы получим такую же систему для оставшихся дуг, в которой снова сможем определить последнюю переменную.

Поясним этот способ вычисления примером Рассмотрим граф, изображенный на рис. 28. Цифры около дуг — стоимости перевозок, цифры около вершин — объемы потребления. На рис. 29 изображено дерево, соответствующее

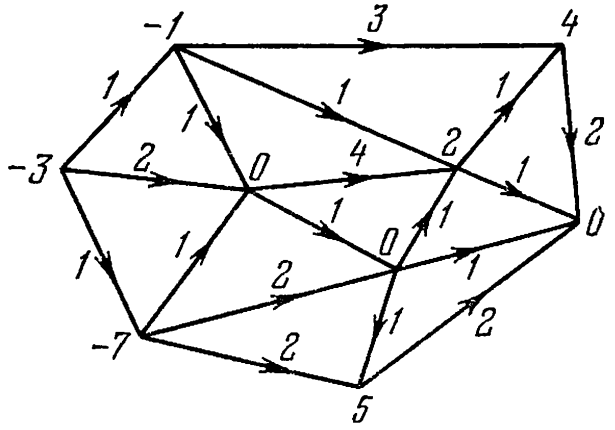

Рис. 28. Транспортная сеть с исходными данными

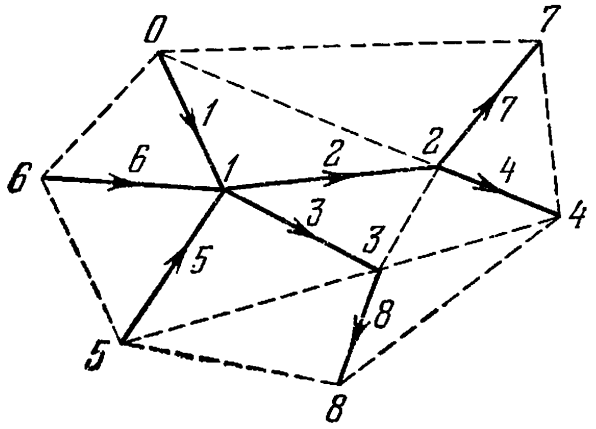

Рис. 29. Начальный базис с правильной нумерацией.

базисному решению. Цифры около вершин и дуг — их номера. Табл. 3 показывает, как при этой нумерации будет выглядеть соответствующая часть матрицы инциденций (рядом в рамке записан вектор $b\,[M]$, а дальше — те его изменения, которые произойдут при последовательном вычислении $x\,[u_k]$). Нахождение значений базисных переменных идет с конца: $x\,[u_8] := 5$. Вторая граничная вершина дуги u_8 — ее начало i_8. Заменяем $x\,[u_8]$ значением $b\,[i_3]$.

Таблица 3

	1	2	3	4	5	6	7	8			8	7	6	5	4	3	2
0	−1									0	−1						
1	1	−1	−1		1	1				1	0			−3	−10		−5	1
2		1		−1			−1			2	2		6			6		
3			1					−1		3	0		5					
4				1						4	0							
5					−1					5	−7							
6						−1				6	−3							
7							1			7	4							
8								1		8	5							

Далее $x\,[u_7] := 4$, аналогично $b\,[i_2] := b\,[i_2] + x\,[u_7]$;

$$x\,[u_6] := 3, \quad b\,[i_1] := b\,[i_1] - x\,[u_6],$$
$$x\,[u_5] := 7, \quad b\,[i_1] := b\,[i_1] - x\,[u_5],$$
$$x\,[u_4] := 0, \quad b\,[i_2] := b\,[i_2] + x\,[u_4],$$
$$x\,[u_3] := 5, \quad b\,[i_1] := b\,[i_1] + x\,[u_3],$$
$$x\,[u_2] := 6, \quad b\,[i_1] := b\,[i_1] + x\,[u_2].$$

Наконец, $x\,[i_1] = 1$.

По рис. 29 хорошо следить за тем, как ведется этот вычислительный процесс — в крайних вершинах графа, которым инцидентно лишь по одной дуге, поток определяется однозначно, дальше можно отбросить дугу, на которой поток определен, и повторить процесс.

Для проверки оптимальности решения нужно по уравнениям

$$v\,[M] \times a\,[M,\ N'] = c\,[N'] \qquad (20)$$

найти систему потенциалов и проверить выполнение неравенств (16).

Система (20) содержит уравнений на одно меньше, чем переменных, и позволяет определить потенциалы с точностью до произвольной постоянной. Из (17) видно, что этой постоянной является слагаемое, одинаковое для всех $i \in M$. Если зафиксировать значение потенциала в одной (нулевой) вершине, остальные по ней можно будет найти однозначно. При этом целесообразно нахождение их в порядке возрастания номеров вершин: при вычислении потенциала l-й вершины используется l-е уравнение (17),

в которое входит потенциал l-й вершины и (уже вычисленный) потенциал вершины с меньшим номером.

Так, в нашем примере полагаем $v[i_0] := 0$, а далее

$$\begin{aligned}
v[i_1] &= v[i_0] + c[u_1], & 0 + 1 &= 1,\\
v[i_2] &= v[i_1] + c[u_2], & 1 + 4 &= 5,\\
v[i_3] &= v[i_1] + c[u_3], & 1 + 1 &= 2,\\
v[i_4] &= v[i_2] + c[u_4], & 5 + 1 &= 6,\\
v[i_5] &= v[i_1] - c[u_5], & 1 - 1 &= 0,\\
v[i_6] &= v[i_1] - c[u_6], & 1 - 2 &= -1,\\
v[i_7] &= v[i_2] + c[u_7], & 5 + 1 &= 6,\\
v[i_8] &= v[i_3] + c[u_8], & 2 + 1 &= 3.
\end{aligned}$$

Таким образом, удачная нумерация вершин и дуг графа предельно упрощает решение и прямой, и двойственной

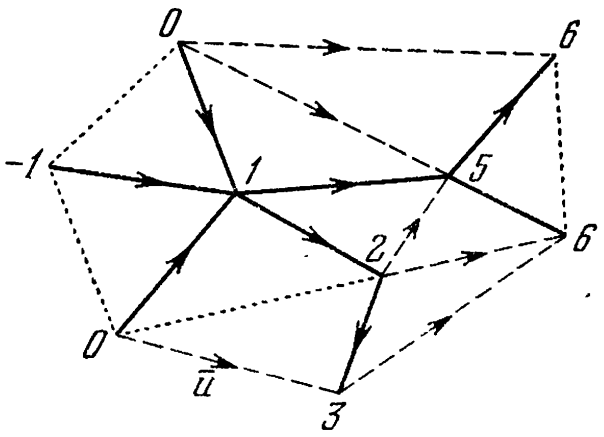

Рис. 30. Потенциалы начального базиса.

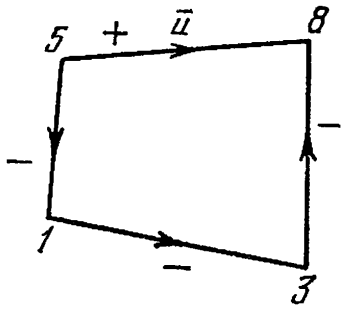

Рис. 31. Цикл исправления.

систем, требуемых для метода последовательного улучшения базисного решения.

Теперь остается для каждого $u \in N \setminus N'$ проверить справедливость неравенства (16). На рис. 30 изображены результаты этой проверки. Базисные дуги изображены сплошными линиями, дуги, на которых нарушается условие (16), — штриховыми линиями, прочие дуги — пунктиром. При нарушениях на некоторых дугах условий (16) полагается выбрать из них какую-либо дугу $\bar{u}$ для введения в базис. Этой дуге соответствует основной цикл, порождающий допустимое направление.

Выберем в нашем случае дугу, идущую из i_5 в i_8. Рассмотрим цикл, порожденный этой дугой (рис. 31, около дуг изображен знак их ориентации при положительной ориентации $\bar{u}$).

Часть базисного плана, соответствующая дугам из цикла, имеет вид (0, 5, 5, 7), а часть циклического вектора (1, —1,—1, —1). Поэтому в выражении

$$x + \lambda \times w$$

максимальное λ, которое мы можем выбрать, сохраняя неотрицательность потоков, это $\lambda = 5$.

В общем случае очевидно

$$\lambda = \min \{x [u] \mid u \in N^-\}, \qquad (21)$$

где N^- — множество отрицательно ориентированных дуг цикла. Пусть $\tilde{u}$ — (какая-либо) дуга из N^-, на которой достигается минимум в (21). Удалив $\tilde{u}$ из базиса (заменив ее дугой $\bar{u}$), мы снова получим дерево. Выберем для примера в качестве $\tilde{u}$ дугу 3.

Если говорить о вычислительной стороне этой части алгоритма, то мы должны рассмотреть два вопроса.

Первый — это способ нахождения цикла по дуге $\bar{u}$, добавляемой к дереву. Здесь нам помогает все та же нумерация: выписав номера граничных вершин $\bar{u}$, мы просматриваем список вершин в порядке номеров (это делается одновременно с вычислением потоков) и, дойдя до выписанного номера k, заменяем его на номер второй граничной вершины k-й дуги. Таким образом, мы строим две цепи, начинающуюся из i ($\bar{u}$) (левую) и начинающуюся из j ($\bar{u}$) (правую). В каждой цепи номера вершин последовательно уменьшаются. В момент, когда эти номера совпадут (обозначим общий номер через m_0), мы и получим цикл. Удаление дуги $\tilde{u}$ разрывает одну из двух цепей на две. Будем называть получившиеся цепи *верхней* (часть, имеющая меньшие номера) и *нижней* (большие номера). Вторую цепь будем называть *целой*.

Теперь мы подошли ко второму вопросу вычислительного характера — как построить для измененного базиса нумерацию, которая оказалась настолько полезной в расчетах.

Поиск цикла и дуги $\tilde{u}$ разбил все вершины на пять подмножеств:

M_0 — не встретившиеся вершины (имеющие номера меньше m_0),

M_1 — вершины целой цепи,

M_2 — вершины верхней цепи,

M_3 — вершины нижней цепи,

M_4 — остальные вершины (имеющие номера больше m_0, но не входящие в цикл).

Некоторые из них могут быть пусты, например:

$$M_0 = \{0\}, \quad M_1 = \{1, 5\}, \quad M_2 = \{1\},$$
$$M_3 = \{3, 8\}, \quad M_4 = \{2, 4, 6, 7\}.$$

Объединив множества M_0, M_1, M_2 в одно $\bar{M}$, выпишем элементы $\bar{M}$ в порядке возрастания их номеров, затем множество M_3 по убыванию номеров, затем множество M_4 по возрастанию номеров. Порядковые номера вершин при этой записи мы и примем за новые номера. Порядковые номера дуг определяются как наибольшие из номеров граничных вершин. В нашем примере получим:

старые номера	0 1 5 8 3 2 4 6 7	вершины,
новые номера	0 1 2 3 4 5 6 7 8	
старые номера	1 2 3 4 5 6 7 8 $\tilde{u}$	дуги.
новые номера	1 5 $\tilde{u}$ 6 2 7 8 4 3	

Этот базис изображен на рис. 32.

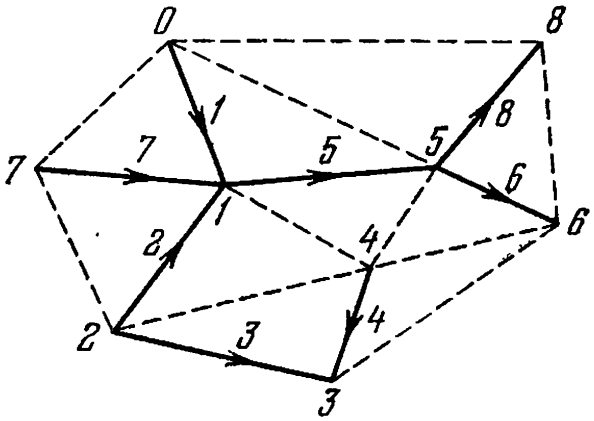

Рис. 32. Результат второй итерации с новой нумерацией.

Построением нового базиса закончена итерация.

Повторим теперь процесс, оставляя лишь необходимые вычислительные действия. Первоначально потоки на дугах нас не должны интересовать.

Проводим проверку оптимальности решения, соответствующего данному базису. Строим потенциалы и вычисляем невязки. Результаты вычисления в прежних обозначениях приведены на рис. 33. Наибольшая невязка соответствует дуге, идущей от вершины 0 к вершине 5 (в нумерации

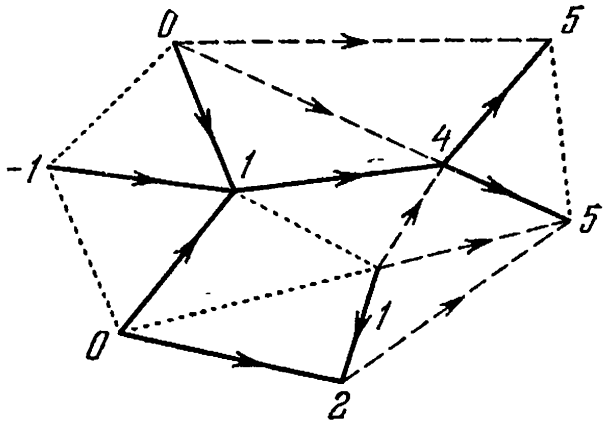

Рис. 33. Потенциалы второй итерации.

рис. 32). Просматривая вершины от больших номеров к меньшим, вычисляем потоки и строим цикл (табл. 4). Минимум потока на отрицательных дугах равен 1 и достигается на дуге 1.

Теперь пересчет потоков не нужен, так как λ уже найдено. Осталось изменить нумерацию. Здесь

$$M_0 = \phi, \quad M_1 = \{0\}, \quad M_2 = \{0\},$$

$$M_3 = \{1, 5\}, \quad M_4 = \{2, 3, 4, 6, 7, 8\}$$

и перенумерация дает (рис. 34):

старые номера	0	5	1	2	3	4	6	7	8	} вершины,
новые номера	0	1	2	3	4	5	6	7	8	
старые номера	1	2	3	4	5	6	7	8	$\tilde{u}$	} дуги.
новые номера	$\tilde{u}$	3	4	5	2	6	7	8	1	

Читателю рекомендуется довести вычисления до конца.

Мы рассмотрели содержание одной итерации метода потенциалов. Для полноты нужно описать построение начального допустимого базисного решения.

Используются два основных метода. Наиболее прост метод «искусственного базиса», который заключается в том,

Таблица 4

Вершина	Поток по дуге	Вторая граничная вершина	Измененная потребность	Цикл	Минимум
8	4	5	6	$0 \to 5$	
7	3	1	—3	$0 \to 5$	
6	0	5	6	$0 \to 5$	
5	6	1	3	$1 \downarrow 0 \to 5$	3
4	0	3	5	$1 \downarrow 0 \to 5$	
3	5	2	—2	$1 \downarrow 0 \to 5$	
2	2	1	1	$1 \downarrow 0 \to 5$	
1	1	0	0	$0 \downarrow 1 \downarrow 0 \to 5$	1

что строится расширение $\langle \bar{M}, \bar{N} \rangle$ исходного графа $\langle M, N \rangle$. Здесь

$$\bar{M} = M + \bar{i}, \qquad \bar{N} = \bar{N} \cup N_{\bar{i}}^{+} \cup N_{\bar{i}}^{-}$$

($\bar{i}$ — новая вершина, $N_{\bar{i}}^{+}$ — множество дуг, ведущих от пунктов производства к i, $N_{\bar{i}}^{-}$ — множество дуг от $\bar{i}$ до всех остальных пунктов). Значение $b\,[\bar{i}]$ полагаем равным 0,

$$c\,[N_{\bar{i}}^{+}] = 0\,[N_{\bar{i}}^{+}], \qquad c\,[N_{i}^{-}] = gz \times 1\,[N_{i}^{-}],$$

где gz — достаточно большое число (например, всегда достаточно $gz > \sum_{u \in N} c\,[u]$). В качестве начального N' можно в этом случае принять множество $\bar{N} \setminus N$.

В нашем примере мы получим таким образом граф, изображенный на рис. 35.

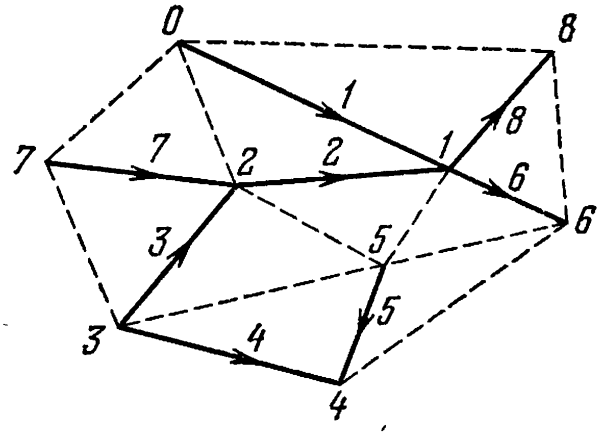

Рис. 34. Оптимальное решение.

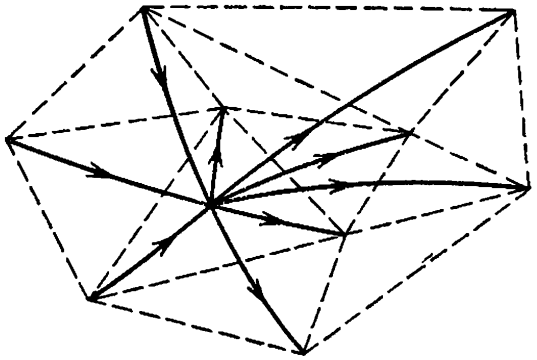

Рис. 35. Искусственный базис.

Для матричной транспортной задачи может быть предложен другой метод:

1°. Выберем какую-либо вершину i (каждая из вершин в матричной задаче — это пункт производства или пункт потребления).

2°. Выберем дугу, инцидентную i. Поток по этой дуге положим равным минимуму из объема производства в начале дуги и объема потребления в конце дуги. Уменьшим эти объемы на величину потока по дуге; одну из вершин, объем которой стал равен нулю, устраним из рассмотрения вместе с инцидентными ей дугами, а другую вершину примем в качестве i. После этого вернемся к выполнению п. 2°.

В случае, когда множества вершин K и L перенумерованы и каждый раз выбирается дуга с наименьшим номером вершин (второй вершины), метод называется *методом северо-западного угла*. Это название широко распространено в популярных сочинениях. Более целесообразно выбирать из оставшихся дуг, инцидентных i, дугу с наименьшей стоимостью.

§ 8. Алгоритмическая реализация метода потенциалов

Поскольку метод потенциалов мы рассматриваем как разновидность метода последовательных улучшений плана, для программного воплощения этого метода нам достаточно, следуя схеме процедуры *LP opt basic solution* из § 6 гл. 2, выписать описания процедур *start*, *new*, *work* и *fin* и связать эти процедуры едиными информационными описаниями.

Рассмотрим сначала реализации метода на алголе-60.

Мы будем задавать граф для простоты списком троек. Обозначим число дуг графа через n и введем три массива

array $c[1:n]$; **integer array** $i, j\ [1:n]$

где $i\ (u)$ — начало дуги u, $j\ (u)$ — конец, $c\ [u]$ — стоимость перевозки по дуге. Множеством вершин будет множество M, вектор потребностей будет задаваться массивом $b\ [M]$,

$$\mathbf{1}\ [M] \times b\ [M] = 0.$$

Нумерацию вершин зададим вектором $w\ [1:m]$, где $w\ [i] \in M$ — истинный номер вершины, получающей i-й номер в этой нумерации. Нумерацию дуг мы задавать не будем, но введем вектор $arc\ [M]$, сопоставляющий каждой вершине (кроме первой) дугу с тем же номером.

Чтобы различать множество M и множество номеров $1:m$, в качестве M выберем $r+1:r+m$, где r — достаточно большое целое число (например, $r = 1000$).

Проще всего написать, как обычно, процедуру *new*. Она состоит из вычисления потенциалов, которое производится просмотром вершин в порядке их номеров, и последующей проверки, которую сейчас произведем в простейшем варианте (без порогов и без циклического порядка просмотра):

```
procedure newT (opt); boolean opt;
begin real max; integer k, s, u;
   array v[1001 : 1000+m]; v[1001] := .0;
   for s := 2 step 1 until m do
     begin k := w[s]; u := arc[k];
       v[k] := if i[u]=k then v[j[u]]-c[u]
                 else v[i[u]]+c[u] end;
   max := .0;
   for u := 1 step 1 until n do
     if v[j[u]]-v[i[u]]-c[u] > max then
       begin max := v[j[u]]-v[i[u]]-c[u]; u0 := u end;
   opt := max < eps
end newT
```

Процедура *work* состоит из двух отдельных блоков. В первом из них находится цикл и выделяются множества M_3 и M_4. Элементы множества M_4 помечаются знаками минус в массиве $w[1 : m]$, о множестве M_3 достаточно знать число его элементов, так как по связи выкидываемой из базиса дуги $u1 = arc[w[s0]]$ и соответствующей ей вершины $i1 = w[s0]$ можно установить, какой ветви принадлежит это множество. (Если $i1 = i[u1]$, то разрывается левая цепь — начинающаяся из $i(\bar{u})$, если же $i1 = j[u1]$, то разрывается правая цепь.) Выпишем вначале заголовок процедуры и первый блок:

```
procedure workT;
begin integer k, k2, l1, l2, r1, r2, s, s0, t, u;
   begin real min, y; array x[1001 : 1000+m];
     r1 := i[u0]; r2 := j[u0]; l1 := l2 := 0; min := gz;
     for s := 1001 step 1 until 1000+m
                                  do x[s] := b[s];
     for s := m step -1 until 2 do
       begin k := w[s]; u := arc[k]; y := x[k];
         k2 := if k=i[u] then j[u] else i[u];
                                  x[k2] := x[k2]+y;
         if r1=k then begin l1 := l1+1; r1 := k2;
           if min > abs(y) and k=i[u] then
             begin min := abs(y); s0 := s; t := l1 end end
         else if r2=k then begin l2 := l2+1; r2 := k2;
           if min > abs(y) and k=j[u] then
```

```
            begin min := abs (y); s0 := s; t := l2 end end
          else w[s] := -w[s];
          if r1 = r2 then go to mks end s;
mks: end x;
```

comment Одна операторная скобка осталась незакрытой — кончился блок, но процедура еще не кончилась.

Во втором блоке происходит перестройка базиса, для чего используется вспомогательный массив $d[1:m]$, в начало которого выписывается в обратном порядке множество M_3, а в конец множество M_4. После этого оставшиеся элементы массива w пододвигаются вверх, переписывается M_3 и наконец M_4;

```
begin integer array d[1:m]; k:=w[s0]; u:=arc[k];
  if k=i[u] then begin r1:=i[u0]; d[1]:=j[u0] end
  else begin r1:=j[u0]; d[1]:=i[u0] end;
  d[2]:=r1; l1:=2; l2:=m;
  for s:=3 step 1 until t+1 do
    begin u:=arc[r1]; d[s]:=r1:=i[u]+j[u]-r1 end;
  for s:=t step -1 until 2 do
                                  arc[d[s+1]]:=arc[d[s]];
  arc[d[2]]:=u0;
  for s:=m step -1 until s0 do
    if w[s]<0 then begin d[l2]:= -w[s];
                                  l2:=l2-1 end
    else if w[s]=d[l1] then begin
                                  w[s]:=0; l1:=l1+1 end;
    for s:=s0 step 1 until m do
      if w[s]>0 then begin w[s0]:=w[s];
                                  s0:=s0+1 end;
    for s:=1 step 1 until t do w[s0+s-1]:=d[s+1];
    for s:=s0+t step 1 until m do w[s]:=d[s]
  end d;
    for s:=1 step 1 until m do w[s]:=abs(w[s]);
  iter:=iter+1
end work
```

После этой процедуры записать какой-либо вывод результатов труда не представляет. Вот одна из простейших процедур, в которой выписывается список базисных дуг с их началами и концами и потоками на них:

```
procedure finT;
begin array x[1001 : 1000+m]; integer k, k2, s, u;
   for s:=1001 step 1 until 1000+m do x[s]:=b[s];
   for s:=m step -1 until 2 do
     begin k:=w[s]; u:=arc[k];
      k2:=i[u]+j[u]-k; x[k2]:=x[k2]+x[k];
      print ('?', u, i[u], j[u], abs (x[k]))
end end
```

Наконец, для процедуры *start* мы используем метод искусственного базиса. Нет нужды вводить для этого много дуг и дополнительную вершину, как об этом писалось выше. Достаточно ввести нулевые компоненты в массивах i, j и c, положив $i[0] := j[0] := 1001$; $c[0] := gz$, но во всех случаях при $arc[k] = 0$ соответственно менять $i[0]$ или $j[0]$:

```
if u=0 then begin i[0]:=j[0]:=1001;
 if b[k]<0 then i[0]:=k else j[0]:=k end;
```

Такую вставку нужно делать в процедурах *new T* и *work T* (в двух местах) после операторов $u := arc[k]$.

Сама же процедура *start T* выглядит очень просто:

```
procedure startT;
begin integer s; iter:=0;
   for s:=1 step 1 until m do
     begin arc[1000+s]:=0; w[s]:=1000+s end
end
```

Приведем теперь всю процедуру решения транспортной задачи полностью:

```
procedure potential method (m, n, i, j, c, b, gz);
   value m, n, gz; integer m, n; real gz;
   integer array i, j; array c, b;
begin integer iter, u0; real eps; boolean opt;
   integer array arc[1001 : 1000+m], w[1 : m];
procedure workT;
begin integer k, k2, l1, l2, r1, r2, s, s0, t, u;
  begin real min, y; array x[1001 : 1000+m];
    r1:=i[u0]; r2:=j[u0]; l1:=l2:=0; min:=gz;
    for s:=1001 step 1 until 1000+m do x[s]:=b[s];
```

```
      for s:=m step -1 until 2 do
        begin k:=w[s]; u:=arc[k]; y:=x[k];
      if u=0 then begin i[0]:=j[0]:=1001;
            if b[k]<0 then i[0]:=k else j[0]:=k end;
            k2:=if k=i[u] then j[u] else i[u];
                                        x[k2]:=x[k2]+y;
            if r1=k then begin l1:=l1+1; r1:=k2;
             if min>abs(y) ∧ k=i[u] then
               begin min:=abs(y); s0:=s; t:=l1 end end
            else if r2=k then begin l2:=l2+1; r2:=k2;
             if min>abs(y) ∧ k=j[u] then
               begin min:=abs(y); s0:=s; t:=12 end end
            else w[s]:=-w[s];
            if r1=r2 then go to mks end s;
mks: end x;

begin integer array d[1:m]; k:=w[s0]; u:=arc[k];
   if u=0 then begin i[0]:=j[0]:=1001;
       if b[k]<0 then i[0]:=k else j[0]:=k end;
   if k=i[u] then begin r1:=i[u0]; d[1]:=j[u0] end
   else begin r1:=j[u0]; d[1]:=i[u0] end;
   d[2]:=r1; l1:=2; l2:=m;
   for s:=3 step 1 until t+1 do
       begin u:=arc[r1]; d[s]:=r1:=i[u]+j[u]-r1 end;
   for s:=t step -1 until 2 do
                                   arc[d[s+1]]:=arc[d[s]];
   arc[d[2]]:=u0;
   for s:=m step -1 until s0 do
       if w[s]<0 then begin d[l2]:=-w[s];
                                        l2:=l2-1 end
       else if w[s]=d[l1] then begin w[s]:=0;
                                        l1:=l1+1 end;
   for s:=s0 step 1 until m do
       if w[s]>0 then begin w[s0]:=w[s];
                                     s0:=s0+1 end;
   for s:=1 step 1 until t do w[s0+s-1]:=d[s+1];
   for s:=s0+t step 1 until m do w[s]:=d[s]
     end d;

   for s:=1 step 1 until m do w[s]:=abs(w[s]);
     iter:=iter+1
end work;
```

```
procedure finT;
begin array x[1001 : 1000+m]; integer k, k2, s, u;
    for s:=1001 step 1 until 1000+m do x[s]:=b[s];
    for s:=m step -1 until 2 do
   begin k:=w[s]; u:=arc[k];
     k2:=i[u]+j[u]-k; x[k2]:=x[k2]+x[k];
     print( ?', u, i[u], j[u], abs(x[k]))
end end;
procedure startT;
begin integer s; iter:=0;
   for s:=1 step 1 until m do
   begin arc[1000+s]:=0; w[s]:=1000+s end
end;
procedure newT (opt); boolean opt;
begin real max; integer k, s, u;
    array v[1001 : 1000+m]; v[1001]:=.0;
    for s:=2 step 1 until m do
     begin k:=w[s]; u:=arc[k];
       if u=0 then begin i[0]:=j[0]:=1001;
       if b[k]<0 then i[0]:=k else j[0]:=k end;
        v[k]:=if i[u]=k then v[j[u]]-c[u]
                  else v[i[u]]+c[u] end;
max:=.0;
for u:=1 step 1 until n do
  if v[j[u]]-v[i[u]]-c[u]>max then
    begin max:=v[j[u]]-v[i[u]]-c[u]; u0:=u end;
opt:=max<eps
end newT;
    eps:=10-5; startT;
mkit: newT(opt); if opt then go to mkfin;
    workT; go to mkit;
mkfin: finT
end potmeth
```

Рассмотрим небольшой пример. Пусть транспортная сеть представляет собой граф, изображенный на рис. 36. Нумерация вершин имеется на рисунке, нумерация дуг может быть установлена по таблице данных, в которую уже добавлен нулевой элемент, требуемый для процедуры *start T*.

Вектор потребления $b\,[M]$ выписан отдельно

u:	0	1	2	3	4	5	6	7	8	9
$i\,[u]$:	1001	1001	1001	1002	1003	1003	1003	1004	1004	1006
$j\,[u]$:	1001	1002	1003	1005	1002	1005	1006	1003	1006	1005
$c\,[u]$:	gz	9	4	3	4	8	6	4	5	1

$b\,[M]$: $-5, 3, 0, -7, 4, 5$

Состояние вычислительного процесса представляется записью, в которую входят целочисленные переменные

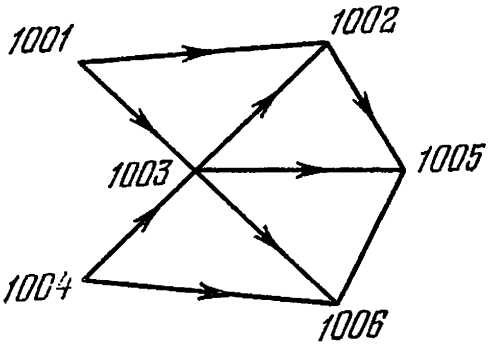

Рис. 36 Транспортная сеть.

iter, $u0$ и векторы $w\,[1 : 6]$, *arc* [1001 : 1006]. После завершения работы *start T* получаем

iter	$u0$	w	1	2	3	4	5	6
0	—		1001	1002	1003	1004	1005	1006

arc	1001	1002	1003	1004	1005	1006
	0	0	0	0	0	0

Для удобства мы будем писать gz вместо его конкретного значения. В процедуре *new T* вычисление потенциалов даст нам

v	1001	1002	1003	1004	1005	1006
	0	gz	gz	$-gz$	gz	gz

после чего проверка оптимальности приведет к значениям $max = 2 \times gz - 4$, $u0 = 7$, $opt \equiv$ **false.**

Выполнение первой части процедуры *work T* даст (цикл состоит, естественно, из трех дуг — положительно ориентированной дуги 7 и двух искусственных дуг 1004 → 1001

и 1001→1003, ориентированных отрицательно; размыкается вторая из этих дуг)

$$s0=3,\quad min=0,\quad t=1$$

w	1	2	3	4	5	6
	1001	1002	1003	1004	−1005	−1006

После второй части процедуры *new T* получаем (мы добавим к стандартной информации массив d [1 : 6], чтобы виднее были действия, выполненные процедурой)

iter	*u*0	w	1	2	3	4	5	6
1	7		1001	1002	1004	1003	1005	1006

arc	1001	1002	1003	1004	1005	1006
	0	0	7	0	0	0

d	1	2	3	4	5	6
	1004	1003			1005	1006

На второй итерации после *new T* получаем

v	1001	1002	1003	1004	1005	1006
	0	gz	$4-gz$	$-gz$	gz	gz

А во время *work T* и после завершения ее работы получаем

iter	*u*0	w	1	2	3	4	5	6
2	8		1001	1002	1004	1003	1005	1006

arc	1001	1002	1003	1004	1005	1006
	0	0	7	0	0	8

Наконец, после 6-й итерации получаем

v	1001	1002	1003	1004	1005	1006
	0	8	4	5	11	10

Вычисленные по процедуре *new T* потенциалы удовлетворяют условиям (16), и мы получили оптимальное решение. Процедура дает окончательно (знак вопроса трактуется в *print* как перевод строки)

```
8  1004  1006  7
9  1006  1005  2
3  1002  1005  2
4  1003  1002  5
2  1001  1003  5
```

Напишем теперь ту же процедуру решения транспортной задачи на алголе-68.

Потребовав, чтобы процедура *new* вырабатывала булево значение «имеется улучшение», мы сможем основную часть алгоритма представить так:

start;
while *new* **do** *work* **od**;
fin

Для задания исходной информации введем три таких же массива, как и раньше, но в процедуре *start* преобразуем их в массив структур. Для этого нового массива мы введем структуру, описывающую дугу

mode arc = struct (**real** *c*, **int** *i*, *j*)

где i и j будут номерами вершин, соответствующих началу и концу дуги. В структуре вершины будут также две ссылки — ссылка на вторую граничную вершину базисной дуги *upv* и еще одна ссылка на вершину *next*, служащая для задания упорядочения вершин с помощью цепных списков. Кроме того, в этой структуре будет с нужным знаком храниться стоимость перевозки по базисной дуге *c* и будет отведено поле *vx* для вычисления потоков и потенциалов

mode vert = struct (**real** *c*, *vx*, **ref vert** *upv*, *next*)

Для информации о дугах мы введем массив [1 : *n*] **arc** *data*, для информации о вершинах — массив [1 : *m*] **vert** *base*.

Теперь уже все готово для того, чтобы написать процедуру *new T*. Осталось только сделать несколько уточнений к самой процедуре. Она будет состоять из двух частей — вычисления потенциалов и проверки оптимальности. Проверка оптимальности будет происходить в точности, как раньше.

Для вычисления потенциалов мы будем двигаться по цепному списку вершин, начиная от вершины *vf*, и последовательно вычислять потенциалы. Однако так как при вычислении потоков нам понадобится противоположный порядок вершин, ссылки в ходе вычислений будут переворачиваться. Начало новой цепи будет содержаться в *ve*. В этой и следующих процедурах мы будем считать описанными

для **vert** те операции, которые мы описываем для **elem** в гл. 1 (т. е. **lin, cons, head, rob** и **join**)

```
proc newT = bool:
begin real d, max:= .0; ref vert k:= next of vf;
   ve:= vf; next of vf:= nil; vx of vf:= .0;
   while lin k do vx of k:= vx of upv of k + c of k;
      ve rob k od;
   for u to n do ref arc a:= data [u];
      if (d:= vx of base [j of a]
                  - vx of base [i of a] - c of a) > max
            then max:= d; u0:= u fi od;
   max > eps
end
```

Процедура *work T* нуждается в более серьезных комментариях. При поисках контура мы будем распределять встретившиеся вершины по трем множествам: левой цепи, правой цепи и множеству M_4. Для каждого из этих множеств будет строиться цепной список с началами и концами соответственно в *lhf* и *lhe*, *rhf* и *rhe*, *fp* (конца не требуется).

Когда будет выяснено, какая из цепей разрывается, мы соединим множества M_0, M_1 и M_2 в одну цепь, но не так, как мы это делали раньше, а просто «прицепим» наши цепи друг к другу. После этого мы перевернем ссылки на получившейся цепи, присоединим цепь, соответствующую M_3, изменив в элементах этой цепи поля, соответствующие стоимости и другой вершине *upv*, и, наконец, прицепим M_4 (чтобы не делать лишнего, эту последнюю цепь мы будем строить прямо в перевернутом виде. На рис. 37—39 показана

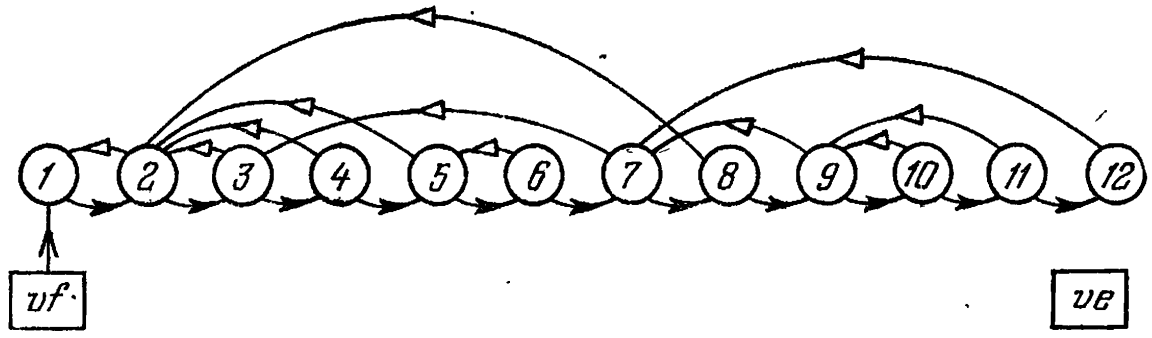

Рис. 37. Связи вершин в начале процедуры.

последовательность всех действий. На рис. 37 изображены связи между вершинами до работы процедуры *new T*.

Сверху изображены связи, задаваемые ссылками *upv*, снизу — ссылками *next*. В прямоугольниках записаны ссылки. Вводится в базис дуга, ведущая из вершины *5* в *10*.

На рис. 38 — состояние связей после работы первой части процедуры *work T*. Указаны лишь ссылки *next* и две ссылки *upv* в концах цепей, так как ссылки *upv* не изменились.

На рис. 39 показаны связи после работы *work T*.

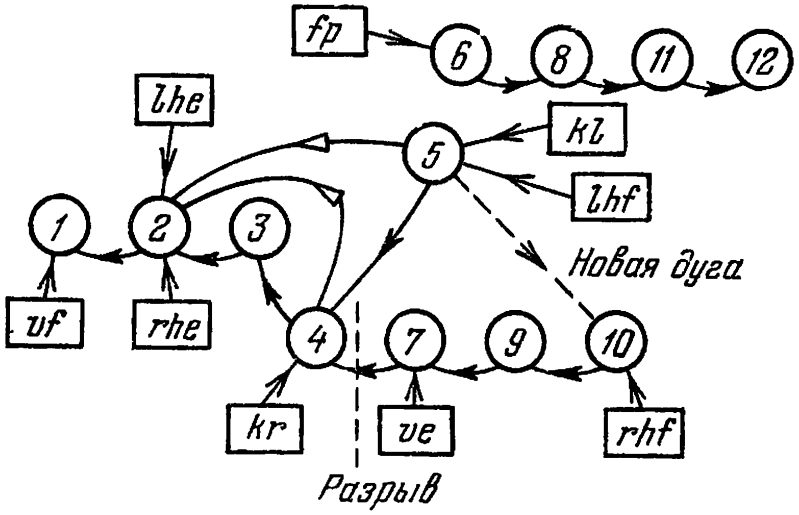

Рис. 38. Связи вершин после нахождения цикла.

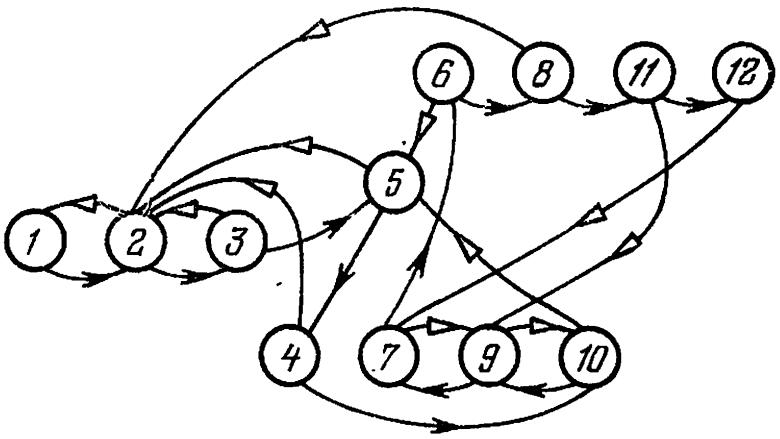

Рис. 39. Связи вершин в начале следующей итерации.

Мы будем пользоваться операцией **dif** сравнения двух ссылок

$$\textbf{op dif} = (\textbf{ref vert } a,\ b)\ \textbf{bool}:\ a\ \textbf{isnt}\ b$$

Смысл введения такой операции в том, чтобы не беспокоиться о нужных уровнях разыменований ссылок.

Дальнейшие пояснения даются прямо в теле процедуры:

```
proc workT = void:
begin real y, y1 := c of data[u0], min := maxreal;
   ref vert k := ve, k1, k1 := nil, kr := nil,
                lhf := base [i of data [u 0]],
             rhf := base [i of data [u0]], lhe := lhf,
                rhe := rhf, fp := nil;
   for s to m do vx of base[s] := b[s] od;
```

co Сделаны начальные присваивания началам и концам цепей, в поля *vx* переписаны потребности вершин. Теперь будет выполняться движение по цепи снизу вверх, которое закончится при совпадении ссылок co

```
while lhe dif rhe do vx of upv of k +:= y := vx of k;
   y := abs y; k1 := head k;
   if k1 dif lhe /\ (bool rh := k1 dif rhe) then fp cons k1
```

co Пересчитывается *vx*, отцепляется в *k*1 (с помощью операции **head**) очередная вершина, и теперь мы выясняем ее судьбу. Если она отличается от тех, которые нам нужны, ее поглотит список *fp*, в противном случае она присоединится к соответствующей ветви. Обратите внимание на условный оператор после **else** — в каждой из альтернатив он выполняет некоторые действия (один оператор присваивания) и вырабатывает первый аргумент операции **join**. Дальше следуют действия, связанные с поиском минимума встречного потока на цикле co

```
   else if rh then kr := rhe; rhe
      else kl := lhe; lhe fi join upv of k1;
   if min > y /\ c of k1 < 0 = rh then min := y;
                                       ve := k1 fi fi od;
if c of ve < 0 then
   if lin kr then next of kr := k; next of k1 := rhf
   else next of kl := k fi; kl := lhf; kr := rhf; y1 := - y1
else if lin kl then next of kl := k; next of kr := lhf
      else next of kr := k fi; kl := rhf; kr := lhf fi;
```

co Здесь производится соединение цепей — по знаку *c* **of** *ve* находится целая ветвь, которая присоединяется к M_0, а к ней присоединяется верхняя цепь. Отдельно рассматривается случай, когда целая ветвь пуста. После

выполнения этого оператора *kl* ссылается на начало нижней цепи, а *kr* — на конец **co**

```
k:=kl;
while k dif ve do k1:=upv of k; upv of k:=kr;
   y:=-c of k; c of k:=y1; y1:=y; kr:=k; k:=k1 od;
k:=next of ve; ref vert (ve):=(y1, skip, kr, fp);
```

co Перевернута нижняя цепь (с довольно сложными изменениями ссылок и передачами информации о базисных дугах). После этого нижняя цепь прикреплена к *fp*. Теперь осталось перевернуть объединенную верхнюю цепь **co**

```
            while k dif vf do kl rob k od;
            next of vf:=kl; iter+:=1
         end
```

Приведем теперь всю процедуру метода потенциалов целиком, опуская для краткости описания *new T* и *work T*

```
proc pot method=(int m, n, ref [ ] arc data,
                                    ref [ ] real b) void:
begin int u, u0, iter:=0;
   real cost:=0, eps:=10-5, gz:=1010;
   mode vert=struct (real c, vx, ref vert upv, next);
   ref vert vf, ve; [1:m] vert base;
   vf:=base[1];
   for s to m do base[s]:=
      (if b[s]>0 then gz else-gz fi, 0,
       if s=1 then nil else vf fi,
       if s=m then nil else base[s+1] fi) od;
   while newT do workT od;
   for s to m do vx of base[s]:=b[s] od;
   while ve dif vf do real y:=vx of ve;
      vx of upv of ve+:=y; cost+:=abs (c of ve×y);
      ve:=next of ve od;
   c of base[1]:=1;
for s from 2 to m do
      print (s, c of upv of base[s], vx of base[s]);
      c of base[s]:=s od;
   print (iter, cost)
end
```

Для простоты здесь процедуры *start T* и *fin T* ликвидированы, — соответствующие действия вписаны непосредственно в основную процедуру.

§ 9. Варианты транспортной задачи

Рассмотрим несколько различных вариантов транспортной задачи, допускающих применение метода потенциалов с некоторыми достаточно простыми модификациями.

а) Задача с ограничениями пропускных способностей. Кроме обычных для транспортной задачи ограничений, вводится еще одна группа ограничений: задается положительный вектор $d\,[N]$ и требуется, чтобы

$$x\,[N] \leqslant d\,[N] \tag{22}$$

(поток $x\,[u]$ по каждой дуге u не превосходит пропускной способности этой дуги $d\,[u]$.)

В двойственной задаче ограничения (22) приводят к появлению новых неотрицательных переменных $\rho\,[N]$, именуемых «рентами».

Двойственная задача приобретает теперь такой вид:

Максимизировать

$$v\,[M] \times b\,[M] + \rho\,[N] \times d\,[N] \tag{23}$$

при условиях

$$\rho\,[N] \geqslant 0\,[N], \tag{24}$$

$$v\,[M] \times a\,[M, N] - \rho\,[N] \geqslant c\,[N]. \tag{25}$$

Неравенство (25) переписывается таким образом: для каждого $u \in N$

$$v\,[j\,(u)] \leqslant c\,[u] + \rho\,[u] + v\,[i\,(u)]$$

Критерий оптимальности для задачи с ограничениями можно, впрочем, написать, исключая ренты.

Обозначим через N_0 множество дуг, на которых поток равен 0, а через N_1 множество «насыщенных» дуг, на которых поток равен $d\,[N_1]$. Иногда N_0 в алгоритме будет включать не все дуги с нулевым потоком, а N_1 — не все насыщенные дуги.

Для того чтобы допустимый план $x\,[N]$ был оптимальным, необходимо и достаточно, чтобы нашлись такие потенциалы $v\,[M]$, чтобы разность

$$v\,[j\,(u)] - v\,[i\,(u)] - c\,[u]$$

была равна нулю, когда $0 < x\,[u] < d\,[u]$, неотрицательна при $u \in N_1$ и неположительна при $u \in N_0$.

Таким образом, может быть два вида нарушений условий оптимальности: как и прежде, на дуге с нулевым потоком может оказаться

$$v\,[j\,(u)] > v\,[i\,(u)] + c\,[u],$$

и, в отличие от прежнего, на насыщенной дуге может оказаться

$$v\,[j\,(u)] < v\,[i\,(u)] + c\,[u].$$

В первом случае надо увеличивать поток по такой дуге, прибавляя соответствующий циклический вектор к вектору потоков; во втором — уменьшать, вычитая соответствующий циклический вектор. Это вполне согласуется со следующей теоремой, являющейся прямым аналогом теоремы 5.

Теорема 6. *Пусть* $\langle M, N'\rangle$ *— дерево в связном графе* $\langle M, N\rangle$, $N \setminus N'$ *разбито на два подмножества* N_0 *и* N_1. *Любое решение системы* (19), *неотрицательное на* N_0 *и неположительное на* N_1, *представимо в виде разности конических комбинаций основных циклических векторов, соответствующих дугам из* N_0 *и из* N_1,

$$z\,[N] = \sum_{u \in N_0} \lambda_u \times \omega_u\,[N] - \sum_{u \in N_1} \lambda_u \times \omega_u\,[N].$$

Доказательство очевидно.

Теперь уже можно просто описать, как решается методом потенциалов задача с ограниченными пропускными способностями.

Базис задается деревом N' и разбиением множества $N \setminus N' = N_0 \cup N_1$. Положив $x\,[N_0] = 0\,[N_0]$, $x\,[N_1] = d\,[N_1]$ и приняв в качестве вектора потребностей вершин «скорректированный вектор»

$$b\,[M] - a\,[M, N_1] \times d\,[N_1],$$

однозначно определяем по дереву потоки на N' (так же, как и раньше).

Дерево $\langle M, N'\rangle$ используется также и для вычисления потенциалов. По потенциалам определяем, оптимален ли план, и если он не оптимален, вводим изменения — положительный основной циклический вектор, соответствующий дуге из N_0, или отрицательный основной циклический вектор, соответствующий дуге из N_1.

Далее определяем максимальный множитель λ, с которым можно прибавить циклический вектор к плану перевозки. В отличие от предыдущего, величина λ лимитируется

не только уменьшением потоков на отрицательно ориентированных дугах цикла, но и увеличением потоков на положительно ориентированных дугах. Таким образом,

$$\lambda = \min \{\min_{u \in N^-} x[u], \quad \min_{u \in N^+} (d[u] - x[u])\}.$$

Дуга, на которой достигается минимум, выводится из базиса и включается в N_0, если она отрицательно ориентирована, и в N_1, если положительно ориентирована.

Начальное приближение в задаче с ограничениями пропускных способностей легко строить тем же методом искусственного базиса.

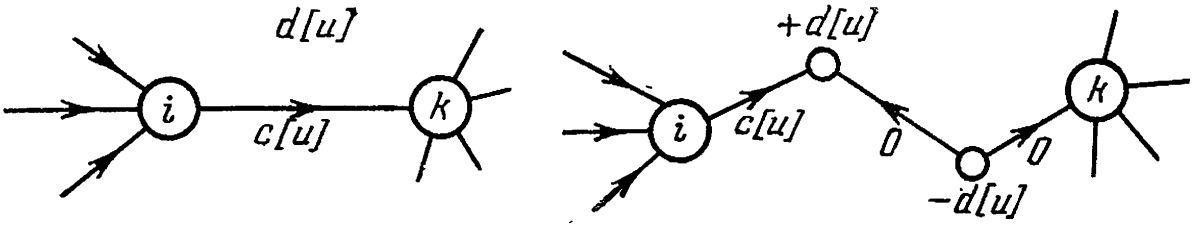

Рис. 40. Замена дуги с ограниченной пропускной способностью.

Отметим, что простым преобразованием сети, заменяющим каждую ограниченную дугу тремя «неограниченными» дугами (рис. 40), можно привести задачу с ограничениями пропускных способностей к эквивалентной задаче без ограничений.

б) Задача об оптимальном контуре. В некоторых случаях требуется оптимизировать замкнутый поток, разумеется, при каких-либо дополнительных ограничениях, так как множество замкнутых потоков представляет собой, как мы знаем, выпуклый конус.

Простейшим из таких ограничений является условие нормировки или взвешенной нормировки потока. Пусть задан граф $\langle M, N \rangle$ и два вектора $c[N]$ и $l[N]$, причем $l[N] > 0[N]$. Рассмотрим задачу.

Задача об оптимальном нормированном потоке.

Найти $x[N]$, минимизирующий

$$c[N] \times x[N] \tag{26}$$

при условиях

$$a[M, N] \times x[N] = 0[M], \quad x[N] \geqslant 0[N], \tag{27}$$

$$l[N] \times x[N] = 1. \tag{28}$$

Планы, удовлетворяющие условию (27), являются по теореме 4 коническими комбинациями контурных векторов. Если определить контурные векторы так, чтобы они удовлетворяли условию (28), то каждый вектор, удовлетворяющий условиям задачи, будет уже выпуклой комбинацией контурных векторов.

Поиск оптимальной крайней точки многогранника (27)—(28) будет, таким образом, эквивалентен поиску такого контура $\langle M', N' \rangle$, на котором достигается минимум отношения

$$\Sigma_{N'} c[u] / \Sigma_{N'} l[u]. \qquad (29)$$

Если $l[u]$ рассматривать как длину дуги u, а $c[u]$ — как затраты на прохождение этой дуги, то мы получим задачу о нахождении *контура с наименьшими затратами на единицу длины.*

Эта задача, как мы увидим в дальнейшем, имеет самостоятельное значение. Однако целесообразно искать именно не контур, а крайнюю точку многогранника (27), (28).

Двойственной к задаче (26)—(28) будет задача:

Найти вектор $v[M]$ и число z, удовлетворяющее условиям

$$v[M] \times a[M, N] + z \times l[N] \geqslant c[N] \qquad (30)$$

и максимизирующие z.

Условие (30), очевидно, преобразуется в условие

$$v[j(u)] - v[i(u)] + l[u] \times z \geqslant c[u] \quad \text{для} \quad u \in N. \qquad (31)$$

Переменные $v[i]$ естественно продолжать называть потенциалами. Переменная z имеет особый смысл минимального удельного убытка (убытка на единицу длины на оптимальном контуре).

Базис в этой задаче включает, очевидно, на одну дугу больше, чем в транспортной задаче. Пусть N' — набор дуг базиса. Граф $\langle M, N' \rangle$, очевидно, связен, и если базису отвечает допустимый план, то он содержит контур $\langle \bar{M}, \bar{N} \rangle$. На дугах N' условия (31) должны выполняться как равенства. Просуммировав по дугам N', получим

$$z \times \Sigma_{\bar{N}} l[u] = \Sigma_{\bar{N}} c[u]$$

и найдем из этого условия z. После этого, зафиксировав один из потенциалов, обычным образом по дереву, содержащемуся в $\langle M, N' \rangle$, найдем весь вектор $v[M]$.

Проверка оптимальности базисного решения производится здесь, как обычно. В случае неоптимальности найдется дуга $\bar{u}$, для которой

$$v\,[j\,(\bar{u})] > v\,[i\,(\bar{u})] - l\,[\bar{u}] \times z + c\,[\bar{u}].$$

Эту дугу мы можем включить в базис. Процесс ее включения в базис зависит от того, где лежат концы дуги $\bar{u}$. Если добавление дуги $\bar{u}$ порождает новый контур, то суммирование по дугам этого контура дает нам соотношение

$$\Sigma\, c\,[u] - z \times \Sigma\, l\,[u] < 0,$$

т. е. средний убыток на этом контуре меньше z. В этом случае необходимо разомкнуть старый контур, исключив из него одну дугу из числа дуг, не входящих в новый контур.

Для случая, когда добавление дуги $\bar{u}$ контура не порождает, процесс сводится также к удалению из базиса дуги $\tilde{u}$, не принадлежащей контуру. Практически выбор $\tilde{u}$ достаточно безразличен, однако для установления сходимости процесса следует подчинить выбор $\tilde{u}$ какому-либо жесткому правилу. Одно из таких правил и рассматривается ниже

Пусть M^* — множество вершин, для которых в $\langle M, N'\rangle$ существует путь, ведущий от контура $\langle \bar{M}, \bar{N}\rangle$ до вершины Пусть $i\,(\bar{u}) \in M^*$. Рассмотрим цепь, ведущую от $j\,(\bar{u})$ к контуру (по этой цепи определяется потенциал вершины $j\,(\bar{u})$) В этой цепи есть отрицательно ориентированные дуги, так как иначе $\bar{u}$ замыкала бы контур. Первую из этих дуг и выберем в качестве $\tilde{u}$.

Если же $i\,(\bar{u}) \notin M^*$, то, построив цепь, соединяющую $i\,(\bar{u})$ с контуром, примем в качестве $\tilde{u}$ последнюю положительно ориентированную дугу в этой цепи.

Теорема 7. *Описанный процесс сходится к оптимальному решению за конечное число шагов.*

Доказательство. Докажем прежде всего, что за конечное число шагов будут построены решение и система потенциалов, удовлетворяющие условиям (30).

Действительно, граф $\langle M, N\rangle$ конечен и имеет конечное число различных контуров. При замене контура, содержащегося в базисном плане, z строго уменьшается, поэтому возвращение к контуру, встречающемуся раньше, исключается, и после конечного числа шагов все изменение базиса

будет относиться к изменению потенциалов для вершин, не входящих в контур.

Легко убедиться в том, что изменения базиса, не затрагивающие контура, могут разве лишь увеличить множество M^*. Поэтому через конечное число шагов множество M^* перестанет изменяться и дальнейшие изменения будут относиться только к значениям потенциалов на вершинах вне контура.

Такие изменения могут быть двух типов. В первом происходит уменьшение потенциала в одной или нескольких вершинах из M^* (обязательно в $j(\bar{u})$). Из монотонности процесса следует, что базисы с одним и тем же контуром и с одним и тем же M^* не могут появляться повторно.

Второй тип изменений не затрагивает значений в M^*. Он лишь увеличивает один или несколько потенциалов в одной или нескольких вершинах из $M \setminus M^*$ (обязательно в $i(\bar{u})$). И здесь из монотонности процесса следует, что возможно лишь конечное число изменений, и значит, через конечное число шагов мы получим систему потенциалов, удовлетворяющую условию (30).

Осталось показать, что полученное решение оптимально. Пусть $\langle \tilde{M}, \tilde{N} \rangle$ — оптимальный контур, а $\langle \bar{M}, \bar{N} \rangle$ — контур из базисного решения. Суммируя неравенство (31) по $u \in \tilde{N}$, получим

$$\Sigma_{u \in \tilde{N}} (v[j(u)] - v[i(u)] + z \times l[u]) \leqslant \Sigma_{u \in \tilde{N}} c[u]$$

или

$$\Sigma_{\tilde{N}} c[u] / \Sigma_{\tilde{N}} l[u] \geqslant z = \Sigma_{\bar{N}} c[u] / \Sigma_{\bar{N}} l[u],$$

что и доказывает оптимальность контура $\langle \bar{M}, \bar{N} \rangle$. ▲

Отметим еще, что из этого неравенства следует, что для любого оптимального контура на всех его дугах неравенство (31) выполняется как равенство.

Следствие. *Пусть N' — оптимальный базис задачи* (26)—(28). *Множество оптимальных контуров этой задачи совпадает с множеством контуров графа $\langle M, N_0 \rangle$, где*

$$N_0 = \{u \mid v[j(u)] - v[i(u)] + z \times l[u] = c[u]\}.$$

Задача об оптимальном контуре может быть подобно транспортной задаче использована для построения методов решения более сложных задач, например задач с ограничениями пропускных способностей.

Нам эта задача потребуется дальше, когда мы будем рассматривать асимптотические свойства процессов динамического программирования.

в) Двухкомпонентная задача. Рассмотрим теперь следующую задачу линейного программирования.

Заданы конечные множества M, N, векторы $c[N]$, $b[M]$ и матрица $a[M, N]$, в каждом столбце которой содержится по одному или по два ненулевых элемента. Требуется найти вектор $x[N]$, удовлетворяющий условиям

$$a[M, N] \times x[N] = b[M], \quad x[N] \geqslant 0[N] \tag{32}$$

и минимизирующий линейную функцию

$$c[N] \times x[N]. \tag{33}$$

Эта задача называется обычно *двухкомпонентной* (по условию на число ненулевых компонент в столбце).

Пусть

$$N_k = \{j \in N \mid \text{в } a[M, j] \text{ содержится } k \text{ ненулевых компонент}\}.$$

Таким образом, $N = N_1 \cup N_2$.

Каждому элементу из N_2 можно сопоставить дугу (безразлично какой ориентации), граничные вершины которой соответствуют ненулевым компонентам.

Будем считать, что ранг матрицы $a[M, N]$ равен $|M|$. Рассмотрим вопрос о базисах системы ограничений (32). В нашем случае базисом является множество N', число элементов которого равно $|M|$.

Если $N' \subset N_2$ и граф, соответствующий N', связен, то он состоит из дерева и еще одной дуги, порождающей цикл. Отождествляя дуги и соответствующие им элементы из N, обозначим этот цикл через $\langle \bar{M}, \bar{N} \rangle$. Оказывается, невырожденность матрицы $a[M, N']$ зависит только от невырожденности $a[\bar{M}, \bar{N}]$. Действительно, в столбцах из $\bar{N}$ все ненулевые элементы сосредоточены в строках $\bar{M}$ и, следовательно, при разложении определителя $a[M, N']$ по столбцам $\bar{N}$ отлично от нуля может быть лишь слагаемое, в котором перемножаются определители матриц $a[\bar{M}, \bar{N}]$ и $a[M \setminus \bar{M}, N \setminus \bar{N}]$. Далее можно пронумеровать вершины из $M \setminus \bar{M}$

и дуги из $N \setminus \bar{N}$ так, что номер каждой дуги будет равен максимальному из номеров вершин (вершины из $\bar{M}$ мы перенумеруем предварительно) — здесь достаточно сослаться на теорему 1 и на связность графа $\langle M, N' \rangle$.

В этой нумерации матрица $a[M \setminus \bar{M}, N' \setminus \bar{N}]$ будет треугольной и, следовательно, невырожденной.

В общем случае условие базисности набора N' выглядит несколько сложнее. Граф $\langle M, N' \rangle$ может быть не связным.

Более того, трудно говорить о графе, так как среди базисных индексов могут быть индексы из N_1, которым не соответствуют дуги. Каждому из таких индексов может, однако, быть поставлена в соответствие вершина из M. Обозначим множество вершин, соответствующих $N_1' = N_1 \cap N'$, через M_1.

Теорема 8. *Пусть* $|N'| = |M|$. *Для того чтобы матрица* $a[M, N']$ *была невырожденной, необходимо и достаточно, чтобы в каждой компоненте связности графа* $\langle M, N_2' \rangle$ *содержалась либо вершина из* M_1, *либо невырожденный цикл, т. е. цикл* $\langle \bar{M}, \bar{N} \rangle$ *с невырожденной матрицей* $a[\bar{M}, \bar{N}]$.

Доказательство. Пусть k — число компонент связности графа. Относя элементы N_1' к той компоненте связности, которой принадлежит соответствующая вершина, мы получим k пар $\langle M_\alpha, N_\alpha \rangle$. Легко видеть, что $|M_\alpha| = |N_\alpha|$. В связном графе число дуг может быть меньше числа вершин разве лишь на единицу. Поэтому в каждой компоненте связности содержится не более чем по одной точке из M_1. Тем компонентам связности, где точек из M_1 нет, соответствует граф с одним циклом, и этот цикл должен быть невырожденным по предыдущему рассуждению. ▲

При решении этой задачи также большую пользу приносит правильная нумерация. Следует иметь в виду, что решение системы

$$a[M, N'] \times x[N'] = b[M]$$

производится для каждой компоненты связности отдельно. Для компонент содержащих вершины из M_1 вычисление идет нормально с единственным условием, чтобы оно заканчивалось расчетом для переменной из N_1'.

Для компонент, содержащих цикл, решение систем производится изящным и своеобразным приемом: выберем

дугу j из цикла и выразим через $x[j]$ все остальные переменные компоненты связности

$$a[M_\alpha, N_\alpha - j] \times x[N_\alpha - j] = b[M_\alpha] - a[M_\alpha, j] \times x[j].$$

Граф $\langle M_\alpha, N_\alpha - j\rangle$ является деревом, и значения всех переменных могут быть последовательно найдены (выражены через $b[M_\alpha]$ и $x[j]$). При этом используются все уравнения, кроме уравнений с наименьшим номером. Это оставшееся уравнение и используется для вычисления значения $x[j]$.

Можно добиться большего единства в изложении, если считать, что элементам из N_1' отвечают петли — циклы состоящие из одной дуги. В этом случае в каждой компоненте связности присутствует ровно один цикл, и последовательность нахождения потоков такова — сначала вычисляются потоки на всех висящих дугах, а затем, когда остается найти только поток на цикле, он находится так, как описано в предыдущем параграфе.

	a	b	c	d	e	f	
1	3						6
2	1	1			1		4
3			3			2	5
4		4					7
5			2	3	4		3
6					2	3	6

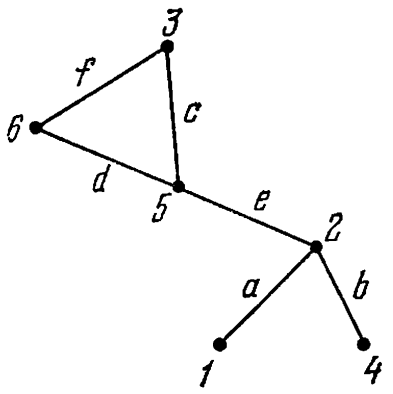

Рис. 41. Пример базиса в двухкомпонентной задаче.

Например, по матрице, изображенной ниже, и соответствующей графу рис. 41, вычисления потоков идут в следующем порядке:

$$4 \times x[b] = 7, \qquad x[b] = {}^7/_4,$$
$$3 \times x[a] = 6, \qquad x[a] = 2,$$
$$x[e] + x[a] + x[b] = 4, \qquad x[e] = {}^1/_4,$$
$$2 \times x[c] + 3 \times x[d] = 3 - 1 = 2, \qquad x[c] = 1 - {}^3/_2 \times x[d],$$
$$3 \times x[c] + 2 \times x[f] = 5, \qquad x[f] = {}^5/_2 - 3 \times (1 - {}^3/_2 \times x[d]) = 1 + {}^9/_2 \times x[d],$$
$$2 \times x[d] + 3 \times x[f] = 6, \qquad {}^{29}/_2 \times x[d] = 3.$$

Отсюда $x[d] = {}^{6}/_{29}$, и теперь мы можем по $x[d]$ найти также $x[c]$ и $x[f]$.

Двойственная система

$$v[M] \times a[M, N'] = c[N']$$

решается в обратном порядке: сначала на цикле $\langle \bar{M}, \bar{N} \rangle$ находится $v[\bar{M}]$, а по $v[\bar{M}]$ вычисляются остальные переменные. Для вычисления $v[\bar{M}]$, так же как раньше, все переменные выражаются через одну, которая затем находится из последнего соответствующего циклу уравнения.

Последний вопрос, связанный со спецификой метода последовательных улучшений плана в данной задаче, — изменение базиса. Добавление дуги u к графу $\langle M, N' \rangle$ либо соединяет две компоненты связности — в этом случае появится компонента связности с двумя циклами — либо прямо породит второй цикл в какой-нибудь компоненте связности.

Результат в обоих случаях один и тот же: компонента связности с двумя циклами. Рассмотрим частичный подграф $\langle M_0, N_0 \rangle$ этой компоненты, состоящий из обоих циклов и соединяющий их цепи. Дуга $\bar{u}$, очевидно, входит в N_0. Мы будем искать вектор $z[N]$, являющийся решением системы $a[M, N] \times z[N] = \mathbf{0}[M]$, положительный на дуге и отличный от нуля лишь на дугах N_0.

Пусть $\langle M_1, N_1 \rangle$ — один цикл, $\langle M_2, N_2 \rangle$ — другой, i_1 — вершина, в которой первый цикл соприкасается с соединяющей цепью, и i_2 — такая же вершина для второго. Не исключено, что $i_1 = i_2$. Система

$$a[M_1, N_1] \times z[N_1] = g[M_1]$$

имеет единственное решение при любом $g[M_1]$, в частности, при $g[M_1]$, отличном от нуля только на компоненте $g[i_1] = \alpha$. Обозначим решение этой системы через $z_\alpha[N_1]$. Выберем значение $z_\alpha[u]$ на дуге u, принадлежащей цепи и инцидентной вершине i_1, таким образом, чтобы восстановился баланс в вершине i_1 (для этого нужно $z[u]$ положить равным $-\alpha/a[i_1, u]$). Для соблюдения баланса во второй граничной вершине u выберем значение $z[u]$ на следующей дуге цепи и т. д. до попадания в вершину i_2. При подходящем выборе вектора $g[M_2]$, точнее, его единственной ненулевой компоненты $g[i_2]$, система

$$a[M_2, N_2] \times z[N_2] = g[M_2]$$

будет иметь такое (единственное) решение, что и в вершине i_2 будет выполняться условие баланса

$$a\,[i_2,\ N_0] \times z\,[N_0] = 0.$$

Вектор $z\,[N_0]$ определяется с точностью до множителя. Выберем его знак таким образом, чтобы значение $z\,[\bar{u}]$ было положительным, и будем искать новый план в виде

$$x_\lambda\,[N] = x\,[N] + \lambda \times z\,[N].$$

При возрастании λ обратится в нуль значение $x_\lambda\,[\tilde{u}]$ на одной из дуг. Удалим эту дугу $\tilde{u}$ из базиса. Удаление $\tilde{u}$ разорвет либо цепь (тогда компонента связности разделится на две с циклом в каждой из них), либо один из циклов, и снова получится базис прежнего вида.

При решении систем мы, так же как в транспортной задаче, пользовались удобно выбранной нумерацией. Однако описание алгоритма во всех деталях потребовало бы слишком много места, и поэтому здесь не приводится.

ГЛАВА 4

ЗАДАЧИ, РОДСТВЕННЫЕ ТРАНСПОРТНОЙ

§ 1. Упорядочения

Рассмотрим граф $\langle M, N, T\rangle$. Будем говорить, что дуги этого графа задают *предшествования* на множестве его вершин: начало каждой дуги *предшествует* его концу. Можно рассматривать цепочки из таких предшествований — последовательности вершин, в которых каждый элемент предшествует в этом смысле следующему за ним элементу, и говорить, что начало последовательности предшествует ее концу. Таким образом, i предшествует j в том и только в том случае, если найдется путь с началом i и концом j.

Такое отношение, которое мы дополним условием «каждый элемент предшествует себе самому», называется *квазипорядком.*

Элементы i и j, каждый из которых предшествует другому, называются *эквивалентными.* Легко убедиться в том, что определенное так отношение эквивалентности обладает свойствами рефлексивности, симметричности и транзитивности. Это отношение порождает разбиение M на множества эквивалентных вершин — *классы эквивалентности.*

Каждый подграф, порожденный классом эквивалентности, по определению вполне связен. Нетрудно убедиться в том, что каждый класс эквивалентности — это максимальный вполне связный подграф, т. е. его нельзя расширить с сохранением свойства вполне связности.

При рассмотрении понятий предшествования и эквивалентности естественным образом наряду с исходным графом $\langle M, N, T\rangle$ появляются некоторые новые графы.

Граф $\langle M, \bar{N}, \bar{T}\rangle$ называется *транзитивным замыканием* графа $\langle M, N, T\rangle$, если каждой дуге в нем соответствует путь в графе $\langle M, N, T\rangle$, имеющий те же начало и конец, а каждому такому пути в свою очередь соответствует дуга в $\langle M, \bar{N}, \bar{T}\rangle$. Граф, являющийся своим транзитивным замыканием, называется *транзитивным.* Граф $\langle \tilde{M}, \tilde{N}, \tilde{T}\rangle$ на-

зывается *редукцией* графа $\langle M, N, T\rangle$, если каждой его вершине $\tilde{i}$ взаимно однозначно сопоставлен класс эквивалентности $M(\tilde{i})$ графа $\langle M, N, T\rangle$, а каждому непустому множеству

$$\{u \mid i(u) \in M(\tilde{i}_1),\quad j(u) \in M(\tilde{i}_2),\quad j(u) \notin M(\tilde{i}_1)\}$$

также взаимно однозначно сопоставлена дуга $\tilde{u} \in \tilde{N}$ с началом $\tilde{i}_1$ и концом $\tilde{i}_2$. Те же термины будут обозначать построение транзитивного замыкания и редукции.

Предложение 1. *Транзитивное замыкание редукции графа есть редукция его транзитивного замыкания.*

Доказательство осуществляется непосредственной проверкой.

Предложение 2. *Число компонент связности при транзитивном замыкании и при редукции графа не изменяется.*

Вершины i и j называются *несравнимыми,* если ни одна из них не предшествует другой.

Граф, не имеющий несравнимых вершин, называется *линейно упорядоченным.*

Предложение 3. *При транзитивном замыкании графа несравнимые вершины остаются несравнимыми, при редукции несравнимыми вершинам соответствуют несравнимые вершины.*

Предложение 4. *В редукции графа $\langle M, N, T\rangle$ контуров не содержится. Если граф $\langle M, N, T\rangle$ сам не содержит контуров, то при редукции он не изменяется.*

В вычислительном отношении представляет интерес построение транзитивного замыкания графа, его редукция, нахождение максимального набора несравнимых вершин графа. Эти задачи и некоторые их обобщения будут рассматриваться в следующих параграфах.

Здесь мы ограничимся лишь изложением простого метода поиска контуров в графе, так как редукция графа сводится к стягиванию каждого контура графа в одну вершину. Эта операция представляет некоторые сложности в удобном представлении необходимой информации, а для дальнейшего содержания книги большого интереса не представляет. В то же время описываемый ниже алгоритм поиска контуров используется в некоторых практических задачах для про-

верки правильности построенного графа (там, где по смыслу задачи этот граф контуров содержать не должен).

Метод заключается в том, что произвольно выбирается вершина $i_1 \in M$ и начинает строиться путь, идущий из i_1. Для этого произвольно выбирается дуга из N_{i_1}, конец которой обозначается через i_2, выбирается дуга из N_{i_2} и т. д. Процесс повторяется до тех пор, пока путь не придет в вершину, которая уже встретилась в пути, — это означает, что путь замкнулся и найден контур, либо пока путь не зайдет в «тупиковую» вершину, из которой не выходит ни одна дуга. В последнем случае дугу, которая завела в этот тупик, следует удалить из N, а путь продолжить с предпоследнего шага (если нам понадобится хранить путь, встречающийся на различных шагах процесса, для этого пути требуется магазинная организация информации, упоминавшаяся в § 4 гл. 1). В том случае, если в пути уже не содержится ни одна дуга и i_1 стала тупиковой вершиной, нужно отбросить эту вершину и поискать в графе новую вершину. Процесс продолжается до тех пор, пока не будет предъявлен контур или пока не будет исчерпан весь граф.

Для описания вычислительного процесса мы зададим граф, как в § 2 гл. 3, двумя массивами:

$$\textbf{integer array } lst[0{:}m],\ j[1{:}n]$$

Рабочая информация о состоянии вычислительного процесса (запись) состоит из двух массивов:

$$\textbf{integer array } fst,\ prec[1{:}m]$$

с начальными значениями

$$fst[i] := lst[i-1]+1, \qquad i \in 1{:}m, \qquad prec[i] = 0$$

и целого числа k с начальным значением 0.

В каждый момент $fst[i]$ будет номером первой невычеркнутой дуги, выходящей из i. Именно эта дуга и будет выбираться для продолжения пути. Массив $prec[1:m]$ будет использован для нескольких целей сразу. Для вершин пути он будет указывать номер предшествующей вершины (кроме начальной вершины, указывающей на себя саму). Равенство $prec[i]$ нулю будет означать, что вершина i в пути еще не встречалась.

Мы ограничимся описанием этого алгоритма на алголе-60:

```
mksearch: for i:=k+1 step 1 until m do
    if fst[i]≤lst [i] then
      begin k:=prec[i]:=i; go to mk end;
    go to mkfin;
mk: i:=j[fst[k]]; if prec[i]>0 then go to mkcontour;
    if fst[i]≤lst[i] then
      begin prec[i]:=k; k:=i; go to mk end;
mk1: fst[k]:=fst[k]+1;
    if fst[k]≤lst[k] then go to mk;
    if prec[k]=k then begin prec[k]:=0;
                            go to mksearch end;
    i:=prec[k]; prec[k]:=0; k:=i; go to mk1;
mkcontour: ⟨контур найден⟩;
mkfin: ⟨контуров нет⟩
```

Непосредственно после метки *mksearch* разыскивается вершина k, не являющаяся тупиковой, для того чтобы с нее начать поиск пути. Добавление к пути новой дуги осуществляется после метки *mk* с выходом на метку *mkcontour*, если был найден контур. Если же контур не найден, то происходит прикрепление найденной дуги к пути и переход в следующую вершину. После метки *mk*1 проверяется, не тупиковая ли это вершина, и если она оказывается тупиковой, то делается шаг назад. Процесс выходит на метку *mkfin*, если в графе не оказалось контуров.

На алголе-68 здесь было бы удобно использовать задание графа в виде списка вершин, каждая из которых имеет список дуг. Разумеется в этом случае при каждой вершине было бы необходимо хранить ссылку и на текущую дугу этого списка.

§ 2. Матрица кратчайших расстояний

В этом параграфе мы займемся вопросом о построении транзитивного замыкания графа. Поскольку элементы k-й степени матрицы смежностей графа равны числу путей, соединяющих соответствующие вершины графа, и путь от одной вершины до другой (если он существует) содержит не более $|M|$ вершин, то для того чтобы существовал путь, ведущий от вершины i к вершине j, необходимо и

достаточно, чтобы был положителен элемент $s[i, j]$ матрицы

$$s[M, M] = r[M, M] + r^2[M, M] + \ldots + r^{|M|-1}[M, M].$$

Таким образом, построение матрицы смежностей транзитивного замыкания графа $\langle M, N\rangle$ формально просто. Более интересной и общей является задача нахождения наилучших путей.

Пусть задан положительный вектор $c[N]$, компоненты которого мы будем рассматривать как длины соответствующих дуг. *Длиной пути* назовем сумму длин входящих в него дуг. Рассмотрим задачу о нахождении для каждой пары i, j кратчайшего пути, ведущего из i в j.

Обозначим через $f^k[M, M]$ матрицу кратчайших путей, состоящих не более чем из k дуг. (Если ни одного пути такой протяженности, ведущего из i в j, не существует, можно считать, что $f^k[i, j] = \infty$.) Очевидно, что

$$f^1[i, j] = \min\{c[u] \mid u \in N_i^- \cap N_j^+\}, \quad f_1[i, i] = 0 \quad (1)$$

(минимум на пустом множестве по определению равен ∞). Далее, легко показать, что для любых k и l

$$f^{k+l}[i, j] = \min\{f^k[i, i_1] + f^l[i_1, j] \mid i_1 \in M\}. \quad (2)$$

Матрица кратчайших расстояний $f[M, M]$ совпадает, очевидно, с $f^{|M|-1}[M, M]$, и путь из i в j существует в том и только в том случае, если $f[i, j] < \infty$. Произведение расчетов по формуле (2) требует порядка $|M|^3$ вычислений. Вычисление переходами от f^k к f^{k+1} может потребовать $|M| - 1$ шагов и общая трудоемкость процесса будет оцениваться порядком $|M|^4$.

Можно вычислять f^k для k, пробегающих целые степени 2, до тех пор, пока мы не дойдем до $k \geqslant |M|$ (алгоритм Беллмана — Шимбела). Трудоемкость такого процесса порядка $|M|^3 \log_2 |M|$ (следовало бы округлить второй сомножитель до ближайшего целого в большую сторону). Каждый из этих вариантов алгоритма требует памяти в $2 \times |M|^2$ чисел.

Тем интереснее, что существует алгоритм, требующий вдвое меньшей памяти и всего $|M|^3$ операций.

Этот алгоритм (алгоритм Флойда) состоит из $|M|$ шагов, выполняемых последовательно для каждого $k \in M$. Рассматривается матрица расстояний $f[M, M]$, первоначально

совпадающая с определенной выше матрицей $f^1[M, M]$. На каждом шаге (соответствующем вершине k) просматриваются все пары $i, j \in M$, значение $f[i, j]$ сравнивается с $f[i, k] + f[k, j]$ и в случае, если сумма оказалась меньше, $f[i, j]$ полагается равным этой сумме.

На алголе-60 мы можем (приняв $M = 1 : m$) записать этот алгоритм одним оператором:

```
for k:=1 step 1 until m do
for i:=1 step 1 until m do
for j:=1 step 1 until m do
   if f[i, j]>f[i, k]+f[k, j] then
   f[i, j]:=f[i, k]+f[k, j]
```

По окончании этого процесса (как будет доказано ниже) матрица f будет матрицей кратчайших расстояний. Прежде чем проводить доказательство, рассмотрим пример Пусть мы имеем матрицу, представленную табл. 1.

Решение представляется в виде набора из шести матриц (табл. 2—7) (изменившиеся элементы выделены).

Рассмотрим в этом примере, как происходит вычисление длины кратчайшего пути от 5 к 2. Этот путь состоит из переходов $5 \to 3 \to 6 \to 2$.

Таблица 1

	1	2	3	4	5	6
1	0	3	*gz*	3	6	*gz*
2	*gz*	0	4	7	*gz*	4
3	3	8	0	5	*gz*	2
4	*gz*	6	*gz*	0	3	*gz*
5	7	*gz*	1	4	0	4
6	5	2	*gz*	*gz*	2	0

Таблица 2

	1	2	3	4	5	6
1	0	3	*gz*	3	6	*gz*
2	*gz*	0	4	7	*gz*	4
3	3	*6*	0	5	*9*	2
4	*gz*	6	*gz*	0	3	*gz*
5	7	*10*	1	4	0	4
6	5	2	*gz*	*8*	2	0

Таблица 3

	1	2	3	4	5	6
1	0	3	*7*	3	6	*7*
2	*gz*	0	4	7	*gz*	4
3	3	6	0	5	9	2
4	*gz*	6	*10*	0	3	*10*
5	7	10	1	4	0	4
6	5	2	*6*	8	2	0

Таблица 4

	1	2	3	4	5	6
1	0	3	7	3	6	7
2	*7*	0	4	7	*13*	4
3	3	6	0	5	9	2
4	*13*	6	10	0	3	10
5	*4*	*7*	1	4	0	*3*
6	5	2	6	8	2	0

Таблица 5

	1	2	3	4	5	6
1	0	3	7	3	6	7
2	7	0	4	7	*10*	4
3	3	6	0	5	*8*	2
4	13	6	10	0	3	10
5	4	7	1	4	0	3
6	5	2	6	8	2	0

Таблица 6

	1	2	3	4	5	6
1	0	3	7	3	6	7
2	7	0	4	7	10	4
3	3	6	0	5	8	5
4	*7*	6	*4*	0	3	*6*
5	4	7	1	4	0	3
6	5	2	*3*	*6*	2	0

Таблица 7

	1	2	3	4	5	6
1	0	3	7	3	6	7
2	7	0	4	7	*6*	4
3	3	*4*	0	5	*4*	2
4	7	6	4	0	3	6
5	4	*5*	1	4	0	3
6	5	2	3	6	2	0

Во время третьего шага непосредственный переход $5 \to 6$ сравнивается по стоимости с $5 \to 3 \to 6$, и, так как второй вариант оказывается дешевле, в табл. 4 и в последующих таблицах длина перехода $5 \to 6$ уже уменьшена. В связи с этим оптимальным стал уже и путь $5 \to 6 \to 2$. На шестом шаге непосредственный переход $5 \to 2$ сравнивается по стоимости с $5 \to 6 \to 2$, и, так как второй вариант оказывается дешевле, в табл. 7 длина перехода $5 \to 2$ уже равна длине перехода $5 \to 3 \to 6 \to 2$ в табл. 1.

Теорема 1. *Матрица f, получаемая в результате работы алгоритма Флойда, является матрицей кратчайших длин.*

Доказательство. Пусть кратчайший путь, идущий из вершины i в вершину j, проходит последовательно через вершины $i_1, i_2, \ldots, i_s$ Пусть k_1 — первый шаг алгоритма, соответствующий какой-либо из этих вершин (назовем ее вершиной i_l; пусть также $i = i_0$, $j = i_{s+1}$). Непосредственный переход от i_{l-1} до i_{l+1} не может быть короче, чем переход от i_{l-1} к i_{l+1} через i_l, так как в противном случае путь $i_0, \ldots, i_{l-1}, i_{l+1}, \ldots, i_{s+1}$ был бы короче кратчайшего. После изменения длины перехода от i_{l-1} до i_l (шага k_1)

путь $i_0, \ldots, i_{t-1}, i_{t+1}, \ldots, i_{s+1}$ также становится кратчайшим. Повторяя этот процесс, мы установим, что в конце и путь $i \to j$ является кратчайшим. ▲

Отметим, что по тому же принципу можно решать задачи о наилучшем пути с некоторыми другими целевыми функциями. Например, если каждой дуге u сопоставлена ее пропускная способность $d[u]$ и требуется найти путь из i в j с наибольшей пропускной способностью, которая понимается как минимум из пропускных способностей дуг, входящих в этот путь, то, обозначив через $g^k[i, j]$ максимальную пропускную способность (не более чем k)-звенных путей из i в j, мы получим

$$g^1[i, j] = \max\{d[u] \mid u \in N_i^- \cap N_j^+\}, \quad g^1[i, i] = gz, \quad (3)$$

$$g^{k+l}[i, j] = \max\{\min\{g^k[i, i_1], g^l[i_1, j]\} \mid i_1 \in M\}. \quad (4)$$

Легко видеть, что алгоритм Флойда применим и здесь:

```
for k:=1 step 1 until m do
for i:=1 step 1 until m do
for j:=1 step 1 until m do
  if g[i, j]<g[i, k] ∧ g[i, j]<g[k, j] then
    g[i, j]:=if g[i, k]<g[k, j]
                then g[i, k] else g[k, j]
```

Методы построения кратчайших длин и путей (для построения путей нужно запомнить еще одну матрицу — матрицу ближайших переходов) не эффективны, если $|M|$ большое число. В этом случае приходится строить отдельные строки матрицы $f[M, M]$. Для их построения разработано много алгоритмов, которые мы разберем в следующем параграфе.

§ 3. Дерево кратчайших путей

Зафиксируем какую-либо вершину i_0 графа $\langle M, N\rangle$ и будем рассматривать задачу нахождения соответствующей этой вершине строки матрицы f. Поскольку речь будет идти только об этой строке, обозначим ее через $\varphi[M]$. Пусть, кроме того, $\varphi^k[j]$ — длина кратчайшего из путей от i_0 до j, состоящих не более чем из k дуг.

Из (2) следует, что

$$\varphi^k[j] = \min\{\varphi^{k-1}[i] + f^1[i, j] \mid i \in M\} \quad (5)$$

или

$$\varphi^k [j] = \min \{\varphi^{k-1} [i (u)] + c [u] \mid u \in N_j^+\}. \qquad (6)$$

Векторы $\varphi^k [M]$ вычисляются по рекуррентному соотношению (6) последовательно, начиная с $\varphi^1 [M] = f^1 [i_0, M]$ либо с вектора $\varphi^0 [M]$, все компоненты которого, кроме $\varphi^0 [i_0] = 0$, полагаются равными ∞. Эти вычисления достаточно проводить до тех пор, пока не совпадут два последовательных набора $\varphi^k [M]$.

Заметим, что в зависимости от того, как расположена информация о графе, можно использовать различные варианты организации вычислений по соотношениям (6). В частности, если для каждого i мы легко можем перебрать дуги из N_i^+, процесс вычислений таков: для каждого k и для каждого $i \in M$ выбирается дуга из N_i^+ с наименьшим значением $\varphi^{k-1} [i (u)] + c [u]$, которое и запоминается в качестве $\varphi^k [i]$.

Другой вариант вычислений таков: в качестве начального приближения для $\varphi^k [M]$ принимается $\varphi^{k-1} [M]$. Последовательно перебираются все дуги, и если для дуги u имеет место неравенство $\varphi^k [j (u)] > \varphi^k [i [u)] + c [u]$, то правая часть этого неравенства принимается в качестве нового значения $\varphi^k [j (u)]$.

Однако возможно и дальнейшее улучшение вычислительной схемы. Оказывается, нет необходимости запоминать отдельно φ^k и φ^{k-1}, а можно при вычислениях ограничиться одним вектором, проводя описанные пересчеты до тех пор, пока при полном просмотре дуг графа происходят изменения.

Соответствующий алгоритм может быть описан следующим образом. Пусть $f_i [M]$ — массив *кратчайших вычисленных длин*, $u1 [M]$ — массив имен дуг, на которых достигается минимум при вычислении, r — переменная, регистрирующая номер последней дуги, вызвавшей изменения в массиве $f_i [M]$. Начальные значения: $f_i [i0] = 0$, $f_i [M - i_0] = gz \times 1 [M - i0]$, $u1 [M] = (n + 1) \times 1 [M]$, $r = 0$. (Массив $u1$ введен для регистрации решений — оптимальных путей от $i0$ до каждой вершины. После того как для каждой вершины зарегистрирована ведущая в нее дуга, набор этих дуг порождает частичный граф, который связен, так как для каждой вершины имеется путь, ведущий в нее из $i0$, — отметим, что начальным присвоением

$u1$ $[M]$ мы неявно ввели искусственные дуги, ведущие из $i0$ в каждую вершину. Очевидно, что на каждом шаге этот граф является деревом.)

```
mk: for u:=r+1 step 1 until n,
                1 step 1 until r-1 do
    if fi[i[u]]+c[u]<fi[j[u]] then
     begin fi[j[u]]:=fi[i[u]]+c[u]; r:=u;
                                  go to mk end
```

Пересчет r обеспечивает здесь необходимое продолжение проверок и изменений до требуемой стабилизации.

Прежде чем доказывать сходимость этого метода, рассмотрим небольшой пример:

Пусть граф задан списком дуг, в котором каждая дуга задана началом, концом и длиной. Составим таблицу, в которой дугам соответствуют столбцы:

имя дуги	А	Б	В	Г	Д	Е	Ж	З	И	К	Л	М	Н	О	П	Р
имя начала	1	2	3	4	5	1	1	2	3	4	4	5	5	4	2	1
имя конца	3	4	2	1	1	5	2	3	5	3	6	6	3	2	1	6
длина дуги	6	3	4	7	8	2	4	1	7	6	5	4	3	4	5	8

Вычисления будут проводиться в таблице, каждая строка которой соответствует вершине графа. Кроме номера вершины, в такой строке содержится пара из вычисленной

Таблица 8

1	0/X
2	gz/X
3	gz/X
4	gz/X
5	gz/X
6	gz/X

Таблица 9

1	0/X		
2	gz/X,	10/В,	4/Ж
3	gz/X,	6/А,	5/З
4	gz/X		
5	gz/X,	2/Е	
6	gz/X,	6/М	

Таблица 10

1	0/X	
2	4/Ж	
3	5/З	
4	gz/X,	6/Б
5	2/Е	
6	6/М	

длины и (через косую черту) дуги, на которой достигается минимум. Табл. 8 представляет начальное состояние вычислений с $i_0 = 1$ (X использовано для искусственных дуг). Первый просмотр дает табл. 9 (в каждой строке последовательно пишутся новые значения изменяемых пар), второй просмотр — табл. 10. Дополнительно ко второму просмотру достаточно проверить, что дуга a (часть списка от начала до последнего изменения при втором просмотре)

изменений не приносит. На рис. 42 изображен граф, составленный из дуг, перечисленных в табл. 10. Это дерево называется *деревом кратчайших путей.*

Сходимость алгоритма мы будем доказывать для случая $c[N] \geqslant 0$ $[N]$.

Теорема 2. *Описанный выше алгоритм сходится к оптимальному решению за конечное число шагов.*

Доказательство. Обозначим через $\varphi^k[M]$ значение $\varphi[M]$ после k-го просмотра дуг, а через $\varphi^0[M]$ — начальное значение $\varphi[M]$. Покажем прежде всего, что последовательность $\{\varphi^k[M]\}$ монотонно не возрастает. Для этого покажем, что если $g[M] \leqslant h[M]$, то после применения одного цикла алгоритма к этим функциям неравенство будет иметь место и для получившихся функций.

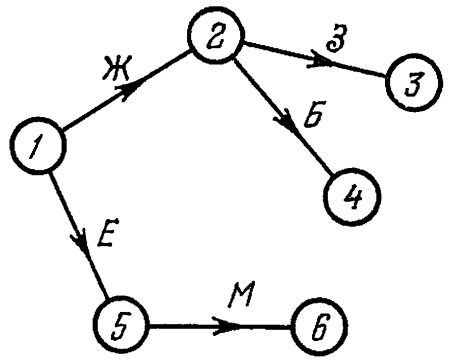

Рис. 42. Дерево кратчайших путей.

Покажем, что неравенство сохраняется при изменении, соответствующем любой дуге u. Это изменение касается разве лишь $g[j(u)]$ и $h[j(u)]$, новыми значениями которых становятся $\min\{g[j(u)],\ g[i(u)] + c[u]\}$ и $\min\{h[j(u)],\ h[i(u)] + c[u]\}$ соответственно. Очевидно, что ввиду справедливости неравенства для старых значений оно сохраняется и для новых значений.

Далее, очевидно, что $\varphi^1[M] \geqslant \varphi^0[M]$. Предположим, что $\varphi^k[M] \leqslant \varphi^{k-1}[M]$. Полагая $g[M] = \varphi^k[M]$ и $h^k[M] = \varphi^{k-1}[M]$, мы по доказанному получим $\varphi^{k+1}[M] \leqslant \varphi^k[M]$ и, следовательно, $\{\varphi^k[M]\}$ монотонно не возрастает.

Вместе с тем эта последовательность ограничена снизу (например, нулем). Следовательно, существует

$$\bar{\varphi}[M] = \lim_{k\to\infty}\varphi^k[M].$$

Покажем, что эта функция удовлетворяет соотношению

$$\bar{\varphi}[i] = \min\{\bar{\varphi}[i(u)] + c[u] \mid u \in N_i^+\}. \qquad (7)$$

Действительно, для любого k мы имеем

$$\varphi^k[i] \leqslant \varphi^{k-1}[i(u)] + c[u], \quad u \in N_i^+$$

и, следовательно,

$$\bar{\varphi}[i] \leqslant \min\{\bar{\varphi}[i(u)] + c[u] \mid u \in N_i^+\}. \qquad (8)$$

Каждому k как мы уже говорили, соответствует некоторое дерево. Так как в графе $\langle M, N\rangle$ имеется лишь конечное число деревьев, то в бесконечной последовательности деревьев найдется дерево, встречающееся бесконечное число раз. Пусть $\bar{u}\,[M]$ — набор дуг, соответствующий этому дереву Для шагов k, при которых набор $u^k\,[M]$ совпадает с $\bar{u}\,[M]$, для любого $i1 \neq i0$ мы имеем

$$\varphi^k\,[i\,(\bar{u}\,[i1])] + c\,[\bar{u}\,[i1]] \leqslant \varphi^k\,[i1] \leqslant \leqslant \varphi^{k-1}\,[i\,(\bar{u}\,[i1])] + c\,[\bar{u}[i1]]$$

Переходя к пределу, получаем

$$\bar{\varphi}\,[i1] = c\,[\bar{u}\,[i1]] + \bar{\varphi}\,[i\,(\bar{u}\,[i1])], \qquad (9)$$

что вместе с (8) дает (7).

Осталось показать, что последовательность $\{\varphi^k\,[M]\}$ сходится к $\bar{\varphi}\,[M]$ за конечное число шагов. Для этого разобьем все вершины графа $\langle M, N\rangle$ на уровни, отнеся к нулевому уровню вершину $i0$, а к уровню r все вершины, путь до которых от $i0$ состоит в дереве $\bar{u}$ из r дуг.

Покажем, что при r-м появлении дерева $\bar{u}$ в последовательности $\{u^k\,[M]\}$ вычисленные для вершин r-го уровня значения φ^k совпадут с $\bar{\varphi}$. Для $i0$ это очевидно. Пусть это верно для уровня $r-1$. Последующие изменения вектора φ вершин $(r-1)$-го уровня не коснутся ввиду того, что $\varphi\,[M]$ достигло на них своей нижней границы. Поэтому при r-м появлении дерева $\bar{u}$ в вершинах r-го уровня произойдет вычисление $\varphi^k\,[M]$ по формуле (9).

Таким образом, если s — максимальный уровень вершин, то после s-го появления графа $\bar{u}$ мы получим окончательно $\bar{\varphi}\,[M]$. ▲

Деревья, значения функции $\bar{\varphi}\,[M]$, соотношения (8) и (9) напоминают нам понятия и конструкции, возникавшие при изучении транспортной задачи на сети. Это не случайно — можно построить сетевую транспортную задачу, решение которой как раз и будет давать дерево кратчайших путей.

Рассмотрим транспортную задачу на графе $\langle M, N\rangle$ с вектором стоимостей $c\,[N]$ и вектором потребностей $b\,[M]$, все компоненты которого равны 1, кроме $b\,[i0] = 1 - |M|$ (для баланса). Оптимальному базисному решению этой задачи соответствует дерево $\langle M, N'\rangle$. Легко видеть, что

для существования допустимого решения необходимо, чтобы в $\langle M, N' \rangle$ существовал путь из $i0$ в любую другую вершину графа. Легко видеть также, что этот путь для оптимальности решения должен быть кратчайшим. Наконец, потенциалы $v[M]$ для базисных дуг должны удовлетворять равенству

$$v[j(u)] = v[i(u)] + c[u],$$

а для остальных дуг неравенству

$$v[j(u)] \leqslant v[i(u)] + c[u], \qquad (10)$$

откуда следует, что

$$v[j] = \min\{v[i(u)] + c[u] \mid u \in N\overline{j}\},$$

т. е. при $v[i0] = 0$ вектор потенциалов является вектором кратчайших длин.

Метод потенциалов для задачи о кратчайшем пути несколько преобразуется. Построение потоков по базису в данном случае лишено смысла. Потенциалы считаются, как и раньше, по дереву, но в каждом случае вычисляется потенциал конца дуги по потенциалу начала. Вводимая в базис дуга (на которой нарушается условие (10)) порождает цикл, состоящий из двух различно ориентированных путей. В отрицательно ориентированном пути потоки по дугам убывают по мере их удаления от $i0$, поэтому дугой с минимальным потоком всегда является последняя дуга отрицательно ориентированного пути.

Таким образом, изменение базиса заключается в том, что для некоторой вершины (именно для $j(\bar{u})$ — конца вводимой в базис дуги) изменяется «прикрепляющая» дуга в точности так же, как это описано для метода рекуррентных соотношений. Отличие же от метода рекуррентных соотношений в том, что при каждом изменении базиса пересчитывается вся система потенциалов, для чего у всех вершин, следующих за $j(\bar{u})$, потенциал уменьшается на одну и ту же величину

$$v[j(\bar{u})] - v[i(\bar{u})] - c[\bar{u}].$$

Отметим, что ни один из описанных методов не требует положительности вектора $c[N]$, требуется лишь, чтобы существовал путь минимальной «длины», понимаемый как сумма «длин» входящих в него дуг. Так как допустимое

решение в случае искусственных дуг всегда существует, то для существования оптимального решения необходимо и достаточно, чтобы в графе не было контуров отрицательной «длины» Задачи, для которых это замечание существенно, встретятся в § 4

Для случая положительного вектора $c[N]$ следует отметить одну особенность метода потенциалов: он получается особенно простым, если начинать решение с искусственного дерева и на каждом шаге включать в базис дугу с наибольшей разностью.

Алгоритм выглядит следующим образом: множество M разделяется на два подмножества; M_0 — вершины, кратчайшее расстояние до которых уже вычислено, M' — остальные вершины. Первоначально $M_0 = \{i_0\}$, $M' = M - i_0$.

1°. Вершинам из M' приписывается расстояние $f[i]$, равное $f^1[i_0, i]$, где f^1 определяется по формуле (1)

2° В M' разыскивается вершина i_1 с минимальным значением $f[i]$ и включается в M_0.

3° Для оставшихся вершин из M' значения $f[i]$ пересчитываются по формуле

$$f[i] := \min\{f[i], f[i_1] + f^1[i_1, i]\}.$$

Этот пересчет удобно производить «счетом от себя»: для всех $u \in N^-_{i_1}$

$$f[j(u)] := \min\{f[j(u)], f[i_1] + o[u]\}.$$

4°. В случае, если M' не пусто, перейти к 2°, если пусто, завершить вычисления

Этот алгоритм называется алгоритмом Дейкстры. Его отличительная особенность — окончательность решений о длинах кратчайших путей до вершин из M_0. Алгоритм Дейкстры в настоящее время считается самым эффективным Имеется много вычислительных разновидностей этого метода, рассмотрим здесь одну из них, которая, хотя мы ее и опишем в терминах алгола-60, будет по используемой технике представления данных находиться вполне на уровне алгола-68

Следуя Дейкстре, разобьем множество M' на два подмножества: M — множество вершин, до которых расстояние уже начало вычисляться, M_2 — все остальные вершины Вершины из M_1 соединим в цепной список,

упорядочив их по возрастанию $f[j]$, и для каждой вершины i укажем предшествующую $pre[i]$ и следующую $next[i]$.

Благодаря этому упорядочению значительно упрощается поиск в п. 2°. Вычисления п. 3° могут проводиться отдельно для каждой дуги $u \in N_{i_1}^-$. Их можно разбить на такие действия:

3.1° Если $j(u) \in M_0 \vee j(u) \in M_1 \wedge f[j(u)] \leqslant f[i_1] + c[u]$, никаких изменений не делать и перейти к п. 3.4°.

3.2° $$f[j(u)] := f[i_1] + c[u].$$

Если $j(u) \in M_1$, то исключить $j(u)$ из цепи (ср. процедуру *exclel* из гл. 1):

$$next[pre[j(u)]] := next[j(u)];$$

$$pre[next[j(u)]] := pre[j(u)]$$

3.3° Найти в цепи k, для которого

$$f[pre[k]] \leqslant f[j(u)] \leqslant f[k],$$

и вставить $j(u)$ в цепь между $pre[k]$ и k:

$$next[pre[k]] := j(u); \qquad pre[j(u)] := pre[k];$$
$$next[j(u)] := k; \qquad pre[k] := j(u).$$

После этих действий перейти к п. 3.4°.

3.4° Если в $N_{i_1}^-$ есть еще дуги, вернуться к п. 3.1° для обработки следующей дуги.

Многократный поиск k в цепном списке несколько облегчается, если при задании графа расположить дуги, входящие в каждое N_i^-, в порядке возрастания $c[u]$ В этом случае каждая следующая из вершин $j(u)$ должна располагаться в цепном списке (если только ее заново включают в список) дальше предыдущей, и на каждой итерации цепной список просматривается только один раз.

Таким образом, общая трудоемкость метода Дейкстры оценивается величиной порядка $s \times |M| + |N|$, где s — максимальная длина цепного списка в ходе вычислений (очевидно, что $s \leqslant |M|$).

Теперь соответствующую процедуру на алголе-60 мы приведем без дальнейших комментариев, предупредив только, что $up[i]$ — это дуга, ведущая в i в оптимальном дереве, а массив f должен по техническим причинам иметь $(m+1)$-ю компоненту. Читателю будет полезно сопоставить текст процедуры с изложением алгоритма:

```
procedure dijkstra(m, c, j, lst, i0, f, up); value m, i0;
    integer m, i0; integer array j, lst, up; array c, f;
begin integer i, i1, i2, u; real f1, f2;
    integer array next, prec[1 : m+1];
    for i:=1 step 1 until m do
      begin f[i]:=gz; next[i]:=prec[i]:=up[i]:=0 end;
    f[i0]:=.0; f[m+1]:=gz+gz; prec[m+1]:=i0;
    next[i0]:=m+1;
    i1:=i0;
mkit: if i1=m+1 then go to mkfin; i0:=i1;
    i:=i1:=next[i1]; f1:=f[i0];
    for u:=lst[i0-1]+1 step 1 until lst[i0] do
      begin i2:=j[u]; f2:=f1+c[u];
        if f2<f[i2] then begin f[i2]:=f2; up[i2]:=u;
          if next[i2]≤m then
            begin next[prec[i2]]:=next[i2];
              prec[next[i2]]:=prec[i2] end;
mki:      if f[i]<f2 then begin i:=next[i];
                                        go to mki end;
          next[prec[i]]:=i2; prec[i2]:=prec[i];
          next[i2]:=i; prec[i]:=i2
      end end u;
    go to mkit;
mkfin: end
```

§ 4. Критический путь

В некоторых моделях возникают задачи о нахождении не самого короткого, а, наоборот, самого длинного пути между двумя заданными вершинами графа. Здесь мы опишем метод решения этой задачи и рассмотрим одно из ее важнейших приложений, которое связано с вопросами управления работами над сложными объектами (стройками, проектированием и т. д.). Однако, прежде чем говорить о приложениях, рассмотрим саму задачу.

Задача о максимальном пути.

Задан граф $\langle M, N\rangle$, не имеющий контуров, и каждой дуге u сопоставлено неотрицательное число $t[u]$ — длина этой дуги. Заданы две вершины i_-, $i_+ \in M$. Требуется найти путь из i_- в i_+, имеющий наибольшую длину, где под длиной пути понимается сумма длин входящих в него дуг.

Изменением знаков у $t[u]$ можно свести задачу к задаче о кратчайшем пути, которая в этом случае будет иметь решение, так как $\langle M, N\rangle$ не имеет контуров. Для вычисления решения пригоден любой из алгоритмов, описанных в § 3, кроме алгоритма Дейкстры. Важно отметить, что из результатов § 3 следует, что можно говорить о дереве максимальных путей, которое решает все задачи о максимальном пути с фиксированной начальной вершиной i_-. Заметим еще, что можно было бы говорить и о дереве максимальных путей, входящих в вершину i_+, которое решало бы все задачи с фиксированной конечной вершиной.

Можно, однако, не пользоваться алгоритмами § 3 непосредственно, а видоизменить их для нашей задачи. Если мы ищем дерево максимальных путей, выходящих из i_-, то мы должны ввести вектор $v[M]$ — максимальных расстояний от i до всех вершин и написать для него уравнение

$$\begin{aligned} &v[i] = \max\{t[u] + v[i(u)] \mid u \in N_i^+\}, \\ &v[i_-] = 0. \end{aligned} \tag{11}$$

Величины $v[i]$, так же как и раньше, естественно называть *потенциалами*. Потенциалы вершин можно находить по уровням дерева максимальных путей: если вычислены потенциалы для всех вершин, непосредственно предшествующих i, то по (11) можно вычислить и $v[i]$.

Однако, вместо того чтобы пользоваться формулой (11), можно всякий раз, как будет установлено, что у вершины $v[i]$ потенциал имеет окончательное значение, для всех дуг $u \in N_i^-$ проводить вычисление

$$v[j(u)] := \max\{v[j(u)],\ t[u] + v[i]\}. \tag{12}$$

Таким образом, к тому времени, все как дуги из N_i^+ будут просмотрены, $v[i]$ будет удовлетворять (11).

Полезно для каждой из вершин помнить, сколько дуг, входящих в эту вершину, еще не просмотрено. Когда это число $next[k]$ обращается в нуль, вершину можно включать в число «готовых» для того, чтобы просмотреть выходящие из нее дуги. Эти «готовые» вершины можно хранить с помощью цепного списка, причем поскольку для них $next[k] = 0$, то ссылки на следующие вершины можно хранить в том же массиве *next*, чем и вызвано такое его название.

Так мы получаем следующую процедуру:

```
procedure maxpath(m, n, t, j, lst, i0, v, up); value m, n, i0;
    integer m, n, i0; integer array j, lst, up; array t, v;
begin integer i, i1, k, m1, u; real z;
    integer array next[0:m];
    for i:=1 step 1until m do
      begin v[i]:=-gz; next[i]:=up[i]:=0 end;
    for u:=1 step 1 until n do next[j[u]]:=next[j[u]]+1;
    m1:=m+1; next[0]:=i1:=i0;
mkit: next[i1]:=m1; if next[0]=m1 then go to mkfin;
    i:=next[0]; z:=v[i];
    for u:=lst[i-1]+1 step 1 until lst[i] do
      begin k:=j[u]; next[k]:=next[k]-1;
        if v[k]<z+t[u] then
          begin v[k]:=z+t[u]; up[k]:=u end;
        if next[k]=0 then i1:=next[i1]:=k
      end u;
    next[0]:=next[i]; go to mkit;
mkfin: end maxpath
```

Легко заметить, что эта процедура для «неправильно составленных» графов, в которых есть контуры или не до всех вершин можно добраться из $i_{\_}$, даст неполное решение. Однако вершины, в которых значение потенциала не вычислено, видны по массиву $v[M]$, где им соответствует значение $-gz$.

Теперь обратимся к задачам управления сложными комплексами работ. Граф, который возникает в таких задачах, трактуется следующим образом (он называется *графом работ* или *сетевым графиком*): его дуги соответствуют выполняемым работам. На очередность выполнения этих работ (множество которых N задано) наложены некоторые ограничения предшествования, — для каждой работы u указано множество работ N_u, которые должны быть завершены до того, как начнется выполнение u. Указание множества предшествующих работ является заданием квазипорядка на множестве N. Можно рассматривать транзитивное замыкание этой системы предшествований. Обозначим множество всех работ, предшествующих u, через $\bar{N}_u$.

Представляет интерес и сама задача построения графа по набору предшествований на множестве N. Анализ этой

задачи несколько громоздок, однако без него мы не сможем выяснить, что представляют собой вершины этого графа и как в принципе выглядит алгоритм построения сетевого графика. Нужно, однако, отдавать себе отчет в том, что практические алгоритмы построения сетевого графика выглядят значительно проще.

Будем здесь предполагать, что каждый класс эквивалентности нашего квазипорядка состоит из одной работы. В противном случае система предшествований работ над объектом является противоречивой и должна пересматриваться. Для проверки непротиворечивости системы предшествований может быть использован алгоритм из § 1.

Итак, мы имеем некоторый набор подмножеств $\{\bar{N}_u\}_{u \in N}$ множества N. Поскольку среди $\bar{N}_u$ могут встретиться одинаковые, введем самостоятельную индексацию для *различных* подмножеств этого набора, обозначая их через N_τ, $\tau \in T$. Естественно, что каждому $u \in N$ соответствует однозначно определяемое $\tau \in T$ такое, что $\bar{N}_u = N_\tau$.

Каждое из множеств N_τ задает разбиение $\mathfrak{N}_\tau$ множества N на два подмножества N_τ, $N \setminus N_\tau$. Рассмотрим произведение этих разбиений

$$\mathfrak{N} : N = \bigcup_{\sigma \in S} N^*_\sigma. \tag{13}$$

По определению для любого $\tau \in T$ существует такое $S(\tau) \subset S$, что

$$\bigcup_{\sigma \in S(\tau)} N^*_\sigma = N_\tau. \tag{14}$$

Обозначим через $\sigma(u)$ то σ, для которого $u \in N^*_\sigma$.

Теперь мы можем строить наш граф. Каждому $u \in N$ сопоставим дугу. Вершинами графа мы будем сейчас считать элементы $\tau \in T$ и $\sigma \in S$. Началом дуги u будем считать $\tau(u)$, а концом $\sigma(u)$. Кроме этих основных дуг, введем еще *фиктивные* дуги, идущие (для каждого $\tau \in T$) из всех вершин $S(\tau)$ в τ. Построенный таким образом граф и является собственно полным и в некотором смысле правильным сетевым графиком. Однако в использовании этот граф неудобен, и используется более простой граф, получающийся из него после некоторых редукций.

В результате редукций из графа удаляются некоторые фиктивные дуги и вершины.

Редукция графа основывается на следующих двух правилах:

1. *Если фиктивная дуга идет от вершины* σ *к вершине* τ, *между которыми есть другой путь, эту дугу следует удалить.*

Фиктивные дуги служат для установления обязательных предшествований между вершинами графа. Если это предшествование является следствием других предшествований, фиктивная дуга не нужна.

2. *Если единственная дуга, выходящая из вершины (входящая в вершину), является фиктивной, эту дугу следует удалить и слить ее граничные вершины в одну.*

Мы различаем граничные вершины по необходимости — только из-за того, что из них выходит или в них входит по нескольку дуг.

Если из вершины выходит только одна дуга — и притом фиктивная, — эта вершина не нужна, а значит, не нужна и дуга, которая устанавливает порядок у двух сливаемых в одно целое вершин.

Прежде чем заниматься обоснованием этого процесса, рассмотрим небольшой пример.

Пример. Пусть $N = \{a, b, c, d, e, f, g, h, i, j, k, l\}$. Множества N_u и полученные транзитивным замыканием множества $\overline{N}_u$ выписаны в табл. 11.

Таблица 11

u	N_u	$\overline{N}_u$
a	l	l
b	j	j
c	b	b, j
d	g, l	g, l
e	g	g
f	g	g
g	—	—
h	f, j	f, j
i	a, c, f, g, h, e	$a, c, f, g, h, e, l, b, j$
j	—	—
k	h, e	h, e, f, j, g
l	—	—

Среди множеств $\bar{N}_u$ имеется девять различных. Перенумеруем их числами от 1 до 9:

$$N_1 = \emptyset,\ N_2 = l,\ N_3 = g,\ N_4 = j,\quad N_5 = \{b, j\},$$
$$N_6 = \{g, l\},\ N_7 = \{f, j, g\},$$
$$N_8 = \{h, e, f, j, g\},\ N_9 = \{a, c, f, g, h, e, l, b, j\}.$$

Произведение соответствующих разбиений состоит из восьми множеств, которые мы занумеруем числами из множества 1 : 8,

$$N_1^* = l,\ N_2^* = g,\ N_3^* = j,\ N_4^* = b,\ N_5^* = f,$$
$$N_6^* = \{h, e\},\ N_7^* = \{a, c\},\ N_8^* = \{d, i, k\}.$$

Действуя по описанному правилу, получаем граф, изображенный на рис. 43.

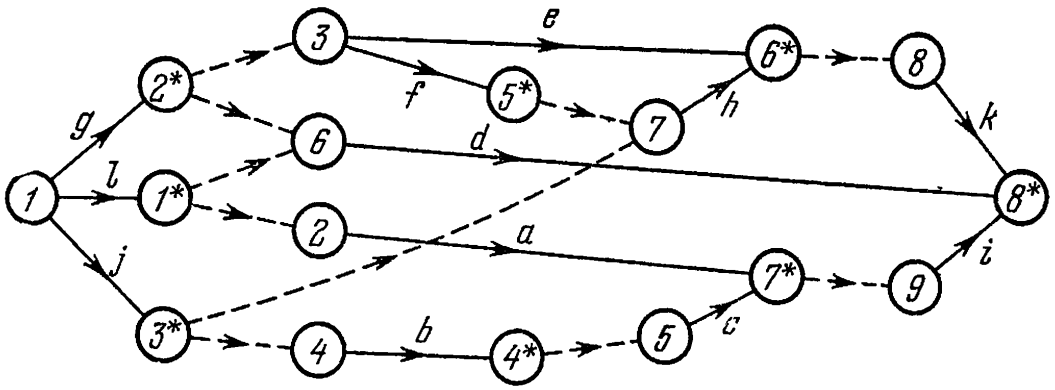

Рис 43. Сетевой график до редукций.

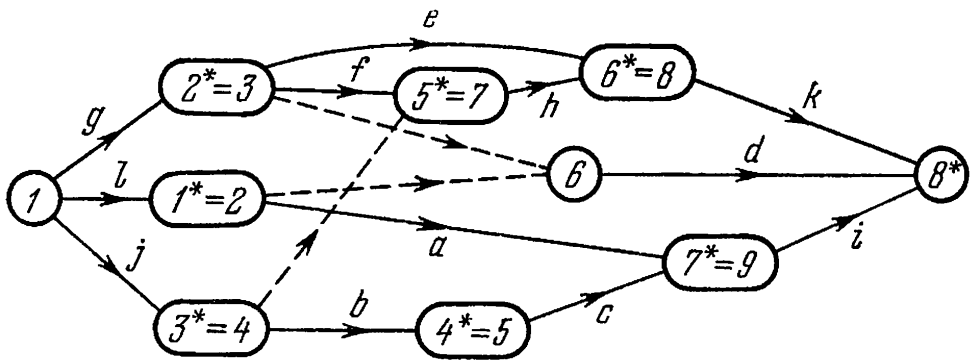

Рис 44 Тот же сетевой график после редукций.

Далее начинается редукция графа, которая приводит к графу, изображенному на рис 44 Фиктивные дуги изображены на этих двух графах штриховыми линиями. В реальных задачах фиктивных дуг обычно оказывается значительно меньше.

Покажем теперь, что последовательность редукций не существенна для окончательного результата

Теорема 3. *Граф, получающийся из графа* $\langle M, N\rangle$ *в результате описанных выше редукций, не зависит от порядка применения этих редукций*

Доказательство Легко видеть, что редукции обоих типов не изменяют свойств связности графа — если до редукции существовал путь из i в j, то после редукции путь из i в j также найдется. Поэтому дуги, которые могут быть удалены по первому правилу, определяются однозначно.

Рассмотрим множество F_1 фиктивных дуг, которые при какой-либо последовательности редукций могут быть удалены по второму правилу. Для каждой такой дуги по определению имеется последовательность предшествующих редукций, которые должны быть проведены (и могут быть проведены) до удаления этой дуги.

Покажем прежде всего, что имеется последовательность редукций, при которой исключаются все дуги из F_1 Взяв одну из дуг, занумеруем предшествующие ей по порядку исключения из графа дуги, после чего присвоим очередной номер ей самой Выберем какую-либо из оставшихся дуг и перенумеруем очередными номерами те из предшествующих дуг, которые еще не получили номеров (а затем и ее самое) Будем повторять этот процесс до полного исчерпания F_1.

По самому построению этой нумерации она дает порядок, в котором фиктивные дуги, принадлежащие F_1, могут исключаться из графа

Пусть при какой-нибудь последовательности редукций удалено некоторое множество $F_0 \subset F_1$. Если $F_1 \setminus F_0$ не пусто, выберем в нем дугу f с минимальным номером. Очевидно, что так как все редукции, предшествующие удалению f, уже сделаны, то f можно удалить Таким образом, никакая последовательность, не исчерпывающая F_1 полностью, не может быть окончательной. ▲

Обоснованность процедуры построения графа может быть проверена следующим образом: если задан некоторый граф, удовлетворяющий всем требованиям, которые предъявляются к сетевому графику, и по этому графу составлена система предпочтений для дуг, то процедура должна восстанавливать этот граф.

К сожалению, описанная процедура этим требованиям не удовлетворяет. На рис. 45 и 46 изображен граф, послуживший оригиналом, и граф, полученный после редукций. Имеющимися редукциями невозможно удалить дугу x.

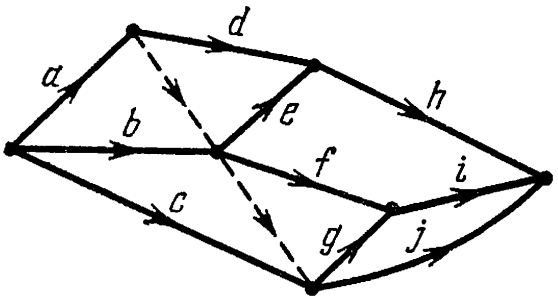

Рис. 45.

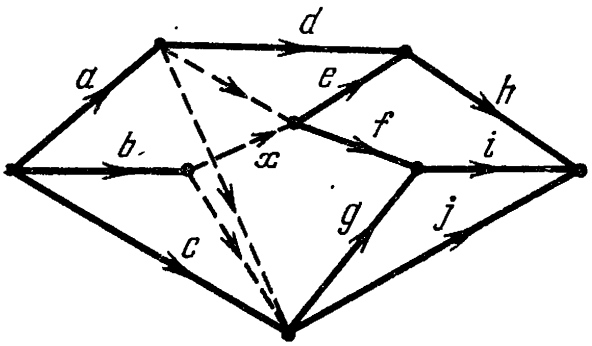

Рис. 46.

Однако удается доказать, что в ограниченном классе случаев эта процедура дает в точности нужный граф.

Теорема 4. *Пусть* $\langle M, N\rangle$ *— граф без фиктивных дуг, не имеющий контуров, причем существуют такие* i_- *и* i_+, *что любая дуга лежит на каком-либо пути из* i_- *в* i_+. *Обозначим через* $\bar{N}_u$ *множество дуг из* N, *предшествующих* u. *Описанная процедура построения графа восстанавливает* $\langle M, N\rangle$ *по семейству подмножеств* $\{\bar{N}_u\}_{u\in N}$.

Доказательство. Транзитивными замыканиями множеств N_u мы здесь ограничиваемся для простоты.

Прежде всего отметим, что в получающемся графе i_- и i_+ определяются однозначно: i_- соответствует элементу $\tau_0 \in T$

$$N_{\tau_0} = \Phi$$

(из i_- выходят дуги, не имеющие предшественников), i_+ соответствует элементу $\sigma_0 \in S$, состоящему из дуг, которые никому не предшествуют

$$N^*_{\sigma_0} = \bigcap_{\tau\in T} (N \setminus N_\tau).$$

В графе, получающемся после построения, будут присутствовать все дуги из N и некоторое множество фиктивных дуг F. Мы должны показать, что все это множество при редукциях будет удалено, а дуги, инцидентные одной вершине, соединятся в ней.

Легко показать, что дуги, выходящие из одной вершины в $\langle M, N\rangle$, будут выходить из одной вершины и в построенном графе, т. е каждой вершине, из которой выходят дуги, соответствует элемент множества T. Действи-

тельно, всем этим дугам предшествует одно и то же множество дуг, а для дуг, выходящих из другой вершины, множество предшественников другое. Аналогично каждой вершине, в которую входят дуги, соответствует элемент $\sigma \in S$. Все дуги, входящие в вершину i, одновременно предшествуют или не предшествуют каждой дуге, поэтому N_i^+ лежит полностью в каждом из N_τ, с которым пересекается. Значит, N_i^+ лежит целиком в одном из элементов разбиения $\mathfrak{N}$. Других дуг в этом элементе быть не может, так как такая дуга $\bar{u}$ предшествовала бы другому набору дуг, а значит, для какого-то u разбиение $\mathfrak{N}_{\tau(u)}$ разделило бы $\bar{u}$ и N_i^+.

Все фиктивные дуги построенного графа можно разделить на два класса — фиктивные дуги, соединяющие вход в вершину и выход из вершины исходного графа, и фиктивные дуги, описывающие ненепосредственные предшествования. Дуги второго типа по самому своему происхождению имеют обходные пути и исключаются по первому правилу. Дуги первого типа (после того как исключены все дуги второго типа) исключаются по второму правилу, так как после этого из каждой вершины выходит не более одной фиктивной дуги. ▲

Сделаем теперь несколько замечаний, относящихся к терминологии и проблематике управления работами с помощью описанной модели.

В сетевом планировании фиктивные дуги, как и настоящие, называются *работами*. Продолжительность настоящих работ считается известной, для фиктивных работ продолжительность принимается равной нулю. Вершины называются *событиями*. Естественно продолжить эту терминологию и считать, что событие наступило, если выполнены все работы, входящие в соответствующее ему множество.

Основная задача сетевого планирования — это задача о нахождении моментов наступления различных событий. Исходными данными в этой задаче являются момент наступления «начального» события и продолжительности работ, которые положительны для настоящих работ и равны 0 для фиктивных. Каждая работа ограничена условием выполнения предшествовавших ей работ. В терминах графа можно сказать, что работа начинается не раньше, чем наступит событие, являющееся ее началом.

Таким образом, для того чтобы найти момент самого раннего наступления события, нужно вычислить максимальную длину путей, ведущих из i_0 в соответствующую вершину. При этом путь максимальной длины от начала проекта i_0 до конца проекта i_+ называется *критическим.*

Ранние начала вычисляются по формуле (11) и о них уже сказано достаточно. По аналогии с ранними началами определяются *поздние моменты* наступления событий — самые поздние сроки наступления событий, при которых общее время выполнения работы не увеличивается. Поздние моменты $w[M]$ удовлетворяют похожему на (11) условию

$$w[i] = \min\{w[j(u)] - t[u] \mid u \in N_i^-\},\ w[i_+] = v[i_+]. \quad (15)$$

Отметим еще, что величина

$$\rho[u] = w[j(u)] - v[i(u)] - t[u] \quad (16)$$

называется *резервом времени* работы u. Дуги, для которых резерв времени равен нулю, называются *критическими.*

Предложение 5. *Для того чтобы дуга была критической, необходимо и достаточно, чтобы она принадлежала какому-нибудь критическому пути.*

Доказательство. Действительно, пусть рассматривается дуга u с началом i и концом j. Рассмотрим самый длинный путь из i_- в i (его длина равна $v[i]$) и самый длинный путь из j в i_+ — (его длина равна $w[i_+] - w[j]$). Очевидно, что для того чтобы $\rho[u] = 0$, необходимо и достаточно, чтобы

$$v[i] + t[u] + w[i_+] - w[j] = v[i_+],$$

т. е. чтобы путь, составленный из u и этих двух путей, был критическим. ▲

Следствие. *В графе $\langle M, N_c\rangle$, составленном из критических дуг, любая дуга лежит на пути из i_- в i_+.*

Таким образом, граф $\langle M, N_c\rangle$ может рассматриваться как самостоятельный сетевой график.

Мы рассмотрели только одну из задач, связанных с организацией работ. Имеется большое число дополнительных проблем, относящихся к той же области, — вероятностные модели, оценки продолжительности работ, составление расписаний при ограниченности числа исполнителей, распределение финансирования по работам.

Один из вариантов этой последней задачи будет рассмотрен в § 7.

§ 5. Кратчайшее дерево

Задача о дереве кратчайших путей естественно наводит на мысль об аналогичной задаче — задаче о построении дерева с минимальной суммой длин дуг.

Итак, пусть задан связный граф $\langle M, N\rangle$ и вектор $c\,[N]$ длин его дуг. Требуется найти $N' \subset N$ так, чтобы граф $\langle M, N'\rangle$ был деревом, а сумма $1\,[N'] \times c\,[N']$ была минимальна.

Для решения этой задачи имеется весьма простой и эффективный алгоритм последовательного построения дерева (похожий на алгоритм, описанный в теореме 1 гл. 3): мы будем рассматривать последовательность деревьев $\langle M', N'\rangle$, увеличивая на каждом шаге M' на одну вершину, а N' — на одну дугу, соединяющую эту вершину с имеющимся деревом. Первоначально мы выберем произвольное $i_0 \in M$ и образуем $M' = \{i_0\}$, $N' = \emptyset$. На каждом шаге процесса, пока $M' \neq M$, из всех дуг, одна граничная вершина которых лежит в M', а другая в $M \setminus M'$, выбирается дуга наименьшей длины. Эта дуга включается в N', а ее начало и конец в M' (что пополняет M' на одну вершину).

Т е о р е м а 5. *Дерево $\langle M, N'\rangle$, полученное в результате работы алгоритма, имеет наименьшую длину.*

Д о к а з а т е л ь с т в о. Пусть $\langle M, N''\rangle$ — некоторое другое дерево. Покажем, что

$$1\,[N'] \times c\,[N'] \leqslant 1\,[N''] \times c\,[N''], \qquad (17)$$

Перенумеруем вершины из M в порядке их включения в M'. Пусть $m\,(i)$ — номер i-й вершины, $m\,(i_0) = 0$. Каждая дуга u из N' имеет номер $n\,(u)$, равный максимуму из номеров граничных вершин.

Найдем в N' дугу с наименьшим номером, не содержащуюся в N'' Обозначим ее через $\bar{u}$, а ее номер через $\tilde{k}$. Добавление этой дуги к N'' порождает в графе $\langle M, N''\rangle$ однозначно определяемый цикл, среди вершин которого есть вершины с номерами, меньшими k, и есть вершины с номерами k и больше. Поэтому в цикле найдется дуга $\tilde{u}$, одна граничная вершина которой имеет номер меньше k, а другая не меньше k. По определению $\bar{u}$ мы имеем $c\,[\tilde{u}] \leqslant c\,[\bar{u}]$, и поэтому замена $\tilde{u}$ на $\bar{u}$ в графе $\langle M, N''\rangle$ не увеличивает длины графа и оставляет его деревом.

Повторяя этот процесс до полного совпадения N'' с N, мы придем к (17), откуда уже и следует утверждение теоремы. ▲

Вычислительная реализация этого алгоритма (алгоритма Прима — Краскала) сложности не представляет. Отметим только, что здесь выгодно использовать те же идеи организации вычислений, которые были в методе Дейкстры.

Важно отдавать себе отчет в том, что задачи с такими простыми алгоритмами — счастливое исключение. Это хорошо видно на этой задаче — достаточно чуть-чуть изменить ее формулировку, и трудность ее существенно изменяется. Вот пример:

Задача о кратчайшем дереве путей.

Найти $N' \subset N$ так, чтобы граф $\langle M, N' \rangle$ был иерархическим деревом с корнем в i_0, а сумма $\mathbf{1}\,[N'] \times c\,[N']$ была минимальна.

Еще один вариант постановки задачи:

Задача о частичном дереве.

Пусть задано непустое подмножество $M_0 \subset M$. Найти частичный связный подграф $\langle M_1, N_1 \rangle$ графа $\langle M, N \rangle$ так, чтобы $M_0 \subset M_1$ и сумма $\mathbf{1}\,[N_1] \times c\,[N_1]$ была минимальна.

Решение первой из этих задач достаточно просто, хотя и существенно отличается от алгоритма Прима — Краскала, о второй задаче и сходных с ней речь пойдет в следующей главе.

Итак, пусть задан граф $\langle M, N \rangle$ и неотрицательный вектор $c\,[N]$. По формуле (1) образуем матрицу $\varphi\,[M, M]$ кратчайших однозвенных переходов от вершины к вершине. Так как в каждую вершину дерева (кроме i_0) должна входить ровно одна дуга, можно пересчитать φ по формуле

$$\varphi\,[M, i] := \varphi\,[M, i] - \mathbf{1}\,[M] \times \min\{\varphi\,[j, i] \mid j \neq i\}.$$

Рассмотрим граф $\langle M, N_0 \rangle$, где $N_0 = \{(i, j) \mid \varphi\,[i, j] = 0, i \neq j\}$. В этом графе может не быть путей из i_0 во все другие вершины. В таком случае в этом графе найдутся контуры. В каждом классе эквивалентности этого графа (в терминологии § 1), как это отмечалось в гл. 3, существует иерархическое дерево с корнем в любой его вершине,

поэтому о переходах в классах эквивалентности уже можно не беспокоиться, и нам нужно построить переходы м е ж д у к л а с с а м и. Проведем редукцию графа $\langle M, N_0\rangle$ и для получившихся вершин положим

$$\varphi[\tilde{i}, \tilde{j}] = \min\{c[u] \mid i(u) \in M_{\tilde{i}},\ j(u) \in M_{\tilde{j}}\}.$$

Для получившейся матрицы повторим процедуру пересчета, построим граф $\langle \tilde{M}, \tilde{N}_0\rangle$ и т. д. до тех пор, пока мы не получим граф, в котором из i_0 можно попасть в любую вершину.

При обратном процессе восстановления графа по его редукции мы будем в каждом классе эквивалентности восстанавливаемого графа строить иерархическое дерево. В качестве корневой вершины этого дерева будет выбираться конец дуги, ведущей в этот класс.

Пусть, например, $M = 1:10$ и $\varphi[M, M]$ задается табл. 12. После пересчета мы получаем табл. 13, соответствующий граф $\langle M, N_0\rangle$ изображен на рис. 47. В этом графе

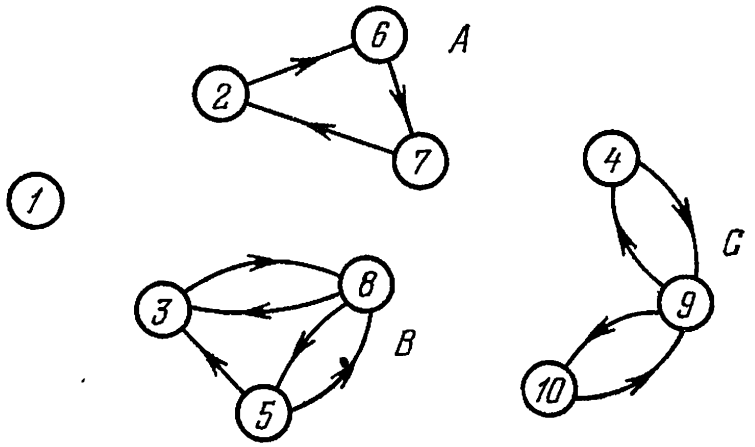

Рис. 47. Граф нулевых переходов первой итерации.

классами эквивалентности являются {2, 6, 7}, {3, 5, 8}, {4, 9, 10}; для них мы получаем матрицу $\varphi[\tilde{M}, \tilde{M}]$, представленную в табл 14. Пересчет дает табл. 15 и рис. 48. В получившемся графе есть путь из *1* во все вершины. При восстановлении получаем граф, изображенный на рис. 49.

Этот алгоритм удобно (с некоторыми изменениями вычислительного характера) описывается на алголе-68. Мы запишем его в виде процедуры, входными параметрами которой будут массивы, задающие информацию о графе, но сначала скажем об изменениях. Граф будет храниться

Таблица 12

	1	2	3	4	5	6	7	8	9	10
1	×			7		8		6	9	
2		×				1	2			6
3			×		3			3		
4		9		×	6	8			3	2
5			2		×	6		3		5
6		7		8		×	1			4
7		2	8			2	×		9	
8			2	5	1		6	×		9
9		6		2			7		×	1
10		6		5	9				3	×

Таблица 13

	1	2	3	4	5	6	7	8	9	10
1	×			5		7		3	6	
2		×	6			0	1			5
3			×		2			0		
4		7		×	5	7			0	1
5			0		×	5		0		4
6		5		6		×	0			3
7		0	6			1	×		6	
8			0	3	0		5	×		8
9		4		0			6		×	0
10		4		3	8				0	×

Таблица 14

	1	*A*	*B*	*C*
1	×	7	3	5
$A = \{2, 6, 7\}$		×	6	3
$B = \{3, 5, 8\}$		5	×	3
$C = \{4, 9, 10\}$		4	5	×

Таблица 15

	1	*A*	*B*	*C*
1	×	3	0	2
A		×	3	0
B		1	×	0
C		0	2	×

в виде набора звезд дуг, входящих в каждую вершину. В каждой такой звезде будет выделено по одной дуге — с минимальной длиной В графе, составленном из выделенных дуг, мы будем пытаться найти путь от i_0 до какой-

нибудь из вершин i'. Если такой путь найдется, то вершина i' и все остальные вершины этого пути выбывают из рассмотрения. Если же в процессе поиска найдется контур, мы его редуцируем, как в § 1, заменяя одной вершиной. Звезды вершин контура объединяются при этом (с удалением появляющихся петель) в одну звезду, и для новой звезды отыскивается новая минимальная дуга.

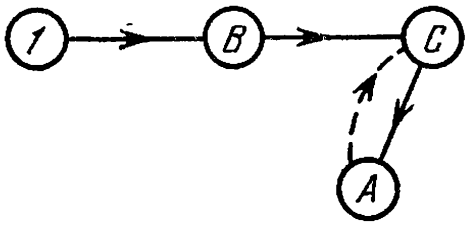

Рис. 48. Граф нулевых переходов второй итерации.

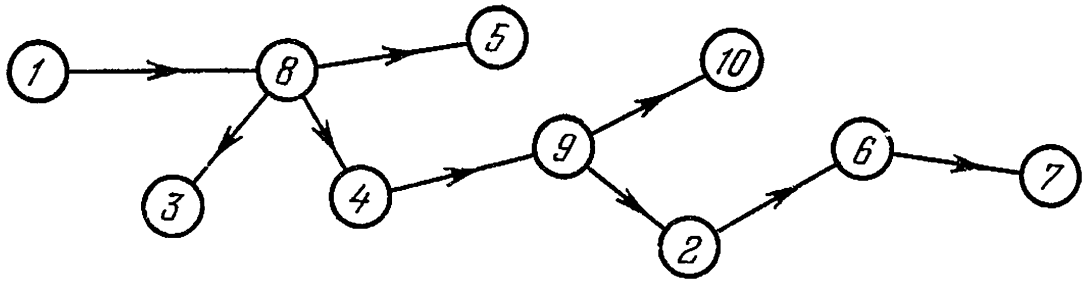

Рис. 49. Кратчайшее иерархическое дерево.

Трудной частью этого алгоритма является обратный ход Поскольку для дальнейшего нам не понадобятся детали этого алгоритма, мы предоставляем читателю в качестве упражнения повышенной сложности разобраться в нем с помощью комментариев:

```
proc arborescence = (ref [] int c, i, j, res, int i0) void:
begin co мы не задали число вершин и дуг графа, их мы получим как границы массивов co
    int m = upb res, n = upb c;
co введем структурные виды для дуг и вершин и еще один для вспомогательной информации co
    mode arc = struct (ref ver vi, vj, ref arc next, int c);
    mode ver = struct (ref arc by, f, ref ver br, im,
                       int a, s, bool act, w);
  mode mem = struct (ref arc u, ref ver v, int s);
```

co Информация о дуге содержит поле *next* для цепных списков, естественные ссылки на вершины и поле для стоимости. Информация о вершине сложнее: *im* — ссылка на вершину, «представляющую» данную в редукции графа, *br* — ссылка для цепного списка класса эквивалентности, *f* — начало цепного списка звезды, *by* — ссылка на минимальную дугу. Смысл информации, помещаемой в поля, *a*, *s*, *act* и *w* и в структуру **mem**, будет разъяснен позже. Введем теперь некоторые знакомые нам операции **co**

```
prio dif = 1, cons = 1;
op dif = (ref ver a, b) bool: a isnt b;
op lin = (ref arc a) bool: a isnt nil;
op cons = (ref ref arc a, ref arc b) ref ref arc:
   begin next of b := a; a := b end;
op cons = (ref ref ver a, ref ver b) ref ref ver:
   begin br of b := a; a := b end;
```

co Работая с классами эквивалентности, мы будем переходить от одной «новой вершины» к другой по дугам старого графа. Часто нужно будет искать нынешнего представителя начальной вершины дуги, выбранной для данной вершины. Для простоты опишем соответствующую процедуру **co**

```
proc prec = (ref ref ver v) ref ref ver:
                                    im of vi of by of v;
[1 : m] arc au; [1 : n] ver v; [1 : m] mem st;
int a1, a2, i1, ist := 0, j1, min;
ref arc u1, u2, u3, umin; ref ver v1, v2, v3, v4;
```

co Описав необходимую память, перейдем к заполнению массивов. Вначале просто заполняется массив вершин **co**

```
for i to m do
  v[i] := (nil, nil, nil, v[i], maxint, 0, i ≠ i0, true) od;
```

co *act* помечает «активные» вершины, представляющие классы эквивалентности, еще не соединенные с *i*0. В *a* будет записана минимальная длина дуги в звезде, которую мы сейчас построим **co**

```
for u to n do
  au[u]:=(v[i[u]], v1:=v[j[u]], skip, a1:=c[u]);
  f of v1 cons au[u];
  if a of v1 > a1 then a of v1:=a1;
                       by of v1:=au[u] fi od;
for i to m do co начался основной перебор вершин co
  while act of v[i] do v1:=v2:=v[i];
    while w of v1 /\ act of v1 do w of v1:=false;
                  v1:=prec(v1) od;
```

co маркируя путь, идем до пассивной или до маркированной вершины co

```
if w of v1 then
 while v2 dif v1 do act of v2:=false;
                                  v2:=prec(v2) od
```

else co начинается разбор случая, когда найден контур. Очистим путь, оставив только контур co

```
      while v2 dif v1 do w of v2:=true;
                                  v2:=prec(v2) od;
      min:=maxint; u3:=umin:=nil; v2:=prec(v1);
v of st[i1:=ist+:=1]:=v1;
      while v2 dif v1 do v3:=br of v2;
                              v of st[i1+:=1]:=v2;
        im of v2:=v1; act of v2:=false; br of v1 cons v2;
      while v3 dif ref ver(nil) do im of v3:=v1;
                                  v4:=v3;
        v3:=br of v3; br of v1 cons v4 od;
      v2:=prec(v2) od;
      for k from ist to i1 do v2:=v of st[k];
        a2:=a of v2; u2:=f of v2; a of v2:=ist;
```

co для каждой вершины в поле a запоминается момент, когда она перестала представлять класс эквивалентности co

```
      while lin u2 do
        if im of vi of u2 dif v1 then u3 cons u2;
          if (a1:=c of u2 -:=a2) < min
          then min:=a1; umin:=u2 fi fi;
        u2:=next of u2 od od;
```

co Мы исключили все вершины контура из «активной» информации, но $v1$ должны сохранить. Для обратного

хода потребуется некоторая информация о $v1$, которая и сохраняется в очередной записи **mem**: это ссылка на $v1$, предыдущая дуга с минимальной стоимостью и номер предыдущей такой же записи **co**

```
st[ist]:=(by of v1, v1, s of v1);
ref ver (v1):=(umin, u3, br of v1, v1, min,
                                    ist, true, true)
fi od od;
```

co Теперь начинается обратный ход. Мы должны оставить у каждой вершины по одной входящей в нее дуге из числа дуг, встречавшихся нам в ходе вычислительного процесса, причем дуги, которые встречались позже, пользуются привилегией **co**

```
for i from ist by -1 to 1 do
  if s of st[i]≥0 then j1:=0; v3:=nil; a1:=i;
v4:=v2:=vj of (u1:=by of (v1:=v of st[i]));
    while v3 dif v1 do i1:=s of (v3:=v2);
      while i1>j1 do i1:=s of st[a1:=i1];
            s of st[a1]:=-1 od;
      j1:=a of v2-1; s of v2:=i1; by of v2:=u
                                    of st[a1];
      v2:=v of st[a of v2] od;
    by of v4:=u1 fi od;
for u to n do c of au[u]:=u od;
for i to m do res[i]:=if i=i0 then 0
                      else c of au[by of v[i]] fi od
end
```

§ 6. Максимальный поток через сеть

Эта задача является частным случаем транспортной задачи с ограничениями пропускных способностей.

Задача о максимальном потоке.

Пусть задан граф $\langle M, N\rangle$, положительный вектор его пропускных способностей $d[N]$ и две вершины i_0 — «источник» и i_+ — «сток». Требуется найти поток $x[N]$, удовлетворяющий во всех вершинах, кроме i_0 и i_+, условию баланса потоков

$$a[i, N] \times x[N] = 0, \qquad (18)$$

условиям ограничения потока на дугах

$$0\,[N] \leqslant x\,[N] \leqslant d\,[N], \tag{19}$$

и максимизирующий поток, выходящий из i_0 (равный потоку, входящему в i_+):

$$1\,[N_{i_0}^-] \times x\,[N_{i_0}^-]. \tag{20}$$

Величина (20) будет здесь называться *величиной потока v.*

Для того чтобы свести эту задачу к транспортной задаче с ограничениями, достаточно добавить к графу еще одну дугу $\bar{u} \notin N$, положив

$$\bar{N} = N + \bar{u},\ i(\bar{u}) = i_0,\ j(\bar{u}) = i_+,$$

и определить $c\,[\bar{N}]$ ($c\,[\bar{u}] = 1$, $c\,[N] = 0\,[N]$) и $b\,[M]$ ($b\,[i_+] = -b\,[i_0] = d\,[\bar{u}] = \bar{d} = 1 + 1\,[N_{i_0}^-] \times x\,[N_{i_0}^-]$, остальные $b\,[i]$ равны 0).

При этом мы получаем на графе $\langle M, \bar{N}\rangle$ транспортную задачу с ограничениями пропускных способностей, эквивалентную задаче (18)—(20). В ней минимизация потоков означает минимальную перевозку по дуге $\bar{u}$, т. е. максимальный поток через граф $\langle M, N\rangle$. Можно выписать для этой задачи двойственную и применить известные результаты теории двойственности.

Интересно, что теорема двойственности для задачи о максимальном потоке приобретает уже совершенно комбинаторный вид.

Назовем *разрезом* такое множество $N^* \subset N$, что в любом пути из i_0 в i_+ в графе $\langle M, N\rangle$ содержится по меньшей мере одна дуга из N^* (иначе удаление N^* разрезает все пути из i_0 в i_+). *Величиной разреза* назовем сумму пропускных способностей включенных в него дуг.

Т е о р е м а 6 (теорема о максипотоке и миниразрезе). *Максимальная величина потока через граф $\langle M, N\rangle$ совпадает с минимальной величиной разреза.*

Д о к а з а т е л ь с т в о. Легко показать, что любой поток, идущий от источника к стоку, представляется в виде суммы потоков v_p, идущих по отдельным путям. Каждый такой путь содержит хотя бы одну дугу из разреза, какой бы разрез N^* мы ни зафиксировали

Величина потока равна сумме величин элементарных потоков. Сумма элементарных потоков, содержащих дан-

ную дугу из разреза, не превосходит ее пропускной способности. Итак,

$$\sum_{u \in N^-_{i_0}} x[u] = v = \sum_{p \in P} v_p \leqslant \sum_{u \in N^*} \sum_{p \in P(u)} v_p \leqslant \sum_{u \in N'} d[u].$$

Здесь P — множество путей, а $P(u)$ — путей, содержащих дугу u.

Итак, максимальная величина потока не превосходит минимальной величины разреза. Осталось показать, что они совпадают. Пусть $\bar{N}'$ — базис оптимального базисного решения транспортной задачи с ограничениями пропускных способностей, N'' — множество насыщенных дуг. Граф $\langle M, \bar{N}' \rangle$, который обязательно включает дугу $\bar{u}$, является деревом. Удаление из этого графа дуги $\bar{u}$ делит его на две компоненты связности — компоненту, содержащую i_0, и компоненту, содержащую i_+. Обозначим множество вершин этих компонент через M_0 и M_+ соответственно. Из $v[i_0] = 0$ и $c[\bar{u}] = 1$ легко получаем $v[M_0] = 0\ [M_0]$, $v[M_+] = 1\ [M_+]$. По теореме двойственности минимум в прямой задаче равен максимуму в двойственной. Равенство

$$c[N] \times x[N] = v[M] \times b[M] - \rho[N] \times d[N]$$

превращается в нашем случае в равенство

$$x[\bar{u}] = \bar{d} - 1\ [N^*] \times d[N^*]$$

или

$$1\ [N^*] \times d[N^*] = \bar{d} - x[\bar{u}] = \sum_{u \in N^-_{i_0}} x[u]. \qquad (21)$$

Здесь N^* — множество дуг с началом в M_0 и концом в M_+. Это множество, очевидно, образует разрез и содержится в N'', так как иначе были бы возможны улучшения оптимального решения. Равенство (21) и доказывает теорему. ▲

Алгоритм для нахождения максимального потока является вариантом метода потенциалов. В нем, разумеется, не требуется реально вводить векторы $c[N]$, $b[M]$ и дугу $\bar{u}$. Метод состоит из двух частей: поиска начального решения, состоящего из дерева (не обязательно кратчайших) путей из вершины i_0 и последовательных улучшений. При каждом таком улучшении заданы две компоненты связности графа — поддерево, содержащее i_0, и поддерево, содержащее i_+ с заданными на них потоками. Заданы потоки и на всех остальных дугах. Если имеется ненасыщенная дуга, ведущая из первой компоненты во вторую, она заменяет цепь, соединяющую i_0 и i_+, поток по которой и пересчитывается

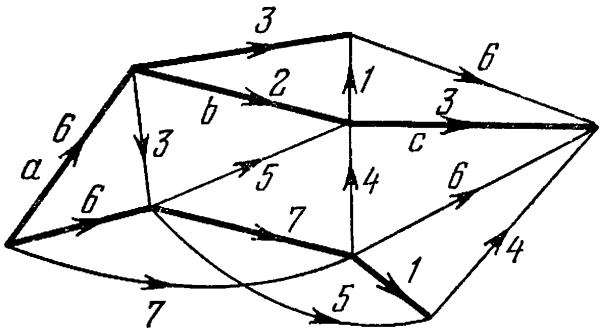

Рис. 50. Дерево кратчайших путей (нулевая итерация построения потока).

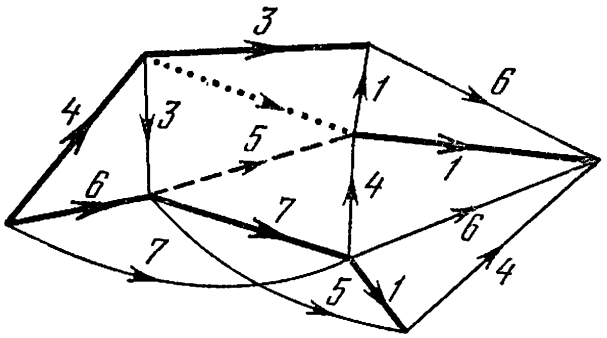

Рис. 51. Первая итерация.

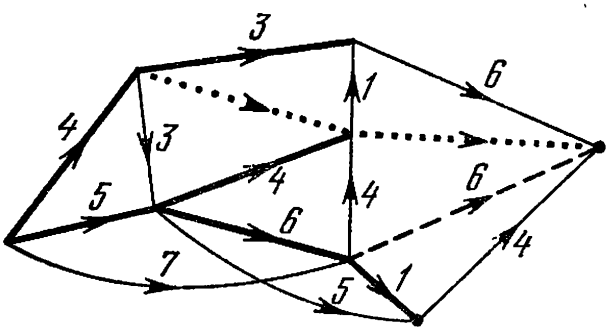

Рис. 52. Вторая итерация

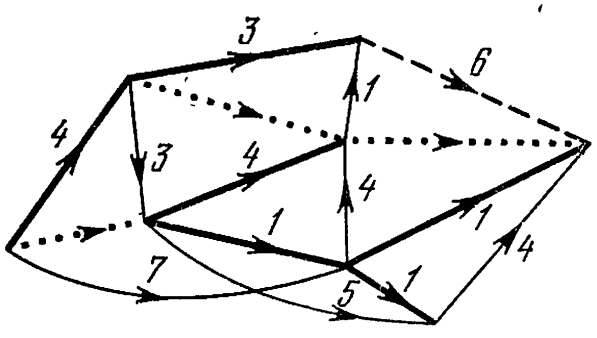

Рис 53. Третья итерация.

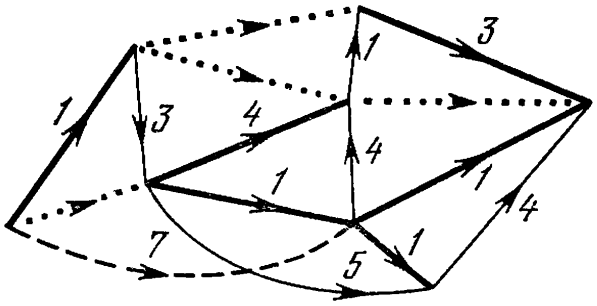

Рис 54. Четвертая итерация.

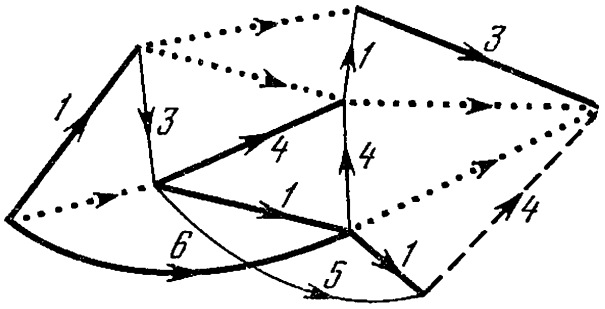

Рис 55. Пятая итерация.

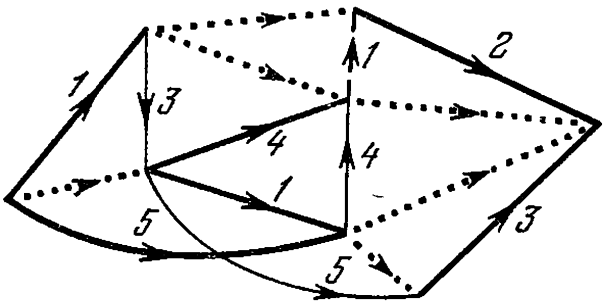

Рис 56. Шестая итерация

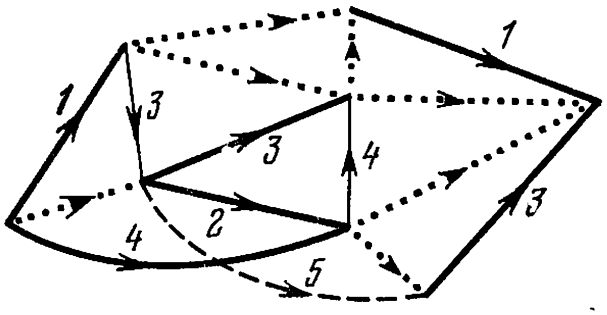

Рис. 57. Седьмая итерация.

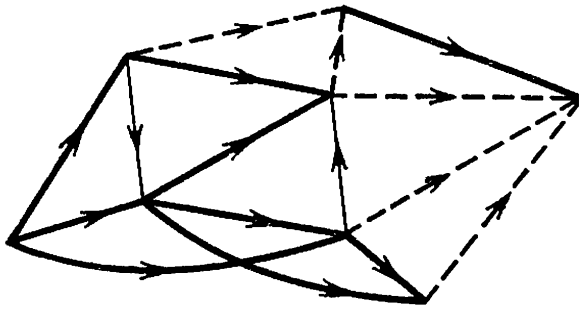

Рис 58. Окончательное решение.

обычным образом до тех пор, пока не насытится какая либо дуга с положительной ориентацией или не станет нулевым поток на дуге с отрицательной ориентацией. Далее эта дуга выбывает из базиса, изменяется деление дуг на компоненты связности и процесс повторяется.

Пример. Рассмотрим граф, изображенный на рис 50 Пропускные способности написаны рядом с дугами Жирными линиями выделены дуги входящие в дерево путей. Путь может использоваться для потока в 2 единицы Дуга b разомкнется, оставив граф, изображенный на рис 51 Штрихами выделена дуга, замыкающая путь. Далее получающиеся графы приводятся без пояснений на рис. 52—58 На рис 58 видно окончательное решение. Штрихами помечены дуги разреза.

§ 7. Распределение ресурсов в сетевом графике

Задача о максимальном потоке через сеть находит неожиданное применение в одной экстремальной задаче, развивающей схему сетевого графика.

Задача распределения ресурсов.

Пусть задан сетевой график $\langle M, N\rangle$, в котором продолжительности работ линейно зависят от параметра ϑ,

$$t(u, \vartheta) = t_0[u] - \vartheta \times \alpha[u], \quad \alpha[u] > 0, \tag{22}$$

который для работы u может меняться на отрезке $[0, d[u]]$. На значения параметров $\vartheta[N]$ наложено условие

$$\vartheta[N] \geqslant \mathbf{0}[N], \qquad \mathbf{1}[N] \times \vartheta[N] \leqslant \Theta. \tag{23}$$

Требуется выбрать такой вектор $\vartheta[N]$, удовлетворяющий (23), при котором длина критического пути была бы минимальна.

Будем решать эту задачу как параметрическую задачу с параметром Θ. При $\Theta = 0$ вектор $t_0[N]$ задает продолжительности всех работ. Найдем множество критических дуг N_c. При малом Θ мы хотим найти такое распределение ресурсов, которое больше всего уменьшало бы критический путь

Очевидно, что дополнительный ресурс при малых Θ должен вкладываться исключительно в дуги из N_c. Эти

вложения будут пропорциональны Θ, и поэтому можно рассматривать вектор $w[N_c]$ скоростей увеличения ресурса $\vartheta[N_c]$ Покажем, что можно ограничиться увеличением ресурса на дугах некоторого разреза графа $\langle M, N_c\rangle$.

Действительно, множество

$$N^+ = \{u \mid u \in N_c,\ w[u] > 0\} \qquad (24)$$

само должно представлять собой разрез, так как от вложения ресурса должны уменьшиться все критические пути, т. е. все пути из i_- в i_+ в графе $\langle M, N_c\rangle$. Выберем в N^+ минимальный разрез N_1 (т. е. разрез, в котором уже нет лишних дуг) и положим

$$z = \min\{w[u] \times \alpha[u] \mid u \in N_1\}. \qquad (25)$$

Уменьшим на z продолжительность всех путей по разрезу N_1. Получится вектор остатков скоростей

$$w[u] := w[u] - \delta(u \in N_1) \times z \times \alpha[u], \quad u \in N. \qquad (26)$$

и новое множество N^+.

Пока это множество является разрезом, будем заново выбирать минимальный разрез (каждый раз он будет другим) и пересчитывать $w[N]$ по формуле (26).

Если же множество N^+ разрезом не является, то ресурсы тратятся нерационально — они не уменьшают какого-то из критических путей, и если уменьшить исходный вектор $w[N_c]$ на этот оставшийся вектор, то при том же уменьшении критического пути затраты ресурса будут меньше, а при тех же затратах ресурса больше будет уменьшение критического пути.

Таким образом, оптимальное распределение ресурсов должно состоять из выпуклой комбинации капиталовложений в минимальные разрезы. При этом в каждый разрез N_1 капиталовложение производится с вектором интенсивностей $\beta[N]$, где

$$\beta[u] = 1/\alpha[u] \qquad (27)$$

(продолжительность всех путей по разрезу уменьшается на одно и то же число). Естественно выбрать тот из минимальных разрезов, на котором достигается минимум затрат на единицу уменьшения времени. Таким образом, мы получаем задачу:

Задача выбора разреза.

Найти разрез N_1 в графе $\langle M, N_c\rangle$, на котором достигается минимум $\sum_{N_1} 1/\alpha\,[u]$.

Обозначим решение этой задачи через R.

Итак, при малых Θ вектор вложений ресурса $z\,[N_c]$ будет выглядеть так:

$$z\,[u] = \gamma \times \delta\,(u \in R)\,/\alpha\,[u]. \qquad (28)$$

Определим теперь, чем определяется максимальное значение $\Theta = \Theta_0$. Во-первых, значение суммарной затраты ресурса ограничено величиной Θ_0

$$\sum_R z\,[u] \leqslant \Theta_0 \text{ или } \gamma \leqslant \Theta/\sum_R (1/\alpha\,[u]) = g_1, \qquad (29)$$

во-вторых, вложение ресурса в каждую из дуг $u \in R$ не должно превысить максимальное значение $d\,[u]$ и, следовательно,

$$\gamma \leqslant \min\{d\,[u] \times \alpha\,[u] \mid u \in R\} = g_2. \qquad (30)$$

Наконец, в графе при уменьшении некоторых дуг могут появиться новые критические пути. Найти субкритический путь просто — нужно в графе $\langle M, N \setminus R\rangle$ найти максимальный путь из i_- в i_+. Пусть его длина на τ меньше длины критического пути. Тогда $\gamma \leqslant \tau$ и окончательно

$$\gamma = \min\{g_1, g_2, \tau\}.$$

В случае, когда $\gamma = g_1$, ресурс исчерпан и решение окончательно; если $\gamma = g_2$, то для дуг, у которых ресурс достиг максимальной границы, мы полагаем $\alpha\,[u] = 0$ и $\beta\,[u] = \infty$, для остальных дуг уменьшаем $d\,[u]$ на $\gamma \times \beta\,[u]$, соответственно изменяем $t_0\,[N]$ и Θ и решаем задачу заново. Для случая $\gamma = \tau$, кроме таких изменений, мы должны добавить к графу $\langle M, N_c\rangle$ новые критические дуги, после чего снова задача решается заново.

Отметим, что от шага к шагу этого процесса минимум в задаче выбора разреза увеличивается (так как увеличивается граф $\langle M, N_c\rangle$ или увеличиваются — до ∞ — значения некоторых из $\beta\,[u]$). Поэтому график изменения длины критического пути выглядит примерно, как указано на рис. 59.

Теорема 7. *Описанный алгоритм дает оптимальное решение задачи распределения ресурсов.*

Доказательство этой теоремы следует из того, что задача распределения ресурсов может рассматриваться как параметрическая задача линейного программирования.

Найти $v[M]$ и $y[N]$, удовлетворяющие условиям

$$v[i(u)] + t_0[u] - \alpha[u] \times y[u] \leqslant v[j(u)], \quad (31)$$

$$\mathbf{1}[N] \times y[N] \leqslant \Theta, \quad (32)$$

$$\mathbf{0}[N] \leqslant y[N] \leqslant d[N] \quad (33)$$

и минимизирующие $v[i_+] - v[i_-]$.

Ограничимся здесь лишь наброском доказательства законности нашего метода, которое пригодно для всех параметрических задач линейного программирования.

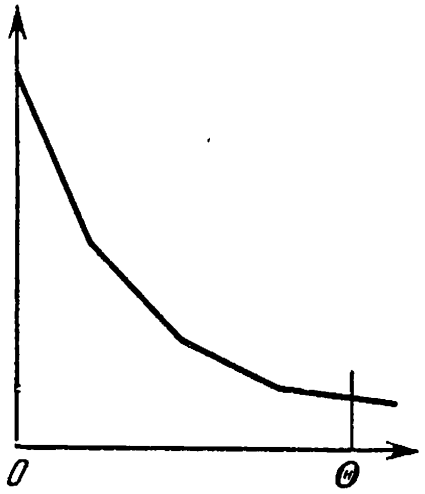

Рис. 59. Зависимость длины критического пути от ресурса.

В параметрической задаче линейного программирования каждому допустимому решению $x[N]$ можно сопоставить пару точек (γ, λ), где γ — значение целевой функции на этом решении и λ — значение параметра, при котором $x[N]$ является решением (если $x[N]$ является решением при нескольких значениях параметра, ему соответствует несколько точек). Когда $x[N]$ пробегает все множество допустимых решений при всех значениях параметра, пара (γ, λ) пробегает выпуклое замкнутое многогранное множество (рис. 60).

При одном и том же значении параметра все оптимальные решения задачи проектируются в одну и ту же точку, принадлежащую нижней границе множества.

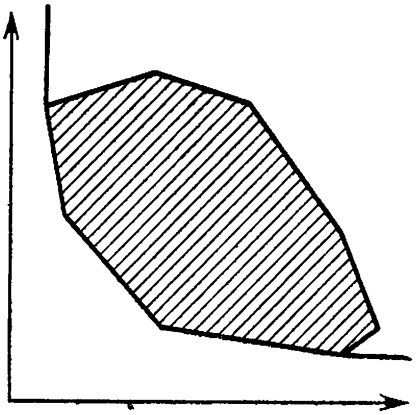

Рис. 60. Область значений параметра и целевой функции.

При изменении параметра для получения оптимального решения следует двигаться из имеющегося оптимального решения по траектории, сохраняющей принадлежность нижней границе, т. е. по направлению наискорейшего убывания целевой функции. Легко показать, что скорость убывания целевой функции не зависит от того, какое именно оптимальное решение имеется при данном состоянии параметра. Специфика рассматриваемой нами задачи используется

лишь в том, что при изменении решения оказывается невыгодным уменьшение капиталовложений в дуги, в которые на предыдущих шагах процесса были сделаны положительные капиталовложения.

§ 8. Покрытие графа путями

Пусть $\langle M, N\rangle$ — транзитивно замкнутый граф без контуров, в котором нет дуг с одинаковыми началом и концом. Мы будем интересоваться вопросом о разбиении множества вершин этого графа на линейно упорядоченные подмножества, т. е. о выделении в графе такого множества путей, чтобы каждая вершина принадлежала хотя бы одному пути.

Транзитивная замкнутость графа позволяет свести эту задачу к эквивалентной задаче о поиске путей, в которых каждая вершина участвует ровно по разу Этот набор путей может быть найден в результате решения транспортной задачи на некотором вспомогательном графе. Что это за задача, будет видно из доказательства следующей теоремы, являющейся основной целью нашего рассмотрения.

Теорема 8 (Дилворта). *Минимальное число путей, содержащих все вершины из M, равно максимальному числу попарно несравнимых вершин.*

Доказательство. Так как несравнимые вершины должны лежать на различных путях, то максимальное число несравнимых вершин не превосходит минимального числа путей. Для того чтобы показать, что эти два числа совпадают, сведем задачу о нахождении минимального числа путей к транспортной задаче.

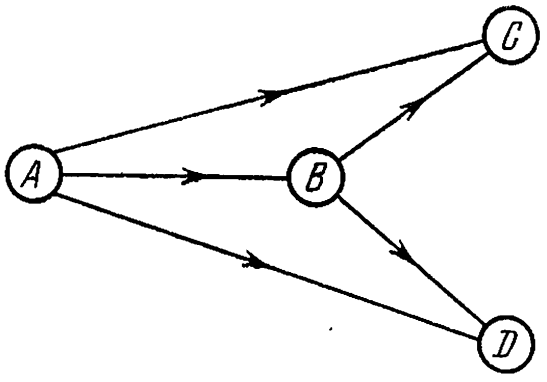

Рис. 61. Транзитивно замкнутый граф

Рассмотрим граф $\langle \bar{M}, \bar{N}\rangle$, в котором множество $\bar{M}$ состоит из двух экземпляров множества M (элементы первого экземпляра $(M, 1)$ мы будем обозначать $(i, 1)$, где $i \in M$, элементы второго экземпляра $(M, 2)$ соответственно $(i, 2)$) и двух вершин i_0 и i_+ (рис. 61 и 62). Дуги мы будем задавать граничными вершинами, поэтому для задания графа можно каждой вершине поставить в соответствие множество

вершин, в которое можно из нее попасть. И для исходного графа положим

$$M_i = \{j \mid j \in M,\ N_i^- \cap N_j^+ \neq \emptyset\}.$$

Во вновь определяемом графе нужно для каждого $i \in \bar{M}$ задать $\bar{M}_i$. Положим

$$\begin{aligned} \bar{M}_i &= \emptyset \quad \text{для } i \in (M, 1) \text{ и } i = i_+, \\ \bar{M}_{i_0} &= (M, 1) + i_1, \\ M_{(i,2)} &= (M_i, 1) + i_+. \end{aligned}$$

Положим теперь $c[u] = 1$ для всех дуг, входящих в вершину i_+ из $(M, 2)$ и $c[u] = 0$ для всех остальных дуг. (Таким образом, $c[u] = \delta(i(u) \in (M, 2) \wedge j(u) = i_+)$.)

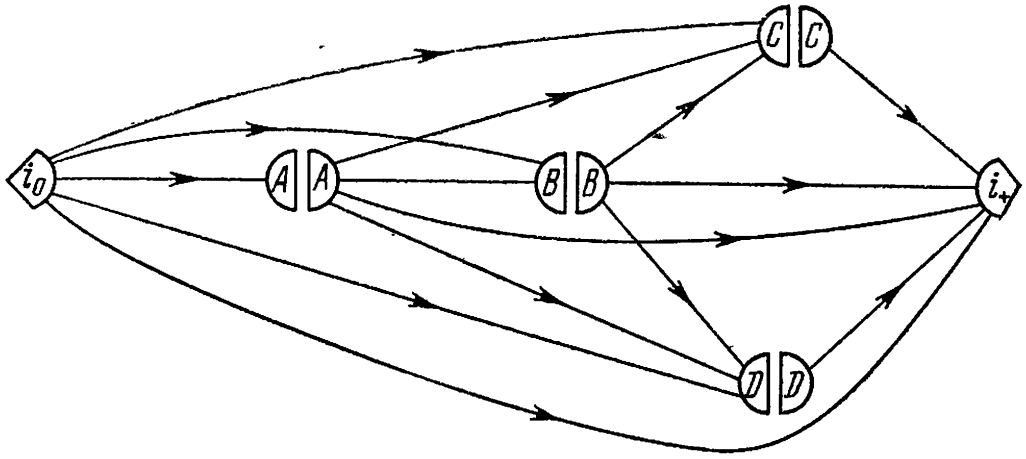

Рис 62. Преобразованный граф.

Потребление зададим вектором

$$b[(M, 1)] = 1[(M, 1)],\quad b[(M, 2)] = -1[(M, 2)],$$
$$b[i_+] = -b[i_0] = |M|.$$

Эта транспортная задача (поскольку вектор цен неотрицателен) имеет оптимальное базисное решение. Легко видеть, что в этом оптимальном базисном решении все ненулевые потоки целочисленны. Если добавить потоки единичной величины из вершины $(i, 1)$ в соответствующие вершины $(i, 2)$, мы получим условия баланса потоков в этих вершинах, и наш поток превратится в поток из i_0 в i_+. Этот поток разлагается на элементарные потоки по путям из i_0 в i_+, причем число их будет равно (минимальному) значению целевой функции в транспортной задаче Легко видеть, что построенные так пути проходят через все вершины графа

$\langle \bar{M}, \bar{N} \rangle$ и им соответствуют пути в исходном графе $\langle M, N \rangle$, также проходящие через все вершины.

Далее, базисность решения означает, что имеется базис-набор $\bar{N}'$ из $|\bar{M}| - 1$ дуг такой, что $\langle \bar{M}, \bar{N}' \rangle$ является деревом Если удалить дугу (i_0, i_+), которая обязательно должна содержаться в любом базисном решении, граф $\langle \bar{M}, \bar{N}' \rangle$ разобьется на две компоненты связности В первой компоненте (содержащей i_0) у всех вершин будет один и тот же потенциал (равный, например, для удобства 1), во второй компоненте у всех вершин, кроме i_+ с потенциалом, равным 1, потенциал будет равен 0. По теореме двойственности для оптимальных решений $v^* [\bar{M}]$ и $x^* [\bar{N}]$ мы должны иметь

$$v^* [\bar{M}] \times b [\bar{M}] = c [\bar{N}] \times x^* [N] \tag{34}$$

и

$$v^* [(i (u), 2)] \geqslant v^* [(j (u), 1)] \quad \text{для } u \in N. \tag{35}$$

Из транзитивности графа $\langle M, N \rangle$ и из неравенства (35) следует, что не существует вершины $i \in M$, для которой $v [(i, 1)] = 0$ и $v [(i, 2)] = 1$. Действительно, в этом случае существовала бы базисная дуга u_1, входящая в i, и, следовательно, $v [(i (u_1), 2)] = 0$, и базисная дуга u_2, выходящая из j, для которой $v [(j (u_2), 1)] = 1$ Но по транзитивности графа тогда имеется и дуга u_3, ведущая из $i (u_1)$ в $j (u_2)$, для которой (35) нарушается. Рассмотрим теперь равенство (34). В правой его части стоит число путей, в левой — число вершин из M, для которых $(i, 1)$ принадлежит первой компоненте, а $(i, 2)$ — второй компоненте (когда эти вершины принадлежат одной и той же компоненте, их суммарный вклад в целевую функцию равен 0). Обозначим множество таких вершин через M^* и покажем, что вершины из M^* попарно несравнимы. Действительно, если бы существовал путь из i_1 в i_2, $i_1, i_2 \in M^*$, то в этом пути нашлась бы такая дуга (i', i''), в которой $(i', 2)$ принадлежит второй компоненте, а $(i'', 1)$ первой компоненте, а для этой дуги нарушилось бы условие (35).

Итак, мы разбили граф на пути таким образом, что в каждом пути найдется вершина из множества попарно несравнимых вершин ▲

К теореме Дилворта сводятся многие комбинаторные теоремы двойственности.

§ 9. Паросочетания в простых графах

Граф $\langle M, N\rangle$ называется *простым* [1]), если множество M разбивается на такие два подмножества M_1 и M_2, что начало любой дуги лежит в M_1, а конец — в M_2. Мы уже несколько раз встречались с простыми графами

Паросочетанием простого графа называется набор дуг, все начала и все концы которых различны Естественно возникает задача о выборе наибольшего такого набора.

Теорема 9 (Кёнига) *Наибольшее число дуг паросочетания равно наименьшему числу вершин в множестве, которому инцидентны все дуги графа.*

Доказательство Дополним граф вершинами i_0 и i_+ и дугами, ведущими из i_0 в M_1 и из M_2 в i_+ с пропускной способностью 1, а на дугах из N сделаем пропускную способность больше $|M|$ Легко видеть, что величина максимального потока через такую сеть как раз и равна максимальному числу дуг в паросочетании По теореме 6 величина этого потока равна величине минимального разреза Разрез состоит из вновь введенных дуг, число которых равно величине потока Эти дуги определяют множество M' вершин из M, которым они инцидентны Любая дуга из N имеет в качестве одной из граничных вершин вершину из M' (иначе нашелся бы путь, увеличивающий поток) Это и доказывает теорему ▲

Следствие 1 (теорема Кёнига — Оре). *Назовем дефицитом графа величину.*

$$\delta = \max \{ |A| - |\Gamma A| \mid A \subset M_1\},$$

где $\Gamma A = \{j \mid N_j^+ \cap (\cup_{i\in A} N_i^-) \neq \emptyset\}$. *Максимальное число дуг в паросочетании графа равно* $|M_1| - \delta$

Доказательство. Действительно, множество вершин, которому инцидентны все ребра, имеет вид

$$B \cup \Gamma (M_1 \setminus B), \text{ где } B \subset M_1.$$

По теореме 9 максимальное число дуг в паросочетании равно

$$s = \min \{ |B| + |\Gamma (M_1 \setminus B)| \mid B \subset M_1\}.$$

Далее имеем

$$s = \min \{ |M_1| - |A| + |\Gamma A| \mid A \subset M_1\} =$$
$$= |M_1| - \max \{ |A| - |\Gamma A| \mid A \subset M_1\}. \blacktriangle$$

[1]) Употребляется также термин «двудольный граф».

Следствие 2 (теорема Кёнига — Холла). *Назовем паросочетание полным, если число дуг в нем равно* $|M_1|$. *Для того чтобы паросочетание было полным, необходимо и достаточно, чтобы* $|\Gamma A| \geqslant |A|$ *для любого* $A \subset M_1$.

Действительно, условие означает, что $\delta = 0$.

Следствие 3. *Все члены*[1]) *определителя произвольной квадратной матрицы* $a[1:n, 1:n]$ *равны нулю в том и только в том случае, если найдутся такие* $M', M'' \subset 1:n$, *что*

$$a[M', M''] = 0[M', M'] \quad и \quad |M'| + |M''| \geqslant n + 1.$$

Доказательство. Если сопоставить строкам матрицы вершины M_1, столбцам — вершины M_2 ($|M_1| = |M_2| = n$), а дугу из $i \in M_1$ в $j \in M_2$ проводить только при $a[i, j] \neq 0$, мы получим простой граф. Ненулевому члену определителя соответствует в нем полное паросочетание. Если полного паросочетания нет, дефицит больше нуля. Множества $M' \cap M_1$, $M' \cap M_2$ и будут обладать требуемыми свойствами. ▲

Матрица $p[M, M]$ называется *бистохастической*, если

$$\begin{gathered} p[M, M] \geqslant 0[M, M], \\ \sum_{i \in M} p[i, M] = \sum_{i \in M} p[M, i] = 1[M]. \end{gathered} \tag{36}$$

Это название возникло в связи с теоретико-вероятностным происхождением таких матриц.

Частный случай бистохастической матрицы — это переставляющая матрица, которая была введена в § 1 гл. 1. Оказывается, переставляющие матрицы — это крайние точки множества бистохастических матриц.

Теорема 10 (Биркгофа — фон Неймана). *Всякая бистохастическая матрица представима в виде выпуклой комбинации переставляющих матриц.*

Доказательство. Покажем, что в определителе бистохастической матрицы найдутся ненулевые члены. Действительно, если бы их не было, то по следствию 3 из предыдущей теоремы нашлись бы такие множества M', $M'' \subset M$, что $p[M', M''] = 0[M', M'']$ и $|M'| + |M''| \geqslant |M| + 1$. Суммируя элементы матрицы $p[M \setminus M', M'']$,

[1]) Членом определителя называется произведение $\Pi_{i \in 1:n} a[i, Ti]$, где T — взаимно однозначное отображение $T: 1:n \to 1:n$.

мы получим из (36)

$$\Sigma_{j \in M''} (\Sigma_{i \in M \setminus M'} p[i, j]) = |M''|.$$

Вместе с тем

$$\Sigma_{i \in M \setminus M'} (\Sigma_{j \in M''} p[i,j]) \leqslant |M \setminus M'| = |M| - |M'|,$$

что дает неравенство $|M'| + |M''| \leqslant |M|$, противоречащее выбору M' и M''.

Далее, каждому ненулевому члену определителя соответствует переставляющая матрица $r[M, M]$. Положив $\lambda = \min\{p[i, j] \mid r[i, j] = 1\}$, мы можем представить матрицу $p[M, M]$ в виде

$$p[M, M] = \lambda \times r[M, M] + (1 - \lambda) \times p1[M, M].$$

Легко показать, что матрица $p1$ является также бистохастической матрицей и что у нее положительных элементов меньше, чем в p. Итерирование этого разложения приводит к утверждению теоремы. ▲

Отметим возможность другого доказательства этой замечательной теоремы: ограничения (36) — это ограничения специфической транспортной задачи, в которой все объемы производства и объемы потребления равны 1. Любое базисное решение такой задачи является целочисленным, следовательно, состоит из нулей и единиц и представляет собой переставляющую матрицу.

Транспортная задача с ограничениями (36) носит название «задача о назначениях». Это название возникло из следующей интерпретации задачи.

> Имеется m должностей и m претендентов на эти должности. Назначение i-го претендента на j-ю должность приводит к убыткам $c[i, j]$. Требуется распределить претендентов по должностям так, чтобы суммарный убыток был минимальным.

Задача о назначениях часто используется как вспомогательная задача при решении дискретных задач, в которых требуется найти оптимальную в каком-либо смысле подстановку. Для решения этой задачи был разработан специальный алгоритм, получивший название «венгерский метод», так как в нем существенно используются результаты венгерского математика Эгервари.

§ 10. Венгерский метод для задачи о назначениях

Общая структура алгоритма такова: он состоит из последовательных шагов, на каждом из которых делается попытка выбрать назначение при неотрицательной матрице убытков так, чтобы все выбранные элементы матрицы были нулевыми. Если это сделать не удается, матрица изменяется с сохранением свойства неотрицательности.

При неотрицательной матрице значение целевой функции задачи должно быть неотрицательно, поэтому если удастся получить «нулевое» назначение, о котором сказано выше, оно, очевидно, будет оптимальным. Для получения «нулевого» назначения нужно таким образом изменить матрицу, чтобы минимальное значение целевой функции равнялось нулю. Таким образом, невозможность получить «нулевое» назначение показывает, что матрица еще недостаточно изменена.

Метод изменения матрицы очень прост — он состоит из прибавления ко всем элементам какой-либо строки или столбца матрицы одного и того же числа. Поскольку в любом допустимом решении должен быть ровно один элемент из этой строки (или этого столбца), к значению целевой функции при этом изменении прибавляется то же число.

Первоначальная подготовка матрицы заключается в том, что от каждого элемента каждой строки матрицы вычитается наименьший в этой строке элемент Матрица становится неотрицательной с нулевым элементом в каждой строке. Вычитание из каждого столбца наименьшего в нем элемента обеспечивает существование нулевых элементов и в столбцах.

Основной шаг алгоритма состоит из поиска «нулевого» назначения и преобразования матрицы. Для «нулевого» назначения лучше воспользоваться терминологией простых графов, считать, что M_1 — множество строк матрицы, M_2 — множество столбцов и дуга из i в j идет в том и только в том случае, если $a[i, j] = 0$. Тогда существование «нулевого» назначения эквивалентно существованию в этом графе полного паросочетания.

Будем разыскивать в нашем графе максимальное паросочетание (дающее наибольшее число «независимых» нулей— нулей, расположенных в различных строках и столбцах матрицы). По теореме 9 число дуг в максимальном паросоче-

тании равно наименьшему числу вершин в множестве, которому инцидентны все дуги графа. В терминах матрицы теорема 9 звучит следующим образом: наибольшее число независимых нулей равно наименьшему числу вертикальных и горизонтальных линий (столбцов и строк), которыми можно перечеркнуть все нули матрицы.

Итак, пусть построено максимальное паросочетание и найдены множества строк M' и столбцов M'' так, что для каждого нуля $a[i, j]$ либо $i \in M'$, либо $j \in M''$. Пусть

$$\lambda = \min \{a[i, j] \mid i \in M_1 \setminus M', \ j \in M_2 \setminus M''\}.$$

Прибавим λ к каждой строке M' и вычтем λ из каждого столбца $M_2 \setminus M''$. Значение целевой функции уменьшится при этом на $\lambda \times (|M_2| - |M''| - |M'|) > 0$. Матрица останется неотрицательной, так как $a[M_1, M'']$ не уменьшилась, $a[M', M_2 \setminus M'']$ осталась без изменений, а в $a[M_1 \setminus M', M_2 \setminus M'']$ наименьший элемент равен λ. Это преобразование матрицы и завершает основной шаг алгоритма.

Осталось описать способ построения максимального паросочетания и множеств M', M''. Этот метод пригоден, разумеется, для любого простого графа, и поэтому мы будем вести построение в терминах простых графов.

Пусть в простом графе $\langle M_1, M_2, N\rangle$ выделено уже паросочетание N^*, $I(N^*) \subset M_1$ — множество начал дуг паросочетания, $J(N^*) \subset M_2$ — множество концов. Определим множества I^* и J^* таким образом, чтобы $I^* \subset I(N^*)$, $J^* \subset J(N^*)$, каждая дуга из N^* была инцидентна ровно одной вершине из $I^* \cup J^*$ Если $k = |N^*|$, то, очевидно,

$$k = |I(N^*)| = |J(N^*)| = |I^*| + |J^*|.$$

Положим $M^* = I^* + J^*$.

Кроме самого паросочетания, для алгоритма требуется еще некоторое множество «кандидатов в паросочетание» N', которое будет постепенно пополняться и в котором для некоторых из элементов N^* будут содержаться «дублеры». От N' мы потребуем, чтобы для каждой его дуги в M^* нашлась инцидентная ей вершина, но чтобы в N' нашлось не больше одной дуги, инцидентной любой вершине из M^*. Кроме того, мы потребуем, чтобы граф $\langle M, N^* \cup N'\rangle$ не содержал циклов. Множество N' будет естественным образом

разбиваться на

$$N'_- = \{u \in N',\ i(u) \in I^*\} \quad \text{и} \quad N'_+ = \{u \in N',\ j(u) \in J^*\}.$$

Перейдем теперь к описанию самого метода. Первоначально, кроме заданного N^*, мы имеем $I^* = \emptyset$, $J^* = J(N^*)$, $N' = \emptyset$. Пополняя множество N' и связывая его элементы с элементами N^*, будем приписывать этим парам положительные ранги $r(u)$.

Опишем теперь очередной шаг алгоритма (до увеличения множества N^* или установления его максимальности).

1° Если все дуги графа инцидентны M^*, то построенное паросочетание максимально и работа алгоритма завершена. Пусть существуют дуги, не инцидентные M^*. Ниже будем считать, что дуга $\bar{u}$ именно такова.

2° Если найдется дуга $\bar{u}$, для которой $i(\bar{u}) \notin I(N^*)$, $j(\bar{u}) \notin J(N^*)$, то $\bar{u}$ можно будет присоединить к N^*, и работа алгоритма завершится.

3° Если найдется дуга $\bar{u}$, для которой $i(\bar{u}) \in I(N^*)$, сделаем ее дублером дуги из N^*. В N^* имеется единственная дуга $\bar{u}^*$, началом которой является $i(\bar{u})$. Сейчас $j(\bar{u}^*) \in J^*$. Исключим конец дуги $\bar{u}^*$ из J^*, а ее начало включим в I^*. Дугу $\bar{u}$ включим в N'. Если концу дуги $\bar{u}$ не инцидентна никакая дуга из N^*, припишем дуге $\bar{u}^*$ ранг $r(\bar{u}^*) = 1$. Если имеется $u^* \in N^*$, $j(u^*) = j(\bar{u})$ и, естественно, $i(u^*) \in I^*$, положим $r(\bar{u}^*) = r(u^*) + 1$ и вернемся к п. 1°.

4° Если найдется дуга $\bar{u}$, для которой $i(\bar{u}) \notin I(N^*)$, но $j(\bar{u}) \in J(N^*)$, то множество N^* можно перестроить с увеличением его размера. Действительно, существует дуга u_1^*, инцидентная $j(\bar{u})$, $i(u_1^*) \in I^*$, а значит, и дуга $u_1 \in N'$, выходящая из $i(u_1^*)$. Если $r(u_1^*) = l$, то легко видеть, что найдется еще $l - 1$ пар $u_i^* \in N^*$, $u_i \in N'$, причем последовательность дуг

$$\bar{u},\ u_1^*,\ u_1,\ u_2^*,\ \ldots,\ u_l^*,\ u_l$$

образует цепь. Эта цепь имеет нечетное число дуг, ее начало $i(\bar{u})$ и конец $j(u_{l+1})$ не инцидентны N^*. Если мы заменим в N^* дуги u_i^* дугами u_i и $\bar{u}$, мы увеличим размер этого множества и закончим работу алгоритма.

На каждом шаге алгоритма множество N^* увеличивается на один элемент до тех пор, пока не обнаружится максимальность паросочетания.

Покажем действие алгоритма на простом примере. Пусть простой граф задается матрицей (табл. 16), строки которой соответствуют M_1, столбцы — M_2, элементами являются названия дуг (там, где нет названия, дуга отсутствует).

Таблица 16

	11	12	13	14	15	16	17
1		a			b		
2		r				t	
3					c	d	x
4		p				s	
5	h			g	e	f	
6		k	q	v		w	l
7				n			m

Первоначально пусть $N^* = \{a, t, c, g, q, m\}$, $N' = \emptyset$, $I^* = \emptyset$, $J^* = 12 : 17$.

α) По п. 3° дуга h становится дублером g, их общий ранг равен 1. Теперь

$$N' = \{h\}, \quad I^* = \{5\}, \quad J^* = \{12, 13, 15, 16, 17\}.$$

β) Дуга v становится дублером q, их ранг равен 2,

$$N' = \{h, v\}, \quad I^* = \{5, 6\}, \quad J^* = \{12, 15, 16, 17\}.$$

γ) Дуга n становится дублером m, их ранг равен 2,

$$N' = \{h, v, n\}, \quad I^* = \{5, 6, 7\}, \quad J^* = \{12, 15, 16\}.$$

δ) Дуга x — дублер c, ранг равен 3,

$$N' = \{h, v, n, x\}, \quad I^* = \{3, 5, 6, 7\}, \quad J^* = \{12, 16\}.$$

ε) Дуга b — дублер a, $r(a) = 4$,

$$N' = \{h, v, n, x, b\}, \quad I^* = \{1, 3, 5, 6, 7\}, \quad J^* = \{16\}.$$

ξ) Дуга r — дублер t, $r(t) = 5$,

$$N' = \{h, v, n, x, b, r\}, \quad I^* = \{1, 2, 3, 5, 6, 7\}, \quad J^* = \emptyset.$$

Теперь дуга s по п. 4° приводит к цепи (подчеркнуты дуги из N^*)

$$s, \underline{t}, r, \underline{a}, b, \underline{c}, x, \underline{m}, n, \underline{g}, h,$$

что дает новое паросочетание (очевидно, максимальное)

$$N^* = \{q, s, r, b, x, n, h\}.$$

Итак, справедливы следующие утверждения.

Теорема 11. *Алгоритм построения максимального паросочетания сходится за конечное число шагов.*

Теорема 12. *Венгерский метод для решения задачи о назначениях сходится за конечное число шагов.*

Теорема 12 следует из предыдущей ввиду того, что на каждом шаге венгерского метода строится новое паросочетание, соответствующее большему значению исходной целевой функции, а число наборов независимых элементов конечно.

Можно оценить трудоемкость построения максимального паросочетания и получить из нее трудоемкость венгерского метода, однако приведенная нами вычислительная схема неоптимальна. Между тем венгерский метод в удачных реализациях оказывается высокоэффективным.

Удачная реализация связана с тем, что последовательно решается m задач о назначениях, в которых число столбцов не больше числа строк и последовательно увеличивается.

При $n = 1$ оптимальное назначение находится тривиально, нужно просто найти минимум в единственном столбце матрицы. При увеличении числа столбцов на 1 мы уже имеем набор из независимых нулей, стоящих в появившихся ранее столбцах. Во вновь появившемся столбце мы находим минимум (с учетом потенциалов строк — тех вычитаемых, которые были определены из предыдущих расчетов), объявляем его новым нулем матрицы или дугой $\bar{a}$ из описанного алгоритма построения паросочетаний. Если этот нуль может дополнить имеющееся паросочетание, итерация кончается, в противном случае он оказывается дублером некоторого нуля из паросочетания и через него соединяет n-й столбец с одним из предыдущих. Они образуют теперь множество неперечеркнутых столбцов, в этом множестве среди неперечеркнутых строк разыскивается элемент минимальной стоимости, после чего изменением потенциалов соответствующих строк и столбцов производится пересчет матрицы.

Новый нуль может также быть в «занятой» строке — в этом случае появляется еще одна неперечеркнутая строка и процесс продолжается. Отметим, что при этом очень просто осуществляется поиск минимума — элементы нового неперечеркнутого столбца сравниваются с уже найденными минимумами строк предыдущей итерации.

Если же строка не занята, то, как в п. 4° алгоритма максимального паросочетания, строится цепочка, по которой паросочетание изменяется с увеличением размера на единицу, что заканчивает процесс для вновь введенного столбца.

В описываемом ниже алгоритме все необходимые связи осуществляются между строками матрицы, поэтому удобно ввести структуру **row**, описывающую строку. Полями этой структуры, кроме самой строки матрицы c, являются потенциал строки v_i, текущий минимум по неперечеркнутым столбцам *min*, столбец, выбранный в данной строке w ($w = 0$, если такого столбца нет), и две ссылки на другие строки: wr — на строку, в которой находится дублер из столбца, выбранного данной строкой, p — для общего цепного списка строк. Этот общий цепной список разбит на две части ссылкой $r1$, строки, идущие до $r1$ включительно, являются неперечеркнутыми, а строки после $r1$ до конца списка $r2$ — перечеркнутые. Оказывается полезным ввести «нулевую» структуру $r0$ и хранить в c **of** $r0$ потенциалы столбцов, в p **of** $r0$ ссылку на начало общего цепного списка, а саму ссылку $r0$ использовать вместо **nil**.

Параметрами процедуры, кроме самой матрицы *cost* [1 : m, 1 : n] (где $m \geqslant n$), являются векторы *row* [1 : n] и *col* [1 : m], в которые записывается оптимальное назначение. Вырабатываемое процедурой значение — это сумма выбранных элементов матрицы *cost*. Дальнейшие комментарии приводятся прямо в тексте процедуры:

```
proc gribov = (ref [,] int cost, ref [ ] int row, col) int:
begin int m = upb col, n = upb row;
 int f := maxint, d, s, vs, g, g1, u, u1;
 mode row = struct (ref row p, wr, int vi, w, min,
                                        [1 : n] int c);
 prio dif = 1;
 op dif = (ref row a, b) bool: a isnt b;
 [0 : m] row a; ref row r, r0 := a[0], r1, r2 := a[m],
                                        t, t1, t2;
 ref [ ] int vj := c of r0;
 for i to m do p of a[i - 1] := a[i];
 vi of a[i] := w of a[i] := 0; c of a[i] := cost[i];
 if (u := (c of a[i])[1]) < f then f := u; t := a[i] fi od;
co Заполнен массив структур и найдено назначение для
```

первого столбца. Осталось запомнить это назначение, после чего пойдет основной цикл добавления столбцов **co**

```
p of r2:=r0; w of t:=1; vj[1]:=f;
for j from 2 to n do g:=0; d:=maxint; r:=r0; r1:=r2;
while r dif r1 do t:=r; wr of (r:=p of r):=r0;
if (u:=min of r:=vi of r+(c of r)[j]<d then
   d:=u; t1:=t fi od;
```

co Сейчас перечеркнутых строк нет, поэтому $r1 = r2$. Выбрали минимальный элемент нового столбца, сформировав попутно значения *min* и запомнили в $t1$ строку, предшествующую выбранной (так удобнее) **co**

```
while (s:=w of (t:=wr of t1))>0 do
  if t dif r1 then p of t1:=p of t;
   p of t:=p of r2; r2:=p of r2:=t
   else r1:=t1 fi;
```

co Выбранный элемент назначается дублером. Соответствующая строка переводится во вторую часть общего списка (случай, когда она стоит в конце первой части, технически более прост) **co**

```
vs:=vj[s]; t2:=r0; g1:=maxint;
while t2 dif r1 do r:=p of t2;
if (u1:=(c of r)[s]-vs+vj of r)<(u:=min of r-g)
then u:=u1; wr of r:=t fi;
if u<g1 then g1:=u; t1:=t2 fi;
min of r:=u, t2:=r od;
```

co Находим минимальный неперечеркнутый элемент, попутно корректируя значения *min*. Если этот минимум оказывается положительным, корректируем потенциалы перечеркнутых строк и соответствующих им столбцов **co**

```
if (g:=g1)>0 then d+:=g;
while r dif r2 do vj[w of (r:=p of r)]+:=g;
                                  vi of r+:=g od
fi od;
```

co Завершающие действия цикла — переброска назначений,

включение нового назначения, фиксация потенциала последнего столбца и изменение целевой функции **co**

```
while (r:=wr of t) dif r0 do w of t:=w of r; t:=r od;
w of t:=j; vj[j]:=d; f+:=d od;
for i to m do if (d:=col[i]:=w of a[i])>0
                                    then row[d]:=i fi od;
f co оформление результата co
end
```

Этот вариант венгерского метода был разработан А. Б. Грибовым и показал себя очень хорошо в экспериментальных расчетах (исходная программа Грибова записана на алголе-60). Высоко эффективной является и программа А. Б. Грибова для транспортной задачи, построенная на тех же принципах.

Следует, однако, иметь в виду, что венгерский метод существенно использует возможность быстрого обращения к любому элементу матрицы стоимостей и его буквальное использование при решении задач большого размера (требующих использования внешней памяти) может оказаться неэффективным.

§ 11. Матрицы из нулей и единиц

Матрица смежности $r[M_1, M_2]$ простого графа может рассматриваться как набор характеристических векторов некоторых подмножеств из M_1 (или как набор характеристических векторов подмножеств из M_2, если рассматривать строки этой матрицы).

В последнее время появилось много работ и постановок задач, связанных с такими объектами. Мы перечислим здесь некоторые из вопросов, которые представляют практический интерес.

Прежде всего терминология.

Множество M_1 с набором его подмножеств $\{R_i\}$, $i \in M_2$, называется *гиперграфом*. В комбинаторной топологии такой же объект называется *комплексом остовов*. Если с каждым множеством R_i гиперграф содержит любое его подмножество, он называется *матроидом* (или соответственно *полным комплексом*).

Естественно возникают также постановки задач.

Задача о различных представителях.

Пусть $|M_2| = m$. Найти в M_1 такое подмножество R, чтобы $|R| = m$ и существовало взаимно однозначное отображение T: $R \to M_2$, при котором $i \in R_{T_i}$ для всех $i \in R$.

Иными словами, требуется выбрать по одному представителю из каждого подмножества так, чтобы все они оказались разными.

Проще решать эту задачу, если говорить о множестве R с максимально возможным числом элементов (это число не превзойдет m). Тогда можно просто убедиться в том, что мы получаем задачу о максимальном паросочетании в простом графе, которая нам уже знакома.

Более сложной оказывается следующая задача.

Задача о наилучшем покрытии.

Пусть задан положительный вектор $c\,[M_2]$. Требуется найти такое подмножество $S \subset M_2$, чтобы

$$U_{i \in S}\, R_i = M_1$$

и величина

$$\mathbf{1}\,[S] \times c\,[S]$$

принимала минимальное значение.

Вариантом этой задачи является задача о наилучшем *разбиении*, в которой требуется дополнительно, чтобы множества R_i, $i \in S$, были дизъюнктны и, следовательно, образовывали разбиение множества M_1. Интересно применение методов линейного программирования для решения этой задачи. Рассмотрим следующую задачу.

Найти вектор $x\,[M_2]$, удовлетворяющий условиям

$$r\,[M_1, M_2] \times x\,[M_2] = \mathbf{1}\,[M_1], \tag{37}$$

$$x\,[M_2] \geqslant \mathbf{0}\,[M_2] \tag{38}$$

и минимизирующий

$$c\,[M_2] \times x\,[M_2]. \tag{39}$$

Предложение 6. *Любое допустимое целочисленное решение задачи* (37) — (39) *является характеристическим вектором некоторого подмножества* $S \subset M_2$.

Базис, которому соответствует целочисленное допустимое базисное решение, можно описать следующим образом.

Он состоит из двух множеств

$$S_1 = \{j \mid x[j] = 1\} \quad \text{и} \quad S_0 = \{j \mid x[j] = 0\}.$$

При этом, очевидно, ни для каких дизъюнктных подмножеств базиса R_1, $R_0 \subset S_1 \cup S_0$ не должно выполняться

$$r[M_1, R_1] \times 1[R_1] = r[M_1, R_0] \times 1[R_0].$$

Можно несколько изменить схему вычислений метода последовательных улучшений и, решая задачу (37) — (39), выполнять переходы от одного базисного решения к другому только в том случае, если получающееся решение окажется целочисленным.

Теорема 13 (Трубина). *Описанный метод решения задачи* (37) — (39) *дает оптимальное решение.*

Доказательство этой теоремы здесь проводиться не будет.

Неожиданные экстремальные задачи, связанные с такими матрицами, возникают при попытках экономного решения линейных систем. Вот пример такой задачи.

Задача о порядке исключения переменных.

Найти взаимно однозначное отображение $T_2: M_2 \to 1:|M_2|$ и отображение $T_1 : M_1 \to 1 : \infty$ таким образом, чтобы $T_1 i \neq T_1 k$ при $i \neq k$,

$$T_1 i \geqslant \max\{T_2 j \mid r[i, j] = 1\}$$

и достигался минимум величины

$$\max\{T_1 i \mid i \in M_1\}.$$

Название этой задачи связано вот с какой ее интерпретацией: при решении линейной системы

$$\alpha[M_1, M_2] \times y[M_2] = \beta[M_1]$$

можно сопоставить матрице этой системы «характеристическую матрицу» $r[M_1, M_2]$, где $r[i, j] = \delta(\alpha[i, j] \neq 0)$. Пусть имеется допустимое решение нашей экстремальной задачи (T_1, T_2). Отображение T_1 задает упорядочение на $M_1 : i \prec k \Leftrightarrow T_1 i < T_2 k$. Единственным образом определяется неулучшаемое отображение T_1^*, согласованное с этим упорядочением

$$T_1^* i = \max\{\max\{T_2 j \mid r[i, j] = 1\}, 1 + \max\{T_1^* k \mid k \prec i\}\}.$$

Теперь легко видеть, что равенства, соответствующие индексам i, для которых

$$T_1^* i = \max \{T_2 j \mid r[i, j] = 1\},$$

могут быть использованы для выражения переменных (с тем же номером) через переменные с меньшими номерами.

Задачи последовательного редуцирования комплекса остовов в последнее время стали появляться в различных областях математики.

Одно из неожиданных применений таких задач встретится нам в конце гл. 6.

Многочисленные экстремальные задачи комбинаторного типа могут быть записаны как задачи линейного программирования с матрицами из нулей и единиц и с дополнительными условиями целочисленности. Хотя теорема 13 дает метод для решения некоторых из таких задач, этот метод оказывается для большинства задач недостаточно эффективным.

В следующей главе будут описаны некоторые комбинаторные задачи, в которых затруднительно использование методов линейного программирования, и будут предложены методы их решения, основанные на улучшенном переборе.

ГЛАВА 5

МНОГОЭКСТРЕМАЛЬНЫЕ ЗАДАЧИ НА ГРАФАХ

§ 1. Характеристические числа графа

Под многоэкстремальными задачами понимаются такие задачи, в которых при выбранном определении окрестности не исключается возможность существования локальных экстремумов, отличных от глобального, т. е. таких допустимых решений, которые не обеспечивают глобального экстремума, но в окрестности которых нет лучших с точки зрения рассматриваемой целевой функции решений. Многочисленные задачи такого типа возникают при введении всякого рода численных характеристик графа. В этом параграфе будут описаны некоторые из них.

Первым следует назвать уже встречавшееся нам число компонент связности графа. Это число p может быть найдено без решения каких-либо экстремальных задач. Не связано с экстремальными задачами и следующее число — *число независимых циклов*. Оно легко выражается через число компонент связности и равно $\nu = |N| - |M| + p$ Используя предложение 5 гл. 3, легко показать, что ν совпадает при таком определении с рангом матрицы, составленной из циклических векторов графа.

Следующим важным числом является *хроматическое* (или *цветовое*) число. Граф $\langle M, N\rangle$ называется k-хроматическим, если каждой его вершине можно приписать число от 1 до k (цвет) таким образом, что начало и конец каждой дуги получают различные номера (будут выкрашены в различный цвет). Наименьшее k, при котором граф является k-хроматическим, и называется хроматическим числом. Оно обозначается через γ.

Теорема 1 (Кёнига). *Граф является 2-хроматическим в том и только в том случае, если он не содержит циклов нечетной длины.*

Доказательство. Граф, имеющий цикл нечетной длины, очевидно, не может быть раскрашен в два цвета, так как среди вершин цикла (число которых нечетно) окажутся одноцветные соседние вершины.

Пусть граф не содержит циклов нечетной длины. Можно предполагать его связным, так как каждую компоненту связности можно раскрашивать отдельно. Выберем дерево в графе, цвет одной из вершин и, двигаясь от нее по дереву, раскрасим все остальные вершины. Получившаяся раскраска будет непротиворечивой с точки зрения остальных дуг, так как их добавление замыкает цикл четной длины, на котором раскраска вершин является чередующейся, и значит, граничные вершины имеют разные цвета. ▲

Цветовое число может быть получено в результате решения следующей задачи линейного программирования в целых числах.

Пусть k — число, при котором граф $\langle M, N\rangle$ заведомо является k-хроматическим, $r > |M|$. Требуется найти неотрицательную целочисленную матрицу $x[1{:}k, M]$, удовлетворяющую условиям

$$\mathbf{1}[1:k] \times x[1:k, M] = \mathbf{1}[M],$$

$$x[1{:}k, i(u)] + x[1{:}k, j(u)] \leqslant \mathbf{1}[1{:}k], \quad u \in N$$

и минимизирующую линейную функцию

$$\Sigma_{j \in 1:k,\, i \in M} r^j \times x[j, i].$$

Переменная $x[j, i]$ здесь равна единице в том случае, если для вершины i выбирается цвет j (для каждой вершины выбирается один цвет, соседние вершины имеют различный цвет). Величина r в целевой функции выбрана таким образом, что любое решение x, использующее l цветов, лучше решения y, использующего $l + 1$ цветов.

Действительно,

$$\Sigma_{j \in 1:k,\, i \in M} r^j \times x[j, i] \leqslant r^l \times \Sigma_{j \in 1:k,\, i \in M} x[j, i] =$$
$$= |M| \times r^l < r^{l+1} \leqslant \Sigma_{j \in 1:k,\, i \in M} r^j \times y[j, i].$$

Следующие два числа — числа внешней и внутренней устойчивости, определения которых требуют введения еще некоторых понятий.

Множество M' вершин графа $\langle M, N\rangle$ называется *внутренне устойчивым*, если никакие две его вершины не смежны, т. е. не являются одновременно граничными вершинами какой-либо дуги В правильно раскрашенном графе множества вершин одного цвета образуют внутренне устойчивые множества. *Числом внутренней устойчивости* α называется максимальное число вершин в M'.

Очевидно, что $\alpha \times \gamma \geqslant |M|$.

Множество M' называется *внешне устойчивым*, если каждая вершина из $M \setminus M'$ смежна с какой-либо вершиной из M'.

Числом внешней устойчивости β называется минимальное число вершин в M'.

Эти числа также можно получить, решая соответствующие целочисленные задачи линейного программирования. Число внутренней устойчивости получается из задачи:

Максимизировать $\mathbf{1}[M] \times x[M]$ при условиях

$$x[i(u)] + x[j(u)] \leqslant 1, \quad u \in N,$$
$$x[i] \in 0:1, \quad i \in M.$$

Число внешней устойчивости — из задачи:

Минимизировать $\mathbf{1}[M] \times y[M]$ при условиях

$$y[i] \in 0:1, \quad i \in M,$$
$$\mathbf{1}[M_i] \times y[M_i] \geqslant 1, \quad i \in M,$$

где M_i — множество вершин, связанных дугой с i, к которому добавлено само i.

Получающиеся при этом векторы являются характеристическими векторами множеств соответственно внутренней и внешней устойчивости.

Отметим, что во всех случаях мы пришли к задаче линейного программирования с дополнительными условиями целочисленности переменных. Более того, во встретившихся нам задачах матрица ограничений состояла из нулей и единиц.

§ 2. Эйлеровы и гамильтоновы пути и циклы

Интересные задачи связаны с построением полных обходов графа. В термин «полный обход» мы можем вкладывать различное содержание, и это приводит к различным задачам. Более простой из них является задача построения обхода, содержащего все дуги графа.

Замкнутая цепь, содержащая по одному разу все дуги графа, называется *эйлеровым циклом* [1]).

[1]) Мы снимаем здесь предположение о том, что в цепь каждая вершина графа входит не более одного раза.

Теорема 2. *Связный граф* $\langle M, N\rangle$ *обладает эйлеровым циклом в том и только в том случае, если в графе нет вершин, которым инцидентно нечетное число дуг.*

Доказательство. В одну сторону доказательство совершенно очевидно. Если в графе есть вершина, которой инцидентно нечетное число дуг («нечетная» вершина), все дуги не смогут войти в эйлеров цикл, поскольку каждое прохождение вершины использует по две инцидентных вершине дуги.

Предположим, что нечетных вершин в графе нет. Построим какой-либо цикл и множество его вершин обозначим через M_0, а множество дуг через N_0. Положим $N_1 = N \setminus N_0$. В графе $\langle M, N_1\rangle$ нет нечетных вершин. Кроме того, найдутся вершины из M_0, которым инцидентны дуги из N_1. Действительно, если бы таких дуг не было, то не было бы связи между $\langle M_0, N_0\rangle$ и $\langle M \setminus M_0, N_1\rangle$, вопреки предположению о связности графа $\langle M, N\rangle$.

Выберем вершину $i \in M_0$ и построим из дуг N_1 цикл, содержащий i. Встроим дуги из этого цикла в замкнутую цепь, соответственно пополнив M_0 и N_0 и снова полагая $N_1 = N \setminus N_0$. Этот процесс можно, очевидно, повторять до полного исчерпания N_1. Построенная в результате замкнутая цепь и будет искомым эйлеровым циклом. ▲

Замкнутый путь, содержащий все дуги графа, называется *эйлеровым контуром.*

Теорема 3. *Для того чтобы в связном графе* $\langle M, N\rangle$ *существовал эйлеров контур, необходимо и достаточно, чтобы*

$$|N_i^+| = |N_i^-| \quad \textit{для всех } i \in M. \tag{1}$$

Доказательство. При выполнении условия теоремы вектор $\mathbf{1}[N]$ является решением уравнения

$$a[M, N] \times x[N] = \mathbf{0}[N]$$

и, следовательно, представим в виде выпуклой комбинации контурных векторов. Легко показать, что $\mathbf{1}[N]$ является суммой нескольких контурных векторов. Покажем, что из соответствующих им контуров можно составить замкнутый путь. Возьмем один из контуров и превратим его в замкнутый путь. Так как граф связен, то найдется вершина, лежащая на этом пути и принадлежащая одному из остальных контуров. Разорвем путь в этой вершине и вставим в разрыв обход контура. Мы снова получим замкнутый путь, для

которого сможем продолжить то же рассуждение до полного исчерпания набора контуров.

Поскольку контуры содержали все дуги графа, то получившийся замкнутый путь является эйлеровым контуром. Необходимость условия очевидна. ▲

Следствие. *Связный граф, удовлетворяющий условию* (1), *вполне связен.*

Цикл (контур, путь) называется *гамильтоновым*, если через каждую вершину графа он проходит ровно по одному разу.

Назовем граф *полным*, если любая пара его вершин соединена дугой хотя бы в одном направлении.

Теорема 4 (Кёнига). *В полном графе всегда существует гамильтонов путь.*

Доказательство. Пусть $i_1, i_2, \ldots, i_k$ — некоторый путь и j не входит в него. Покажем, что путь можно расширить добавлением j. Если не существует дуги, ведущей из j в i_1, то имеется дуга из i_1 в j. При наличии дуги из j в i_2 можно поставить j между i_1 и i_2. В противном случае найдется дуга из i_2 в j. Повторяя этот процесс, мы включим j между какими-либо двумя вершинами пути либо убедимся в том, что есть дуга из i_k в j и припишем j как последнюю вершину пути.

Последовательно расширяя путь, мы включим в него все вершины графа. ▲

Нахождение гамильтонова пути (или контура) в произвольном графе может быть приведено к задаче линейного программирования с условием целочисленности переменных. Мы сформулируем, однако, более важную и общую задачу — задачу нахождения гамильтонова контура с наименьшей длиной.

Эта задача называется *задачей о бродячем торговце* или *задачей о коммивояжере.* При формулировке этой задачи целесообразно освободиться от трудности поиска допустимого решения и, добавив искусственные дуги, считать, что в графе имеются переходы из любой вершины в любую другую. Тогда задача принимает следующий вид.

Задача о коммивояжере.

Задана матрица $c\,[M, M]$. Требуется найти такой порядок обхода вершин (элементов множества) $i_0, i_1, \ldots, i_m = i_0$, $m = |M|$, чтобы сумма $\sum_{k \in 1:m} c\,[i_{k-1}, i_k]$ была минимальной.

Эта задача также может быть записана как задача линейного программирования с целочисленными переменными:

Найти целочисленную матрицу $x\,[M, M]$, удовлетворяющую условиям

$$x\,[M, M] \geqslant 0\,[M, M], \tag{2}$$

$$1\,[M] \times x\,[M, M] = x\,[M, M] \times 1\,[M] = 1\,[M], \tag{3}$$

$$\Sigma_{i, j \in A}\, x\,[i, j] \leqslant |\,A\,| - 1, \quad A \neq M, \quad A \subset M \tag{4}$$

и минимизирующую значение

$$\Sigma_{i, j}\, c\,[i, j] \times x\,[i, j], \tag{5}$$

Условия (2), (3) вместе с условием целочисленности определяют переставляющую матрицу. Условие (4) является условием одноцикловости этой подстановки — если бы в подстановке был цикл с множеством вершин $A \neq M$, условие (4), очевидно, не выполнялось бы.

Задачу (2) — (5) не удается использовать непосредственно для решения задачи о коммивояжере. Более успешным в решении этой и других задач сходного типа оказались методы *улучшенного перебора,* один из которых мы опишем в следующем параграфе.

§ 3. Метод ветвей и границ

Когда мы занимаемся решением дискретных задач, трудно четко сформулировать, что значит построить метод решения этой задачи. Один метод решения всегда есть — это метод, заключающийся в полном просмотре всех элементов множества.

Один из подходов заключается в том, чтобы оценить трудоемкость предлагаемого метода через какие-то внешние параметры задачи — число переменных, число ограничений, количество битов информации, описывающих все исходные данные, и т. д. Некоторые оценки такого рода уже нам встречались. Однако этот корректный подход не может последовательно проводиться в жизнь. Во-первых, параметры, по которым оценивается трудоемкость задачи, часто плохо характеризуют задачу, — среди задач с данным значением параметров есть классы «очень трудных» и «очень

легких» задач. Во-вторых, кроме принципиальных оценок нас интересует фактическая трудоемкость численных методов на задачах данного класса, на вычислительных машинах с такой-то памятью и быстродействием. Надеяться на получение априорных оценок в этом случае пока не приходится. Поэтому нашей целью сейчас является в большей степени разработка и экспериментальная проверка методов, чем оценка их трудоемкости.

Перебор множества допустимых решений часто может быть различными улучшениями доведен до высокоэффективных методов Разумеется, для успешного использования этих методов нужно в значительной мере пользоваться спецификой задачи. Мы опишем здесь несколько идей такого улучшения и несколько вариантов использования этих идей.

Большая группа методов улучшенного перебора объединяется под названием «метод ветвей и границ». Общая идея этого метода очень проста. Предположим, что мы решаем задачу:

| Найти минимум $f(x)$ по $x \in A$.

Пусть имеется некоторое разбиение множества A

$$\mathfrak{A}: A = A_0 \cup A_1 \cup \ldots \cup A_k$$

и известно, что в множестве A_0 оптимальных решений быть не может. Тогда

$$\min \{f(x) \mid x \in A\} = \min \{\min \{f(x) \mid x \in A_i\} \mid i \in 1:k\}. \quad (6)$$

Разбиение множества A на подмножества преследует несколько целей. Прежде всего, выделение множества A_0 сокращает задачу, и мы, конечно, стремимся как можно большую часть множества A передать в A_0. Далее, разбиение на подмножества A_i оставшейся части A может упростить задачу нахождения минимума $f(x)$ (во всяком случае, полезно выбирать такие разбиения, при которых задача упрощается).

Отметим два важных способа увеличения множества A_0.

Если мы знаем, что для двух элементов разбиения A_i и A_j имеет место

$$\min \{f(x) \mid x \in A_i\} > \min \{f(x) \mid x \in A_j\}, \quad (7)$$

то в A_i заведомо оптимальных решений нет и $A_0 := A_0 \cup A_j$.

Если мы имеем какое-либо допустимое решение x_0 и для некоторого $i \in 1 : k$ установили, что

$$\min \{f(x) \mid x \in A_i\} > f(x_0), \tag{8}$$

то в A_i заведомо оптимальных решений нет и $A_0 := A_0 \cup A_i$.

Для того чтобы получить неравенство типа (8), полезно использовать оценки снизу для минимума $f(x)$ на A_i. Один из методов получения таких оценок — использование расширений получающихся экстремальных задач.

Имея некоторое разбиение $\mathfrak{A}$ можно проводить его дальнейшее измельчение, в котором тот или иной элемент разбиения заменяется несколькими элементами (составляющими его разбиение). При этом последовательно уточняются оценки и находятся новые допустимые решения, а некоторые из элементов разбиения переводятся в множество A_0. Реализация этого общего принципа имеет много сложностей, часто меняющихся от задачи к задаче, и прежде чем о них говорить, рассмотрим какую-нибудь реализацию.

Покажем, как можно осуществить эти принципы на задаче о коммивояжере, иллюстрируя рассмотрение и конкретным примером. Рассмотрим задачу с $M = 1 : 8$ и с матрицей стоимостей переездов, представленной табл. 1.

Таблица 1

	1	2	3	4	5	6	7	8
1	*gz*	5	3	2	4	5	2	7
2	7	*gz*	1	6	7	8	9	6
3	3	2	*gz*	3	2	4	5	7
4	4	6	2	*gz*	1	3	6	2
5	4	5	1	2	*gz*	4	2	3
6	3	8	9	6	4	*gz*	1	5
7	1	6	6	4	2	1	*gz*	0
8	7	8	7	2	8	3	0	*gz*

Таблица 2

	1	2	3	4	5	6	7	8	
1	*gz*	3	1	0	2	2	0	5	2
2	5	*gz*	0	5	6	6	8	5	1
3	0	0	*gz*	1	0	1	3	5	2
4	2	5	1	*gz*	0	1	5	1	1
5	2	4	0	1	*gz*	2	1	2	1
6	1	7	8	5	3	*gz*	0	4	1
7	0	6	6	4	2	0	*gz*	0	
8	6	8	7	2	8	2	0	*gz*	
				1			1		

На каждом шаге вычислений одно из множеств разбиения будет делиться на два множества. Принцип деления для каждого из подмножеств в этой задаче оказывается возможным сделать единым — таким же, как для всего множества A. Множеством A в задаче о коммивояжере является множество гамильтоновых контуров или — в матричных терминах — множество одноцикловых подстановок.

Способ разбиения множества A гамильтоновых контуров на подмножества будет использоваться только один: зафиксировав дугу u, мы будем относить к одному множеству все пути, содержащие эту дугу, а к другому — все пути, не содержащие этой дуги. В первом случае матрица (и задача) может быть уменьшена, так как из $i\,(u)$ и $j\,(u)$ можно составить «вершину» i', для которой $c\,[i', i'] = \infty$, $c\,[i, i'] = c\,[i, i\,(u)]$ и $c\,[i', i] = c\,[j\,(u), i]$ при $i \neq i'$. Во втором случае можно положить $c\,[i\,(u), j\,(u)] = \infty$ и рассмотреть задачу о коммивояжере с оставшимися дугами. Итак, в обоих случаях мы получаем задачу о коммивояжере, и поэтому нам достаточно иметь метод оценки минимума лишь для этой задачи.

Отметим, что при неотрицательной матрице $c\,[M, M]$ нуль будет оценкой снизу для минимума. Поэтому редукции, проводившиеся над матрицей $c\,[M, M]$, при решении задачи о назначениях венгерским методом, дают оценку снизу для минимума целевой функции [1]). Преобразуя нашу матрицу, получаем табл. 2 (справа от строк и под столбцами записаны вычитаемые).

Целевая функция уменьшена на 10, и это число и является оценкой снизу для минимума целевой функции. Отметим, что запрещение некоторых из нулевых переходов

Таблица 3

	1	2	4	5	6	7	8	
1	*gz*	0	0	2	2	0	5	
3	0	*gz*	1	0	1	3	5	
4	2	2	*gz*	0	1	5	1	
5	1	0	0	*gz*	1	0	1	1
6	1	4	5	3	*gz*	0	4	
7	0	3	4	2	0	*gz*	0	
8	6	5	2	8	2	0	*gz*	
					3			

Таблица 4

	1	2	3	4	5	6	7	8	
1	*gz*	3	1	0	2	2	0	5	
2	0	*gz*	*gz*	0	1	1	3	0	5
3	0	0	*gz*	1	0	1	3	5	
4	2	5	1	*gz*	0	1	5	1	
5	2	4	0	1	*gz*	2	1	2	
6	1	7	8	5	3	*gz*	0	4	
7	0	6	6	4	2	0	*gz*	0	
8	6	8	7	2	8	2	0	*gz*	

[1]) Задача о назначениях может рассматриваться как расширение задачи о коммивояжере. Очень эффективной оценкой для минимума является минимум целевой функции в задаче о назначениях. Но его получение слишком трудоемко. Используемая нами оценка представляет собой не что иное, как допустимое решение задачи, двойственной задаче о назначениях.

повышает эту оценку. Так, запрет перехода $7 \to 6$ позволяет вычесть 1 из 6-го столбца, запрет $8 \to 7$ вычесть 2 из 8-й строки. Больше всего (на 5) повышается оценка при запрете перехода $2 \to 3$. Этот переход мы и используем для дробления A. Итак, получаем два варианта (табл. 3 и 4). В варианте A_1 запрещен переход $3 \to 2$, так как он замыкает контур. Вариант A_1 имеет меньшую оценку и развивается дальше. Больше всего повышает оценку удаление перехода

Таблица 5

	1	2	4	5	6	8	
1	*gz*	0	0	2	2	4	
3	0	*gz*	1	0	1	4	
4	2	2	*gz*	0	1	0	
5	1	0	0	*gz*	1	0	
6	0	3	4	2	*gz*	2	1
7	0	3	4	2	0	*gz*	
							1

Таблица 6

	1	2	4	5	6	7	8	
1	*gz*	0	0	2	2	0	5	
3	0	*gz*	1	0	1	3	5	
4	2	2	*gz*	0	1	5	1	
5	1	0	0	*gz*	1	0	1	
6	1	4	5	3	*gz*	0	4	
7	0	3	4	2	0	*gz*	0	
8	4	3	0	6	0	*gz*	*gz*	2

$8 \to 7$ (табл. 5 и 6). Теперь наименьшую оценку имеет вариант A_2, который целесообразно дробить по переходу $3 \to 2$ (табл. 7 и 8). Делим далее A_{21} по переходу $8 \to 7$ (табл. 9 и 10). Сейчас

$$A = A_{11} \cup A_{12} \cup A_{211} \cup A_{212} \cup A_{22}.$$

Делим A_{11} по $6 \to 1$ (табл. 11 и 12).
Делим A_{111} по $7 \to 6$ (табл. 13 и 14).
Наконец, делим A_{1111} по $3 \to 5$ (табл. 15 и 16).

Таблица 7

	1	3	4	5	6	7	8
1	*gz*	1	0	2	2	0	5
2	0	*gz*	0	1	1	3	0
4	2	1	*gz*	0	1	5	1
5	2	0	1	*gz*	2	1	2
6	1	8	5	3	*gz*	0	4
7	0	6	4	2	0	*gz*	0
8	6	7	2	8	2	0	*gz*

Таблица 8

	1	2	3	4	5	6	7	8
1	*gz*	0	1	0	2	2	0	5
2	0	*gz*	*gz*	0	1	1	3	0
3	0	*gz*	*gz*	1	0	1	3	5
4	2	2	1	*gz*	0	1	5	1
5	2	1	0	1	*gz*	2	1	2
6	1	4	8	5	3	*gz*	0	4
7	0	3	6	4	2	0	*gz*	0
8	6	5	7	2	8	2	0	*gz*

Таблица 9

	1	3	4	5	6	8	
1	*gz*	1	0	2	2	5	
2	0	*gz*	0	1	1	0	
4	2	1	*gz*	0	1	1	
5	2	0	1	*gz*	2	2	
6	0	7	4	2	*gz*	3	1
7	0	6	4	2	0	*gz*	

Таблица 10

	1	3	4	5	6	7	8	
1	*gz*	1	0	2	2	0	5	
2	0	*gz*	0	1	1	3	0	
4	2	1	*gz*	0	1	5	1	
5	2	0	1	*gz*	2	1	2	
6	1	8	5	3	*gz*	0	4	
7	0	6	4	2	0	*gz*	0	
8	4	5	0	6	0	*gz*	*gz*	2

Таблица 11

	2	4	5	6	8
1	0	0	2	*gz*	4
3	*gz*	1	0	1	4
4	2	*gz*	0	1	0
5	0	0	*gz*	1	0
7	3	4	2	0	*gz*

Таблица 12

	1	2	4	5	6	8	
1	*gz*	0	0	2	2	4	
3	0	*gz*	1	0	1	4	
4	2	2	*gz*	0	1	0	
5	1	0	0	*gz*	1	0	
6	*gz*	1	2	0	*gz*	0	2
7	0	3	4	2	0	*gz*	

Таблица 13

	2	4	5	8
1	0	0	2	*gz*
3	*gz*	1	0	1
4	2	*gz*	0	0
5	0	0	*gz*	0

Таблица 14

	2	4	5	6	8	
1	0	0	2	*gz*	4	
3	*gz*	1	0	0	4	
4	2	*gz*	0	0	0	
5	0	0	*gz*	0	0	
7	1	2	0	*gz*	*gz*	2

1

Таблица 15

	2	4	8
1	0	0	*gz*
4	2	*gz*	0
5	*gz*	0	0

Таблица 16

	2	4	5	8	
1	0	0	2	*gz*	
3	*gz*	0	*gz*	0	1
4	2	*gz*	0	0	
5	0	0	*gz*	0	

Теперь очевидно оптимальное решение в множестве A_{11111}: $2 \to 3 \to 5 \to 4 \to 8 \to 7 \to 6 \to 1 \to 2$. Все остальные множества имеют оценку минимума не меньше 16 и поэтому не могут дать решений с меньшим значением целевой функции. Получившуюся схему ветвлений удобно представить деревом, изображенным на рис. 63.

Описанный метод называется методом ветвей и границ, так как в нем поиск оптимального решения идет по ветвям дерева вариантов, и при этом ветвлении используются границы — оценки.

При выборе метода мы можем распоряжаться следующими его характерными особенностями:

— способом оценки минимума целевой функции на элементе разбиения,

— способом дробления элемента разбиения,

— способом отнесения элементов разбиения к A_0,

— стратегией выбора варианта для ветвления и конкретных параметров ветвления.

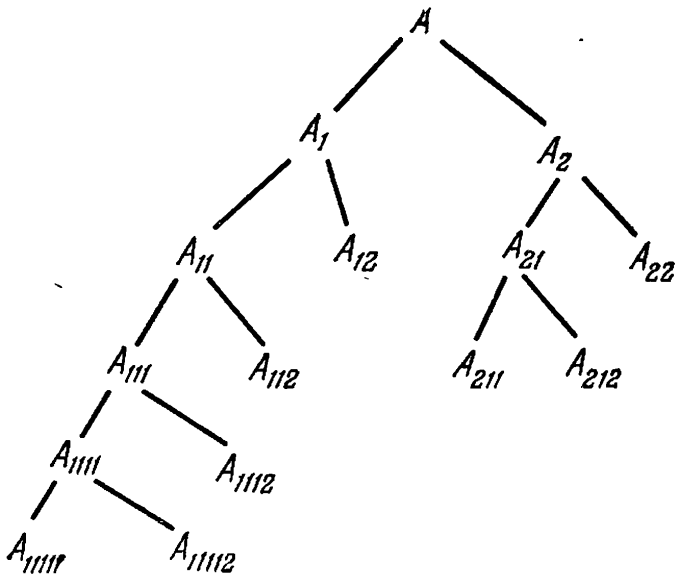

Рис. 63. Дерево вариантов в задаче о коммивояжере.

Так, в задаче о коммивояжере в качестве оценок можно было использовать и решение задачи о назначении, и решение задачи о кратчайшем иерархическом дереве. Способ дробления в этой задаче определяется, по-видимому, однозначно — разумных замен ему не видно. Способ «отбраковки» элементов мы использовали один — по оценке минимума. Эффективность этой отбраковки повысится, если взять достаточно хорошее допустимое решение и сравнивать с ним оценки (действовать по формуле (8)). Например, если строить обход вершин, выбрав какую-то вершину для начала и переходя из каждой вершины в самую близкую из оставшихся (по таблице 2), мы получим при начальной вершине 1 путь

$$1 \to 4 \to 5 \to 3 \to 2 \to 8 \to 7 \to 6 \to 1$$

стоимости 18, что позволит нам сразу забраковать подмножества A_{112}, A_{1112}, A_{22}.

Важнейшие особенности метода — выбор варианта для ветвления и общая политика ветвления. Главная цель ветвления — выделение «неперспективных» частей множества A, частей с большими оценками, до которых скорее всего перебор не должен и доходить. С этой точки зрения тот выбор решающей дуги, который описан выше, обладает высокими вычислительными достоинствами.

Что же касается политики ветвления, то ясно, что трудно построить вычислительный процесс, в котором количество крупных записей о возможных продолжениях процесса менялось бы и могло сильно вырастать. Так, в нашей задаче о коммивояжере каждый элемент разбиения A_α представлен информацией о принятых решениях, оценкой и матрицей получившихся стоимостей. Часть этой информации не является необходимой — она может быть восстановлена по другим данным, но удобна для проведения вычислений. Например, вместо матрицы можно хранить несколько линейных массивов и первоначальную матрицу c $[M, M]$. В том случае, когда элемент разбиения выбран для дальнейшего дробления и требуется его матрица, она может быть воссоздана по этой информации.

Однако такое сокращение необходимой информации приводит к увеличению вычислительной работы. Для того чтобы реже переходить с одной матрицы на другую, можно использовать более устойчивые политики выбора дробимых множеств.

Один из таких вариантов, оказавшийся очень эффективным именно в задаче о коммивояжере, заключается в том, что в качестве дробимого элемента на каждом шаге выбирается одно из подмножеств элемента, дробившегося на предыдущем шаге. После того как развитие этой линии закончено, оценка минимума оказалась слишком высокой или в рабочем элементе разбиения найдено оптимальное для него решение, — из остальных элементов разбиения выбирается элемент с наименьшей оценкой, информация о нем приводится в рабочее состояние и развивается новая линия.

Другой вариант, который особенно удобен с точки зрения хранения информации о вычислительном процессе, — это вариант одностороннего обхода дерева вариантов. В этом варианте при каждом разбиении рабочего элемента для продолжения выбирается один из получившихся «осколков» так же, как и раньше, но при завершении работы над

осколком происходит возвращение к предыдущему шагу для исследования второго осколка (или к ближайшему из предыдущих шагов, где еще сохранилась альтернатива).

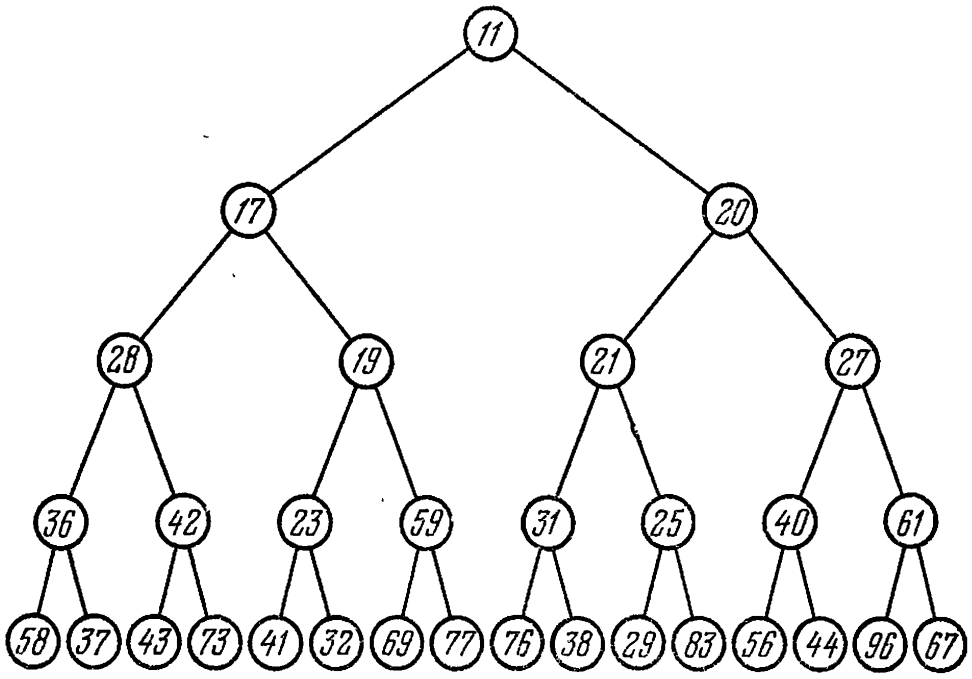

Рис. 64. Пример для иллюстрации ветвлений.

В рассмотренном нами примере это соответствует последовательному рассмотрению следующих вариантов:

$\emptyset$ — ветвление,
1 — ветвление,
11 — ветвление,
111 — ветвление,
1111 — ветвление,
11111 — получено допустимое решение со значением 16 (рекорд), переход к альтернативе,
11112 — оценка хуже рекорда. Возврат на один шаг с переходом к альтернативе,
1112 — то же действие,
112 — то же действие,
12 — то же действие,
2 — ветвление,
21 — ветвление,
211 — оценка не лучше рекорда. Возврат с переходом к альтернативе,
212 — то же действие,
22 — оценка хуже рекорда. Альтернатив нет. Конец перебора.

Для того чтобы сопоставить описанные три политики ветвления, рассмотрим пример, изображенный на рис. 64. Здесь показано полное дерево вариантов, в котором вершины самого нижнего уровня — допустимые решения, а все прочие вершины — элементы разбиения. Номера вершин являются значениями целевой функции для допустимых решений и оценками для элементов разбиения. Для краткости мы будем писать PA, когда появляется новый рекорд со значением A, $O(A, B)$ — когда отменяется несколько элементов (A, B), $A = B + C$, если A делится на B и C.

Итак, ветвление «полным фронтом»:

$$11 = 17 + 20,\quad 17 = 19 + 28,\quad 19 = 23 + 59,$$
$$20 = 21 + 27,\quad 21 = 31 + 25,\quad 23 = 41 + 32,$$
$$P32,\ O(41, 59),\ 25 = 29 + 83,\ O83,$$
$$P29,\ O31,\ 27 = 40 + 61,\ O(40, 61),$$
$$28 = 36 + 42,\ O(36, 42).$$

Ветвление «побегами»:

$$11 = 17 + 20,\quad 17 = 19 + 28,\quad 19 = 23 + 59,$$
$$23 = 41 + 32,\quad P32,\quad O(41, 59),$$
$$21 = 25 + 31,\quad 25 = 29 + 83,\quad 25 = 29 + 83,$$
$$O83,\quad P29,\quad O31,\quad 27 = 40 + 61,\quad O(40, 61),$$
$$28 = 36 + 42,\quad O(36, 42).$$

Односторонний обход дерева:

$$11 = 17 + 20,\quad 17 = 19 + 28,\quad 28 = 36 + 42,$$
$$36 = 58 + 37,\quad P58,\quad P37,\quad O4,\quad 19 = 23 + 59,$$
$$23 = 41 + 32,\quad O41,\quad P32,\quad O59,\quad 20 = 21 + 27,$$
$$21 = 31 + 25,\quad 31 = 76 + 38,\quad O(7, 38),$$
$$25 = 29 + 83,\quad P29,\quad O83,\quad 27 = 40 + 61,\quad O(40, 61),$$

Как мы видим, политика полного фронта дает меньше всего ветвлений, а односторонний обход — больше всего. Это общая закономерность (что легко показать). Однако в вычислительном отношении метод одностороннего обхода представляется настолько удобным, что мы опишем здесь общую схему его реализации.

Чтобы упростить запись введем сразу некоторые обязательные переменные — целочисленную переменную *level*, указывающую уровень перебора — количество звеньев в пу-

ти от корня дерева до текущей вершины, *rec* — «рекорд» — наилучшее из встретившихся значений целевой функции, *est* — оценка минимума на текущей вершине дерева.

Состоянием мы будем называть информацию о вершине вместе с информацией о возможности продолжений.

Обозначим через *bestsol* лучшее из встретившихся допустимых решений ($f(bestsol) = rec$), через *solinit* — начальное допустимое решение, через *sol* — допустимое решение, встречающееся в ходе работы алгоритма.

На начальных шагах работы над алгоритмом полезно использовать полуалгольскую запись вычислительного процесса, заменяя еще не готовые части алгоритма словесными вставками (квазиоператорами).

С помощью такой смеси схема одностороннего обхода примет следующий вид:

```
        bestsol: = solinit; rec: = f (solinit); level: = 0;
        определение состояния соответствующего корню;
mkest:  вычисление est;
        if найдено допустимое решение sol с f (sol) = est
        then begin
          if est < rec then begin bestsol: = sol; rec: = est end
            end
          if est ⩾ rec then go to mkback;
mkforw: выбор продолжения; изменение состояния;
        level: = level + 1; go to mkest;
mkback: if level > 0 then
        begin изменение состояния отменой последнего
        продолжения; level: = level — 1; go to mkback
         end
        else begin изменение последнего продолжения;
                go to mkest end;
mkfin:
```

Ниже приводится процедура, в которой эта схема реализуется для задачи о коммивояжере. Для запоминания решения выделен вектор $sol\,[1 : m]$, в котором $sol\,[i] = j$, если из i нужно перейти в j. На промежуточных шагах процесса мы имеем «частичное решение» — граф, состоящий из нескольких отдельных путей. Число этих путей обозначено через *mc*, их концы выписаны в векторе $ni\,[1 : mc]$, их начала (в другом порядке) — в векторе $nj\,[1 : mc]$.

Для того чтобы связать начала с концами вводится вектор концов *end* [1 : *m*], в котором *end* [*i*] — конец пути, начинающегося в *i*. Такой смысл имеют только те компоненты, которым соответствуют начала из массива *nj*. Аналогично вводится массив начал *beg* [1 : *m*].

Для того чтобы матрица *c* [1 : *m*, 1 : *m*] оставалась неизменной, вычитаемые по строкам и столбцам можно хранить в отдельных массивах (*ai* и *aj*). Наконец, траектория ветвления запоминается в магазине *st*, где на каждый шаг ветвления приходится по два числа — номера строки и столбца перехода, по которому производится ветвление. При этом номер строки хранится со своим знаком, если переход запрещается, и со знаком минус, если переход включается в решение (это — условность, могло быть и наоборот). *nst* показывает заполненность магазина:

```
procedure travsal (m, c, solinit, gz, bestsol, rec);
    value m, gz; integer m, rec, gz;
    integer array c, solinit, bestsol;
begin integer i, j, ib, jb, ic, jc, mc, nst, level, est, f1, f2, f3, f4;
    integer array ni, nj, ei, ej, ai, aj, beg,
                                  end, sol [1 : m], st [1 : 2 × m × m];
    rec: = est: = level: = 0; mc: = m; nst: = 1;
    for i: = 1 step 1 until m do
      begin ni [i]: = nj [i]: = end [i]: = beg [i]: = i;
        ai [i]: = aj [i]: = 0; c [i, i]: = gz;
        j: = bestsol [i]: = solinit [i]; rec: = rec + c [i, j] end;
mkest: for ic: = 1 step 1 until mc do
    begin i: = ni [ic]; f1: = gz;
      for jc: = 1 step 1 until mc do
        begin j: = nj [jc]; f2: = c [i, j] − ai [i] − aj [j];
          if f2 < f1 then f1: = f2 end;
        ai [i]: = ai [i] + f1; est: = est + f1
    end ic;
    for jc: = 1 step 1 until mc do
      begin j: = nj [jc]; f2: = f1: = gz;
        for ic: = 1 step 1 until mc do
          begin i: = ni [ic]; f3: = c [i, j] − ai [i] − aj [j];
            if f3 < f1 then begin f2: = f1; f1: = f3 end
            else if f3 < f2 then f2: = f3 end ic;
        aj [j]: = aj [j] + f1; est: = est + f1; ej [j]: = f2 − f1
      end jc;
```

```
    if mc = 2 ∧ rec > est then
      begin rec: = est; sol[ni[1]]: = beg[ni[2]];
        sol[ni[2]]: = beg[ni[1]];
        for i: = 1 step 1 until m do bestsol[i]: = sol[i] end;
    if est ≥ rec then go to mkback;
    f4: = —1;
    for ic: = 1 step 1 until mc do
      begin i: = ni[ic]; f1: = f2: = gz;
        for jc: = 1 step 1 until mc do
          begin j: = nj[jc]; f3: = c[i, j] — ai[i] — aj[j];
            if f3 < f1 then begin f2: = f1; f1: = f3 end
            else if f3 < f2 then f2: = f3
          end jc; ei[i]: = f2: = f2 — f1;
        for jc: = 1 step 1 until mc do
          begin j: = nj[jc];
            if c[i, j] — ai[i] — aj[j] = 0 then
                                        begin f3: = f2 + ej[j];
            if f3 < f4 then begin ib: = ic;
                                        jb: = jc; f4: = f3 end
    end end end ic;
    i: = ni[ib]; st[nst]: = — i; st[nst + 1]: = j: = nj[jb];
                                        nst: = nst + 2;
    ni[ib]: = ni[mc]; nj[jb]: = nj[mc]; mc: = mc — 1;
    sol[i]: = j; f4: = beg[i]; f1: = end[j];
                                        end[f4]: = f1; beg[f1]: = f4;
    c[f1, f4]: = c[f1, f4] + gz; level: = level + 1; go to mkest;
mkback: if level > 0 then
      begin nst: = nst — 2; i: = st[nst]; j: = st[nst + 1];
      if i > 0 then begin c[i, j]: = c[i, j] — gz;
                                        level: = level — 1;
                    go to mkback end
      else begin st[nst]: = i: = — i; f4: = beg[i]; f1: = end[f4];
            c[i, j]: = c[i, j] + gz; end[f4]: = i; end[j]: = f1;
            c[f1, f4]: = c[f1, f4] — gz; beg[f1]: = j;
          mc: = mc + 1; ni[mc]: = i; nj[mc]: = j; nst: = nst + 2;
          go to mkest end
      end
end travsal
```

В следующих параграфах мы с разной степенью детализации опишем применение метода ветвей и границ еще в нескольких экстремальных задачах.

§ 4. Размыкание контуров в графе

Вот характерная комбинаторная задача, которая хорошо решается методом ветвей и границ.

Задача о размыкании контуров.

Заданы граф $\langle M, N \rangle$ и положительный вектор $w[N]$ весов дуг. Требуется выбрать такой частичный граф $\langle M, N_1 \rangle$, который не имел бы контуров и имел максимальный вес

$$1[N_1] \times w[N_1].$$

Можно говорить об удалении из графа некоторого множества дуг N' таким образом, чтобы граф $\langle M, N \setminus N' \rangle$ не имел контуров и чтобы суммарный вес удаленных дуг был минимален. Нам будет удобнее решать задачу именно в этой постановке.

Для того чтобы применить в этой задаче метод ветвей и границ мы должны описать метод получения оценки и политику ветвления. При описании оценки полезно ввести одно обобщение нашей задачи — задачу о размыкании контуров с запрещением удалять дуги из некоторого подмножества R множества N. Обозначим минимальный вес, необходимый для размыкания контуров в этой задаче через $w(N \mid R)$. Очевидно, что если $N \subset \bar{N}$, то $w(N \mid R) \leqslant w(\bar{N} \mid R)$ и если $R \subset \bar{R}$, то $w(N \mid \bar{R}) \geqslant w(N \mid R)$.

Перейдем теперь к построению оценки снизу для $w(N \mid R)$. Зафиксируем в графе $\langle M, N \rangle$ какой-либо контур C. На этом контуре имеется дуга u^*, входящая в оптимальное множество N'. Естественно, что

$$w[u^*] \geqslant \min\{w[u] \mid u \in C \setminus R\}. \tag{9}$$

Имеем

$$w(N \mid R) \geqslant \min\{w[u] \mid u \in C \setminus R\} + w(N \setminus (C \setminus R) \mid R).$$

Теперь алгоритм построения оценки выглядит так. Положим $est := 0$; $U := N$;

1° Если в графе $\langle M, U \rangle$ нет контуров, перейти к 4°.

2° Выбрать какой-либо контур C.

3° $est := est + \min\{w[u] \mid u \in C \setminus R\}$; $U := U \setminus (C \setminus R)$.

Перейти к 1°

4° Взять est в качестве оценки для $w(N \mid R)$.

Разумеется, величина оценки *est* будет зависеть от того, как выбирались в 2° контуры C.

Политика ветвления может быть различной. Проще всего при данном множестве R дуг, которые нельзя размыкать, и множестве дуг $Q = N \setminus R$, которые можно размыкать, найти какой-либо контур C, выбрать ту из его дуг $\bar{u}$, на которой достигался минимум в (9) и рассмотреть две возможности:

1) дуга $\bar{u}$ удаляется из графа,

2) дугу $\bar{u}$ запрещается удалять из графа.

При этом полезен контроль за тем, чтобы в множестве R не появлялось контуров (для этого нужно, чтобы в каждом контуре оставалась хотя бы одна дуга из Q).

В каждом промежуточном состоянии N' можно проводить дополнительный анализ свободных дуг Q. Те дуги, которые содержатся в транзитивном замыкании графа $\langle M, R\rangle$, могут прямо включаться в R, дуги, которые замыкают контуры при добавлении их в R, должны прямо удаляться.

Дальнейшие вычислительные детали здесь не представляют интереса, целесообразнее рассмотреть возможные применения этой задачи.

Прежде всего начнем с сетевых графиков. Может показаться очень удачным применением для нашей задачи упоминавшееся размыкание контуров при построении сетевого графика. Нужно предостеречь от такого применения.

Составление сетевого графика должно основываться на правильных представлениях о последовательности работ, механический выбор в контуре той связи, которая должна быть отменена, совершенно противопоказан!

Вот более серьезное применение.

Задача о круговой расстановке станков.

Имеется множество станков M, которые должны быть расставлены по кругу, на котором имеется $m = |M|$ мест. Расположение станков задается, таким образом, отображением $T : M \to 1 : m$. На станках должны обрабатываться детали различных типов из множества типов K. Для каждой детали задана последовательность операций, причем каждой операции отвечает свой станок

из M. Деталь выходит из кладовой (между m-м и первым местом) и движется по кругу в фиксированном направлении от одного станка к другому. При каждом расположении станков T каждая деталь $k \in K$ проходит целое число кругов $l(k, T)$. Требуется выбрать такое расположение станков, при котором достигался бы минимум величины

$$\varphi(T) = \sum_{k \in K} \omega_k \times l(k, T), \qquad (10)$$

где ω_k — некоторые веса, зависящие от частоты встречаемости и от физического веса деталей.

Для того чтобы формализовать эту задачу, нужно яснее представить себе, от чего зависит $l(k, T)$. Последовательность операций $i_1, i_2, \ldots, i_{r_k}$ переводится отображением T в последовательность целых чисел из $1 : m$

$$Ti_1, Ti_2, \ldots, Ti_{r_k}.$$

Назовем главными инверсиями последовательности пары соседних чисел, расположенные в убывающем порядке. Так, в последовательности

$$3, 8, 5, 9, 4, 1, 4, 5, 9, 2$$

главными инверсиями будут (8, 5), (9, 4), (4, 1) и (9, 2). Число главных инверсий в последовательности s обозначим через $I(s, T)$. Легко видеть, что число кругов, которые деталь k должна проделать при ее обработке, равно $I(s_k, T) + 1$. Вычитая из (10) постоянную $\sum_{k \in K} \omega_k$, мы приходим к задаче минимизации $\sum_{k \in K} \omega_k \times I(s_k, T)$.

Рассмотрим пару станков $i, j \in M$. Положим

$$w[(i, j)] = \sum_{k \in K} \omega_k \times r_k(i, j), \qquad (11)$$

где $r_k(i, j)$ — количество непосредственных переходов от станка i к станку j в последовательности обработки k-й детали.

Рассмотрим теперь граф $\langle K, N \rangle$, в котором множество дуг — это множество пар (i, j), которым соответствует положительный вес $w[(i, j)]$. Каждое упорядочение вершин этого графа можно представить как два действия: разрыв контуров и затем доупорядочение получающегося частич-

ного порядка. Очевидно, что главные инверсии определяются лишь выбором дуг, которые будут противоречить устанавливаемому порядку, и мы приходим к задаче о размыкании контуров.

§ 5. Кратчайшее частичное дерево

В этом параграфе мы вернемся к задаче, которая была сформулирована в § 5 гл. 4. В ней требовалось найти кратчайший связный частичный подграф $\langle M_1, N_1\rangle$ графа $\langle M, N\rangle$, содержащий множество вершин M_0. Очевидно, что этот подграф должен быть деревом.

Опишем характерные черты метода ветвей и границ в применении к этой задаче.

Прежде всего — оценка. Рассмотрим следующий процесс, являющийся модификацией процесса Прима — Краскала из § 5 гл. 4. Выберем произвольно вершину $i_0 \in M_0$ и будем присоединять к ней последовательно вершины с инцидентными им дугами по одной. Получающуюся компоненту связности мы будем называть *основной*.

В отличие от собственно алгоритма Прима — Краскала, если на каком-либо шаге мы сможем присоединить к основной компоненте вершины из $M \setminus M_0$, мы будем присоединять их в первую очередь, *считая длины соответствующих дуг нулевыми.*

В отношении вершин из M_0 будет в точности сохраняться алгоритм Прима — Краскала. Процесс завершится, когда все вершины из M_0 будут присоединены к основной компоненте связности. Обозначим полученный граф через $\langle M^*, N^*\rangle$.

Теорема 5. *Сумма длин дуг множества N^* при сделанной оговорке относительно дуг, которыми присоединялись вершины из $M^* \setminus M_0$, является оценкой снизу для минимума в задаче о кратчайшем частичном дереве.*

Доказательство. Мы повторим по существу рассуждения, делавшиеся при доказательстве теоремы 5 гл. 4. Перенумеруем дуги и вершины графа $\langle M^*, N^*\rangle$ в порядке их включения в основную компоненту связности, после чего произвольно продолжим нумерацию на все остальные дуги из N и вершины из M.

Пусть теперь $\langle M_1, N_1\rangle$ — оптимальное частичное дерево. Мы будем последовательно изменять это дерево, не увели-

чивая его длины, так, чтобы в результате получилось дерево, содержащее $\langle M^*, N^* \rangle$.

Пусть k — наименьший из номеров дуг множества $N^* \setminus N_1$. Одна из граничных вершин дуги u_k имеет номер k, другая — меньший номер и, следовательно, принадлежит M_1. Если $i_k \notin M_1$, то $i_k \notin M_0$ Дуга u_k имеет в этом случае нулевую длину и ее добавление (вместе с вершиной i_k) к графу $\langle M_1, N_1 \rangle$ не увеличивает суммарной длины дуг.

Предположим, что $i_k \in M_1$. В этом случае добавление u_k к $\langle M_1, N_1 \rangle$ приводит к появлению цикла. В этом цикле должны быть дуги с номерами, большими чем k, и, в частности, такая дуга $\bar{u}$, одна из граничных вершин которой имеет номер меньше чем k. Замена этой дуги дугой u_k по определению u_k может лишь уменьшить сумму длин дуг графа, оставив его связным.

Повторяя эту процедуру дальше, мы получим некоторый граф, содержащий $\langle M^*, N^* \rangle$ и, следовательно, имеющий разве лишь большую сумму длин дуг. ▲

Описанная процедура дает достаточно простую и эффективную систему оценки.

Ветвление будет проводиться следующим образом: выбрав какую-либо вершину $i \in M \setminus M_0$, мы в одной альтернативе включим ее в M_0, а в другой удалим (вместе со всеми инцидентными ей дугами) из графа.

В обоих случаях мы получаем новые задачи такого же типа и можем в них воспользоваться алгоритмом Прима — Краскала. При этом можно пользоваться этим алгоритмом в режиме «досчета», сохраняя от предыдущего расчета все дуги и вершины с номерами меньшими, чем у вершины ветвления.

§ 6. Задача об оптимальном раскрое

В конце гл. 2 в связи с обсуждением метода генерирования столбцов нам встретилась следующая экстремальная задача, которая может рассматриваться как задача линейного программирования с целочисленными коэффициентами.

Задача о наилучшем раскрое.

Задано конечное множество N, положительные векторы c $[N]$ и l $[N]$ и число $l0$. Требуется найти неотрицательный целочисленный вектор x $[N]$, удовлетворяющий

условию

$$l[N] \times x[N] \leqslant l0 \qquad (12)$$

и максимизирующий $c[N] \times x[N]$.

Будем решать эту задачу методом ветвей и границ с односторонним обходом дерева вариантов. И в этой задаче не представляет труда организовать процесс ветвления так, чтобы каждый элемент разбиения представлял собой такую же задачу о раскрое. Поэтому достаточно, если мы сможем оценить максимум целевой функции в исходной задаче.

Если расширить задачу, отказавшись от условия целочисленности, мы получим очень простую задачу линейного программирования — с одним ограничением:

Максимизировать $c[N] \times x[N]$ по неотрицательным $x[N]$, удовлетворяющим (12).

Легко видеть, что решение этой задачи таково: нужно выбрать $j_0 \in N$, на котором достигается максимум отношения $c[j]/l[j]$, и положить

$$x[j_0] = l0/l[j_0], \quad x[N - j_0] = 0[N - j_0],$$

так что оценкой максимума в исходной задаче будет

$$l0 \times c[j_0]/l[j_0].$$

Ветвление вариантов мы будем проводить, выбирая число экземпляров самой выгодной детали (детали с наибольшим отношением $c[j]/l[j]$) и выбирая в первую очередь вариант с наибольшим возможным числом экземпляров, а затем последовательно его уменьшая.

Как ни странно, в этих немногочисленных фразах содержится все описание работы алгоритма. Шаг вперед состоит в том, что выбирается самая выгодная деталь, а затем она отрезается столько раз, сколько поместится, и для остатка делается то же самое (рекурсия), а шаг назад заключается в том, что отменяется одна из последних отрезанных деталей (последний экземпляр последней отрезанной детали), оценивается максимум и в случае целесообразности продолжения делается шаг вперед (для оставшихся деталей), а при нецелесообразности делается шаг назад.

На этой простой задаче удобно наблюдать, как вырабатывается экономная запись о состоянии вычислительного

процесса. Зная действие алгоритма в принципе, проследим за решением небольшого примера

Пусть мы имеем сырье длины 100 и пять деталей с длинами 40, 9, 21, 37 и 20 и соответственно ценами 45, 10, 23, 40 и 20 (отношения длин к ценам равны соответственно 1.125, 1.111, 1.095, 1.081 и 1.000). На первом шаге оценка максимума равна $100 \times 45/40 = 112.5$. Первых деталей может уместиться в этом сырье не больше двух. В качестве начального решения отрезаем две первые детали. Получаем выручку 90 и остаток сырья 20.

Шаг. 2. Оценка остатка $20 \times 10/9 = 22.222$ Оценка всего варианта должна учитывать накопленную выручку

$$90 + 22.222 = 112.222.$$

Вторых деталей можно уместить две. Накопленное решение (2, 2), выручка 110, остаток сырья 2. Из этого сырья ничего не выкроить. Следует сделать шаг назад, отказавшись от одного экземпляра детали 2.

Накопленное решение (2, 1), выручка 100, остаток сырья 12. Учитывая, что деталь 2 вырезать уже нельзя, этот остаток никак не использовать, следует сделать еще шаг назад.

Накопленное решение (2, 0) выручка 90, остаток 20. В этом остатке может уместиться только деталь 5, что дает оценку и значение целевой функции 110. Поскольку решение со значением 110 уже имеется, следует сделать новый шаг назад.

Накопленное решение (1), выручка 45, остаток сырья 60. Оценка варианта $45 + 60 \times 1.11 = 111.667$, накопленное значение (1, 6), выручка 105, остаток 6.

Шаг назад: (1, 5), 95, 15.

Шаг назад: (1, 4), 85, 24 Оценка варианта $85 + 24 \times 23/21 = 111.286$.

Шаг назад: (1, 4, 0), 85, 24, оценка 109 (по детали 5).

Шаг назад: (1, 3), 75, 33, оценка 111.143 (по детали 3).

Шаг вперед: (1, 3, 1), 98, 12, оценка 98.

Шаг назад: (1, 3, 0), 75, 33, оценка 108 (по детали 5).

Шаг назад: (1, 2), 65, 42, оценка 111 (по детали 3).

Шаг вперед: (1, 2, 2), 111, 0 — вот новый рекорд!

Шаг назад: (1, 1), 55, 51, оценка 110.857 (по 3).

Шаг назад (0), 0, 100, оценка 111.111 (по 2).

Шаг вперед: (0. 11), 110, 1.

Шаг назад: (0, 10), 100, 10

Шаг назад: (0, 9), 90, 19.

Шаг назад: (0, 8), 80, 27, оценка 109.571 (по 3).

Следующий шаг назад невозможен, и перебор этим завершается с наилучшим решением: (1, 2, 2, 0, 0).

В ходе решения примера у нас выработалась приемлемая запись, состоящая из вектора $x\,[N]$, накопленной выручки γ, остатка сырья λ и оценки *est*. Дополнительно к этому желательно иметь k — номер последней включенной в решение переменной и информацию о рекорде: $x^*\,[N]$ и *rec* — рекордное значение целевой функции.

В терминах этой записи легко описать процесс вычислений, лишь в незначительных деталях отличающийся от проводившегося.

Начальная запись: $x^*[N] = x[N] = 0[N]$; $k = 0$; $\lambda = l0$; $\gamma = 0$; $rec = 0$. Пусть $N = 1 : m$.

Правила переработки записей мы приведем в виде алгольского текста. Общая структура этого текста примерно соответствует схеме, описанной в § 3. Однако есть некоторые отличия.

Так как решается задача максимизации, то знаки неравенств направлены в другую сторону. Так как любой из промежуточных раскроев дает допустимое решение, немного переставлены действия после вычисления оценки:

```
mkest: if k = m then go to mkback else k:= k + 1;
    if l[k] > lam then go to mkest;
    est: = gam + lam × c[k]/l[k];
    if est ≤ rec then go to mkback;
    x[k]: = entier (lam/l[k]); lam: = lam − x[k] × l[k];
    gam: = gam + x[k] × c[k];
    if res < gam then begin rec: = gam;
      for i: = 1 step 1 until m do bestsol[i]: = x[i] end;
    go to mkest;
mkback: k: = k − 1;
    if x[k] > 0 then
      begin x[k]: = x[k] − 1; lam: = lam + l[k];
        gam: = gam − c[k]; go to mkest end
    else if k > 1 then go to mkback
```

Отметим, что этот алгоритм работает тем лучше, чем больше различия в отношениях $c[j]/l[j]$. Для задач, в которых $c[N] = l[N]$, он сильно уступает другим методам, которые встретятся нам ниже.

§ 7. Задачи с бивалентными переменными

Мы уже отмечали, что многие дискретные задачи оптимизации легко сводятся к задачам линейного программирования, в которых переменные принимают значения 0 или 1. Очень легко приспособить для решения таких задач метод ветвей и границ. Простейший такой метод под названием «аддитивный алгоритм» был предложен Э. Балашем.

Рассмотрим задачу линейного программирования с бивалентными переменными.

Бивалентная линейная задача.

Найти вектор $x\,[N]$, компоненты которого принимают значения 0 и 1, удовлетворяющий условию

$$a\,[M, N] \times x\,[N] \geqslant b\,[M] \tag{13}$$

и минимизирующий $c\,[N] \times x\,[N]$.

Делая, если нужно, замены переменных, можно привести эту задачу к эквивалентной задаче, в которой все $c\,[j]$ неотрицательны. В дальнейшем мы будем предполагать, что необходимые замены уже сделаны и $c\,[N] \geqslant 0\,[N]$.

Каждый элемент разбиения будет определяться фиксацией значений нескольких компонент вектора $x\,[N]$. Для оценки минимума по оставшимся переменным можно, воспользовавшись неотрицательностью вектора c, всегда брать 0. Таким образом, в каждом элементе разбиения мы будем задавать *частичное решение* — набор значений переменных на некотором подмножестве N' множества N и стараться продолжить это частичное решение до полного решения нулями. В тех случаях, когда это продолженное решение окажется допустимым, т. е. удовлетворяющим условию (13), оно будет оптимальным среди допустимых решений данного варианта. Если же это продолженное решение не является допустимым, т. е. если вектор $\beta\,[M] = a\,[M, N'] \times x\,[N'] - b\,[M]$ имеет отрицательные компоненты, то следует проверить, имеются ли у данного частичного решения допустимые продолжения.

Во всяком случае для существования таких продолжений необходимо, чтобы для любого $i \in M^- = \{i \mid \beta\,[i] < 0\}$ выполнялось неравенство

$$\Sigma_{j \in N_i} a\,[i, j] + \beta\,[i] \geqslant 0, \tag{14}$$

где

$$N_i = \{j \mid j \notin N',\ a\,[i, j] > 0\}$$

Если для какого-либо i это неравенство не выполняется, то данный вариант должен быть отброшен. Если же оно выполняется для всех таких индексов, то вариант должен быть развит. Для этого рекомендуется выбрать один из индексов отрицательности $i \in M^-$ и в множестве N_i выбрать индекс j, при котором $c\,[j]$ имеет наименьшее значение.

По значению компоненты решения $x\,[j]$ и следует произвести ветвление.

При варианте перебора, соответствующем одностороннему обходу дерева, рекомендуется в первую очередь исследовать вариант, в котором переменной приписывается значение 1.

Для полного детального описания алгоритма нам не достает только способа задания информации о дереве вариантов. Удобный метод задания этой информации был предложен А. Джеффрионом. Вся информация задается целочисленным вектором $g\,[N]$, в котором $g\,[N \setminus N'] = 0\,[N \setminus N']$, $g\,[j] = k$, если $x\,[j]$ было k-й по счету единицей в текущем частичном решении (отмененные единицы здесь не учитываются), $g\,[j] = -k$, если $x\,[j]$ было сделано нулем между назначением $(k-1)$-й и k-й единицы в данном частичном решении.

Запись в алгоритме состоит из информации о рекорде: значение целевой функции rec и вектор $x^*\,[N]$; информации о текущем решении: вектор Джеффриона $g\,[N]$, количество k единиц, зафиксированных в частичном решении, вектор $\beta\,[M]$, накопленное значение целевой функции γ.

Начальное значение записи:

$$k = 0,\ \beta\,[M] = -b\,[M],\ \gamma = 0,\ g\,[N] = x^*\,[N] = 0\,[N],$$
$$rec = 1 + c\,[N] \times 1\,[N].$$

Правила переработки записей:

1. Образовать множество $M^- = \{i \mid \beta\,[i] < 0\}$. Если это множество пусто, записать новый рекорд $rec := \gamma$; $x^*\,[j] := \delta\,(g\,[j] > 0)$ для каждого $j \in N$ и перейти к п. 5.

2. Для каждого $i \in M^-$ образовать множество N_i и проверить выполнение неравенства $\sum_{j \in N_i} a\,[i, j] + \beta\,[i] \geqslant 0$. Если для какого-либо i неравенство не выполняется, перейти к п. 5.

3. Выбрать какое-либо $i \in M^-$ и $j \in N_i$, на котором достигается минимум $c\,[j]$. Если $\gamma + c\,[j] > rec$, перейти к п. 5.

4. Увеличить k на единицу, пересчитать запись:

$$\beta\,[M] := \beta\,[M] + a\,[M, j];\quad \gamma := \gamma + c\,[j];\quad g\,[j] := k.$$

Перейти к п. 1.

5. Если $k \geqslant 1$, то сделать шаг назад, отменив последнюю единицу и назначенные после нее нули. Для этого

пересчитать запись:

for $j \in N$ **do if** $g[j] = -(k+1)$ **then** $g[j] := 0$
else if $g[j] = k$ **then begin** $j1 := j$; $g[j] := -k$ **end**;
$k := k-1$; $beta[M] := beta[M] - a[M, j1]$; $gam := gam - c[j1]$

и перейти к п. 1.

6. В случае если процесс дошел до этого пункта, в $x^*[N]$ содержится оптимальное решение (причем, если $rec > c[N] \times 1[N]$, задача допустимых решений не имеет).

Обоснование этого алгоритма вряд ли требуется — очевидно, что он эквивалентен (по поисковым способностям) полному перебору.

Рассмотрим небольшой пример. Пусть требуется минимизировать $3x_1 + 4x_2 + 6x_3 + 2x_4 + 3x_5$ при условиях

$$3x_1 + 6x_2 - 2x_3 - 3x_4 - 5x_5 \geqslant 1,$$
$$-2x_1 + 4x_2 + x_3 - 4x_4 + 3x_3 \geqslant 2.$$

Результаты преобразования записей от шага к шагу будут сведены в табл. 17. В последних двух графах таблицы указывается номер причины пересчета и комментарий к этой причине.

Таблица 17

№	1 2	k	1 2 3 4 5	est	rec	1 2 3 4 5	n	Пояснения
0	−1 −2	0	0 0 0 0 0	0	19	0 0 0 0 0	4	$i=1, j=1$
1	2 −4	1	1 0 0 0 0	3	19	0 0 0 0 0	4	$i=2, j=5$
2	−3 −1	2	1 0 0 0 2	6	19	0 0 0 0 0	4	$i=1, j=2$
3	3 3	3	1 3 0 0 2	10	10	1 1 0 0 1	5	
4	−3 −1	2	1 −3 0 0 2	6	10	1 1 0 0 1	3	
5	2 −4	1	1 0 0 0 −2	3	10	1 1 0 0 1	4	$i=2, j=2$
6	8 0	2	1 2 0 0 −2	7	7	1 1 0 0 0	5	
7	2 −4	1	1 −2 0 0 −2	3	7	1 1 0 0 0	3	
8	−1 −2	0	−1 0 0 0 0	0	7	1 1 0 0 0	4	$i=1, j=2$
9	5 2	1	−1 1 0 0 0	4	4	0 1 0 0 0	5	
10	−1 −2	0	−1 −1 0 0 0	0	4	0 1 0 0 0	6	

Из примера хорошо видны достоинства и недостатки метода: его легко осуществить, он не требует сложных вычислений и объемов памяти. Однако этот метод плох тем,

что мало сокращает перебор в сравнении с полным, использует слабые оценки, примитивный отсев и не дает достаточно аргументированной стратегии ветвления.

Многие из этих недостатков устраняются при использовании вспомогательных задач линейного программирования. Чтобы не усложнять дело техническими деталями, мы покажем, как выглядит это использование, на специальных задачах более простого вида.

Прежде всего следует назвать простую задачу, имеющую большое сходство с задачей наилучшего раскроя.

Задача о ранце.

Имеется множество N (множество предметов) и два положительных вектора $c[N]$ — стоимостей предметов и $w[N]$ весов предметов. Требуется найти такое подмножество $N' \subset N$, чтобы

$$1[N'] \times w[N'] \leqslant w0$$

(вес отобранных предметов не превосходил заданного) и достигался максимум стоимости

$$1[N'] \times c[N'].$$

Для характеристического вектора $x[N]$ множества N' мы получаем соответственно неравенство

$$w[N] \times x[N] \leqslant w0 \qquad (15)$$

и значение целевой функции $c[N] \times x[N]$.

Реализовать в этой задаче метод ветвей и границ даже проще, чем в задаче линейного раскроя — не надо вычислять максимальное количество j-х предметов, которые могут уместиться в остаток веса, можно взять не больше одного предмета.

Сложнее оказывается вариант этой задачи, в котором вместо (15) имеется целый набор ограничений.

Многомерная задача о ранце.

Заданы множества M и N, положительные векторы $b[M]$ и $c[N]$ и неотрицательная матрица $a[M, N]$. Требуется найти вектор $x[N]$, удовлетворяющий условиям

$$x[j] \in 0:1, \quad j \in N, \qquad (16)$$

$$a[M, N] \times x[N] \leqslant b[M] \qquad (17)$$

и максимизирующий $c[N] \times x[N]$.

На этой задаче мы и будем демонстрировать использование линейного программирования. Если в многомерной задаче о ранце заменить условие (16) условием

$$\mathbf{1}\,[N] \geqslant x\,[N] \geqslant \mathbf{0}\,[N],$$

мы получим задачу линейного программирования, двойственная к которой выглядит следующим образом.

Двойственная оценочная задача.

Найти $v\,[M]$ и $v1\,[N]$, минимизирующие

$$v\,[M] \times b\,[M] + v1\,[N] \times \mathbf{1}\,[N]$$

при условиях

$$v\,[M] \geqslant \mathbf{0}\,[M], \quad v1\,[N] \geqslant \mathbf{0}\,[N],$$
$$v\,[M] \times a\,[M, N] + v1\,[N] \geqslant c\,[N].$$

Когда мы находим оптимальное решение непрерывного расширения многомерной задачи о ранце, то одновременно находятся оптимальные векторы $v^*\,[M]$ и $v1^*\,[N]$ из двойственной задачи, и для оптимального $x^*\,[N]$ мы имеем по соотношениям двойственности

$$\begin{gathered} x^*\,[j] = 1 \quad \text{при} \quad v1^*\,[j] > 0, \\ v^*\,[M] \times a\,[M, j] + v1^*\,[j] = c\,[j] \\ \text{при} \quad x^*\,[j] > 0. \end{gathered} \tag{18}$$

Если обозначить $\gamma\,[N] = v^*\,[M] \times a\,[M, N]$, то (18) можно переписать так: $\gamma\,[j] \leqslant c\,[j]$ при $x^*\,[j] = 1$, $\gamma\,[j] \geqslant c\,[j]$ при $x^*\,[j] = 0$ и $\gamma\,[j] = c\,[j]$ при $0 < x^*\,[j] < 1$.

Это небольшое отступление в линейное программирование потребовалось нам вот зачем: нам недостаточно для получения оценок избавиться от условия целочисленности. Задача линейного программирования общего вида — слишком громоздка, чтобы ее решать на каждом шаге вычислительного процесса. Мы упростим и эту задачу, заменив ограничения (17) одним ограничением

$$(\nu\,[M] \times a\,[M, N]) \times x\,[N] \leqslant (\nu\,[M] \times b\,[M]),$$

где $\nu\,[M]$ — любой неотрицательный вектор. Имеется много наводящих соображений в пользу того, чтобы в качестве $\nu\,[M]$ взять $v^*\,[M]$. Мы не будем здесь описывать эти аргументы — они не носят вполне определенного характера, главное соображение — вычислительные удобства, которые будет целесообразнее обсудить после описания алгоритма.

Итак, введя, кроме вектора $\gamma\,[N]$, число $\beta = v^*\,[M] \times \times b\,[M]$, мы рассмотрим следующую задачу линейного программирования.

Оценочная задача.

$$\left|\begin{array}{c}\text{Максимизировать } c\,[N] \times x\,[N] \text{ при условиях} \\ \mathbf{1}\,[N] \geqslant x\,[N] \geqslant \mathbf{0}\,[N], \\ \gamma\,[N] \times x\,[N] \geqslant \beta.\end{array}\right. \tag{19}$$

Решение этой задачи $\bar{x}\,[N]$ выглядит следующим образом: $\bar{x}\,[j] = 1$ при $\gamma\,[j] < c\,[j]$ и $\bar{x}\,[j] = 0$ при $\gamma\,[j] > c\,[j]$, остальные значения выбираются произвольно с соблюдением ограничений (19). Однако, поскольку нас будет интересовать эта задача и при других значениях β, а также при уменьшенном множестве N, мы опишем общий алгоритм получения оптимальных решений в таких оценочных задачах.

Идея состоит в том, что нужно выбрать такое $j_0 \in N$, что, разбив множество N на N_1, N_0 и j_0 по следующему принципу:

$$N_1 \subset \{j \mid c\,[j]/\gamma\,[j] \geqslant c\,[j_0]/\gamma\,[j_0]\},$$

$$N_0 \subset \{j \mid c\,[j]/\gamma\,[j] \leqslant c\,[j_0]/\gamma\,[j_0]\}$$

и положив

$$x\,[N_1] = \mathbf{1}\,[N_1], \quad x\,[N_0] = \mathbf{0}\,[N_0],$$

$$x\,[j_0] = \beta - \gamma\,[N_1] \times \mathbf{1}\,[N_1],$$

мы получили бы допустимое решение. Если это решение будет допустимым, то оно будет и оптимальным, так как

$$z = c\,[j_0]/\gamma\,[j_0] \quad \text{и} \quad v1\,[j] = c\,[j]/\gamma\,[j] - z, \quad j \in N_1$$

дают соответствующее решение задачи, двойственной оценочной.

Нужный индекс j_0 легко найти, расположив элементы по убыванию отношения $c\,[j]/\gamma\,[j]$. При таком расположении можно, просматривая индексы, последовательно увеличивать до единицы значения $x\,[j]$ и переходить к следующему индексу или, если ограничение (19) будет мешать, взять максимальное возможное значение $x\,[j]$ и на этом прекратить процесс. В частности, если элементы уже расположены

в таком порядке и $N = 1 : n$, мы можем описать решение этой задачи следующим образом:

```
j:=0; rem:=beta;
for j:=j+1 while gam[j]<rem do
   begin x[j]:=1; rem:=rem-gam[j] end;
x[j]:=rem/gam[j];
for j:=j+1 step 1 until n do x[j]:=0
```

Однако точно придерживаться такого расположения переменных оказывается неудобно. Подчеркнем поэтому, что нам важно лишь, чтобы к моменту выбора заключительного индекса j_0 все индексы с большим значением отношения $c[j]/\gamma[j]$ были уже просмотрены.

Итак, для оценки будет решаться оценочная задача. Ветвление будет проводиться простым и естественным образом — будет выбираться переменная $x[j]$ и в одной альтернативе мы положим $x[j] = 1$, а в другой $x[j] = 0$. Политика выбора переменной для ветвления заслуживает отдельного разговора.

Из неотрицательности матрицы $a[M, N]$ и положительности векторов $b[M]$ и $c[N]$ следует, что по оптимальному решению линейной задачи $x^*[N]$ мы можем построить допустимое решение исходной задачи $\bar{x}[j] = entier(x^*[j])$. Значение целевой функции этой задачи будет оценкой снизу для максимума, а оценкой сверху будет максимум оценочной задачи, который будет равен

$$c[N] \times \bar{x}[N] + c[N_b] \times x^*[N_b] = \\ = c[N] \times \bar{x}[N] + (\beta - \gamma[N] \times \bar{x}[N])$$

(как уже указывалось в оценочной задаче, получающейся вначале, $\beta = v^*[M] \times b[M]$). Таким образом, речь идет о сравнительно небольшом добавлении к значению целевой функции

$$\delta = (\beta - \gamma[N] \times \bar{x}[N]).$$

Отказ от каждой единицы $j \in N_1$, где $N_1 = \{j \mid c[j] > \gamma[j]\}$, приводит к потере части этого возможного приращения, именно к потере не меньше чем $c[j_1] - \gamma[j_1]$ (поскольку $x[j_1]$ будет заменяться переменными с отношением $c[j]/\gamma[j]$, не превосходящим единицы). По этой причине ценность элементов множества N_1 полезнее мерить не отношением $c[j]/\gamma[j]$, а разностью $c[j] - \gamma[j]$. Ветвление с крупными значениями этой разности полезно проводить

на самых верхних уровнях дерева, чтобы альтернативы получались по возможности неперспективные.

Наконец, при одной и той же разности $c[j] - \gamma[j]$ более высокое место должны занимать переменные с большим значением. Действительно, при отказе от переменной с большими затратами ресурса возрастает количество переменных, которые получают значение 1 в оценочной задаче, а следовательно, привлекаются переменные с худшим значением $c[j]/\gamma[j]$ и теряется большая часть возможного приращения целевой функции.

Читателю, безусловно, видна вся нестрогость и приблизительность этих рассуждений. Это — рассуждения не математика, а проектировщика вычислений, учитывающего различные характерные свойства проектируемого процесса, проводящего эксперименты для проверки тех или иных рабочих решений и не имеющего времени дождаться, когда будет поставлена и решена задача о выборе наивыгоднейшей политики ветвления для решения многомерных задач о ранце данного класса.

Итак, конструируя свой метод решения этой задачи, мы выберем следующее расположение переменных:

$$\begin{aligned} i < j \equiv c[i] \geqslant gam[i] \wedge (c[i] - gam[i] > c[j] - gam[j] \vee \\ \vee (c[i] - gam[i] = c[j] - gam[j] \wedge c[i] > c[j])) \vee \\ c[i] < gam[i] \wedge c[i]/gam[i] \geqslant c[j]/gam[j] \end{aligned}$$

(сначала множество N_1, затем N_b, затем N_0, элементы $N_1 \cup N_b$ расположены в порядке убывания $c[j] - \gamma[j]$, а при одинаковых значениях разности по убыванию $\gamma[j]$, элементы N_0 — по убыванию $c[j]/\gamma[j]$).

Каждый элемент разбиения множества допустимых решений будет состоять из всех решений, продолжающих некоторое частичное допустимое решение, т. е. в каждом таком элементе α будут фиксированы некоторое число k_α и вектор $x_\alpha[1 : k_\alpha]$, и в этот элемент будут входить все допустимые решения $y[1 : n]$, для которых $y[1 : k_\alpha] = x_\alpha[1 : k_\alpha]$.

Информация о состоянии вычислительного процесса будет состоять из информации о рабочем частичном решении: числа $k1$, векторов $x[1 : k1]$ и $b[M] := b[M] - a[M, 1 : k1] \times x[1 : k1]$, $k2$ — номера переменной, после которой останавливается решение оценочной задачи, *est* — оценки

максимума, полученной из решения этой задачи, $c1 = c[1:k1] \times x[1:k1]$ и rem — значения $x[k2+1] \times w[k2+1]$, а, кроме того, из обычной информации о рекорде $bestsol[N]$ — наилучшее допустимое решение и $rec = c[N] \times bestsol[N]$.

Теперь мы можем прямо описать этот метод в виде алгольской процедуры. Решение двойственной задачи $v[M]$ будет считаться заданным. Примем $M = 1:m$, $N = 1:n$.

```
procedure multiknapsack (m, n, a, b, c, v, bestsol, rec);
                                                value m, n;
   integer m, n; real rec; integer array a, b, bestsol;
                                                array c, v;
begin integer i, j, k, k1, k2; real rem, c1, g1, d1, ci, gi,
                                                di, est, e1;
   integer array b1[1:m], x, nom[1:n]; array gam [1:n];
   rec:=rem:=0;
   for i:=1 step 1 until m do rem:=rem+b[i]×v[i];
   for k:=1 step 1 until n do begin g1:=0;
                                        x[k]:=bestsol[k]:=0;
      for i:=1 step 1 until m do g1:=g1+a[i, k]×v[i];
      c1:=c[k]; d1:=c1/g1; gam[k]:=g1;
      for j:=k-1 step -1 until 1 do
        begin i:=nom[j]; ci:=c[i]; gi:=gam[i]; di:=ci/gi;
          if c1>g1∧(c1-g1>ci-gi∨c1-g1=
                                        =ci-gi∧c1>ci)
             ∨c1≤g1∧(d1>di∨d1=di∧c1>ci)
          then nom[j+1]:=nom[j]
          else begin nom[j+1]:=k; go to mkk
                                        end end j; nom[1]:=k;
mkk:  end k;
   c1:=est:=0; k1:=k2:=0;
mkest: for k:=k2+1 step 1 until n do
      begin j:=nom[k]; g1:=gam[j];
        if g1≤rem then begin est:=est+c[j];
                                        rem:=rem-g1 end
        else begin k2:=k-1; e1:=rem×c[j]/g1;
                                        go to mke1 end
      end k; k2:=n; e1:=0;
mke1: if est+e1≤rec then go to mkback;
   for k:=k1+1 step 1 until k2 do
      begin j:=nom[k]; k1:=k;
```

```
    for i:=1 step 1 until m do begin b1[i]:=b[i]-a[i, j];
            if b1[i]<0 then go to mkzero end;
    for i:=1 step 1 until m do b[i]:=b1[i];
        c1:=c1+c[j]; x[k]:=1;
        if c1>rec then begin rec:=c1;
          for j:=1 step 1 until n do
                                  bestsol [nom[j]]:=x[j] end
      end k;
    if k2<n then begin k2:=k2+1; x[k2]:=0; k1:=k2;
      if k2<n then go to mkest end;
mkback: for k:=k1 step -1 until 1 do
      if x[k]=1 then begin j:=nom[k]; k1:=k;
                                              c1:=c1-c[j];
        for i:=1 step 1 until m do b[i]:=b[i]+a[i, j];
        go to mkzero end k;
    go to mkfin;
mkzero: x[k]:=0; rem:=rem+gam[j]; est:=est-c[j];
    go to mkest;
mkfin: end mks
```

Аналогичный метод может быть предложен и для задачи с бивалентными переменными общего вида, а при некоторых модификациях и для общей задачи линейного программирования с условием целочисленности всех или некоторых переменных.

Отметим, что многие комбинаторные задачи, похожие на уже встречавшиеся, можно свести к задачам с бивалентными переменными. Однако непосредственное «шаблонное» сведение приносит мало пользы. Нужно помнить, что метод улучшенного перебора по сути дела дает лишь идею алгоритма. Его реализация может оказаться и очень плохой, и очень хорошей в зависимости от оценок, политики и способа ветвления, формы представления данных, причем насколько удачны эти все параметры процесса определяется не только задачей, но и сочетанием исходных данных.

Например, уже отмечалось, что алгоритм нахождения наилучшего раскроя окажется неудачным при $c[N] = l[N]$. В только что описанной процедуре ее трудоемкость существенно зависит от того, насколько велик разброс в значениях $c[i]/\gamma[i]$ (чем больше разброс, тем лучше) и разброс в значениях $\gamma[i]$ (чем меньше, тем лучше).

Для комбинаторных задач лучше использовать возможности организации перебора непосредственно в терминах исходной задачи.

Рассмотрим, например, задачу о нахождении числа внутренней устойчивости из § 1. Мы можем просто перебрать с помощью одностороннего обхода дерева все внутренне устойчивые множества, включая или не включая в них очередной элемент. Для оценки здесь удобно использовать число элементов накопленного множества плюс число элементов, с которыми не связан очередной элемент.

Информацию о графе при такой оценке удобно представить в виде набора списков вершин, не связанных с данной вершиной и имеющих больший номер, чем она. В начальной части процедуры такое представление информации готовится по традиционному представлению графа в виде размеченного списка концов дуг. Списки вершин содержатся в массиве a, а в $l1$ содержится его разметка.

Массив t используется для магазинного хранения кандидатов во внутренне устойчивые множества для различных уровней перебора, в $l\,[0 : m-1]$ хранится разметка этого массива, а в $f\,[0 : m-1]$ — степень использования списка кандидатов каждого шага.

```
integer procedure intstab (m, lst, j); value m;
   integer m; integer array lst, j;
begin integer i, i1, i2, j1, fp, k, rec, est;
   integer array f, l, l1[0:m-1], a, t[1:m×(m-1)/2];
   fp:=m×(m-1)/2; k:=m+m-4;
   for i:=1 step 1 until fp do a[i]:=1;
   for i:=1 step 1 until m do
     begin t[i]:=i;
       for j1:=lst[i-1]+1 step 1 until lst[i] do
         begin i1:=j[j1]; if i1<i then
                                 begin i2:=i1; i1:=i end
                               else i2:=i;
           a[(k+i2)×i2/2+i1]:=0 end end i;
   l1[0]:=fp:=0; i1:=0;
   for i:=1 step 1 until m do begin
     for i2:=i+1 step 1 until m do
       begin i1:=i1+1;
         if a[i1]=1 then begin fp:=fp+1; a[fp]:=i2 end;
       end; l1[k]:=fp end i;
```

```
    fp:=m+1; f[0]:=1; k:=rec:=0; l[0]:=est:=m;
mkest: if est≤rec then go to mkback;
    i1:=t[f[k]]; i2:=f[k]+1; j1:=l1[i1-1];
                                        f[k+1]:=fp;
    for i:=i2 step 1 until l[k] do
      begin for j1:=j1+1 while a[j1]<t[i] do
                if j1>l1[i1] then go to mkf;
        if a[j1]>t[i] then j1:=j1-l
        else begin t[fp]:=a[j1]; fp:=fp+1 end end i;
mkf: k:=k+1; l[k]:=fp-1; est:=k+fp-f[k];
    if k>rec then rec:=k; go to mkest;
mkback: if k>0 then begin k:=k-1; f[k]:=f[k]+1;
          fp:=l[k]+1; est:=k+fp-f[k]; go to mkest end;
    intstab:=rec
end intstab
```

§ 8. Задача размещения производства

Мы рассмотрим здесь один из простейших вариантов задачи размещения производства и на нем продемонстрируем, что и схема одновременного ветвления имеет достаточно удобные реализации.

Задача размещения производства.

Пусть $M = 1 : m$ — множество пунктов, в которых могут располагаться заводы, производящие некоторый однородный продукт, а $N = m + 1 : m + n$ — пункты потребления этого продукта. Заданы объемы потребления $b[N]$, матрица стоимостей перевозки $ct[M, N]$, а кроме того для каждого $i \in M$ задано множество вариантов строительства завода D_i. Пусть $D = \bigcup_{i \in M} D_i$. Каждому $d \in D$ сопоставляются две величины — объем производства $a[d]$ и стоимость осуществления этого варианта $c[d]$.

Требуется найти такой набор вариантов строительства $d[M]$ и такой план перевозки $x[M, N]$, чтобы достигался минимум суммарных затрат (c и ct считаются приведенными к единым измерителям)

$$1[M] \times c[d[M]] + \sum_{i \in M, j \in N} ct[i, j] \times x[i, j], \quad (20)$$

причем

$$x[M, N] \geqslant 0[M, N], \tag{21}$$

$$d[i] \in D_i, \quad i \in M, \tag{22}$$

$$\mathbf{1}[M] \times x[M, N] = b[N], \tag{23}$$

$$x[M, N] \times \mathbf{1}[N] \leqslant a[d[M]]. \tag{24}$$

В этой задаче метод одностороннего обхода не дает обычно хороших результатов из-за большого числа примерно равноправных вариантов, разница между которыми начинает чувствоваться лишь при больших уровнях ветвления, поэтому схема одновременного ветвления здесь предпочтительнее.

В качестве частичных решений будут рассматриваться векторы $d[1 : k]$, где $k < m$, удовлетворяющие условию (22). Решение, продолжающее $d[1 : k]$, описывается вектором $d[k + 1 : m]$ и матрицей $x[1 : m, N]$, удовлетворяющей (21), (23), (24). Для построения оценки на множестве продолжений $d[1 : k]$ мы будем решать следующую расширенную задачу.

Найти $d[k + 1 : m]$ и $x[M, N]$, удовлетворяющие условиям (21), (22), (24),

$$x[i, N] \leqslant b[N], \tag{25}$$

$$\mathbf{1}[M] \times x[M, N] \times \mathbf{1}[N] = \mathbf{1}[N] \times b[N] = \beta \tag{26}$$

и максимизирующие (20).

Будем, кроме $d[M]$, выбирать вектор объемов производства $y[M]$, удовлетворяющий условиям

$$y[M] \leqslant a[d[M]], \quad \mathbf{1}[M] \times y[M] = \beta. \tag{27}$$

Тогда мы сможем переписать нашу задачу (25), (26) следующим образом.

Найти $d[k + 1 : m]$, $y[M]$ и $x[M, N]$, удовлетворяющие условиям (21), (22), (26), (27) и

$$x[M, N] \times \mathbf{1}[N] = y[M] \tag{28}$$

и минимизирующие (20).

Отметим, что при фиксированном $y[i]$ поиск $x[i, N]$ превращается в этой задаче в линейное расширение задачи о ранце, встречавшееся нам в § 4 гл. 1 и в § 7 этой главы.

Мы можем решить эту задачу предварительно и сопоставить каждому i и каждому значению

$$y \in 0 : \max \{a[d] \mid d \in D_i\}$$

некоторое число $c1(i, y)$. Тогда наша оценочная задача будет выглядеть так:

Оценочная задача.

Найти $d[k+1:m]$ и $y[M]$, удовлетворяющие (22), (27) и минимизирующие

$$\sum_{i \in k+1:m} c1(i, y[i]). \qquad (29)$$

Для решения этой задачи мы сведем ее к эквивалентной задаче о кратчайшем пути. Пусть M' — множество всех пар (i, j), где $i \in 0 : m$, $j \in 0 : \beta$. Проведем из $(i-1, l)$ в (i, l') дугу стоимости $c1(i, l'-l)$, если $0 \leqslant l'-l \leqslant r(i)$, где $r(i) = \max \{a[d] \mid d \in D_i\}$ при $i > k$ и $r(i) = a[d[i]]$ при $i \leqslant k$. Легко видеть, что кратчайший путь из (0, 0) в (m, β) как раз и дает решение оценочной задачи.

Мы описали способ оценки. Способ ветвления будет таков: выбрав вариант с наименьшей оценкой, возьмем вектор $d[1:m]$ из оценочной задачи и найдем для полученного набора решений оптимальный план перевозки. Полученное допустимое решение надо, разумеется, сравнить с рекордом.

Затем образуем m альтернативных вариантов — при каждом $k \in 1 : m$ зафиксируем $d[1 : k-1]$ и запретим использовать $d[k]$. Таким образом, каждый из элементов разбиения множества решений будет описываться вектором $d[1:k]$ и набором *запретов* в множествах D_i, $i \in 1 : k+1$.

Пусть теперь множества D_i — непересекающиеся отрезки натурального ряда, $\bigcup D_i = 1 : ndec$, $lst[0:m]$ — вектор разметки этого множества, векторы $a, c[1:ndec]$ таковы, как они были описаны в задаче размещения производства, $N = m+1 : m+n$, $b[N]$ — вектор потребностей, $ct[1:m, m+1:m+n]$ — матрица стоимостей перевозки.

Опишем наш алгоритм в виде процедуры на алголе-68 с необходимыми комментариями:

```
proc EmeliKomlik = (ref[ ] int a, b, c, d, lst, jt, xt,
                    ref[,] int ct, int cost, costt) void:
begin int m = upb lst, n = upb b, ndec = upb a;
  mode cand = struct (ref cand next, ref ban f,
                      int est, [1:m] int d)
```

co это структура для описания вариантов. Главное — вектор решений *d* и оценка *est*. *next* используется для цепных списков. Имеется ссылка на запрет *ban*, который сам будет ссылаться на другие запреты, составляющие иерархическое дерево. Пройдя по этому дереву от *f* до корня, мы найдем все запреты данного варианта co;

```
mode ban = struct (ref ban pre, int dec, fol);
```

co *dec* — это сам запрет, *fol* — число ссылок на данный запрет (которое нужно, чтобы утилизировать ненужные запреты).

Мы не будем для экономии места включать в эту процедуру блоки нахождения кратчайших путей co

```
ban zeroban:= (nil, 0, 0); co это корень дерева co
ref ban fban:=nil, co fban — начало списка пустых
                мест co bc, bc1, newb;
cand cand0:=(nil, zeroban, 0, skip);
ref cand fcan:=nil, co тоже список пустых мест co
  first:=cand0, cand, cand1, cand2, cd, cd1;
int rec:=maxint, btot:=0, a2, a3, c2, c3, dmin,
                                    d1, i1, i2, min;
for i from m+1 to n do btot+:=b[i] od;
[1:ndec] bool p; [0:m] int c1, a1;
shortestpath(1, cand0);
```

co эта процедура находит кратчайший путь и записывает его в *d* of *cand* 0, а его оценку в *est* of *cand* 0 co

```
while lin first do
  if est of (cand:=head first) < rec then
    study(cand, jt, xt, rec, costt);
```

co эта процедура формирует по варианту *cand* данные для транспортной задачи, решает ее и, если суммарные затраты меньше рекорда, записывает в *jt* и *xt* оптимальный план, в *costt* — транспортные затраты, новый рекорд в *rec* и вектор решений в *d* of *cand*. Теперь будем строить альтернативные варианты co

```
for j to ndec do p[j]:=true od;
i1:=ndec; bc:=f of cand; c1[0]:=a1[0]:=c2:=a2:=0;
for i to m do c1[i]:=c2+:=c[d1:=(d of cand)[i]];
  a1[i]:=a2+:=a[d1]; p[d1]:=false od;
```

co запрещаем то, что использовалось в нашем варианте.

Обратите внимание, как производится вырезка из мультизначения, являющегося полем в структуре. Массивы $c1$ и $a1$ нам потом пригодятся **co**

```
    for i from m by −1 to 1 do bc1:=bc; i2:=i1;
      i1:=lst[i − 1];
      while (d1:=dec of bc)>i1 do
        p[d1]:=false; bc:=pre of bc od;
  min:=maxint; d1:=(d of cand)[i];
  for d0 from i1+1 to i2 do if p[d0] then
    (d of cand)[i]:=d0;
    shortestpath(i, cand);
    if est of cand<min then
    min:=est of cand; best:=d of cand fi fi od;
    if min<rec then
    cand1:=if lin fcan then head fcan else heap cand fi;
    newb:=if lin fban then head fban else heap ban fi;
    ref ban(newb):=(bc1, d1, 0); fol of bc1+:=1;
    ref cand(cand1):=(nil, newb, min, best);
```

co мы образовали новый вариант, который сохраняет все запреты варианта *cand* в пунктах $1 : i$ и, кроме того, имеет собственный запрет $d1 = (d$ **of** $cand)[i]$. Место для нового варианта и нового запрета берется из «мусора» или генерируется заново. «Наследование» записей достигается тем, что новый запрет *newb* сразу «прицепляется» к первому из запретов *cand* в п. *i*. Далее новый вариант включается в цепной список *first* так, чтобы сохранялось неубывание оценок (см. процедуру *inclel* в первой главе) **co**

```
    if lin first then inclel(first, cand)
    else first:=cand1 fi fi od fi;
```

co теперь осталось только собрать мусор **co**

```
    bc:=f of cand; fcan cons cand;
    while fol of bc=0 do fban rob bc; fol of bc−:=1 od od
end
```

§ 9. Выбор наилучшего разбиения

В этом параграфе мы рассмотрим одну задачу построения наилучшего разбиения множества. Интересен метод, которым будет решена эта задача — в нем метод ветвей и границ сочетается с наивным и на первый взгляд совершенно непер-

спективным подходом. Однако алгоритм получается чрезвычайно эффективным.

Будет рассматриваться следующая задача.

Задача о куче камней.

Имеется множество камней N, каждому из которых j сопоставлено положительное число — его вес $w[j]$. Требуется разделить это множество на заданное число куч m таким образом, чтобы вес самой тяжелой из куч был минимален.

Иными словами, требуется найти такое разбиение $\mathfrak{N}$: $N = \bigcup_{i \in 1:m} N_i$, которое минимизирует

$$\max \{1[N_i] \times w[N_i] \mid i \in 1:m\}. \tag{30}$$

Пусть нам требуется решить задачу:

Минимизировать $f(x)$ по $x \in A$

Предположим, что мы имеем два числа *est* и *rec*, для которых известно, что

$$est < \max \{f(x) \mid x \in A\} \leqslant rec.$$

Выберем какое-либо число $z \in (est, rec)$ и рассмотрим задачу:

z-задача.

Найти $x \in A$, для которого $f(x) \leqslant z$.

Если нам удастся найти решение этой задачи, то z можно взять в качестве нового значения *rec*, после чего можно повторить процесс. Если мы покажем, что z-задача решения не имеет, то z можно взять в качестве нового значения *est* и также повторить процесс.

Главная трудность здесь разумеется лежит в способе решения получающейся z-задачи. Оказывается в некоторых случаях, в том числе и в задаче о куче камней, можно искать решение z-задачи методом ветвей и границ.

Действительно, z-задача, соответствующая задаче о куче камней, выглядит следующим образом.

Найти разбиение $\mathfrak{N} : N = \bigcup_{i \in 1:m} N_i$, при котором

$$1[N_i] \times w[N_i] \leqslant z, \quad i \in 1:m. \tag{31}$$

Будем решать эту задачу методом ветвей и границ с односторонним обходом дерева вариантов. Можно пред-

ставить себе m «ящиков» для камней, каждый из которых имеет вместимость z. Таким образом, в ящиках должно оставаться свободное место $slack = z \times m - \mathbf{1}\,[N] \times w\,[N]$. Будем укладывать камни в ящики, последовательно заполняя один ящик за другим и стремясь каждый раз уложить в ящик самый большой из оставшихся (и помещающихся) камней. Остающееся в ящике свободное место будет вычитаться из остатка свободного места *slack*, а этот остаток должен будет в течение всего вычислительного процесса оставаться неотрицательным. Если *slack* станет отрицательным, мы вернемся и отменим последнее из назначений, как это и полагается делать при одностороннем обходе.

Покажем действие этого алгоритма на небольшом примере. Пусть исходная куча состоит из 16 камней:

72, 70, 69, 64, 58, 54, 52, 49, 36, 35, 34, 31, 29, 23, 21, 20,

суммарный вес которых 717. Пусть $m = 4$, таким образом, максимальный вес должен быть больше чем 179. Оценка сверху для минимума максимального веса есть 717. Явно, что эта оценка завышена, поэтому z мы выберем поближе к 180. Попробуем решить эту задачу с $z = 183$. Свободное место составляет $183 \times 4 - 717 = 15$. Заполняем первый ящик. Будем записывать наши действия двумя строками, в верхней — остаток места в ящике, в нижней — выбранный камень. Итак,

1 ящик. 183, 111, 41, 5, остаток свободного места $15 - 5 = 10$.
72, 70, 36

Аналогично

2 ящик. 183, 114, 50, 1, остаток $10 - 1 = 9$.
69, 64, 49

3 ящик. 183, 125, 71, 19, слишком большой остаток,
58, 54, 52

отменяем назначение 52 и назначаем следующий камень 35.

Итак,

3 ящик. 183, 125, 71, 36, 2.
58, 54, 35, 34

Четвертый ящик можно не заполнять — в него уместится все остальное со свободным местом 7.

Теперь мы знаем, что минимум лежит между 180 и 183. Попробуем решить z-задачу с $z = 181$. Первоначально $slack = 7$.

1 ящик. 181, 109, 39,3, $slack = 4$.
72, 70, 36

2 ящик. 181, 112, 48, 13.
69, 64, 35

После ряда попыток получаем

2 ящик. 181, 112, 48, 25, 4, $slack = 0$,
69, 64, 23, 21

и также после перебора

3 ящик. 181, 123, 69, 34, 0.
58, 54, 35, 34

Читателю предоставляется возможность найти решение z-задачи с $z = 180$.

При реализации этого алгоритма на ЭВМ существенно организовать информацию так, чтобы достаточно просто производился выбор наибольшего из оставшихся камней, вес которых не превосходит свободного места в ящике.

Мы сделаем справочник, указывающий для каждого размера w наибольший камень $pl[w]$, а для каждого камня будем помнить ссылки на левого и правого соседа. При этом правая ссылка будет указывать на очередного меньшего невычеркнутого соседа у всех камней, а левая ссылка — только у невычеркнутых. Такая организация позволяет просто восстанавливать информацию при возвращении камней из ящиков.

Значение z на первом шаге определяется как $est + d$, где d — параметр процедуры. В дальнейшем в качестве z принимается полусумма est и rec.

Результат получается в массиве $res[1:n]$, в котором последовательно выписаны элементы множеств разбиения, причем первые элементы помечены знаком минус.

```
procedure bestpartition (n, c, m, d, res, s);
  value n, m, d; integer n, m, d, s; integer array c, res;
begin integer i, j, k, l1, n1, r, r1, sim, sp, sum, slack, tail;
  integer array list, t, rt, t [0:n+1], fst [1:m];
    for i:=1 step 1 until n-1 do
    for j:=i+1 step 1 until n do
      if c[i]<c[j] then
       begin k:=c[i]; c[i]:=c[j]; c[j]:=k end; n1:=n+1;
begin integer array pl[1:c[1]]; j:=c[1]; sum:=0;
    for i:=1 step 1 until n do
      begin r:=c[i]; sum:=sum+r;
        for k:=j step-1 until r do pl[k]:=i;
        j:=r-1 end;
    for k:=j step-1 until 1 do pl[k]:=n+1;
    sim:=(sum-1)÷m; sp:=sum; s:=sim+d;
mks: if s≥sp then begin s:=(sim+s)/2; go to mks end;
    for i:=1 step 1 until n+1 do
      begin lt[i]:=i-1; rt[i-1]:=i end;
    slack:=s×m-sum; k:=0; j:=1;
mknew: k:=k+1; fst[k]:=j; list[j]:=r:=rt[0]; j:=j+1;
    tail:=s-c[r]; lt[rt[r]]:=0;
    for i:=0 step 1 until r-1 do rt[i]:=rt[r];
  if j=n+1 then begin t[k]:=tail; go to mk6 end;
```

```
mkd: if tail ≥ c[lt[n+1]] then begin
     r:= if tail > c[1] then 0 else if pl[tail] > r
                 then (pl[tail] - 1) else r;
mk7:   r:= rt[r]; if r = n+1 then go to mk5; list[j]:= r;
       j:= j+1; tail:= tail - c[r]; lt[rt[r]]:= lt[r];
   for i:= lt[r] step 1 until r-1 do rt[i]:= rt[r];
  if j = n+1 then begin t[k]:= tail; go to mk6 end;
       go to mkd
    end else
mk5: if slack > tail then
      begin slack:= slack - tail; t[k]:= tail;
        if k < m-1 then go to mknew end
     else begin j:= j-1;
      if j = fst[k] then
        begin k:= k-1; r:= list[j]; j:= j-1;
                                    lt[rt[r]]:= r;
        for i:= lt[r] step 1 until r-1 do rt[i]:= r;
            if k = 0 then go to mk1 else tail:= t[k] end;
        r:= list[j]; lt[rt[r]]:= r; tail:= tail + c[r];
      for i:= lt[r] step 1 until r-1 do rt[i]:= r;
        go to mk7;
mk1: end;
mk6: if k = 0 then begin sim:= s; if sp = sum then
       begin s:= s+d; go to mk6 end end
      else begin fst[k+1]:= j; t[m]:= slack; i:= 0;
        for i:= rt[i] while i < n+1 do
          begin list[j]:= i; j:= j+1 end;
        for i:= 1 step 1 until n do res[i]:= c[list[i]];
        for i:= 1 step 1 until k+1 do
                                res[fst[i]]:= - res[fst[i]];
         for i:= k step -1 until 1 do
           if slack > t[i] then slack:= t[i];
         sp:= s - slack; if sp > sim+1 then go to mks end;
        s:= sp end
end bpart
```

ГЛАВА 6

РЕКУРРЕНТНЫЕ МЕТОДЫ (МОДЕЛИ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ)

§ 1. Линейная задача раскроя

В этой главе будут рассматриваться главным образом модели, в которых удается использовать рекуррентные соотношения типа соотношений, вводившихся при решении задачи о кратчайшем пути.

Мы начнем с уже знакомой нам задачи наилучшего линейного раскроя. Итак, как и в прошлой главе, требуется найти целочисленный вектор, удовлетворяющий условиям

$$x\,[1:n] \geqslant 0\,[1:n], \qquad l\,[1:n] \times x\,[1:n] \leqslant l0 \qquad (1)$$

и максимизирующий $c\,[1:n] \times x\,[1:n]$. Теперь мы пойдем в решении этой задачи другим путем. Будем для простоты считать все $l\,[i]$ и $l0$ целыми числами. Это позволит ограничиться в дальнейшем рассмотрении конечным множеством различных размеров сырья.

Рассмотрим граф $\langle M, N\rangle$ с множеством вершин $0 : l0$. Каждой вершине этого графа i сопоставим множество деталей N_i, которые можно отрезать от заготовки длины

$$N_i = \{j \in 1 : n \mid l\,[j] \leqslant i\},$$

а каждой из таких деталей j сопоставим дугу, ведущую из i в вершину $i - l\,[j]$ и приносящую доход $c\,[j]$. Добавим, кроме того, в каждой из вершин $i \in 1 : l0$ дугу, по которой можно из i перейти в $i - 1$ с нулевым доходом.

Покажем, что исходная задача о наивыгоднейшем раскрое эквивалентна задаче о нахождении пути наибольшей длины в построенном графе. Действительно, каждый такой путь можно интерпретировать как последовательность деталей, отрезаемых от единицы сырья. С другой стороны, каждому раскрою — набору отрезаемых деталей — можно сопоставить (не единственным способом) последовательность,

в которой эти детали отрезаются и, следовательно, путь от вершины $l0$ до вершины 0 в графе $\langle MN\rangle$.

Поскольку в графе $\langle M, N\rangle$ нет контуров, то задача о поиске пути наибольшей длины имеет смысл и может решаться методами, описанными в гл. 4. При этом каждой вершине $i \in 1 : l0$ сопоставляется исходящая из нее дуга k_i (деталь k_i, отрезаемая от остатка сырья длины i, — в терминах нашей задачи *политика кроя*). Принимая $v[0] = 0$, мы можем при данной политике кроя вычислить набор потенциалов $v[1 : l0]$

$$v[i] = c[k_i] + v[i - l[k_i]].$$

Для того чтобы эта политика кроя была оптимальной, необходимо и достаточно, чтобы эти потенциалы удовлетворяли условиям

$$v[i] + c[k] \leqslant v[i + l[k]]. \qquad (2)$$

Потенциалы здесь имеют ясный смысл: $v[i]$ — доход (а в оптимальном решении максимальный доход), который можно получить при раскрое сырья длины i. Сопоставляя (1) и (2), мы получаем для оптимальных $v[i]$ уравнение

$$v[i] = \max\{c[k] + v[i - l[k]] \mid k \in N_i\} \qquad (3)$$

Это уравнение называется *функциональным уравнением Р. Беллмана* или *функциональным уравнением динамического программирования.* Если мы сможем решить это функциональное уравнение и одновременно для каждого i выберем управление, на котором в (3) достигается максимум, мы получим оптимальную политику кроя.

Фактически уравнения Беллмана нам встречались и раньше, когда речь шла о кратчайших и самых длинных путях в графе. В этой главе мы будем систематически иметь дело с такими уравнениями.

В рассматриваемой сейчас задаче имеется простой способ решения уравнения Беллмана: достаточно вычислять $v[i]$ последовательно от $v[1]$ до $v[l0]$, запоминая управления, на которых достигался максимум.

В примере, рассматривавшемся в предыдущей главе, мы получаем табл. 1 значений $v[i]$ и управлений (в число деталей включена деталь длины 1 с нулевой ценой). В этом

примере хорошо видна характерная особенность решения — кусочное постоянство $v[i]$ как функции от i и монотонное неубывание этой функции. Можно воспользоваться этим обстоятельством и сократить информацию, необходимую для задания приведенной таблицы, ограничившись лишь строками, в которых не используется деталь 1 (табл. 2).

Таблица 1

Размер	Оценка	Деталь	Размер	Оценка	Деталь	Размер	Оценка	Деталь	Размер	Оценка	Деталь
1	0	1	26	23	21	51	56	21	76	85	40
2	0	1	27	30	9	52	56	21	77	85	40
3	0	1	28	30	9	53	56	21	78	86	21
4	0	1	29	30	9	54	60	9	79	88	40
5	0	1	30	33	21	55	60	9	80	90	40
6	0	1	31	33	21	56	60	9	81	90	9
7	0	1	32	33	21	57	63	21	82	91	40
8	0	1	33	33	21	58	65	40	83	91	40
9	10	9	34	33	21	59	65	40	84	93	21
10	10	9	35	33	21	60	66	21	85	95	40
11	10	9	36	40	9	61	68	40	86	95	40
12	10	9	37	40	9	62	68	40	87	96	21
13	10	9	38	40	9	63	70	9	88	98	40
14	10	9	39	43	21	64	70	9	89	100	40
15	10	9	40	45	40	65	70	9	90	100	9
16	10	9	41	45	40	66	73	21	91	101	40
17	10	9	42	46	21	67	75	40	92	101	40
18	20	9	43	46	21	68	75	40	93	103	21
19	20	9	44	46	21	69	76	21	94	105	40
20	20	9	45	50	9	70	78	40	95	105	40
21	23	21	46	50	9	71	78	40	96	106	21
22	23	21	47	50	9	72	80	9	97	108	40
23	23	21	48	53	21	73	80	9	98	110	40
24	23	21	49	55	40	74	80	9	99	110	9
25	23	21	50	55	40	75	83	21	100	111	40

Более того, оказывается возможным построить вычислительный процесс таким образом, что в нем будет получаться непосредственно эта сокращенная таблица. Мы будем задавать ее списком, каждый элемент которого состоит из *размера* сырья, получаемого из этого сырья *дохода* и *детали*, которую следует отрезать от сырья этого размера. Список

будет предполагаться упорядоченным по возрастанию размеров сырья. При этом он должен оказаться строго упорядоченным и по доходам. Действительно, записи, в которых сырья тратится больше, чем в других, а доход получается меньше, интереса не представляют. Мы будем говорить, что запись с размером сырья λ_1 и доходом γ_1 *доминирует* запись с сырьем λ_2 и доходом γ_2, если $\lambda_1 \leqslant \lambda_2$ и $\gamma_1 \geqslant \gamma_2$.

Таблица 2

Размер	Оценка	Деталь	Размер	Оценка	Деталь	Размер	Оценка	Деталь
9	10	9	57	63	21	80	90	40
18	20	9	58	65	40	82	91	40
21	23	21	60	66	21	84	93	21
27	30	9	61	68	40	85	95	40
30	33	21	63	70	9	87	96	21
36	40	9	66	73	21	88	98	40
39	43	21	67	75	40	89	100	40
40	45	40	69	76	21	91	101	40
42	46	21	70	78	40	93	103	21
45	50	9	72	80	9	94	105	40
48	53	21	75	83	21	96	106	21
49	55	40	76	85	40	97	108	40
51	56	21	78	86	21	98	110	40
54	60	9	79	88	40	100	111	40

На каждом шаге к самой первой из имеющихся в рабочем списке записей будет прибавляться каждая из деталей: размер сырья будет при этом равен сумме размера в исходной записи и длины прибавляемой детали, доход будет равен сумме имеющегося дохода и цены прибавляемой детали, в качестве отрезаемой детали будет указана эта прибавляемая деталь. Полученный набор новых записей вливается в рабочий список с соблюдением условий упорядочения: по длине и по доходу записи, которые доминируются другими, из списка исключаются. Записи, в которых размер сырья превосходит $l0$, в список также не включаются. Использованная уже первая запись рабочего списка передается в архивный список, а процесс повторяется до исчерпания рабочего списка.

Проследим на нескольких шагах того же примера работу этого алгоритма.

Шаг	Рабочий список	Новый список
1	0, 0, 0	(9, 10, 9), (20, 20, 20), (21, 23, 21), (37, 40, 37), (40, 45, 40)
2	9, 10, 9 20, 20, 20 21, 23, 21 37, 40, 37 40, 45, 40	(18, 20, 9), (29, 30, 20), (30, 33, 21), (46, 50, 37), (49, 55, 40)
3	18, 20, 9 21, 23, 21 29, 30, 20 30, 33, 21 37, 40, 37 40, 45, 40 46, 50, 37 49, 55, 40	(27, 30, 9), (38, 40, 20), (39, 43, 21), (55, 60, 37), (58, 65, 40)

После завершения третьего шага рабочий список состоит из

(21, 23, 21), (27, 30, 9), (30, 33, 21), (37, 40, 37),
(39, 43, 21), (40, 45, 40), (46, 50, 37), (49, 55, 40),
(55, 60, 37), (58, 65, 40),

а архив — из записей (0, 0, 0), (9, 10, 9), (18, 20, 9).

Отметим, что для задачи линейного раскроя можно, изменяя множество состояний, писать различные варианты уравнения Беллмана, соответственно разнообразя вычислительную схему. Например, считая состоянием пару из длины сырья i и наибольшего номера детали k, разрешенной к отрезанию, мы получим функциональное уравнение

$$v[i, k] = \max \{c[k] \times r + v[i - r \times l[k], k - 1] \mid 0 \leqslant r \leqslant entier\,(i/l[k])\}, \quad (4)$$

$$v[0, k] = v[i, 0] = 0.$$

Нас интересует $v[l0, n]$. Нетрудно представить себе вычислительную схему, соответствующую этому уравнению (здесь используется *dec* — целочисленный массив

решений, принимаемых в состояниях процесса, x — вектор раскроя):

```
for i:=0 step 1 until l0 do v[0, i]:=0;
for k:=1 step 1 until n do
  begin ck:=c[k]; lk:=l[k];
  for i:=0 step 1 until l0 do
    begin max:=v[k-1, i]; d:=0;
      for r:=1 step 1 until entier (i/lk) do
        if max<ck×r+v[k-1, i-r×lk] then
          begin max:=ck×r+v[k-1, i-r×lk];
            d:=r end;
        v[k, i]:=max; dec[k, i]:=d
        end i end k; d:=l0;
for k:=n step-1 until 1 do
    begin x[k]:=dec[k, d]; d:=d-x[k]×l[k] end
```

Бросается в глаза вычислительная нерациональность этой схемы. Во-первых, из матрицы v на каждом шаге фактически используются две строки, причем можно обойтись и одной, если сменить направление в цикле по i. Во-вторых, можно написать более удобный в вычислительном отношении вариант уравнения (4)

$$v[i, k]=\begin{cases}\max\{v[i, k-1], c[k]+v[i-l[k], k]\} \\ \qquad\qquad\qquad\qquad \text{при } i\geqslant l[k] \\ v[i, k-1] \text{ при } i<l[k].\end{cases} \quad (5)$$

Его решение реализуется способом, использующим вместо матрицы v вектор, в котором последовательно получаются все строки этой матрицы:

```
  for i:=0 step 1 until l0 do
    begin v[i]:=0; dec[i]:=0 end;
  for k:=1 step 1 until n do
    begin ck:=c[k]; lk:=l[k]; x[k]:=0;
      for i:=lk step 1 until l0 do
        if v[i-lk]+ck>v[i] then
          begin v[i]:=v[i-lk]+ck; dec[i]:=k end
    end k;
  d:=l0;
  for i:=dec[d] while i>0 do
    begin x[i]:=x[i]+1; d:=d-l[i] end
```

Для случая, когда в памяти машины нетрудно разместить одновременно массивы *v* [0: *l*0] и *dec* [0: *l*0], этот алгоритм, по-видимому, является одним из наиболее эффективных. Но возможен его вариант (тоже достаточно экономный) и в форме переработки списка основных состояний.

Переработкой списка удобно заниматься, пользуясь возможностями языка алгол-68 — структурами, связанными в цепной список и возможностями генерирования новых экземпляров из кучи.

Несколько пояснений: **state** — структура, включающая запись списка (решение *d*, длина *l*, стоимость *c*) и ссылку *p* на следующую запись; **dom** — операция, сравнивающая записи (*a* **dom** *b* вырабатывает булевское значение «*a* доминирует *b*»), **lin**, как обычно, вырабатывает значение «ссылка не есть **nil**».

На каждом шаге алгоритма (в цикле по *i*) мы имеем три цепных списка: входящий список, полученный на предыдущем шаге, ссылка на его начало записана в *in*, выходящий список — его начальная ссылка в *out*, а конечная — в *outf*,— и рабочий список с началом в *w* и концом в *wf*. Для того чтобы генерировать новые записи, используется оператор **loc state**. Освобождающиеся места от доминирующих записей объединяются в четвертый цепной список, действующий как магазин. Начало этого списка запоминается ссылкой *free*, при необходимости создания новой записи берется запись, на которую ссылается *free*. Если же такой записи нет, то используется **loc**.

При выработке результата окончательный список обращается, после чего за один просмотр накапливается решение.

```
proc lcutch = (int m, l0, ref [ ] real c, ref [ ] int l, res) void:
begin mode state = struct (ref state p, int d, l, real c);
  int i, l1; ref state in, out, outf, w, wf, cur, free;
  prio dom = 1;
  op dom = (ref state a, b) bool:
       l of a ≤ l of b ∧ c of a ≥ c of b;
  op lin = (ref state a) bool: a isnt nil;
  in: = loc state: = (nil, 0, 0, .0); free: = w: = nil; i: = 1;
mki1: res [i]: = 0; out: = outf: = wf: = nil;
mk: if lin in then cur: = in;
```

```
mk1: if lin w then
       if cur dom w then free rob w; go to mk1
       elif w dom cur then free rob in; go to mk
       elif l of w < l of cur then cur: = head w;
                                        go to mkw fi;
     in: = p of in fi
     elif lin w then cur: = head w
     else go to mki fi;
mkw: join (out, outf, cur);
  if (l1: = l[i] + l of cur) > l0 then go to mk fi;
  if lin free then cur: = head free; go to mkf fi;
  cur: = loc state;
mkf: join (w, wf, cur);
     ref state (cur): = (nil, i, l1, c[i] + c of cur); go to mk;
mki: in: = out; if (i + : = 1) ≤ m then go to mki1 fi;
  in: = listinvert (in); l1: = l of in;
  while lin in do if l1 = l of in then
    l1 - : = l[d of in]; res[d of in] + : = 1 fi;
    in: = p of in od
end
```

Мы использовали здесь операции **head** и **rob** и процедуру *join* (и, следовательно, операции **cons** и **join**), описанные в § 4 гл. 1, применительно к структурному виду *state* со ссылочным полем *p*. Разумеется, все их описания наравне с описанием **lin** должны быть включены в программу, причем для двуместных операций должен быть указан их приоритет (как для **dom**).

§ 2. Детерминированная задача управления запасами

Применим теперь тот же подход к новой задаче.

Рассмотрим предприятие, которое для своего нормального функционирования должно получать в моменты времени $t \in 1\_ : t1$ некоторый продукт (сырье или топливо) в количествах $r[t]$. Эти количества известны и неотрицательны. Известно также наличие продукта $s[0]$ на складе в момент времени 0. Считается, что продукт можно запасать заранее и неограниченное время в неограниченном размере хранить на складе, неся лишь некоторые убытки, зависящие от объема хранимого продукта. Требуется для каждого

момента $t \in 1 : t1$ определить объем закупок продукта $y[t]$ так, чтобы запас продукта $s[t]$ в каждый момент времени t, равный по очевидным балансовым соображениям

$$s[t] = s[t-1] + y[t] - r[t] = \\ = s[0] + \Sigma_{\tau \in 1:t} (y[\tau] - r[\tau]), \quad (6)$$

был неотрицателен и чтобы достигался минимум общих затрат. Эти затраты включают затраты на приобретение и хранение продукта, и их обычно представляют в виде

$$\Sigma_{\tau \in 1:t1} (\varphi_\tau (y[\tau]) + \gamma_\tau (s[\tau])), \quad (7)$$

где φ и γ — заданные неубывающие функции, описывающие зависимость затрат соответственно от объема закупок и от объема запаса.

Таким образом, мы приходим к следующей задаче.

Детерминированная задача управления запасами.

Найти неотрицательные векторы $y[1 : t1]$ и $s[1 : t1]$, удовлетворяющие условию (6) и минимизирующие значение целевой функции (7).

Если все $y[\tau]$ и $s[\tau]$ считать целыми числами, то и эту задачу можно свести к задаче о кратчайшем пути в графе. Состояниями в этом графе будут состояния склада, т. е. пары (t, σ), где t — момент времени, а σ — количество продукта на складе в этот момент. Для того чтобы не рассматривать всех мыслимых состояний неограниченного склада, можно считать, что $\sigma \leqslant R_t = \Sigma_{\tau > t} r[\tau]$. При $t = 0$ достаточно рассмотреть $\sigma = s[0]$, а при $t = t1$ — только $\sigma = 0$. Дуга в каждом таком состоянии будет соответствовать возможному решению о закупке продукта, т. е. из состояния (t, σ) можно перейти в любое состояние $(t + 1, \sigma')$, где

$$\sigma - r[t+1] \leqslant \sigma' \leqslant R_{t+1}.$$

Длина такой дуги принимается равной

$$\varphi_{t+1} (\sigma' - \sigma + r[t+1]) + \gamma_{t+1} (\sigma').$$

Легко видеть, что каждому пути от $(0, s[0])$ до $(t1, 0)$ в этом графе отвечает допустимая политика закупок, затраты при которой совпадают с длиной пути и, следовательно, задача о выборе кратчайшего пути эквивалентна задаче о выборе наилучшей политики закупок.

Вводя в задаче потенциалы, мы получаем для них, как раньше, функциональное уравнение

$$v[i, j] = \min(\gamma_i(j) + \varphi_i(k) + v[i-1, j-k+r[i]]), \quad (8)$$

где минимум берется по всем неотрицательным k, удовлетворяющим условию $k \leqslant j + r[i]$. Если принять $v[0, s[0]] = 0$, то потенциал $v[i, j]$, вычисленный из уравнения, оказывается равным минимальным затратам, необходимым для того, чтобы при удовлетворении потребности в продукции в моменты от 1 до i включительно после i-го момента на складе оставался запас j. Так как и в этом графе нет контуров, потенциалы могут вычисляться последовательно, слоями для $i = 1, 2$ и т. д.

Отметим, что при некоторых дополнительных предположениях детерминированная задача управления запасами может быть сильно упрощена. Например, пусть

$$\gamma_\tau(x) = g \times x \quad \text{при всех } \tau \quad (9)$$

и

$$\varphi_\tau(x) = h_\tau \times \delta(x > 0) + c \times x.$$

Это соответствует случаю, когда затраты на приобретение продукта состоят из платы за продукт, пропорциональной объему заготовки и не зависящей от момента времени, и из «организационных расходов». Сумма слагаемых, пропорциональных объему, не зависит от политики закупок, и ее можно не учитывать при сравнении политик, т. е. принять $c = 0$.

В этом случае оказывается, что заготовку можно производить каждый раз на несколько моментов времени полностью, т. е. ни в какой момент не делать закупок, если продукция для этого момента была частично закуплена раньше. Исключение может составить лишь закупка, первая от начала процесса

Теорема 1. *При предположениях* (9) *оптимальная политика такова, что для всех моментов, кроме разве лишь первого, из* $y[i] > 0$ *следует* $s[i-1] = 0$.

Доказательство. Покажем, что любую политику закупок, не удовлетворяющую условию теоремы, можно улучшить. Пусть существует такое i, что $y[i] > 0$, $s[i-1] > 0$ и i_1 — последний момент времени перед i, когда делалась заготовка. Уменьшим $y[i_1]$ на $s[i-1]$ и увеличим на то же число $y[i]$ При этом политика закупок останется допустимой, затраты на закупки разве лишь

уменьшатся (если закупка в момент i_1 станет равной нулю), затраты на хранение уменьшатся на $g \times s[i-1] \times (i-i_1)$. ▲

Рассмотрим граф Γ_1 с множеством вершин $0 : t1+1$ и дугами, идущими из любой вершины i в любую вершину j, где $j \geqslant \max\{i+1, j1\}$ и $i > 0$. Из вершины 0 дуги будут идти только в вершины $j \in 1 : j1$, где $j1$ определяется условием

$$\Sigma_{i \in 1 : j1} r[i] > s[0] \geqslant \Sigma_{i \in 1 : j1-1} r[i].$$

Длиной $c[i, j]$ дуги (i, j) будем считать затраты на хранение и покупку продукта, купленного в момент i и полностью обеспечивающего потребности в моменты $i : j-1$ (затраты на покупку в момент 0 принимаются равными 0).

Теорема 2. *При предположениях* (9) *задача управления запасами эквивалентна задаче о кратчайшем пути в описанном выше графе.*

Доказательство. Рассмотрим какую-либо политику запасания, удовлетворяющую условию, описанному в теореме 1 Пусть $i_1, i_2, \ldots, i_k$ — точки, в которых $y[i] > 0$. Поставим в соответствие этой политике путь

$$0 \to i_1 \to i_2 \to \ldots \to i_k \to t1. \qquad (10)$$

Легко видеть, что $i_2 \geqslant j1$, и поэтому такой путь принадлежит графу Также очевидно, что длина этого пути равна затратам, связанным с исходной политикой запасания.

С другой стороны, каждый путь вида (10) порождает политику запасания, в которой вершины пути являются моментами пополнения запасов и затраты в которой равны длине пути. ▲

Теперь мы можем перенести на задачу о запасах методы нахождения кратчайшего пути.

Обозначим через $v[i]$ минимальные затраты, необходимые для обеспечения продуктом до момента $i-1$ включительно. Тогда, так как эти $v[i]$ являются потенциалами в задаче о кратчайшем пути, мы получаем

$$v[j] = \min\{c[i, j] + v[i] \mid i < j\}. \qquad (11)$$

Вычислительный процесс может строиться здесь так же, как и в предыдущих случаях — последовательным вычислением $v[i]$ для $i = 1, 2, \ldots$ Отметим, что вычисления могут быть существенно сокращены благодаря отбрасыванию неко-

торых дуг. Так, например, заведомо нецелесообразно заготавливать в момент i продукт для момента $j > i$, если

$$g \times r\,[j] \times (j - i) > h_i. \qquad (12)$$

В следующей главе мы рассмотрим другие, более сложные варианты этой задачи, в которых спрос на продукцию случаен (с известной функцией распределения).

Отметим еще, что уравнение (11) могло быть написано в другом варианте: вместо дерева кратчайших путей *от* вершины 0 мы могли строить дерево кратчайших путей *к* вершине $t1 + 1$. Тогда $v\,[j]$ имело бы смысл минимальных затрат, необходимых для обеспечения продуктом с момента j включительно, и для $v\,[j]$ мы получили бы уравнение

$$v\,[i] = \min\{c\,[i,\ j] + v\,[j] \mid j \geqslant i\}. \qquad (13)$$

Такая форма уравнения Беллмана со счетом «с конца» процесса часто оказывается более удобной, особенно для случайных процессов.

§ 3. Общая схема динамического программирования. Детерминированный случай

Мы рассмотрели две экстремальные задачи, которые оказались частными случаями задачи о нахождении наилучшего пути для некоторых специальным образом построенных графов. В обеих этих задачах оказалось возможным придать четкий смысл вводимым потенциалам вершин графа, и в обеих задачах решение первоначальной задачи оказалось замененным решением целого семейства однотипных задач, в которое первоначальная задача входит в качестве одного из элементов. Впрочем, то же самое верно и по отношению к задаче о кратчайшем пути. Более того, можно отметить еще и весьма тесную связь задач каждого семейства: например, зная кратчайшие пути до всех вершин, непосредственно предшествующих вершине i, мы можем найти кратчайший путь и до i (перебирая для этого всех ее предшественников и наращивая их кратчайшие пути, переходами до i). Такие же действия мы проводили фактически при любом последовательном нахождении потенциалов в других задачах.

В этой главе мы опишем те общие свойства и особенности моделей, которые позволяют использовать технику

рекуррентного решения задачи. Мы будем говорить о *процессе* с множеством *состояний* M. Этот процесс будет переходить из одних состояний в другие под воздействием *управлений* или *решений* (какого-то лица или каких-то лиц). Переход из состояния в состояние может быть детерминированным — в таком случае он однозначно определяется управлением — или случайным — в этом случае мы будем считать, что управление u задает на M распределение вероятностей $p_u[M]$ и при выборе этого управления u случайный механизм в соответствии с распределением $p_u[M]$ выбирает новое состояние процесса. При этом результат работы этого случайного механизма стохастически не связан с его действиями на предыдущих шагах и целиком определяется выбором управления. Впрочем, в этой главе мы будем заниматься исключительно детерминированной схемой, а к стохастической обратимся в следующей.

В предыдущем абзаце неявно сделано несколько важных предположений об изучаемом процессе. Подчеркнем эти предположения:

Мы считаем, что *в процессе выделены такие состояния, а каждому состоянию сопоставлены такие управления, что управление, выбранное в данном состоянии при любой предыстории процесса, определяет* (в детерминированном случае полностью, а в стохастическом случае с точностью до выбора некоторого случайного воздействия) *следующее состояние процесса.*

Выбор таких состояний и управлений не всегда возможен, а когда возможен, то часто неоднозначен, и от успешности этого выбора существенно зависит эффективность применения методов динамического программирования в конкретных задачах.

Итак, пусть каждому состоянию $i \in M$ поставлено в соответствие множество управлений N_i и каждому управлению $u \in N_i$ соответствует $j(u) \in M$ — состояние, в которое процесс попадает в результате использования u. Пусть процесс находится в состоянии i_0. Выбор $u_0 \in N_{i_0}$ переводит процесс в состояние $i_1 = j(u_0)$, выбор $u_1 \in N_{i_1}$ — в состояние $i_2 = j(u_1)$ и т. д. Мы получаем последовательность пар

$$(i_0, u_0), \quad (i_1, u_1), \ \ldots,$$

где для всех k

$$u_k \in N_{i_k}, \quad i_{k+1} = j(u_k). \tag{14}$$

Такая последовательность называется *траекторией* процесса. Траектория может быть конечной (и в этом случае она заканчивается не парой из состояния и управления, а отдельным состоянием, которое будет называться *концом траектории*) или бесконечной. В некоторых случаях нам придется предполагать, что бесконечные траектории у рассматриваемого нами процесса невозможны, или добиваться этого искусственно.

Задачи, которые мы будем решать в этой главе, выглядят примерно следующим образом.

Типовая задача.

Задана функция f на множестве траекторий T. Требуется выбрать траекторию $t \in T$, на которой достигается минимум (или максимум) функции f.

Функции f, о которых будет идти обычно речь, будут функциями из довольно узкого класса, который нам уже встречался в § 4 гл. 1.

Мы можем рассматривать траекторию как список пар (i, u). Очевидно, что после отделения от траектории t начальной пары (i_0, u_0) остаток траектории $tail\ (t)$ сам может рассматриваться как траектория с началом в состоянии $i_1 = j\ (u_0)$. Потребуем от множества T, чтобы из $t \in T$ следовало $tail\ (t) \in T$.

Будем предполагать, что функция f, заданная на T, удовлетворяет условию:
если траектория $t \in T$ представлена в виде

$$t = (head\ (t),\ tail\ (t)) = ((i,\ u),\ t'), \tag{15}$$

то

$$f\ (t) = c\ (u) + \beta\ (u) \times f\ (t'), \tag{16}$$

т. е. $f\ (t)$ *рекуррентна* в смысле § 4 гл. 1. Будет предполагаться, что $\beta \geqslant 0$.

Множество траекторий будет определяться способом выбора управлений в состояниях процесса. Наиболее общий способ задания правила для выбора управления состоит в том, что задается решающая функция.

Рассмотрим функцию d, определенную для всех конечных траекторий. Обозначим конец траектории t через $tip(t)$. Функция d называется *решающей функцией* (или *решающим правилом*), если для всех $t \in T$

$$d\ (t) \in N_{tip(t)}. \tag{17}$$

Будем говорить, что траектория t определяется решающим правилом d, если для любого k справедливо соотношение

$$u_k = d(t_k),$$

где

$$t_k = (i_0, u_0), (i_1, u_1), \ldots, (i_{k-1}, u_{k-1}), i_k$$

— конечная траектория, совпадающая с началом траектории t.

Множество траекторий, определяемых произвольными решающими правилами, обозначим через T°.

Назовем решающее правило *стационарным*, если

$$d(t) = d(tip(t)), \tag{18}$$

т. е. если управление зависит только от конца пройденной части траектории. Стационарные решающие правила будут называться также *политиками*. Множество траекторий, определяемых политиками, обозначим через T^p.

В некоторых случаях мы будем ограничивать число шагов в траектории, в других — траектории будут конечными по смыслу задачи без особых предположений.

В рассматривавшихся нами задачах всюду $\beta \equiv 1$ и траектории конечны. Условие конечности траекторий процесса может быть легко интерпретировано в терминах теории графов. Действительно, описанному нами процессу можно сопоставить граф Γ с множеством вершин M и множеством дуг $N = \bigcup N_i$, причем дуга u, принадлежащая множеству N_i, имеет в качестве начала вершину i и в качестве конца вершину $j(u)$. Траектории процесса, очевидно, сопоставляется путь в этом графе. Таким образом, в случае конечного графа для того чтобы процесс динамического программирования не имел бесконечных траекторий, необходимо и достаточно, чтобы соответствующий ему граф не имел контуров.

Пусть мы решаем задачу максимизации функции f на траекториях множества T, начинающихся из i. Обозначим множество таких траекторий через $T(i)$, а множество траекторий, начинающихся с пары (i, u), через $T(i, u)$.

Положим

$$v[i] = \max\{f(t) \mid t \in T(i)\}. \tag{19}$$

Так как $T(i, u)$, $u \in N_i$ образуют разбиение множества $T(i)$, то

$$v[i] = \max\{\max\{f(t) \mid t \in T(i, u)\} \mid u \in N_i\}. \quad (20)$$

Но для траекторий из $T(i, u)$ справедливо (16), и поскольку первое слагаемое для всех траекторий одинаково, его можно вынести за знак максимума. Точно так же можно вынести и неотрицательный множитель $\beta(u)$.

Оставшийся максимум будет максимумом по некоторому подмножеству траекторий из $T(j(u))$. Если оно совпадает с $T(j(u))$, то этот максимум в соответствии с (19) будет равен $v[j(u)]$.

Таким образом, мы имеем окончательно: если множество T таково, что для любого i

$$T(i) = \bigcup T(i, u), \quad T(i, u) = \{(i, u), t' \mid t' \in T(j(u))\}, \quad (21)$$

то

$$v[i] = \max\{c[u] + \beta[u] \times v[j(u)] \mid u \in N_i\}, \quad (22)$$
$$v[i] = f[i] \text{ при } N_i = \emptyset.$$

Уравнение (22) называется *уравнением Беллмана* или *уравнением динамического программирования*. Разрешимостью этого уравнения в общем случае мы займемся в следующей главе, а здесь ограничимся рассмотрением нескольких важных частных случаев.

Предложение 1. *Множество* T^0 *удовлетворяет условию* (21).

Доказательство следует из того, что любой путь в графе можно разбить на начальную дугу и остаток пути, и, наоборот, присоединение к концу дуги некоторого пути образует путь, идущий из начала дуги.

Определенные нами решающие правила таковы, что любой путь в графе переходов процесса может соответствовать некоторому решающему правилу.

Отметим, что множество T^p условию (21) не удовлетворяет.

В дальнейшем мы всюду будем рассматривать задачи максимизации f на T^0, но, как правило, оптимальное решающее правило будет политикой.

§ 4. Достаточные условия разрешимости уравнения Беллмана

Здесь мы рассмотрим два вида достаточных условий. В первом будет использована конечность траекторий, обеспечиваемая отсутствием контуров в графе переходов, во втором существенно используется возможность накладывать условия на множитель β в представлении (16).

Теорема 3. *Если в графе переходов* Г, *соответствующем рассматриваемому процессу динамического программирования, нет контуров, то уравнение* (22) *имеет единственное решение и набор управлений, на которых достигается максимум в* (22), *порождает для любого начального состояния траекторию, на которой достигается максимум функции* f.

Доказательство. Разрешимость и единственность решения уравнения (22) очевидны. Для $M_0 = \{i \mid N_i = \varnothing\}$ значения $v[i]$ известны. Поэтому можно вычислить $v[i]$ для тех i, из которых процесс на следующем шаге обязательно попадает в M_0. Обозначим объединение M_0 из этого нового множества через M_1. Теперь можно вычислить $v[i]$ для состояний, из которых процесс попадает на следующем шаге в M_1. Так как граф не имеет контуров и конечен, то за конечное число таких итераций будут исчерпаны все его вершины.

Сопоставим теперь каждому $i \notin M_0$ управление $\bar{u}_i \in N_i$, на котором достигается максимум в (22). Тогда

$$v[i] = c(\bar{u}_i) + \beta(\bar{u}_i) \times v[j(\bar{u}_i)] \tag{23}$$

и

$$v[i] \geqslant c(u) + \beta(u) \times v[j(u)] \quad \text{для} \quad u \in N_i. \tag{24}$$

Рассмотрим какую-либо траекторию t, ведущую из вершины i

$$(i, u_0), (i_1, u_1), \ldots$$

Складывая вдоль по траектории неравенства (24), взятые с такими множителями, чтобы взаимно уничтожились слагаемые с $v[i_k]$, мы получим

$$v[i] \geqslant c(u_0) + \beta(u_0) \times (c(u_1) + \\ + \beta(u_1) \times (c(u_2) + \ldots)) = f(t).$$

Вместе с тем на траектории $\bar{t}$, порожденной управлениями

$\bar{u}$, ввиду равенств (23) мы будем иметь

$$v[i] = f(\bar{t}),$$

что и доказывает теорему. ▲

Теорема 4. *Если существует такое* $\bar{\beta} > 1$, *что для всех* $u \in N$ *справедливо* $\beta(u) \geqslant \bar{\beta}$, *то уравнение* (22) *имеет единственное решение и набор управлений, на которых достигается максимум в* (22), *порождает для любого начального состояния траекторию, на которой достигается максимум функции* f.

Доказательство. Нам часто потребуется по вектору $x[M]$ строить вектор $x_1[M]$, где

$$x_1[i] = \max\{c(u) + \beta(u) \times x[j(u)] \mid u \in N_i\} \quad (25)$$

(применять к x преобразование Беллмана). Мы будем в таком случае писать

$$x_1 = \mathbf{Bel}(x). \quad (26)$$

Рассмотрим теперь два произвольных вектора $v[M]$ и $w[M]$ и построим по ним векторы $v_1 = \mathbf{Bel}(v)$ и $w_1 = \mathbf{Bel}(w)$. Для разности этих векторов мы имеем

$$\begin{aligned} v_1[i] - w_1[i] = \\ = \max\{c(u) + \beta(u) \times v[j(u)] \mid u \in N_i\} - \\ - \max\{c(u) + \beta(u) \times w[j(u)] \mid u \in N_i\} \leqslant \\ \leqslant \max\{c(u) + \beta(u) \times v[j(u)] - \\ - (c(u) + \beta(u) \times w[j(u)]) \mid u \in N_i\} = \\ = \max\{\beta(u) \times (v[j(u)] - w[j(u)]) \mid u \in N_i\}, \end{aligned}$$

так как разность максимумов не превосходит максимума разности. Отсюда

$$\begin{aligned} |v_1[i] - w_1[i]| \leqslant \\ \leqslant \max\{\beta(u) \times |v[j(u)] - w[j(u)]| \mid u \in N_i\} \leqslant \\ \leqslant \bar{\beta} \times \max_{u \in N_i} |v[j(u)] - w[j(u)]| \leqslant \\ \leqslant \bar{\beta} \times \max_{j \in M} |v[j] - w[j]|, \end{aligned}$$

и, следовательно,

$$\max_{i \in M} |v_1[i] - w_1[i]| \leqslant \bar{\beta} \times \max_{i \in M} |v[i] - w[i]|. \quad (27)$$

Из этого неравенства вытекает невозможность двух различных решений уравнения (18): векторы, вычисленные по этим решениям в соответствии с (21), должны с ними совпадать, а максимум разности компонент этих векторов a

должен по (22) удовлетворять неравенству $0 < a \leqslant \bar{\beta} \times a$, что невозможно.

Рассмотрим теперь последовательность векторов $v_k\,[M]$, где $v_0\,[M]$ — произвольный вектор, а

$$v_k = \mathbf{Bel}\,(v_{k-1}), \qquad k \leqslant 1. \tag{28}$$

Если $v_0\,[M]$ выбрать так, чтобы выполнялось неравенство

$$|\,v_0\,[i]\,| \leqslant \bar{c}/(1 - \bar{\beta}), \tag{29}$$

где $\bar{c} = \max\{\,|\,c\,(u)\,|\,|\,u \in N\}$, то этому же неравенству будет удовлетворять любой из векторов $v_k\,[M]$. Действительно, предполагая, что (29) выполнено для v_{k-1}, мы получаем из (28) и (25)

$$|\,v_k\,[i]\,| \leqslant \max\{\,|\,c\,(u) + \beta\,(u) \times v_{k-1}\,[j\,(u)]\,|\,|\,u \in N_i\} \leqslant$$
$$\leqslant \bar{c} + \bar{\beta} \times \bar{c}/(1 - \bar{\beta}) = \bar{c}/(1 - \bar{\beta}).$$

Следовательно, при выполнении неравенства (29) последовательность $\{v_k\,[M]\}$ ограничена. Взяв $v\,[M] = v_k\,[M]$, $w\,[M] = v_{k+1}\,[M]$, мы заключаем из (22), что она сходится в себе и, следовательно, имеет предел. Этот предел, очевидно, удовлетворяет уравнению (22).

Отметим, что из сходимости такой последовательности следует и сходимость последовательности с любым начальным вектором $v_0\,[M]$.

Вторая часть доказательства очень похожа на рассуждения в теореме 3. Обозначим через $\bar{u}_i$ управление, на котором достигается максимум в уравнении (20). Для $\bar{u}$ имеет место равенство (23), и, кроме того, справедливы неравенства (24). Для любой траектории

$$t\,(i_0) = (i_0,\ u_0),\ (i_1,\ u_1),\ \ldots$$

значение $f\,(t\,(i_0))$ в силу сделанного в теореме предположения является суммой абсолютно сходящегося ряда

$$f\,(t\,(i_0)) = c\,(u_0) + \sum_{k=1}^{\infty} c\,(u_k) \times \Pi_{l \in 0:k-1}\,\beta\,(u_l). \tag{30}$$

Из (24) имеем

$$c\,(u) \leqslant v\,[i] - \beta\,(u) \times v\,[j\,(u)] \text{ для } u \in N_i$$

и, следовательно,

$$f(t(i_0)) \leqslant \sum_{k=0}^{\infty} (v[i_k] - \beta(u_k) \times v[i_{k+1}]) \times$$
$$\times \prod_{l \in 0:k-1} \beta(u_l) = \sum_{k \geqslant 0} v[i_k] \times \prod_{l \in 0:k-1} \beta(u_l) -$$
$$- \sum_{k \geqslant 0} v[i_{k+1}] \times \prod_{l \in 0:k} \beta(u_l) = v[i_0],$$

причем для $\{\bar{u}_l\}$ здесь выполняется равенство. ▲

Введенные в этом доказательстве последовательные приближения для потенциалов $v[M]$ могут быть использованы и для численного решения уравнения (22), причем и в условиях теоремы 3. Эффективность алгоритма последовательных приближений существенно зависит от удачного выбора начального приближения.

Отметим, что если $v_0[M]$ удовлетворяет неравенству

$$v_0 \geqslant \mathbf{Bel}\, v_0 \tag{31}$$

или неравенству

$$v_0 \leqslant \mathbf{Bel}\, v_0, \tag{32}$$

то последовательность $v_k[M]$ удовлетворяет соответственно неравенству

$$v_k[M] \geqslant v_{k+1}[M] \tag{33}$$

или

$$v_k[M] \leqslant v_{k+1}[M]. \tag{34}$$

Оба неравенства доказываются просто и совершенно одинаково по индукции. Действительно, например, для (33) имеем

$$v_k[i] = \max\{c(u) + \beta(u) \times v_{k-1}[j(u)] \mid u \in N_i\} \geqslant$$
$$\geqslant \max\{c(u) + \beta(u) \times v_{k-2}[j(u)] \mid u \in N_i\} = v_{k-1}[i].$$

Имея вектор $v_0[M]$, удовлетворяющий (31) или (32), можно, таким образом, построить монотонную последовательность $\{v_k[M]\}$.

Особенно удачный метод имеется для нахождения функции, удовлетворяющей (32). Зафиксируем какую-либо политику $u0[M]$ и решим систему уравнений

$$w[i] = c(u0[i]) + \beta(u0[i]) \times w[j(u0[i])]. \tag{35}$$

Выполнение (32) для так определенного $w[M]$ очевидно. Более сложным вариантом этого подхода является постепенное расширение множеств N_i: если в каждом непустом N_i выделить непустое подмножество N_i' и затем решить

уравнение

$$w[i] = \max \{c(u) + \beta(u) \times w[j(u)] \mid u \in N_i'\}, \quad (36)$$

то его решение будет также удовлетворять (32).

Настолько же универсальный метод построения решения неравенства (31) неизвестен. В конкретных задачах успех достигается расширением множеств N_i (за счет произвольных дополнительных управлений) и увеличением $c(u)$ на имеющихся управлениях.

§ 5. Асимптотическое поведение последовательных приближений

Естественно, что должно существовать много процессов, в которых последовательность $v_k[M]$ не сходится. Оставляя в стороне полную классификацию возможных случаев, рассмотрим один полезный класс таких процессов.

Ограничимся сейчас для простоты случаем, когда все $N_i \neq \emptyset$. Если бы нам потребовалось найти траекторию t, начинающуюся из вершины i, состоящую из k переходов и максимизирующую $f(t) + v_0[i_k]$, то, как легко убедиться, искомый максимум как раз и равнялся бы $v_k[i]$.

Легко представить себе процесс, который на каждом шаге приносит доход, в точности или приблизительно равный некоторому $\lambda > 0$. Для такого процесса $v_k[i]$ должно расти пропорционально k.

Более точно эта ситуация описывается следующей теоремой.

Теорема 5. *Пусть* $\beta(u) \equiv 1$ *и граф переходов процесса вполне связен. Пусть* λ *— максимум отношений суммы доходов* $c(u)$ *на контуре к числу дуг контура, найденный на контурах графа переходов. Тогда:*

а) *найдутся такие константы* a_1 *и* a_2, *что для всех* $i \in M$ *и всех* $k = 1, 2, \ldots$ *выполняется неравенство*

$$a_1 \leqslant v_k[i] - k \times \lambda \leqslant a_2; \quad (37)$$

б) *найдется такая константа* r, *что при любом* k *оптимальная траектория будет состоять из обходов контуров с максимальным отношением и еще не более* r *переходов;*

в) *система*

$$v[i] = \max\{c(u) - \lambda + v[j(u)] \mid u \in N_i\}, \quad i \in M \qquad (38)$$

имеет решение.

Доказательство. Мы опробуем в этом доказательстве технику использования неравенств (31) и (32) и начнем его с п. в). Как мы знаем из § 9 гл. 3, в задаче об оптимальном контуре всегда есть оптимальное решение, а следовательно, и оптимальное решение $(\lambda, z[M])$ двойственной задачи Это решение удовлетворяет неравенствам

$$z[i] + \lambda \geqslant c(u) + z[j(u)] \text{ для } u \in N_i,\ i \in M, \qquad (39)$$

что эквивалентно (31). Вместе с тем, если зафиксировать какой-либо базис, состоящий из оптимального контура и выходящих на него путей, мы получим некоторую политику $\bar{u}[M]$, для которой разрешимо уравнение (35) для двойственных переменных

$$w[i] + \lambda = c(\bar{u}[i]) + w[j(\bar{u}[i])]$$

(действительно, на оптимальном контуре эта система разрешима, а все остальные переменные вычисляются по найденным ранее). Эта система разрешима с точностью до постоянного слагаемого. Выберем такое решение, чтобы $w[M] \leqslant z[M]$, и построим по $w_0 = w$ последовательность $\{w_k[M]\}$

$$w_k[i] = \max\{c(u) - \lambda + w_{k-1}[j(u)] \mid u \in N_i\}. \qquad (40)$$

Из монотонности этой последовательности и из $w_k[M] \leqslant z[M]$ мы получаем ее ограниченность

$$w_{k+1}[i] = \max\{c(u) - \lambda + w_k[j(u)] \mid u \in N_i\} \leqslant$$
$$\leqslant \max\{c(u) - \lambda + z[j(u)] \mid u \in N_i\} = z[i].$$

Последовательность сходится к некоторой функции $\bar{w}[M]$, являющейся решением уравнения (38). Вместе с $\bar{w}[M]$ решением этого уравнения будет любой вектор $\bar{w}[M] + a \times \mathbf{1}[M]$. Выберем a_1 и a_2 так, чтобы

$$\bar{w}[M] + a_1 \times \mathbf{1}[M] \leqslant v_0[M] \leqslant \bar{w}[M] + a_2 \times \mathbf{1}[M].$$

Докажем теперь, что для любого $k > 0$ справедливо неравенство

$$w[M] \times a_1 \times \mathbf{1}[M] \leqslant v_k[M] - k \times \lambda \times \mathbf{1}[M] \leqslant$$
$$\leqslant \bar{w}[M] + a_2 \times \mathbf{1}[M]$$

Пусть оно верно для какого-либо $k \geqslant 0$. Соотношение

$$v_{k+1}[i] = \max \{c(u) + v_k[j(u)] \mid u \in N_i\}$$

можно переписать в виде

$$v_{k+1}[i] - (k+1) \times \lambda = \\ = \max \{c(u) - \lambda + (v_k[j(u)] - k \times \lambda) \mid u \in N_k\}.$$

Воспользовавшись индукционным предположением, убеждаемся, что неравенство имеет место и для $k+1$. Из этого неравенства уже легко получить (подправляя a_1 и a_2) требуемое неравенство (37).

Для доказательства п. б) рассмотрим оптимальную k-шаговую траекторию. Эта траектория, представляющая собой путь в графе переходов, может быть разбита (неоднозначно, это для нас несущественно) на путь без самопересечений и на некоторое число простых контуров. Путь состоит из r_1 (не более чем из $|M| - 1$) переходов, и получаемый на нем доход не превосходит $\bar{c} + r_1$, где $\bar{c} = \max c(u)$. Среди простых контуров возможно некоторое число неоптимальных. Если обозначить через λ' вторую по величине после максимальной характеристику простого контура, то доход, получаемый на этих контурах, можно оценить сверху величиной $\lambda' \times r_2$, где r_2 число переходов, попавших в неоптимальные контуры. Доход на оптимальных контурах будет равен $(k - r_1 - r_2) \times \lambda$. Итак,

$$v_k[i] \leqslant (k - r_1 - r_2) \times \lambda + \bar{c} \times r_1 + \\ + \lambda' \times r_2 + \max \{v_0[j] \mid j \in M\}.$$

Используя левую часть (37), получаем

$$a_1 - \max \{v_0[j] \mid j \in M\} \leqslant (\bar{c} - \lambda) \times r_1 - (\lambda - \lambda') \times r_2.$$

Отсюда

$$r_2 \leqslant (\max \{v_0[j] \mid j \in M\} - a_1 + \\ + (\bar{c} - \lambda) \times (|M| - 1))/(\lambda - \lambda'),$$

и в качестве r можно принять эту оценку для r_2, увеличенную на $|M| - 1$. ▲

Пункт б) доказанной теоремы известен под названием «теорема о магистрали» (в слабой форме). Его можно истолковывать следующим образом: самый выгодный путь почти целиком состоит из движения по «магистрали» — грубо

говоря, скоростному пути, на который траектория должна выйти и с которого она должна сойти в конце.

Вообще говоря, по различным причинам у траектории могут быть отклонения от такой структуры, но при дополнительных предположениях о процессе удается установить, что она имеет в точности такой вид.

Условия, наложенные в следующей ниже теореме, возможно выглядят несколько искусственно, но это впечатление обманчиво — они существенно связаны с природой задачи.

Теорема 6. *Если частичный подграф* $\langle M', N' \rangle$ *графа переходов, включающий только дуги, входящие в оптимальные контуры, вполне связен и ОНД длин оптимальных контуров равен* 1, *то для достаточно больших* k *и любого* $i_0 \in M'$

$$v_k[i] = z[i] + k \times \lambda + \overline{w}, \quad i \in M, \tag{41}$$

где $z[i]$ *— максимальный доход, получаемый на пути от* i *до* i_0 *(считаем* $c'(u) = c(u) - \lambda$ *доходом на дуге* u*),*

$$\overline{w} = \max\{w_l[i_0] \mid l \geqslant 1\}, \tag{42}$$

а последовательность $\{w_l\}$ *определена соотношением* (40).

Доказательство. Прежде всего отметим, что $z[i]$ и w существуют. Значения $z[i]$ получаются, например, методом потенциалов при нахождении наилучших путей, и метод применим, так как по выбору λ контуров с положительным доходом c' в графе нет. Максимум в (42) достигается. Действительно, так как доход, получаемый на контурах, не положителен, то можно ограничиться траекториями, не имеющими самопересечений, а число таких траекторий конечно.

Далее вместо (41) мы можем доказывать эквивалентное равенство

$$w_k[i] = z[i] + \overline{w}.$$

Покажем сначала, что $w_k[i] \leqslant z[i] + \overline{w}$. Рассмотрим траекторию, на которой достигается доход $w_k[i]$. По теореме 5 при достаточно большом k в этой траектории найдется вершина $i_1 \in M'$. Если в траектории не участвует i_0, то можно добавить путь от i_1 до i_0 и от i_0 до i_1 (такие пути существуют вследствие того, что граф $\langle M', N' \rangle$ вполне связен). Суммарный доход, получаемый на этом добавленном пути, который составлен из оптимальных контуров, равен нулю и,

следовательно, доход на получившейся траектории также равен $w_k[i]$. Разобьем теперь эту траекторию на части до i_0 и от i_0 до конца. Доход на первой части не превосходит $z[i]$, а на второй не превосходит $\overline{w}$.

Построим теперь траекторию, доход на которой равен $w_k[i]$. Началом этого пути будет траектория, на которой достигается $z[i]$, концом — траектория, на которой достигается $\overline{w}$. Середина будет представлять собой замкнутый путь от i_0 до i_0, составленный из оптимальных контуров. Как отмечалось в § 3 гл. 3, во вполне связном графе с общим наибольшим делителем длин контуров, равным 1, можно выбрать замкнутый путь любой достаточно большой длины с началом и концом в данной точке.

Так как доход, получаемый в начале пути, равен $z[i]$, на конце $\overline{w}$, а середина пути представима в виде суммы оптимальных контуров и поэтому доход на ней равен 0, общий доход на траектории действительно равен $w_k[i]$. ▲

Более сложные случаи существования и несуществования решений уравнений Беллмана и асимптотического поведения функций v_k мы отложим до следующей главы, где изучаются произвольные уравнения с $0 \leqslant \beta(u) \leqslant 1$. Ниже будут рассмотрены варианты уравнений Беллмана для некоторых задач, не входящие в это общее рассмотрение. Но прежде приведем еще несколько примеров, в которых возникают уравнения в точности рассмотренного типа.

§ 6. Задача замены оборудования

Предположим, что требуется обеспечить присутствие на некотором производстве работоспособного станка в моменты времени $1 : t$. Станок может выдержать не более s моментов. От его возраста (который измеряется целым числом из $0 : s$) зависят эксплуатационные затраты. Пусть заданы $c[k]$ — затраты на эксплуатацию в течение единицы времени станка возраста k. Задана также стоимость нового станка $c0$. Для сопоставления затрат, производимых в различные моменты времени, обычно используется *приведение затрат* — затраты в момент τ считаются меньше затрат в момент 0 и для сопоставления умножаются на некоторый множитель, называемый *коэффициентом приведения*. Обычно коэффициент приведения из экономических соображений принимается равным β^τ, где β — заданное число, $0 < \beta < 1$

Состояние этого процесса естественно описать возрастом станка $k \in 1 : s$ и временем, оставшимся до конца процесса $\tau \in 0 : t$. При $\tau = 0$ предстоящие затраты равны 0. Управление во всех состояниях (k, τ), где $k < s$, одно и то же: купить новый станок или оставить старый (переход в $(1, \tau - 1)$ с затратами $c0 + c[0]$ или в $(k + 1, \tau - 1)$ с затратами $c[k]$), в состоянии (s, τ) возможна только покупка нового станка (переход в $(1, \tau - 1)$ с затратами $c0 + c[0]$).

Для этого процесса мы, естественно, получаем функциональное уравнение

$$\begin{aligned} v[k, \tau] &= \min \{c0 + c[0] + \beta \times v[1, \tau - 1], \\ &\qquad c[k] + \beta \times v[k + 1, \tau - 1]\} \text{ при } k < s, \\ v[s, \tau] &= c0 + c[0] + \beta \times v[1, \tau - 1], \\ v[k, 0] &\equiv 0. \end{aligned} \tag{43}$$

Это уравнение может рассматриваться как рекуррентное соотношение для последовательных приближений, сходящихся (по теореме 4) к решению функционального уравнения

$$\begin{aligned} v[k] &= \min \{c0 + c[0] + \beta \times v[1], \\ &\qquad c[k] + \beta \times v[k + 1]\}, \\ v[s] &= c0 + c[0] + \beta \times v[1]. \end{aligned} \tag{44}$$

Здесь, очевидно, $v[k]$ имеет смысл минимума приведенных затрат на бесконечно продолжающийся процесс, начинающийся с состояния k.

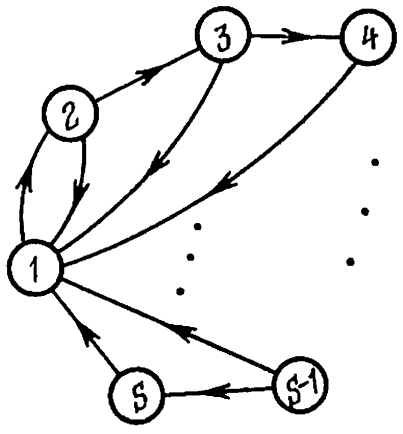

Рис. 65. Граф переходов

Структура оптимальной политики для (44) легко устанавливается по графу переходов данного процесса (рис. 65). Состояние 1 в любом случае участвует в процессе, исходящая из него траектория в каком-то состоянии k_0 обязательно возвращается в 1 и замыкает контур, только этот контур и представляет для нас интерес. Для этого контура мы получаем

$$v^{k_0}[i] = c[i] + \beta \times v^{k_0}[i + 1]$$

при $i < k_0$ и

$$v^{k_0}[k_0] = c0 + c[0] + \beta \times v^{k_0}[1],$$

откуда

$$v^{k_0}[1] = \Sigma_{i \in 1 : k_0 - 1} c[i] \times \beta^{i-1} + \\ + (c0 + c[0]) \times \beta^{k_0} + v^{k_0}[1] \times \beta^{k_0+1}$$

и окончательно

$$v^{k_0}[1] = (\Sigma_{1 : k_0 - 1} c[i] \times \beta^{i-1} + (c0 + c[0]) \times \beta^{k_0})/(1 - \beta^{k_0+1}).$$

Оптимальная политика здесь определяется выбором k_0, при котором $v^{k_0}[1]$ достигает минимума.

Эта оптимальная политика и найденное по ней решение уравнения (44) могут служить хорошим приближением для t-шаговой оптимальной политики и решений уравнения (43).

§ 7. Задачи оптимального резервирования и надежности

Многочисленные экстремальные задачи, приводящие к схеме динамического программирования, возникают также в связи с вопросом обеспечения максимальной надежности сложных устройств. В этом параграфе мы рассмотрим несколько таких задач.

Несколько схем обеспечения надежности можно объединить следующей формулировкой.

Рассматривается устройство, состоящее из множества элементов N. Вероятность выхода из строя элемента $j \in N$ равна $p[j]$. Устройство выходит из строя, если выходит из строя какой-либо из его элементов. Таким образом, вероятность выхода из строя устройства равна (в предположении независимости поломок элементов)

$$1 - \Pi_{j \in N} (1 - p[j]).$$

Предположим, что $p[j]$ зависит от некоторого параметра $\vartheta[j]$ и можно выбирать набор параметров $\vartheta[N]$ из некоторого множества Θ. Естественно возникает задача:

Минимизировать

$$p(\vartheta) = 1 - \Pi_{j \in N} (1 - p[j, \vartheta[j]]) \qquad (45)$$

по

$$\vartheta = \vartheta[N] \in \Theta.$$

Один из простейших параметров, управляющих надежностью, — число резервных элементов. Для каждого элемента в схему вводятся дополнительные его экземп-

ляры, и элемент считается не вышедшим из строя, пока действует хотя бы один экземпляр данного элемента. При k дополнительных экземплярах j-го элемента принимается $p(j, k) = p[j]^{k+1}$ и

$$p(\vartheta) = 1 - \Pi_{j \in N}(1 - p[j]^{1+k_j}). \quad (46)$$

Такая форма зависимости возникает в схеме *горячего резервирования* (все элементы находятся под нагрузкой). Минимизация $p(\vartheta)$ эквивалентна максимизации вероятности исправной работы

$$q(\vartheta) = \Pi_{j \in N}(1 - p[j]^{1+k_j}). \quad (47)$$

Рассмотрим метод решения этой задачи с помощью рекуррентных соотношений Беллмана, задавая область Θ линейным неравенством

$$\Sigma_{j \in N} w_j \times k_j \leqslant \overline{w} \quad (48)$$

(это может быть, например, ограничение по весу или по стоимости дополнительных экземпляров элементов). Пусть $N = 1 : n$.

В духе предыдущего рассмотрим задачу оптимального резервирования для схемы с $N = 1 : \nu$ и $\overline{w} = \omega$. Обозначим через $v_\nu(\omega)$ максимум функции (47) при условиях (48). Тогда, очевидно,

$$v_0(\omega) \equiv 1,$$

$$v_\nu(\omega) = \max\{(1 - p[\nu]^{1+k}) \times v_{\nu-1}(\omega - k \times w_\nu) \mid 0 \leqslant k \leqslant \omega/w_\nu\}. \quad (49)$$

По рекуррентным соотношениям (49), так же как и в предыдущих параграфах, можно найти значения $v_\nu(\omega)$ и оптимальную политику для всех значений параметра. Однако и здесь можно воспользоваться идеями переработки списка основных состояний. За счет этого удается избежать сложностей, связанных с непрерывной шкалой изменения параметра ω, и построить метод даже для значительно более сложного случая нескольких ограничений вида (48), практически не поддающийся исследованию с помощью обычных рекуррентных соотношений.

Пусть, кроме (48), мы имеем еще ограничение

$$\Sigma_{j \in N} c_j \times k_j \leqslant \overline{c}. \quad (50)$$

В этом случае вместо уравнений (49) мы имеем

$$v_0(\omega, \gamma) \equiv 1,$$

$$v_\nu(\omega, \gamma) = \max\{(1 - p[\nu]^{1+k}) \times v_{\nu-1}(\omega - k \times w_\nu, \gamma - k \times c_\nu)\}.$$

Рассмотрим на примере, как выглядит применительно к этой задаче метод переработки списка состояний. Пусть $n = 3$, $p[1:3] = (0.1, 0.2, 0.3)$, $w[1:3] = (3, 5, 4)$, $c[1:3] = (1, 1, 1)$, $w0 = 14$, $c0 = 4$. Возможные состояния после выбора k_1 таковы (первое число k_γ, второе $v_\gamma(\omega, \gamma)$, третье ω, четвертое γ):

0 (0,0.9 , 0,0),
1 (1,0.99 , 3,1),
2 (2,0.999 , 6,2),
3 (3,0.9999 , 9,3),
4 (4,0.99999,12,4).

После выбора k_2 получаем 11 состояний, из которых 4 оказываются хуже других. Чтобы провести сравнение, они помечены буквами:

a (0,0.72 , 0,0),
b (1,0.864 , 5,1),
c (2,0.8928 ,10,2),
d (0,0.792 , 3,1),
e (1,0.9504 , 8,2),
f (2,0.98208 ,13,3),
g (0,0.7992 , 6,2),
h (1,0.95904 ,11,3),
i (0,0.79992 , 9,3),
j (1,0.959904,14,4),
k (0,0.799992,12,4),

Состояние *b* доминирует *g* и *i*, *f* доминирует *j*, *c* доминирует *k*.

Из оставшихся 7 состояний выбором k_3 получаем 16 состояний, из которых 8 оказываются хуже других:

а (0,0.504 , 0,0),
б (1,0.6552 , 4,1),
в (2,0.70056 , 8,1),
г (3,0.714168,12,3),
д (0,0.6048 , 5,1),
е (1,0.78624 , 9,2),
ж (2,0.840672,13,3),
з (0,0.62496 ,10,2),
и (1,0.812448,14,3),
й (0,0.5544 , 3,1),
к (1,0.72072 , 7,2),
л (2,0.770616,11,3),
м (0,0.66528 , 8,2),
н (1,0.864864,12,4),
о (0,0.687456,13,3),
п (0,0.671328,11,3).

Вариант «н» оказывается наилучшим. Он соответствует вектору резервных элементов (1, 1, 1).

Отметим еще, что рассматриваемые нами состояния образуют дерево вариантов (рис. 66) — такое же, как в методе ветвей и границ, — и отличие метода переработки списка состояний заключается в том, что отбрасывание вариантов производится по результатам их попарного сравнения.

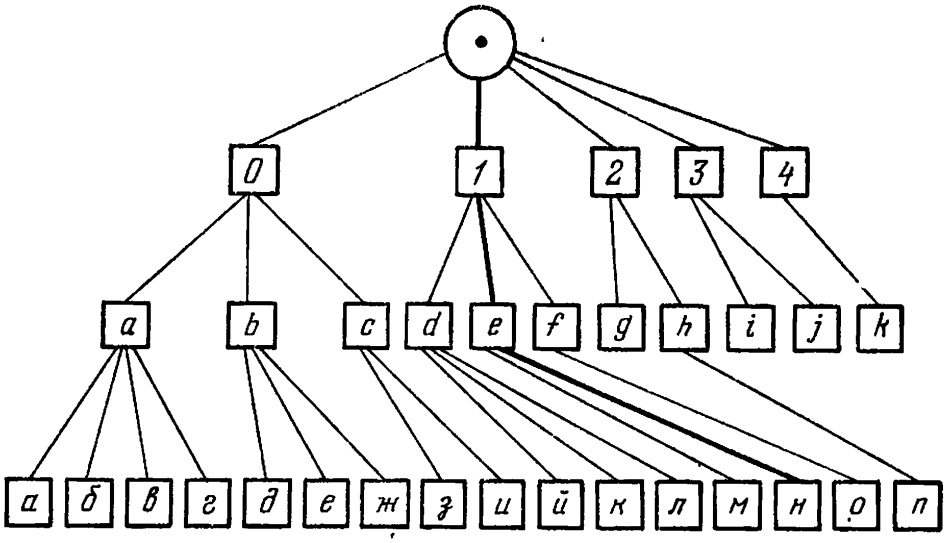

Рис. 66 Дерево вариантов в задаче резервирования

Другим часто встречающимся вариантом схемы резервирования является схема *холодного резервирования*, в которой резервные элементы включаются последовательно, по мере выхода из строя работающих элементов. Если требуемое время функционирования устройства равно t, а функция распределения времени безотказной работы j-го элемента $F_j(x)$, то

$$p[j] = F_j(t), \quad p(j, k) = F_j^{k+1}(t),$$

где F^r — r-кратная свертка функции распределения $F(x)$. Для решения задач оптимального холодного резервирования, а также и оптимального смешанного резервирования (для одних деталей — холодного, а для других — горячего) может использоваться тот же метод переработки списка состояний. Отличие будет лишь в способах перехода от состояния к состоянию.

В схеме холодного резервирования уже в большей степени проявляется процесс функционирования системы — в ней участвует время. Большое число задач надежности

связано со временем еще теснее — в них требуется определить оптимальный режим контроля и профилактики системы. Несколько таких задач будет рассмотрено в следующей главе.

§ 8. Поиск неисправности

Здесь будет рассмотрена модель несколько иного типа. Именно, в ней значение потенциала будет находиться не по одному, а по нескольким предшествующим состояниям.

Эта модель принадлежит к распространившемуся в последнее время классу моделей поиска неисправностей в сложных системах. Предположим, что требуется найти неисправность в агрегате, состоящем из множества элементов M (M можно считать и множеством различных *неисправностей*). Будем считать, что в агрегате неисправен ровно один элемент, причем вероятность того, что это элемент i, известна и равна $p[i]$ ($p[M]$ — вероятностный вектор).

Для обнаружения неисправности используются диагностические *тесты*. Множество тестов мы обозначим через N. Применение теста j приводит к появлению одного из двух ответов (например, 0 или 1) в зависимости от того, какая неисправность имеется в агрегате. Обозначим через M_j^0 множество неисправностей, приводящих к ответу 0 для теста j, и через M_j^1 множество неисправностей, дающих ответ 1. Задание M_j^0 и M_j^1 для всех $j \in N$ эквивалентно заданию матрицы $a[M, N]$, составленной из нулей и единиц. Естественно, что для разрешимости диагностической задачи необходимо, чтобы произведение разбиений (M_j^0, M_j^1), $j, \in N$, состояло из одноэлементных множеств. Легко видеть, что это условие эквивалентно тому, чтобы были различны все строки матрицы $a[M, N]$.

Каждому тесту j ставится в соответствие число $c[j]$ — затраты на использование теста. Требуется выбрать диагностическую процедуру, при которой математическое ожидание затрат было бы минимальным. Нужно подчеркнуть, что сами затраты в реальном диагностическом процессе случайны и поэтому для сравнения диагностических процедур нужно выбирать какие-то характеристики, описывающие процедуру в целом.

В данном случае математическое ожидание затрат представляется достаточно удобной величиной.

Диагностическая процедура представляет собой дерево, вершинами которого являются подмножества множества M. Само M входит в дерево в качестве начальной вершины, а концевыми вершинами дерева являются одноэлементные множества. Каждой неконцевой вершине соответствует некоторый тест, которым множество, представленное этой вершиной, разбивается на два подмножества, представленные двумя другими вершинами графа.

Пример. Пусть $M = 1:6$, $N = 1:9$, $c[N] = (3, 2, 4, 5, 6, 4, 4, 3, 3)$, $p[M] = (0.3, 0.2, 0.1, 0.2, 0.1, 0.1)$,

Тесты задаются табл. 3.

Таблица 3

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

Легко построить *какую-нибудь* диагностическую процедуру. Например, применение теста 1 делит M на $M_1 = \{1, 4, 5, 6\}$ и $M_2 = \{2, 3\}$. Далее M_1 делится тестом 2 на «концевое» множество $M_{11} = \{1\}$ и множество $M_{12} = \{4, 5, 6\}$. Множество M_{12} делится тестом 3 на $M_{121} = \{4\}$ и $M_{122} = \{5, 6\}$. Наконец, тестом 5 множество M_{122} делится на одноэлементные. Этим же тестом делится и множество M_2. Итак, получено диагностическое дерево, изображенное на рис. 67.

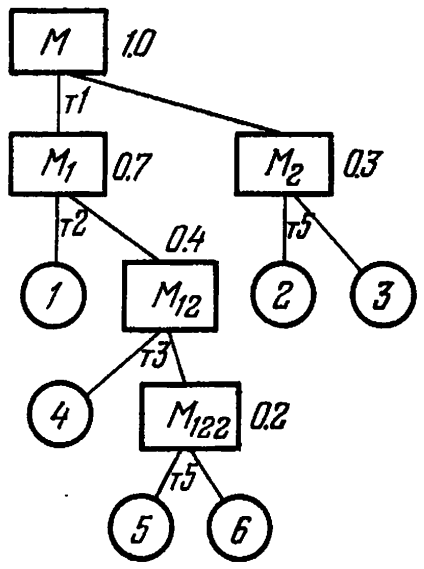

Рис. 67. Возможное дерево проверок.

Около каждой неконцевой вершины записана вероятность ее появления и, следовательно, применения соответствующего теста. Таким образом, ожидаемые затраты составят

$$3 + 0.7 \times 2 + 0.4 \times 4 + 0.2 \times 6 + 0.3 \times 6 = 9.$$

Для нахождения оптимального диагностического правила обозначим через $v(R)$ минимум ожидаемых затрат при условии, что неисправен элемент из множества R. Тогда

$$v(R) = \min \{c[j] + p_R(R \cap M_j^0) \times v(R \cap M_j) + \\ + p_R(R \cap M_j^1) \times v(R \cap M_j^1) \mid j \in N\}, \quad (51)$$
$$v(R) = 0 \quad \text{при} \quad |R| = 1,$$

где

$$p_A(B) = \Sigma_{i \in B} p[i] / \Sigma_{i \in A} p[i].$$

В вычислительном отношении удобнее рассматривать функцию $g(R) = p_M(R) \times v(R)$, для которой рекуррентное соотношение (51) выглядит более просто

$$g(R) = \min \{c[j] \times p_M(R) + g(R \cap M_j^0) + g(R \cap M_j^1) \mid j \in N\}. \quad (52)$$

Результаты вычислений по этой рекуррентной формуле для нашего примера представлены в табл. 4, а оптимальная диагностическая процедура изображена на рис. 68.

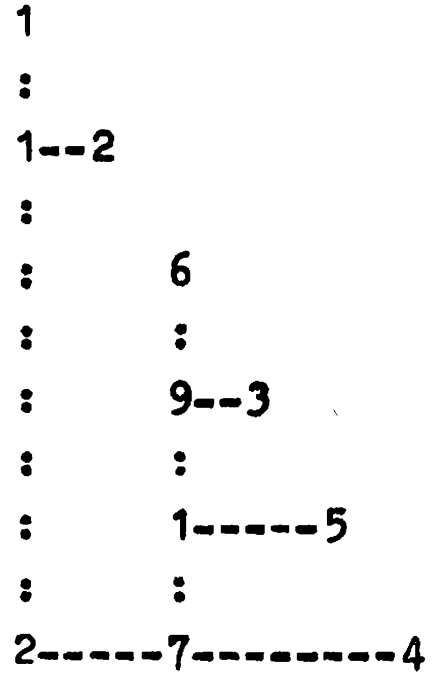

Рис. 68. Оптимальное дерево проверок (подлинная выдача из вычислительной машины).

Программная реализация этого алгоритма возможна только при достаточно малых $m = |M|$ (порядка 10—15). В этих условиях возможно удобное представление состояний — в виде характеристических векторов соответствующих подмножеств. Эти характеристические векторы в свою очередь могут рассматриваться как двоичные представления целых чисел из $0 : 2^m - 1$. Поэтому в качестве состояний процесса можно взять это множество целых чисел. Тестовые множества M_j^1 также можно представить целыми числами из этого же диапазона. Обозначим j-е число через $test[j]$. Нахождение по множеству R множеств $R \cap M_j^0$ и $R \cap M_j^1$ при таком представлении множеств достаточно просто, хотя и выходит за рамки «чистого» алгола-60:

Таблица 4

R	$p_M(R)$	Тест	Затраты	R	$p_M(R)$	Тест	Затраты
1	.3	—	.0	61	.4	2	.8
2	.2	—	.0	62	.3	2	.6
21	.5	1	1.5	621	.6	8	2.4
3	.1	—	.0	63	.2	9	.6
31	.4	2	.8	631	.5	2	1.6
32	.3	2	.6	632	.4	2	1.4
321	.6	8	2.4	6321	.7	2	3.5
4	.2	—	.0	64	.3	7	1.2
41	.5	2	1.0	641	.6	2	2.4
42	.4	2	.8	642	.5	2	2.2
421	.7	8	2.9	6421	.8	2	4.3
43	.3	9	.9	643	.4	7	2.2
431	.6	2	2.1	6431	.7	2	3.6
432	.5	2	1.9	6432	.6	2	3.4
4321	.8	2	4.0	64 321	.9	2	5.5
5	.1	—	.0	65	.2	1	.6
51	.4	2	.8	651	.5	2	1.6
52	.3	2	.6	652	.4	2	1.4
521	.6	1	2.4	6521	.7	1	3.5
53	.2	1	.6	653	.3	1	1.5
531	.5	2	1.6	6531	.6	2	2.7
532	.4	2	1.4	6532	.5	2	2.5
5321	.7	1	3.5	65 321	.8	1	4.6
54	.3	1	.9	654	.4	7	2.2
541	.6	2	2.1	6541	.7	2	3.6
542	.5	2	1.9	6542	.6	2	3.4
5421	.8	1	4.0	65 421	.9	2	5.5
543	.4	1	2.1	6543	.5	7	3.5
5431	.7	2	3.5	65 431	.8	2	5.1
5432	.6	1	3.3	65 432	.7	1	4.9
54 321	.9	1	5.4	654 321	1.0	2	7.0
6	.1	—	.0				

нужно логически перемножить числа, соответствующие R и N_j^1, и получится число, соответствующее $R \cap N_j^1$ (процедура *and* в описанной ниже процедуре [1])). Для получения числа, соответствующего $R \cap N_j^0$, нужно результат просто вычесть из числа, соответствующего R. Потен-

[1]) Небольшая техническая подробность — для этого требуется еще, чтобы использовался транслятор, в котором целые числа имеют представление с фиксированной запятой.

циалы вычисляются в процедуре в массиве $v[1:2^m-1]$, который первоначально используется для хранения вероятностей $p_M(R)$. Результаты записываются в *cost* (минимум целевой функции, т. е. $v[2^m-1]$), и в двух целочисленных массивах *dec*, $q[1:2\times m-1]$, задающих политику диагностики. В ячейке *dec* [1] записывается номер первого (корневого) теста. Если $q[i]$ равно нулю, то *dec* $[i]$ — номер состояния, в противном случае — номер проверки. В том случае, если в *dec* $[i]$ записан номер проверки, то при результате проверки, равном 1, следующее состояние становится равным $i+1$, а при результате 0 следующее состояние $q[i]$. Так, для дерева на рис. 68 значения компонент массива будут следующими:

i	1	2	3	4	5	6	7	8	9	10	11
dec $[i]$	2	1	2	1	7	4	1	5	9	3	6
$q[i]$	5	4	0	0	7	0	9	0	11	0	0

В приводимой процедуре *test* $[1:m]$ — матрица тестов, *cost* — минимальная стоимость эксперимента, остальные обозначения соответствуют использовавшимся:

```
procedure diagn (m, n, p, test, c, dec, q, cost, gz);
   value m, n, gz; integer m, n; real cost, gz;
   integer array test, q, dec; array p, c;
begin integer i, i1, i2, j, k, m1; real p1, p2, min;
   array v[1:2↑m-1]; integer array r[1:2↑m-1];
   m1:=i2:=1;
  for k:=1 step 1 until 2↑m-1 do
     begin v[k]:=.0; r[k]:=0 end;
   for i:=0 step 1 until m-1 do
     begin r[m1]:=i-m; m1:=m1+m1 end;
   for k:=1 step 1 until m do
     begin p1:=p[k]; i1:=i2; i2:=i2+i2;
       for j:=i2 step i2 until m1 do
        for i:=j-i1 step 1 until j-1 do
            if r[i]≥0 then v[i]:=v[i]+p1
     end k;
   for k:=1 step 1 until m1-1 do if r[k]=0 then
        begin min:=gz; p1:=v[k]; i1:=n+1;
           for i:=1 step 1 until n do
             begin i2:=and (k, test[i]);
                if i2>0∧i2<k then
                  begin p2:=c[i]×p1+v[i2]+v[k-i2];
```

```
                    if p2 < min then
                      begin min := p2; i1 := i end
              end end i;
            v[k] := min; r[k] := i1
      end k;
    i1 := 1; k := m1 - 1;
    for i := 1 step 1 until 2×m - 1 do dec[i] := q[i] := 0;
mk: j := r[k];
    if j > 0 then begin dec[i1] := j; i2 := and (k, test[j]);
      q[i1] := i2 - k; k := i2; i1 := i1 + 1; go to mk end;
    dec[i1] := m + r[k] + 1; q[i1] := 0;
    if i1 ≤ m + m - 1 then begin
      for i := i1 step - 1 until 1 do
        if q[i] < 0 then
          begin k := - q[i]; q[i] := i1 := i1 + 1;
                                        go to mk end
      end;
    cost := v] n + m - 1]
end diagn.
```

§ 9. Плоская задача раскроя

В задаче раскроя, которую мы рассматривали в конце гл. 2, было несущественно, что представляет собой множество способов раскроя. Надо было лишь, чтобы на этом множестве было удобно решать вспомогательную экстремальную задачу, в которой выбирается наивыгоднейший раскрой.

Такой подход позволяет решить многие другие задачи раскроя.

Как практически важное обобщение задачи линейного раскроя часто рассматривается задача раскроя прямоугольного листа на прямоугольные заготовки сквозными резами, параллельными краям. Это условие объясняется технологией, при которой для раскроя используются так называемые гильотинные ножницы. Нам это очень удобно, так как позволяет обойтись рассмотрением прямоугольных листов и свести задачу к процессу динамического программирования, в котором состояние описывается парой целых чисел — длиной и шириной листа в выбранной единице измерения.

Задача выбора наивыгоднейшего гильотинного раскроя и будет нас сейчас интересовать.

Задача гильотинного раскроя.

Пусть заданы пара положительных целых чисел $(l0, w0)$ — длина и ширина исходного прямоугольного листа и еще n троек чисел $(l[i], w[i], c[i])$, где $l[i]$, $w[i]$ — положительные целые числа, задающие длину и ширину заготовок, а $c[i]$ — стоимость такой заготовки, положительное вещественное число. Требуется выбрать политику раскроя, т. е. для всех размеров листа (l, w), где $l \in 1 : l0$, $w \in 1 : w0$, указать способ раскроя — по длине или по ширине — расстояние резов от края так, чтобы сумма стоимостей получившихся из листа заготовок была максимальной.

Прежде всего займемся исследованием рекуррентных соотношений Беллмана. Как обычно, наряду с исходной задачей мы будем решать такую же задачу со всеми меньшими (целочисленными) значениями $l0$ и $w0$.

Обозначив максимум этой суммы для листа (l, w) через $v[l, w]$, мы получим для потенциалов v рекуррентное соотношение

$$v[l, w] = \max\{\max_{\lambda \in 0:l}(v[l-\lambda, w]+v[\lambda, w]), \max_{\omega \in 0:w}(v[l, \omega]+v[l, w-\omega])\}. \quad (53)$$

Это рекуррентное соотношение имеет много решений, от чего нас не спасает ни условие неотрицательности v ни условие на границе

$$v[0, w] = v[l, 0] = 0. \quad (54)$$

Условие

$$v[l[i], w[i]] \geqslant c[i] \quad (55)$$

также не устраняет лишних решений (это видно, например, из того, что любое решение при умножении на множитель, больший 1, остается решением). Оказывается, для обеспечения единственности нужно потребовать *минимальность решения.*

Теорема 7. *Пусть* v_1 *и* v_2 *— два решения рекуррентных соотношений* (53) — (55). *Существует решение* $\tilde{v}$, *для которого*

$$\tilde{v} \leqslant v_1, \quad \tilde{v} \leqslant v_2. \quad (56)$$

Существует решение тех же рекуррентных соотношений, которое не больше любого другого решения, и по этому решению может быть построен оптимальный гильотинный раскрой.

Доказательство. Пусть $v_3[l, w] = \min\{v_1[l, w], v_2[l, w]\}$. Поскольку из (53) следуют неравенства

$$\begin{aligned} v[l, w] &\geqslant v[l-\lambda, w]+v[\lambda, w], \quad \lambda \in 0:l, \\ v[l, w] &\geqslant v[l, w]+v[l, w-\omega], \quad \omega \in 0:w, \end{aligned} \tag{57}$$

то из выполнения этого неравенства для v_1 и v_3 следует его выполнение для v_3. Неравенство (57) является частным случаем неравенства (31). Выбрав v_3 в качестве начального приближения, мы можем построить монотонно невозрастающую ограниченную снизу последовательность, которая сходится к функции, являющейся решением (53) и удовлетворяющей (56). (Такая структурная операция для решений уравнения Беллмана очень характерна. Мы вернемся к этому вопросу в следующей главе.)

Рассмотрим множество F функций, удовлетворяющих условиям (54) — (57). Так как это выпуклое множество, которое содержит, в частности, функции, удовлетворяющие (53) — (55), в нем для любой пары (l, w) найдутся функции с минимальным значением $v[l, w]$. Взяв функцию v_1, принимающую наименьшее возможное значение в точке (l_1, w_1), и функцию v_2, принимающую наименьшее возможное значение в точке (l_2, w_2), построим по ним функцию v_{12}, для которой $v_{12}[l_1, w_1] = v_1[l_1, w_1]$ и $v_{12}[l_2, w_2] = v_2[l_2, w_2]$. Повторяя этот процесс нужное число раз, получим функцию из F, минимальную одновременно по всем компонентам. Эта функция, очевидно, является решением (53). В противном случае по ней можно было бы, как выше, построить решение (53), которое еще меньше.

Покажем теперь, что это решение $\hat{v}$ для каждой пары (l, w) дает максимум стоимости деталей, которые можно выкроить из листа (l, w). Действительно, для каждой пары (l, w) либо найдется $i \in N$ такое, что

$$l = l[i], \quad w = w[i], \quad \hat{v}[l, w] = c[i], \tag{58}$$

либо максимум в (53) достигается при $\lambda \in 1 : l-1$ или при $\omega \in 1 : w-1$. Если бы это было не так, то $\hat{v}[l, w]$ можно было бы уменьшить и получившаяся функция по-прежнему удовлетворяла бы (55) и (57). Используя ее

в качестве начального приближения для монотонно невозрастающей последовательности, мы получили бы, переходя к пределу, функцию $\bar{v}$, удовлетворяющую (53)—(55).

Рассмотрит теперь произвольный гильотинный раскрой листа ($l\ 0$, $w\ 0$). В нем участвует некоторое множество листов с размерами (l_α, w_α), причем для каждого α задан либо номер детали $i(\alpha)$, которая получается из этого листа, либо номера α_1 и α_2 получающихся листов. В первом случае воспользуемся неравенством

$$\hat{v}[l_\alpha, w_\alpha] \geqslant c[i(\alpha)],$$

во втором — неравенством

$$\hat{v}[l_\alpha, w_\alpha] \geqslant \hat{v}[l_{\alpha_1}, w_{\alpha_1}] + \hat{v}[l_{\alpha_2}, w_{\alpha_2}]. \qquad (59)$$

Если эти неравенства просуммировать по всем промежуточным листам, в левой части у нас останется $\hat{v}[l\ 0, w\ 0]$, а в правой части — сумма стоимостей получившихся деталей. С другой стороны, мы можем построить гильотинный раскрой, сопоставляя каждому из встречающихся размеров либо разрез, на котором достигается максимум в (53) для функции $\hat{v}$, либо номер детали из (58). Для этого раскроя в (59) всюду достигается знак равенства, и, следовательно, значение $v[l\ 0, w\ 0]$ р а в н о сумме стоимостей деталей. ▲

Отметим, что единственности решения можно было сразу достичь, модифицируя уравнение (53) и полагая

$$v[l, w] = \max\{\max_\lambda, \max_\omega, \max\{c[i] \mid l[i] \leqslant l\, w[i] \leqslant w\}, 0\}. \qquad (60)$$

Однако мы хотели показать еще одно использование рассуждений, основанных на неравенстве (31).

В вычислениях, разумеется, следует брать за основу уравнение (60), решение которого можно, как обычно, находить последовательно, начиная от $v[0, 0] = 0$. Схема вычислений, в которой участвуют все пары из $1 : l\ 0$ и $1 : w\ 0$, особого интереса не представляет. Она просто реализуется, но очень трудна в практически интересных задачах, где $l\ 0$ и $w\ 0$ обычно имеют значения порядка сотен.

Между тем легко видеть, что в практически интересных случаях функция $v[l, w]$ обладает хорошими свойствами,

в частности, она монотонна по каждой из переменных и кусочно постоянна.

Например, при $m = 2$, $l0 = 10$, $w0 = 5$, $l[1:2] = (2, 3)$, $w[1:2] = (7, 3)$, $c[1:2] = (3, 2)$ эта функция будет иметь график, изображенный на рис. 69.

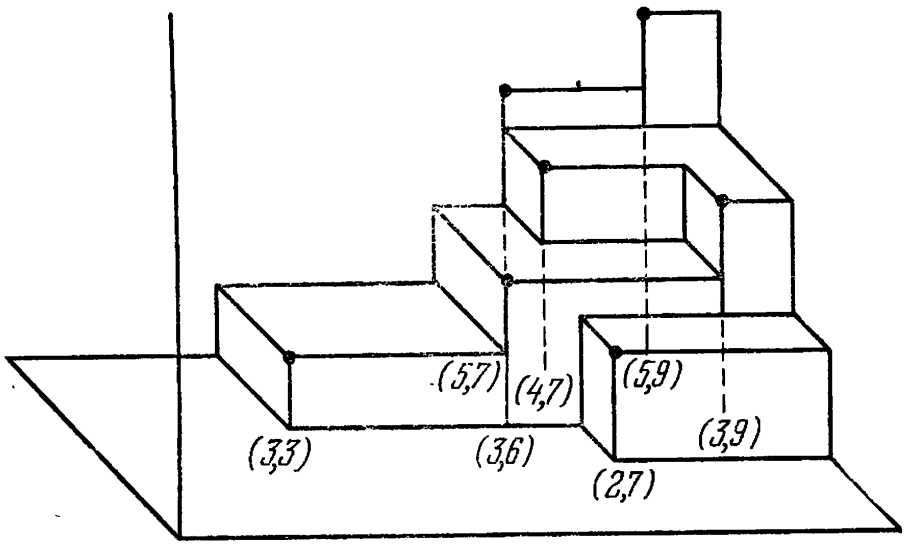

Рис. 69. График функции Беллмана.

Таким образом, чтобы задать функцию v, достаточно задать список основных точек — троек $(l_\alpha, w_\alpha, c_\alpha)$, соответствующих скачкам функции (в нашем примере их восемь, включая (0, 0, 0)).

Мы опишем процедуру, в которой реализуется схема переработки списков, в общих чертах. Список будет строиться «слоями» — для каждой интересующей нас ширины отдельно.

Интересующие нас ширины будут также выписаны в виде списка, в который вначале будут включены ширины исходных деталей, а затем этот список будет «замкнут» — в него будут включены все комбинации исходных ширин с натуральными коэффициентами.

Для каждой ширины ее «слой» будет формироваться в два этапа — сначала будет задана исходная информация и начальные детали слоя, а затем, почти как в задаче линейного раскроя, из начальных деталей будут сформированы их комбинации. Исходная информация состоит из трех составляющих — детали данной ширины, список точек предыдущей ширины (который к этому моменту уже должен быть сформирован) и пары списков, соответству-

ющие парам ширин, в сумме дающим рассматриваемую. На рис. 70 показано, какой именно список мы хотим делать из пары списков.

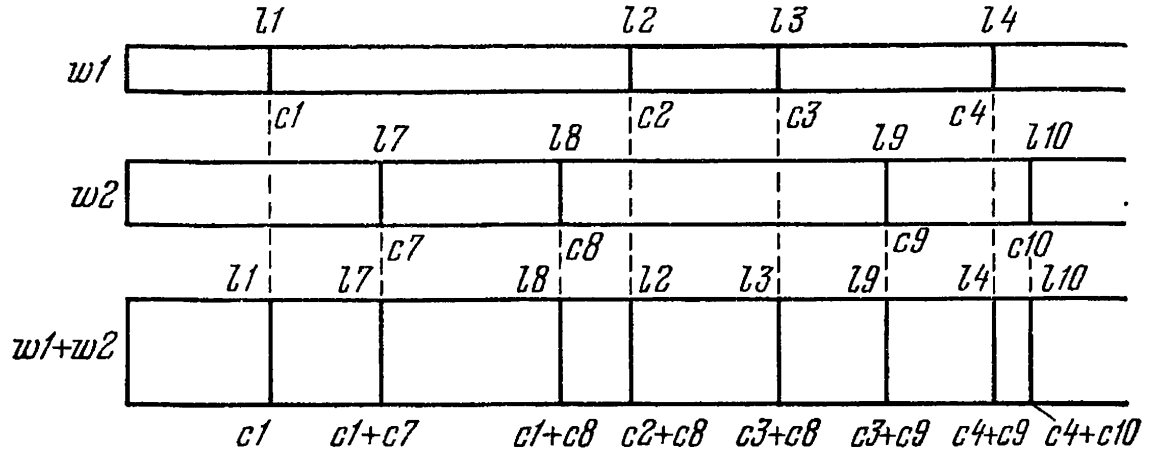

Рис. 70. Схема соединения списков.

Рассмотрим детали алгоритма на небольшом примере.

Пример. $m = 3$, $l0 = 205$, $w0 = 100$.

Прежде всего образуем список основных ширин (табл. 5). Вначале в него входит три числа 31, 41, 51. После замыкания списка получаем 31, 41, 51, 62, 72, 82, 92, 93.

Таблица 5

i	1	2	3
$l[i]$	71	31	81
$w[i]$	31	41	51
$c[i]$	8	5	16

Для ширины 31 мы имеем одну деталь с длиной 71 и ценой 8. Предыдущего списка здесь нет, пар ширин, в сумме дающих 31, тоже нет.

Замыкание добавляет в этом списке удвоение первой детали. Запишем окончательный список так:

31 : (71, 8), (142, 16).

Для ширины 41 получаем исходный список 41 : (31, 5). После добавления предыдущего списка 41 : (31, 5), (71, 8), (142, 16). После замыкания получим: 41 : (31, 5), (62, 10), (93, 15), (124, 20), (155, 25), (186, 30) (предыдущий список весь доминируется).

Для ширины 51 выпишем прямо окончательный список 51 : (31, 5), (62, 10), (81, 16), (112, 21), (143, 26), (162, 32), (193, 87).

Для ширины 62 в исходном списке, кроме списка 51, будет «удвоенный» список 31: 62 : (71, 16), (142, 32).

Мы получим начальный список: 62 : (31, 5), (62, 10), (71, 16), (112, 21), (142, 32), (193, 37) и окончательный 62 : (31, 5), (62, 10), (71, 16), (102, 21), (133, 26), (142, 32), (173, 37), (204, 42).

Список 72 строится аналогично по 62 и 31 + 41. Последнее сочетание дает: 72 : (31, 5), (62, 10), (71, 18), (93, 23), (124, 28), (142, 36), (155, 41), (186, 46). В результате получаем тот же список.

Теперь список 82 должен в качестве исходной информации брать списки 72, 41 + 41 и 31 + 51, список 92 — списки 82 и 41 + 51, список 93 — списки 92 и 31 + 62 и т. д.

Окончательно получаем:
82 : (31, 10), (62, 20), (81, 24), (93, 30), (112, 34), (124, 40), (143, 44), (155, 50), (174, 54), (186, 60), (205, 64),
92 : (31, 10), (62, 20), (81, 26), (93, 31), (112, 37), (124, 41), (143, 46), (155, 51), (162, 57), (186, 62), (193, 67),
93 : (31, 10), (62, 20), (71, 24), (81, 26), (93, 31), (112, 37), (124, 41), (133, 44), (142, 48), (152, 50), (155, 51), (162, 57), (173, 58), (183, 61), (186, 62), (193, 67), (204, 68).

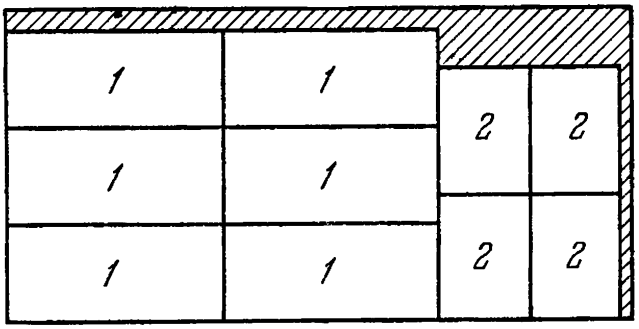

Рис. 71. Оптимальный раскрой.

Наилучший раскрой, получающийся в этом примере, изображен на рис. 71.

§ 10. Связь динамического программирования и улучшенного перебора

В этом параграфе мы хотим рассмотреть основную идею динамического программирования — идею рекурсии и вложения исходной экстремальной задачи в класс родственных задач с другой точки зрения — с точки зрения улучшенного перебора.

Пусть мы решаем (как в § 3 гл. 5) некоторую дискретную экстремальную задачу.

| Найти $\min, \{f(x) \mid x \in A\}$.

Пусть на каком-то шаге процесса мы получили разбиение $\mathfrak{A}$ множества допустимых решений A:

$$\mathfrak{A}: A = A_0 \cup \bigcup_{\alpha \in \Omega} A_\alpha.$$

Назовем α-задачей задачу:

| Найти $\min \{f(x) \mid x \in A_\alpha\}$.

Предложение 2. *Если α-задача является расширением β-задачи, т. е. если существует отображение $T: A_\beta \to A_\alpha$ и*

$$f(x) \geqslant f(Tx) \quad \text{для} \quad x \in A_\beta, \tag{61}$$

то существует оптимальное решение исходной задачи в множестве $A \setminus A_\beta$.

Доказательство следует из того, что от расширения задачи минимум разве лишь уменьшается. ▲

Мы уже пользовались вычислительными возможностями, которые нам представляет предложение 2. Так, например, в задаче линейного раскроя рассматриваются «частичные раскрои» — наборы деталей, использующие какую-то часть сырья и дающие какой-то доход. В качестве множеств допустимых решений можно рассматривать раскрои, продолжающие данный частичный раскрой.

Частичный раскрой, в котором использована меньшая, чем у данного, длина сырья и получен большой доход, предоставляет больше возможностей для продолжения, поэтому откидывание доминируемых состояний в решении задачи линейного раскроя и является использованием предложения 2.

Ситуация, с которой мы здесь встречаемся, весьма типична. Во многих задачах все элементы разбиения множества допустимых решений могут быть охарактеризованы сравнительно небольшой информацией, которую мы и называем состоянием процесса. Фактически, пользуясь рекуррентными соотношениями или уравнениями динамического программирования, мы добиваемся того, чтобы в каждый момент вычислительного процесса было не больше одного состояния процесса.

В случае, если состояний процесса не очень много, мы можем вести вычисления для всех состояний, если их слишком много, можно попытаться ввести доминирование на множестве состояний и удалить лишнее.

Однако в действительно сложных задачах успех может быть достигнут лишь при сочетании различных идей и подходов. Например, в задаче о поиске неисправности, где метод рекуррентных соотношений практически без-

надежен при n порядка 20, оказывается целесообразным сочетать его с методом ветвей и границ для того, чтобы выбрать начальные разбиения множества неисправностей на достаточно малые подмножества.

§ 11. Схема исключения Бертеле — Бриоши

Метод рекуррентных соотношений Беллмана допускает еще одну трактовку, которая позволяет значительно расширить число задач, решаемых с его помощью.

Рассмотрим следующую задачу.

Пусть $T = 0 : t$ и для каждого $i \in T$ задано конечное множество X_i. Пусть на произведении $X_{i-1} \times X_i$, $i \in 1 : t$, задана функция c_i. Требуется каждому $i \in T$ сопоставить $x[i] \in X_i$ так, чтобы максимизировать

$$\sum_{i \in 1:t} c_i (x[i-1],\ x[i]). \tag{62}$$

Эта задача легко сводится к схеме Беллмана, но не уступает ей в общности. Будем решать ее знакомым методом рекуррентных соотношений: обозначим через $v_k(y)$ максимум

$$\sum_{i \in 1:k} c_i (x[i-1], x[i])$$

при условии $x[k] = y$, мы получим рекуррентное соотношение

$$\begin{aligned} v_{k+1}(y) &= \max \{c_{k+1}(x,\ y) + v_k(x) \mid x \in X_k\}, \\ v_0(x) &\equiv 0, \end{aligned} \tag{63}$$

причем нас должен интересовать $\max \{v_t(y) \mid y \in X_t\}$. Можно тот же подход трактовать иначе, как способ *последовательного исключения переменных*. Рассмотрим, например, в целевой функции (62) переменную $x(t)$. Она входит только в слагаемое $c_t(x[t-1], x[t])$. Найдем при каждом $x[t-1]$ максимум по $x[t]$. Мы получим некоторую новую функцию

$$\gamma_{t-1}(x) = \max \{c_t(x,\ y) \mid y \in X_t\}, \tag{64}$$

которую теперь можно добавить к функции $c_{t-1}(x, y)$. В получившейся целевой функции можно также исключить $x[t-1]$ и т. д.

Теперь мы можем перейти к варианту этой задачи, который уже не удается привести к классической схеме, рассмотренной выше.

Задача максимизации на дереве.

Пусть $\langle M, N\rangle$ — дерево, каждой вершине которого i сопоставлено конечное множество X_i, а каждой дуге u — функция c_u на произведении $X_{i(u)} \times X_{j(u)}$. Требуется выбрать набор $x[M]$, $x[i] \in X_i$, максимизирующий

$$\sum_{u \in N} c_u (x[i(u)],\ x[j(u)]).$$

В этой задаче мы можем применить ту же технику последовательного исключения переменных.

Поскольку в дереве всегда найдется «висячая» вершина, т. е. вершина, инцидентная лишь одной дуге, здесь всегда возможна предложенная выше редукция задачи. При исключении такой вершины и инцидентной ей дуги мы снова получим дерево и т. д.

Сходство между этим процессом и процессом рекуррентного счета такое же, как между цепью и деревом, — последовательно исключая переменные, мы решаем всю задачу до конца.

Если $\langle M, N\rangle$ состоит из одного цикла, исключение может быть проведено чуть иначе — каждая вершина i будет инцидентна двум дугам и, следовательно, $x[i]$ будет входить только в два слагаемых целевой функции. Мы получим, например (обозначения зависят от ориентации, которая здесь несущественна),

$$c_{u_1} (x[i(u_1)],\ x[i]) + c_{u_2} (x[i],\ x[j(u_1)]).$$

Для каждой пары значений $x[i(u_1)]$, $x[j(u_2)]$ можно выбрать то значение $x[i]$, на котором достигается максимум этой суммы. Таким образом, сумма будет заменена некоторой функцией $c_{u_1 u_2} (x[i(u_1)],\ x[j(u_2)])$, а граф преобразуется в цикл с меньшим числом дуг. Те же рассуждения проходят для графа, состоящего из дерева и связанных им циклов. Если же два цикла имеют пересечение, то при исключении вершины, на которой эти циклы расходятся, мы получим функцию, зависящую от трех переменных, которую в нашей графической интерпретации задачи следовало бы изобразить треугольником.

Таким образом, произвольный граф является не вполне удачным объектом для развития метода исключений и должен быть заменен произвольным комплексом остовов (или гиперграфом).

Итак, рассматривается следующая задача.

Задача Бертеле — Бриоши.

Имеются два конечных множества M и N. Каждому $u \in N$ сопоставлено непустое множество $M_u \subset M$. Каждому $i \in M$ сопоставлено конечное и непустое множество X_i. Задан набор функций $c_u(x[M_u])$. Требуется найти $x[M]$, на котором достигается максимум

$$\sum_{u \in N} c_u(x[M_u]).$$

При решении этой задачи методом исключения переменных производится следующая редукция комплекса: выбирается какая-либо из вершин $i \in M$ и звезда этой вершины (так называется совокупность множеств M_u, содержащих i) заменяется объединением этих множеств, за исключением самого i. Соответствующая этому новому множеству M^* функция c^* определяется по формуле

$$c^*(x[M^*]) = \max\{\sum_{u,\, M_u \ni i} c(x[M_u]) \mid x[i] \in X_i\}. \quad (65)$$

При такой редукции мы снова получаем комплекс остовов, и вид задачи сохраняется.

Трудоемкость решения задачи Бертеле — Бриоши зависит от того, насколько сложно производить вычисления по формуле (65). Например, можно считать, что трудоемкость определяется размером наибольшей из встречающихся таблиц, задающих функции c.

Так мы приходим к задаче о выборе последовательности редукций, при которой задача Бертеле — Бриоши оказывается наименее трудоемкой. Авторы назвали эту задачу «вторичной задачей оптимизации».

Вторичная задача оптимизации.

Пусть задан комплекс остовов $K_0 = \langle M, N, \{M_u\}\rangle$, и каждому $i \in M$ сопоставлено положительное число $r[i]$ ($r[i] = \ln(|X_i|)$). Назовем *характеристикой комплекса* величину

$$s(K) = \max\{1[M_u] \times r[M_u] \mid u \in N\}. \quad (66)$$

Требуется найти такую последовательность комплексов

$$K_0, K_1, \ldots, K_m, \quad (67)$$

в которой каждый следующий получался бы из предыдущего описанной операцией исключения вершины и

достигался минимум максимальной из встречающихся в последовательности характеристик.

Оказывается возможным применить при решении этой задачи динамическое программирование в его обычном виде. Эта возможность базируется на следующем факте.

Предложение 3. *Комплекс K_j, получающийся на j-м шаге в последовательности* (66), *зависит от того, какие вершины были исключены на первых j шагах, и не зависит от последовательности их исключения.*

Мы опустим доказательство этого предложения, которое достаточно просто. Из этого предложения следует, что для задания комплекса мы можем ограничиться множеством его вершин.

Обозначим минимум при решении вторичной задачи оптимизации для комплекса $R \subset M$ через $v(R)$. Тогда, очевидно,

$$v(R) = \max \{s(R), \min \{v(R - i) \mid i \in R\}\}. \qquad (68)$$

Это уравнение представляет собой обычное уравнение динамического программирования, которое уже встречалось нам неоднократно.

ГЛАВА 7

МАРКОВСКИЕ ПРОЦЕССЫ РЕШЕНИЯ

§ 1. Некоторые сведения о марковских цепях

Эта глава несколько отличается по своей проблематике от предыдущих. Но неявно в них участвовали понятия и конструкции, которые становятся яснее и естественнее, если перейти к рассмотрению хотя бы простейших вероятностных задач управления.

Такие простейшие процессы — марковские цепи, в которых выбором управляющих параметров можно влиять на законы распределения, — и будут рассмотрены в этой главе.

Мы начнем с некоторых простых фактов, относящихся к обычным марковским цепям.

Будем говорить, что задана (конечная однородная) *марковская цепь*, если заданы множество M состояний этой цепи, вектор $p_0[M]$ начальных вероятностей (распределение вероятностей на M в начальный момент времени), матрица переходных (условных) вероятностей $p[M, M]$. При этом

$$\begin{array}{ll} p_0[M] \geqslant 0[M], & \mathbf{1}[M] \times p_0[M] = 1, \\ p[M, M] \geqslant 0[M, M], & p[M, M] \times \mathbf{1}[M] = \mathbf{1}[M] \end{array} \tag{1}$$

(как мы уже говорили, вектор $p_0[M]$, удовлетворяющий этим условиям, называется *вероятностным вектором*, матрица $p[M, M]$ — *стохастической матрицей*. Отсюда и идет термин «бистохастическая матрица», встретившийся нам в гл. 4). Свяжем с матрицей $p[M, M]$ матрицу $r[M, M]$:

$$r[i, j] = \delta(p[i, j] > 0). \tag{2}$$

Эта матрица может рассматриваться как матрица смежностей некоторого графа $\langle M, N \rangle$, называемого *графом переходов* марковской цепи. Этот граф переходов задает систему предшествований на M (в смысле § 1 гл. 4), в соответствии с которой M разбивается на классы эквивалент-

ности. Рассмотрим граф $\langle \tilde{M}, \tilde{N} \rangle$, являющийся редукцией графа $\langle M, N \rangle$. В этом графе нет контуров и, следовательно, найдутся «концевые» вершины — вершины, из которых не выходят дуги.

Множество $M' \subset M$ называется *эргодическим классом* марковской цепи, если

$$p\,[M', M'] \times 1\,[M'] = 1\,[M'] \qquad (3)$$

и никакое собственное подмножество M' этим свойством не обладает.

Предложение 1. *Классы эквивалентности, соответствующие концевым вершинам графа* $\langle \tilde{M}, \tilde{N} \rangle$, *и только они образуют эргодические классы марковской цепи.*

Доказательство. Если M' — класс эквивалентности, соответствующий концевой вершине, то для него выполнение (3) очевидно. Так как в классе эквивалентности существует путь из любой его вершины в любую, это означает, что вероятность перехода из любого состояния в любое (за несколько шагов) положительна, и подмножество, обладающее свойством (3), из M' не выделить.

Если M' — эргодический класс, то $p\,[M', M \setminus M'] = 0\,[M', M \setminus M']$. Следовательно, в $M \setminus M'$ нет элементов, которым предшествовали бы какие-либо элементы из M'. Значит, M' состоит только из вершин, входящих в «концевые» классы эквивалентности, причем каждый такой класс может войти в M' только целиком. Но по уже доказанному каждый такой класс эквивалентности сам представляет собой эргодический класс и, следовательно, совпадает с M'. ▲

Состояния, не входящие в эргодические классы, мы будем называть *переходными* [1]). Классы эквивалентности переходных состояний частично упорядочены, их можно разбить на уровни, приписав нулевой уровень эргодическим классам и $(k+1)$-й уровень классу эквивалентности, из которого можно попасть в класс k-го и меньшего уровней. Таким образом, мы задаем на $\tilde{M}$ функцию $lev\,[\tilde{M}]$, удовлетворяющую соотношению (это тоже рекуррентное

---

[1]) В литературе по марковским цепям, к сожалению, распространен термин «несущественные состояния». Этот термин мало удобен в тех многочисленных задачах, где только эти состояния и существенны.

соотношение динамического программирования)

$$\begin{aligned} lev[i] &= 0 \quad \text{при} \quad \tilde{N}_{\bar{i}} = \emptyset, \\ lev[i] &= 1 + \max\{lev[j(u)] \mid u \in \tilde{N}_{\bar{i}}\}. \end{aligned} \tag{4}$$

Пример. Пусть $M = 1:8$. В табл. 1 и 2 выписаны матрицы p и r.

Таблица 1

	1	2	3	4	5	6	7	8
1	0.4						0.6	
2					1			
3	0.8							0.2
4	0.7			0.3				
5		1						
6		0.1		0.3	0.3	0.3		
7	0.9						0.1	
8	0.2			0.4			0.4	

Таблица 2

	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0	0
3	1	0	0	0	0	0	0	1
4	1	0	0	1	0	0	0	0
5	0	1	0	0	0	0	0	0
6	0	1	0	1	1	1	0	0
7	1	0	0	0	0	0	1	0
8	1	0	0	1	0	0	1	0

Граф переходов и редукция этого графа представлены на рис. 72 и 73. Около классов эквивалентности (вершин графа на рис. 73) написаны их уровни.

Легко видеть, что любая траектория марковской цепи переходит из состояния в состояние с понижением своего уровня и с вероятностью единица за конечное число шагов доходит до какого-либо эргодического класса. Поэтому все асимптотические рассмотрения для марковских цепей

проводятся именно для эргодических классов. Мы введем сейчас понятия, необходимые для будущего.

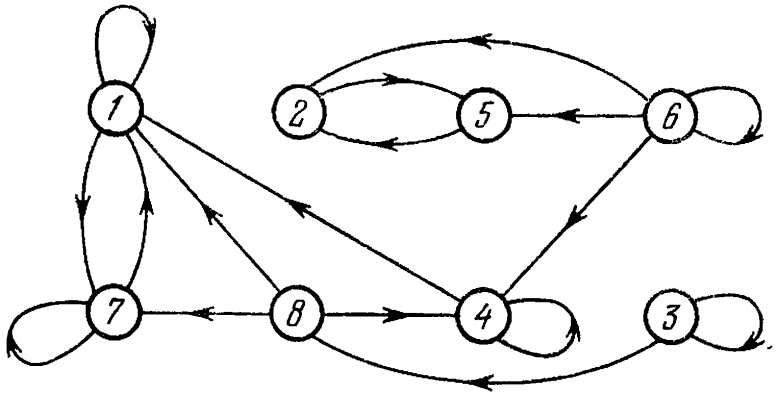

Рис 72. Граф возможных переходов цепи.

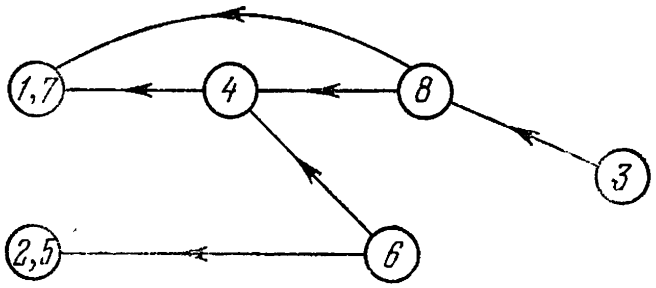

Рис. 73. Граф классов эквивалентности.

Для каждого эргодического класса M_α можно рассмотреть систему линейных уравнений

$$\begin{gathered}\pi[M_\alpha]\times p[M_\alpha,\ M_\alpha]=\pi[M_\alpha],\\ \pi[M_\alpha]\times 1[M_\alpha]=1.\end{gathered}\tag{5}$$

Предложение 2. *Система* (5) *имеет единственное решение, и это решение положительно.*

Доказательство. Известны многочисленные доказательства этого утверждения, как правило основывающиеся на теореме Перрона о характеристических корнях неотрицательных матриц. Мы здесь используем методы, более близкие по духу всему остальному изложению.

В (5) ограничений на одно больше, чем переменных. Однако, как это видно из (3), между уравнениями первой группы имеется линейная зависимость, поэтому система (5) всегда разрешима. Более того, так как преобразование $T:\pi\to\pi\times p$ непрерывно отображает симплекс вероятностных векторов в себя, то по теореме Брауэра существует непрерывная точка этого преобразования π, которая и является неотрицательным решением (5).

Вектор $\bar{\pi}[M_\alpha]$ должен быть строго положительным. Действительно, если найдется непустое множество $M' \subset \subset M_\alpha$, для которого $\bar{\pi}[M'] = 0[M']$, то из (5) следует, что $p[M_\alpha \setminus M', M'] = 0[M_\alpha \setminus M', M']$, и значит, $M_\alpha \setminus M'$, вопреки предположениям, образует эргодический класс.

Осталось доказать единственность решения. Пусть π_0 и π_1 — два различных положительных решения. Обозначим через $\pi_\lambda[M_\alpha]$, где λ — *любое* вещественное число, вектор $\lambda \times \pi_1[M_\alpha] + (1 - \lambda) \times \pi_0[M_\alpha]$, который, очевидно, также является решением системы (5). Выбором соответствующего λ можно добиться того, чтобы вектор $\pi_\alpha[M_\alpha]$ был неотрицателен и имел хотя бы одну нулевую компоненту. Между тем, как мы уже показали, это невозможно. Следовательно, неверно предположение о существовании двух различных решений (5).▲

Вектор $\pi[M_\alpha]$, являющийся решением системы (5), называется *вектором стационарных вероятностей* или *стационарным распределением* эргодического класса M_α. (Если начальное распределение цепи p_0 совпадает с π, то распределение на каждом следующем шаге также будет совпадать с π.) Любое стационарное распределение исходной цепи должно быть сосредоточено только на состояниях из эргодических классов и, как легко убедиться, является выпуклой комбинацией стационарных распределений для эргодических классов цепи.

Соответствующая (5) неоднородная система

$$y[M_\alpha] \times p[M_\alpha, M\alpha] = b[M_\alpha] + y[M_\alpha] \qquad (6)$$

имеет (при условии $\mathbf{1}[M_\alpha] \times b[M_\alpha] = 0$) однопараметрическое семейство решений

$$y[M_\alpha] = \rho[M_\alpha] + \gamma \times \pi[M_\alpha]. \qquad (7)$$

В качестве $\rho[M_\alpha]$ выберем для определенности неотрицательное решение, обладающее свойством $\min\{\rho[i] \mid i \in \in M_\alpha\} = 0$. Если на эргодическом классе M_α задать начальное распределение $\pi_0[M_\alpha]$, решению ρ, соответствующему правой части $b[M_\alpha] = \pi_0[M_\alpha] - \pi_0[M_\alpha]$, можно придать физический смысл потока вероятностей, переводящего распределение π_0 в распределение π.

В приведенном примере два эргодических класса $M_1 = \{1, 7\}$ и $M_2 = \{2, 5\}$. Им соответствуют векторы $\pi[M_1] = (0.6, 0.4)$, $\pi[M_2] = (0.5, 0.5)$ и векторы $y[M_1] =$

$= (\alpha, {}^{2}/_{3}\alpha + {}^{10}/_{9}b_1)$, $y[M_2] = (\beta, \beta + 2 \times b_2)$. Таким образом, при $\pi_0[1] \geqslant 0.6$ имеем $\rho[M_1] = (0, (0.6 - \pi_0[1]) \times {}^{10}/_{9}))$, при $\pi_0[1] \geqslant 0.6$ будет $\rho[M_1] = ((\pi_0[1] - 0.6) \times {}^{5}/_{3}, 0)$, два аналогичных варианта будет и у вектора $\rho[M_2]$.

Наряду с вектором стационарных вероятностей иногда целесообразно рассматривать матрицу стационарных безусловных вероятностей перехода $\pi[M_\alpha, M_\alpha]$

$$\pi[i, j] = \pi[i] \times p[i, j].$$

Для этой матрицы

$$1[M_\alpha] \times \pi[M_\alpha, M_\alpha] = \pi[M_\alpha, M_\alpha] \times 1[M_\alpha], \qquad (8)$$

т. е. сумма элементов каждой строки равна сумме элементов соответствующего столбца. Матрицы, обладающие этим свойством, встречаются в задачах оптимизации, связанных с марковскими цепями. Условие (8) фактически является давно знакомым нам условием замкнутости потока — поток (вероятность), входящий в каждую вершину, равен потоку, выходящему из этой вершины. Полезно вспомнить по этому поводу задачу об оптимальном замкнутом потоке, которая встречалась нам в § 9 гл. 3, а затем использовалась в § 5 гл. 6.

Когда мы будем заниматься асимптотикой рекуррентных соотношений в марковских процессах решения, нам понадобится аналог задачи об оптимальном замкнутом потоке, и в этой задаче существенную роль будет играть условие типа (8).

Т е о р е м а 1. *Если общий наибольший делитель длин контуров в графе переходов* $\langle M_\alpha, M_\alpha \rangle$ *эргодического класса* M_α *равен* 1, *то при любом начальном распределении на этом классе* $p_0[M_\alpha]$ *распределение* $p_n[M_\alpha]$ *на* n*-м шаге* (*где* $p_n[M_\alpha] = p_{n-1}[M_\alpha] \times p[M_\alpha, M_\alpha]$) *стремится к* $\pi[M_\alpha]$ *при* $n \to \infty$.

Д о к а з а т е л ь с т в о этой теоремы такое же, как в известной теореме Маркова. Нужно только убедиться в том, что найдется такое k, что вероятность перейти из i в j за k шагов положительна при любых $i, j \in M_\alpha$. Очевидно, что по определению эргодического класса граф $\langle M_\alpha, N_\alpha \rangle$ вполне связен. Поэтому к нему применимо предложение 7 гл. 3, которое и гарантирует нам существование такого k.

Далее, полагая

$$\gamma = \min \{p^k [i, j] \mid i, j \in M_\alpha\} > 0,$$

мы получаем

$$\max \{| p_n [i] - \pi [i] | \mid i \in M_\alpha\} =$$
$$= \max \{| (p_{n-k} [M_\alpha] - \pi [M_\alpha]) \times p^k [M_\alpha, i] | \mid i \in M_\alpha\} \leqslant$$
$$\leqslant (1 - \gamma) \times \max \{| p_{n-k} [i] - \pi [i] | \mid i \in M_\alpha\},$$

откуда уже без труда следует утверждение теоремы. ▲

Теорема 2. *Пусть M_α — эргодический класс и i_0, i_1, ... , i_T — траектория марковского процесса, принадлежащая M_α. Обозначим через $k [i, j]$ число моментов времени, для которых $i_t = i$, $i_{t+1} = j$, а через $\nu [M_\alpha, M_\alpha]$ матрицу, составленную из элементов $k [i, j]/T$. Тогда*

$$\mathsf{P} \{\nu [M_\alpha, M_\alpha] \to \pi [M_\alpha, M_\alpha] \text{ при } T \to \infty\} = 1.$$

Эту теорему (которая называется эргодической теоремой Бирхгофа — Хинчина и имеется в стандартных курсах по теории вероятностей) мы доказывать не будем.

Естественно, что любая линейная комбинация частот ν с вероятностью 1 сходится к соответствующей линейной комбинации стационарных вероятностей. В частности, если задана матрица «доходов» $c [M, M]$ и суммируются доходы по траектории, соответствующие переходам из состояния в состояние, то средний доход на траектории, принадлежащей M_α,

$$\sum_{i \in M_\alpha,\, j \in M_\alpha} c [i, j] \times \nu [i, j]$$

с вероятностью 1 стремится к математическому ожиданию дохода на одном переходе в стационарном режиме

$$\bar{c}_\alpha = \sum_{i \in M_\alpha,\ j \in M_\alpha} c [i, j] \times \pi [i, j]. \tag{9}$$

Если же начало траектории является переходным состоянием, то средний доход на траектории будет зависеть от того, в какой эргодический класс попадет траектория. Вероятности попадания процесса из данного переходного состояния i в эргодические классы M_α вычисляются, очевидно из уравнений

$$q [i, \alpha] = p [i, M] \times q [M, \alpha], \tag{10}$$

где

$$q [i, \alpha] = \delta (\beta = \alpha) \text{ при } i \in M_\beta.$$

Предложение 3. *Система* (10) *имеет единственное решение, и траектория марковской цепи, начинающаяся из* i, *с вероятностью* $q[i, \alpha]$ *попадает в* M_α.

Доказательство. Пусть M' — множество переходных состояний. Для этих состояний система (10) может быть переписана в виде

$$q[M', \alpha] - p[M', M'] \times q[M', \alpha] = p[M', M_\alpha] \times \mathbf{1}[M_\alpha]. \quad (11)$$

Нам достаточно убедиться в невырожденности матрицы

$$a[M', M'] = E[M', M'] - p[M', M'].$$

У этой матрицы на диагонали стоят положительные элементы, все прочие элементы неположительны, сумма элементов в каждой строке неотрицательна и хотя бы одна из этих сумм положительна.

Допустим, что столбцы матрицы $a[M', M']$ линейно зависимы, т. е. найдется $\lambda[M'] \neq \mathbf{0}[M']$, для которого $a[M', M'] \times \lambda[M'] = \mathbf{0}[M']$. Не умаляя общности, можно считать, что $\max\{\lambda[i] \mid i \in M'\} = \lambda[i_0] > 0$. Пусть $M^+ = \{i \in M' \mid \lambda[i] \geqslant 0\}$. Тогда

$$\begin{aligned} 0 = a[i_0, M'] \times \lambda[M'] &\geqslant a[i_0, M^+] \times \lambda[M^+] \geqslant \\ &\geqslant \lambda[i_0] \times (a[i_0, M^+] \times \mathbf{1}[M^+]) \geqslant \\ &\geqslant \lambda[i_0] \times (a[i_0, M'] \times \mathbf{1}[M']) \geqslant 0. \end{aligned}$$

Очевидно, во всех этих неравенствах достигается знак равенства. Покажем теперь, что $M'' = \{i \in M' \mid \lambda[i] = \lambda[i_0]\}$ содержит эргодический класс. Действительно, для любого $i_0 \in M''$ мы по предыдущему имеем

$$a[i_0, M''] \times \mathbf{1}[M''] = 0,$$

что совпадает с (3), и, следовательно, само M'' либо минимальное его подмножество, удовлетворяющее (3), является эргодическим классом. Это противоречит определению M' и опровергает предположение о вырожденности матрицы $a[M', M']$.

Матрица $a[M', M']$ имеет обратную матрицу $a^-[M', M']$, которая, очевидно, может быть представлена рядом

$$a^-[M', M'] = E[M', M'] + \sum_{k \geqslant 1} p^k[M', M'].$$

(Мы всегда имеем право написать этот ряд формально. Его сходимость следует из существования матрицы $a^-[M',$

$M']$ и неотрицательности членов ряда.) Далее

$$q[M', \alpha] = a^{-}[M', M'] \times p[M', M_\alpha] \times \mathbf{1}[M_\alpha] = \\ = \sum_{k \geqslant 1} p^k [M', M_\alpha] \times \mathbf{1}[M_\alpha],$$

т. е. $q[M', \alpha] = \sum_{k \geqslant 1} q_k [M', \alpha]$, где $q_k [i, \alpha]$ — вероятность попасть из i в M_α ровно за k шагов. Отсюда следует второе утверждение теоремы. ▲

Можно говорить о математическом ожидании среднего дохода на траектории, начинающейся из состояния i. Оно, очевидно, будет равно

$$\gamma_i = \sum_\alpha q [i, \alpha] \times \bar{c}_\alpha. \qquad (12)$$

Однако в отличие от предыдущего γ_i не будет равно пределу среднего дохода на траектории, который должен совпадать с одним из $\bar{c}_\alpha$. Самостоятельный интерес представляет случай, когда все эти $\bar{c}_\alpha$ одинаковы и (для определенности и для удобства) равны 0. Здесь естественно появляется новая система показателей, характеризующих относительную доходность состояний, родственная системе двойственных переменных для задач линейного программирования и поэтому важная для понимания смысла показателей, возникающих при решении оптимизационных задач, связанных с марковскими цепями. Такого рода параметры марковской цепи изучаются в теории потенциала для марковских процессов, некоторые понятия которой приводятся в следующем параграфе.

В заключение рассмотрим уравнение

$$y [M] \times p [M, M] = b [M] + y [M], \qquad (13)$$

аналогичное (6), но уже для всех без исключения состояний цепи. Примем

$$b[M] = \sum_\alpha p_0 [M] \times q[M, \alpha] \times \pi_\alpha [M] - p_0 [M]. \qquad [14]$$

Здесь уменьшаемое имеет смысл математического ожидания предельного распределения вероятностей, и его связь с γ_i из (12) аналогична связи $\pi_\alpha [i]$ с $\bar{c}_\alpha$.

Для переходных состояний уравнение (13) однозначно разрешимо, как это было установлено при доказательстве предложения 3. Зная $y [\bar{M}]$ для множества переходных состояний $\bar{M}$, получаем для каждого эргодического

класса M_α автономную систему

$$y[M_\alpha] \times p[M_\alpha, M_\alpha] = y[M_\alpha] + (b[M_\alpha] - y[\bar{M}] \times p[\bar{M}, M_\alpha]), \tag{15}$$

совпадающую с (6). Обозначим через $\rho[M]$ вектор, составленный из однозначно определенного $y[\bar{M}]$ и решений систем (15), выбранных, как сказано выше. Отметим, что такой вектор можно построить по любому начальному распределению $p_0[M]$. Аналог системы (13) для оптимизационных задач встретится в § 6.

§ 2. Марковские цепи и потенциалы

Мы ограничим себя самыми простыми моделями, которые в теории потенциала используются лишь как предварительные для введения основных понятий и фактов.

Рассмотрим вначале марковскую цепь с матрицей перехода $p[M, M]$ и единственным эргодическим классом, состоящим из единственного состояния i_+ (*состояния поглощения*). Пусть задана матрица $c[M, M]$ доходов при переходах из состояния в состояние, причем $c[i_+, i_+] = 0$.

Обозначим через $f[i]$ математическое ожидание дохода, получаемого на бесконечной траектории, идущей из состояния i. Значения $f[i]$ для различных i связаны управлением

$$f[i] = \Sigma_{j \in M} (c[i, j] + f[j]) \times p[i, j]. \tag{16}$$

Естественно, полагая

$$f[i_+] = 0, \tag{17}$$

мы можем однозначно разрешить эти уравнения. Действительно, (16) можно переписать в виде

$$f[i] - p[i, M] \times f[M] = \gamma[i], \tag{18}$$

где $\gamma[i] = \Sigma_j c[i\ j] \times p[i, j]$. Матрица системы для $M' = M - i_+$ нам уже встречалась в доказательстве предложения 3, где было показано, что она невырожденная.

Решение системы (18) можно представить в виде ряда

$$f[M] = \Sigma_{k \geqslant 0} p^k[M, M] \times \gamma[M], \tag{19}$$

где $p^0[M, M] = E[M, M]$,

$$p^k[M, M] = p[M, M] \times p^{k-1}[M, M].$$

Матрица

$$s[M', M'] = \Sigma_{k \geqslant 0} p^k [M', M'] \qquad (20)$$

называется *резольвентой* нашей марковской цепи. Элементы матрицы s имеют простой вероятностный смысл: $s[i,j]$ — математическое ожидание числа попаданий в j бесконечной траектории, начинающейся из i.

Вектор $f[M]$, удовлетворяющий системе (16), называется *потенциалом* или, если $\gamma[M] \geqslant \mathbf{0}[M]$, *потенциалом заряда* γ.

Функция $f[i]$, удовлетворяющая неравенству

$$f[M] \geqslant p[M, M] \times f[M], \qquad (21)$$

называется *эксцессивной* функцией.

Предложение 4. *Потенциал заряда является эксцессивной функцией.*

Доказательство следует непосредственно из (18) и (21).

Предложение 5. *Минимум из двух эксцессивных функций является эксцессивной функцией.*

Доказательство. Действительно,

$$\min\{f[i], g[i]\} \geqslant$$
$$\geqslant \min\{p[i, M] \times f[M], p[i, M] \times g[M]\} \geqslant$$
$$\geqslant \Sigma_{j \in M} p[i, j] \times \min\{f[j], g[j]\}. \blacktriangle$$

Введем еще функцию $f[M]$, удовлетворяющую условию

$$f[M] = p[M, M] \times f[M], \qquad (22)$$

и назовем ее *регулярной*. В рассматриваемом сейчас случае функциями, удовлетворяющими (22), будут лишь константы.

В случае, когда $p[M, M]$ имеет несколько эргодических классов, потребуем, чтобы на каждом эргодическом классе M_α равнялось нулю математическое ожидание дохода за один шаг в стационарном режиме

$$\bar{c}_\alpha = \pi_\alpha[M_\alpha] \times \gamma[M_\alpha]. \qquad (23)$$

Сохраняя определения эксцессивной функции и регулярной функции, назовем *потенциалом* функцию $f[M]$, удовлетворяющую условию (16) и для каждого эргодического класса M_α условию

$$\pi[M_\alpha] \times f[M_\alpha] = 0. \qquad (24)$$

Если при этом функция $\gamma[M]$ неотрицательна (а следовательно, равна 0 на любом эргодическом классе), $f[M]$ называется *потенциалом заряда* γ.

В заключение приведем несколько знаменитых свойств потенциала заряда. Для краткости слово «заряд» будет в дальнейшем опускаться. Множество положительности заряда назовем *носителем потенциала.*

Обозначим через $q_A[i, j]$ вероятность того, что марковская цепь, двигаясь из состояния i в качестве своего первого состояния в множество A будет иметь состояние j. Таким образом,

$$q_A[M, A] \geqslant \mathbf{0}[M, A], \quad q[A, A] = E[A, A],$$
$$q_A[M, A] \times \mathbf{1}[A] \leqslant \mathbf{1}[M]. \tag{25}$$

Теорема 3 (Ф. Рисса). *Любая эксцессивная функция однозначно представима в виде суммы регулярной функции и потенциала.*

Доказательство. Пусть f — эксцессивная функция. Умножив неравенства (21) на вектор стационарных вероятностей $\pi[M_\alpha]$, получим

$$\pi[M_\alpha] \times f[M_\alpha] \geqslant \pi[M_\alpha] \times (p[M_\alpha, M] \times f[M]) =$$
$$= \pi[M_\alpha] \times f[M_\alpha],$$

откуда следует, что на M_α неравенства (21) выполняются как равенства. Обозначим $d_\alpha = \pi[M_\alpha] \times f[M_\alpha]$ и положим

$$g[i] = \sum_\alpha q[i, \alpha] \times d_\alpha.$$

Очевидно (ср. (10)),

$$g[i] = p[i, M] \times g[M]$$

и, следовательно, функция $g[M]$ регулярна. Покажем, что функция $h[M] = f[M] - g[M]$ является потенциалом. Действительно, $h[M_\alpha] = \mathbf{0}[M_\alpha]$, так как на M_α функции f и g совпадают. Кроме того,

$$h[M] - p[M, M] \times h[M] =$$
$$= (f[M] - p[M, M] \times f[M]) - (g[M] - p[M, M] \times$$
$$\times g[M]) \geqslant \mathbf{0}[M].$$

Однозначность представления следует из того, что на эргодических классах g должна совпадать с f, а по значе-

ниям на эргодических классах регулярная функция определяется однозначно. ▲

Предложение 6 (принцип максимума). *Пусть* $h[M]$ — *потенциал и* A — *его носитель. Тогда для любого* $i \in M$

$$h[i] \leqslant \max\{h[j] \mid j \in A\}.$$

Доказательство. Очевидно, $h[i] = q_A[i, A] \times h[A]$. Отсюда и из (25) следует требуемое неравенство. ▲

Предложение 7 (принцип доминирования). *Если* $f[M]$ *и* $g[M]$ — *потенциалы,* A — *носитель потенциала* g *и*

$$f[A] \geqslant g[A], \text{ то } f[M] \geqslant g[M].$$

Доказательство. Утверждение следует из того, что для $i \in A$

$$g[i] = q_A[i, A] \times g[A], \quad f[i] \geqslant q_A[i, A] \times f[A]$$

(к $g[i]$ на пути из i в A ничего не прибавляется, а к $f[i]$ добавляется что-то неотрицательное). ▲

Предложение 8 (принцип выметания). *Пусть* $g[M]$ — *потенциал заряда* γ *и* $A \subset M$ — *непустое множество. Функция* $f[M] = q_A[M, A] \times g[A]$ *обладает следующими свойствами:*

а) $f[M]$ — *потенциал,*

б) $f[M] \leqslant g[M]$ *и* $f[A] = g[A]$,

в) *носитель* f *содержится в* A *и однозначно определяется по этим свойствам.*

Доказательство. Равенство $f[A] = g[A]$ очевидно. Возьмем какое-либо $i \in M \setminus A$. Значение $g[i]$ может быть представлено в виде двух слагаемых — неотрицательного математического ожидания доходов до попадания в множество A и математического ожидания доходов после попадания в A, которое равно $q_A[i, A] \times g[A]$. Таким образом,

$$g[i] \geqslant q_A[i, A] \times g[A] = f[A],$$

и утверждение б) доказано. Для $i \in M \setminus A$ имеем

$$f[i] = q_A[i, A] \times g[A] =$$
$$= p[i, M] \times q_A[M, A] \times g[A] = p[i, M] \times f[M], \qquad (26)$$

а для $i \in A$

$$f[i] = g[i] = \gamma[i] + p[i, M] \times g[M] =$$
$$= \gamma[i] + p[i, M] \times (g[M] - f[M]) + p[i, M] \times f[M] =$$
$$= \gamma'[i] + p[i, M] \times f[M],$$

откуда следует, что f — потенциал неотрицательного заряда γ' и его носитель содержится в A.

Обратное утверждение следует из равенств $f[A] = g[A]$ и (26). ▲

§ 3. Пример управляемого случайного процесса: стохастическая задача управления запасами

Задача управления запасами уже встречалась в предыдущей главе. Здесь мы рассмотрим более сложную и интересную модель, в которой заранее неизвестен спрос на продукт в каждый момент времени. Этот спрос будет предполагаться случайным, причем вероятностные характеристики спроса будут считаться заданными. Модели, в которых создается запас, обеспечивающий этот случайный спрос с вероятностью единица, особого интереса не представляют. Чаще бывает нужна постановка задачи, в которой нехватка запаса допустима, но приводит к некоторым потерям — связанным с ожиданием этого запаса (и условным или реальным штрафом за потери) либо со срочной дополнительной постановкой. В любом случае будем считать, что нехватка запаса приводит к появлению некоторых дополнительных затрат, соизмеримых с затратами на хранение и на пополнение запаса. Для простоты мы будем сейчас рассматривать задачи со стационарным марковским процессом спроса и уровни спроса будем считать дискретными.

Итак, пусть задана марковская цепь с конечным множеством состояний X, элементы которого суть неотрицательные числа, т. е. заданы начальное распределение $\pi_0[X]$ и матрица переходных вероятностей $p[X, X]$. Состояние x_t этой цепи в момент времени t задает спрос на продукцию, предъявляемый складу в этот момент времени. Склад, имея запас продукции s_t, по возможности удовлетворяет этот спрос, т. е. в случае, если $s_t \geqslant x_t$, удовлетворяет спрос полностью и остается с запасом $z_t = s_t - x_t$, а в случае $s_t < x_t$ остается с запасом $z_t = 0$ и оставляет неудовлетворенным спрос $x_t - s_t$.

Могут рассматриваться различные варианты удовлетворения остатка спроса. Здесь для простоты примера будем считать, что этот спрос теряется, приводя к убытку $\psi(x_t - s_t)$, где $\psi(x)$ — заданная функция $\psi(0) = 0$.

По известному остатку запаса z_t и по состоянию x_t марковской цепи в момент времени t производится пополнение запаса. Пополнение его до уровня y требует затрат $\varphi(y - z_t)$, где φ — заданная функция.

Общие затраты складываются из убытков от потерянного спроса, затрат на пополнение склада и затрат на хранение остатка продукции, которые равны $\gamma(z_t)$ ($\gamma(0) = 0$).

Таким образом, от запаса s_t мы перешли к запасу s с затратами

$$\lambda_t = \psi((x_t - s_t)^+) + \gamma((s_t - x_t)^+) + \varphi(s - (s_t - x_t)^+). \quad (27)$$

Когда рассматривается функционирование процесса в моменты $t \in 0 : t1$, получается траектория из состояний $s_0, s_1, \ldots, s_{t1}$ с суммарными затратами, которые равны

$$\sum_{t \in 0 : t1-1} \lambda_t.$$

Эти суммарные затраты случайны. Нам уже встречались в предыдущей главе постановки задачи, в которых показатель качества управления был случаен. Здесь это — одна из основных особенностей моделей. Мы пойдем по обычному (но далеко не единственному пути) и *будем в качестве целевой функции рассматривать математическое ожидание затрат.*

С выбором целевой функции в какой-то мере связан выбор состояний описываемого процесса. В терминах s_t и s мы не можем записать математическое ожидание затрат, так как они зависят от x_t, а распределение x_{t-1} в свою очередь зависит от x_t. Кроме того, сам запас s мы должны выбрать в зависимости от x_t, и поэтому нежелательно было бы прятать «внутренние переменные» перехода от s_t к $s = s_{t+1}$. Естественнее в качестве состояний процесса взять не s_t, а как раз эти внутренние переменные — пары $\zeta_t = (x_t, z_t)$ — и описывать переход от пары ζ_t к паре ζ_{t+1}.

Представляется естественным формально ввести функцию $\sigma(\zeta)$, описывающую зависимость будущего запаса s_{t+1} от пары $\zeta_t = (x_t, z_t)$. Зафиксировав эту функцию, мы получим марковскую цепь с множеством состояний $Z = X \times S$ (где S — множество возможных уровней запаса — должно быть таким, чтобы $s - (s - x)^+ \in S$ для любых $s \in S$,

$x \in X$), матрицей переходных вероятностей $p[Z, Z]$, где

$$p[\zeta, \zeta'] = p[x, x'] \times \delta((\sigma(\zeta) - x')^+ = z') \qquad (28)$$

и при переходе из состояния ζ в состояние ζ' возникают убытки

$$c[\zeta, \zeta'] = \gamma(z) + \varphi(\sigma(\zeta)) + \psi((x' - \sigma(\xi))^+). \qquad (29)$$

Изучение получившейся марковской цепи проводится обычным путем с помощью техники и понятий, описанных в предыдущих параграфах. Можно рассматривать эргодические классы цепи, вычислить на них стационарные распределения, определить для каждого эргодического класса средние убытки на один шаг, а для каждого переходного состояния вектор вероятностей попадания в эти эргодические классы.

Если вычесть средние убытки $\bar{\gamma}$ из одношаговых убытков $c[\zeta, \zeta']$, приносимых переходом из одного состояния в другое, для измененных убытков

$$\gamma'[\zeta] = \Sigma_\zeta p[\zeta, \zeta'] \times c[\zeta, \zeta'] - \bar{\gamma}, \qquad (30)$$

мы получим возможность вычислить потенциалы

$$v[\zeta] = \gamma[\zeta] + p[\zeta, Z] \times v[Z], \qquad (31)$$

характеризующие относительную «убыточность» тех или иных состояний цепи.

Нас, однако, должно интересовать не такое описание, а оптимальное управление запасами. Здесь возможны две постановки задачи. При одной мы можем искать такую функцию $\sigma[Z]$, которой соответствуют минимальные средние убытки за один шаг. При другой постановке, более характерной для способа рассуждений из предыдущей главы, мы можем управлять отрезком цепи ограниченной длины, меняя от шага к шагу управляющую функцию $\sigma[Z]$. Эти постановки задачи различны и изучаются совершенно различными методами, однако, разумеется, между ними имеется связь такого типа, как это было описано в § 3 гл. 6.

Отметим еще, что при беглом взгляде на эти постановки задачи возникает два естественных (но несколько противоположных по своему характеру) вопроса. Во-первых, когда в многошаговой постановке мы рассматриваем меняющиеся от шага к шагу функции управления, не может ли получиться уменьшения ожидаемых затрат от учета предыстории

процесса? Ответ на этот вопрос отрицателен. Марковость процесса и рекуррентные свойства процессов динамического программирования оказываются здесь достаточными для того, чтобы можно было учитывать всю предысторию процесса заданием последнего состояния.

Второй вопрос таков: если задание будущих спросов требует меньшей информации, чем последнее состояние процесса x_t, нельзя ли ограничиться этой меньшей информацией? Например, нельзя ли вообще обойтись без этой информации в случае, когда спрос в последовательные промежутки времени образует независимые случайные величины? А на этот вопрос следует ответить положительно. Мы требуем включения в состояние нашего управляемого процесса лишь той минимальной информации, которая позволяет описать вероятностные характеристики последующего процесса, в том числе, конечно, и дальнейшее получение самой этой минимальной информации.

Ответ на оба эти вопроса будет дан в следующем параграфе.

§ 4. Марковские и полумарковские процессы решения

Здесь будет описан основной объект этой главы — управляемый случайный процесс или *процесс решения*. Основой для определений будет марковская цепь или некоторое обобщение марковской цепи — цепь со случайными продолжительностями переходов из состояния в состояние. Такое обобщение называется обычно *полумарковским процессом*.

Будет рассматриваться процесс с множеством состояний M. Каждому состоянию $i \in M$ сопоставлено непустое множество управлений N_i, доступных в этом состоянии. Каждому $u \in N_i$ ставится в соответствие распределение вероятностей μ_u на множестве $M \times R^+ \times R^1$, где R^1 — вещественная прямая, а R^+ — положительный луч. Предполагается, что при выборе управления u некоторый случайный механизм, независимый от предыстории процесса, выбирает в соответствии с распределением μ_u случайную тройку $\langle \varkappa, \tau, \gamma \rangle$, где $\varkappa \in M$ — новое состояние процесса, $\tau > 0$ — продолжительность перехода из i в $\varkappa$, γ — доход, получаемый при этом переходе.

Математическое ожидание по μ_u будет обозначаться через E_u. Таким образом,

$$c_u = \mathsf{E}_u\gamma \quad \text{и} \quad t_u = \mathsf{E}_u\tau \tag{32}$$

обозначают соответственно математическое ожидание получаемого при управлении u дохода и продолжительности перехода в следующее состояние.

Функционирует этот процесс следующим образом: в начальном состоянии i_0 выбирается управление $u_0 \in N_{i_0}$. В соответствии с распределением μ_{u_0} находится тройка $\langle i_1, t^1, c^1\rangle$, выбирается $u_1 \in N_{i_1}$, находится в соответствии с μ_{u_1} тройка $\langle i_2, t^2, c^2\rangle$ и т. д., пока процесс не оборвется в каком-либо состоянии i_p. Получающаяся траектория может быть представлена списком

$$\mathfrak{r} = (r_0, r_1, \ldots, r_{p-1}, i_p),$$

где элемент списка r_k состоит из состояния i_k, управления u_k и чисел t^k и c^k. Число p будет называться длиной траектории и обозначаться через $lgh\ (\mathfrak{r})$.

Определенный таким образом процесс называется *полумарковским процессом решения*, а в случае $\tau \equiv 1$ при всех u— *марковским процессом решения.*

Нас будут интересовать задачи выбора управления, приносящего наибольший ожидаемый доход. Определения понятия «управление» могут быть в этой схеме весьма различными.

Наиболее общий способ задания управления состоит в том, что управление определяется как функция $u\ (\mathfrak{r})$, заданная на всех конечных траекториях $\mathfrak{r}$. Пусть $j\ (\mathfrak{r})$ — последнее состояние траетории $\mathfrak{r}$. Если для всех $\mathfrak{r}$ выполняется $u\ (\mathfrak{r}) \in N_{j(\mathfrak{r})}$, функция u называется *детерминированной решающей функцией* (или просто *решающей функцией*), если же значением $x\ (\mathfrak{r})$ является распределение вероятностей на $N_{j(\mathfrak{r})}$, она называется *рандомизированной решающей функцией* (от английского random — случайный. Использование рандомизированных функций решения предполагает, что лицо, принимающее решение, выбирает не конкретное решение, а распределение вероятностей, в соответствии с которым некоторый независимый случайный механизм — рандомизатор — уже и выбирает конкретное решение).

Решающая функция общего вида — даже детерминированная — это чрезвычайно громоздкая конструкция. Естественно возникает желание рассмотреть более простые функции от траекторий, ограничиваясь некоторой рекуррентно вычисляемой информацией о предыстории процесса. Отметим, что каждую траекторию $\mathfrak{r}$ можно представить (если только длина траектории больше 0) в виде траектории на единицу меньшей длины и записи, состоящей из последнего управления, последнего полученного дохода, последней продолжительности перехода и последнего состояния

$$\mathfrak{r} = (\mathfrak{r}', \; tip\,(\mathfrak{r})), \quad tip\,(\mathfrak{r}) = (\bar{u}, \bar{t}, \bar{c}, \bar{i}). \tag{33}$$

Естественно, при этом $\bar{u} \in j\,(\mathfrak{r}')$.

Назовем функцию траектории $\varphi\,(\mathfrak{r})$ *марковской статистикой* [1]) траектории, если

$$\varphi\,(\mathfrak{r}) = \psi\,(\varphi\,(\mathfrak{r}'), \; tip\,(\mathfrak{r})). \tag{34}$$

Примерами марковских статистик могут служить: длина траектории, последнее состояние процесса $j\,(\mathfrak{r})$, пара последних состояний $j\,(\mathfrak{r})$, $j\,(\mathfrak{r}')$, накопленные суммы $t\,(\mathfrak{r})$ и $c\,(\mathfrak{r})$ величин t^i и c^i на траектории $\mathfrak{r}$.

Можно упростить решающую функцию, потребовав, чтобы она зависела только от некоторой марковской статистики траектории. Мы будем называть такие решающие функции *марковскими решающими функциями*. Ограничиваясь каким-либо классом решающих функций, мы сужаем задачу и, вообще говоря, уменьшаем максимум целевой функции. Будем говорить, что марковская статистика $\varphi\,(\mathfrak{r})$ является *марковской достаточной статистикой* для некоторого класса экстремальных задач, если в любой из этих задач сужение множества решающих функций до марковских решающих функций, зависящих от $\varphi\,(\mathfrak{r})$, не уменьшает максимума целевой функции.

Как правило, в конкретных классах задач удается указать марковскую достаточную статистику, и эта статистика обычно выглядит очень просто. Рассмотрим в качестве примера следующую задачу.

---

[1]) В математической статистике функция от наблюдений называется *статистикой*. Нам естественно использовать здесь эту терминологию, так как теория решающих функций первоначально возникла как раздел математической статистики.

Максимизация дохода на траекториях фиксированной длины.

Найти решающую функцию, при которой математическое ожидание дохода на траекториях длины n, начинающихся из состояния i_0, будет максимальным.

Мы будем, естественно, рассматривать эту задачу с различными n и i_0.

Теорема 4. *Марковской достаточной статистикой в задаче максимизации дохода на траекториях фиксированной длины является пара* $(i_0, n - lgh(\mathfrak{r}))$.

Доказательство. Возьмем какую-либо решающую функцию u (для простоты детерминированную; рассуждения с рандомизированными решающими функциями проводятся так же, но более громоздки) и будем последовательно превращать эту функцию в марковскую решающую функцию, не уменьшая при этом ожидаемого дохода. Вначале для любого состояния $i \in M$ обозначим через $\bar{u}(i)$ то управление из N_i, на котором достигается максимум ожидаемого дохода t_u (см. (32)). Пусть u_1 — решающая функция, совпадающая с u при $lgh(\mathfrak{r}) < n$ и равная $\bar{u}(j(\mathfrak{r}))$ при $lgh(\mathfrak{r}) = n$. Так как доход $c(\mathfrak{r})$ на траектории $\mathfrak{r}$ является марковской статистикой и

$$c(\mathfrak{r}) = c(\mathfrak{r}') + \bar{c},$$

то, обозначив через E_u математическое ожидание функции от траектории, получающейся при использовании решающей функции u, мы получим

$$\mathsf{E}_u c(\mathfrak{r}) = \mathsf{E}_u c(\mathfrak{r}') + \mathsf{E}_u \bar{c} =$$
$$= \mathsf{E}_u c(\mathfrak{r}') + \sum_{i \in M} p_u(j(\mathfrak{r}') = i) \times \mathsf{E}_u(\gamma \mid j(\mathfrak{r}') = i) \leqslant$$
$$\leqslant \mathsf{E}_u c(\mathfrak{r}') + \sum_{i \in M} p_u(j(\mathfrak{r}') = i) + c_{\bar{u}(i)} = \mathsf{E}_{u_1} c(\mathfrak{r}). \quad (35)$$

Таким образом, при отыскании оптимального управления можно ограничиться решающими функциями, значения которых на последнем шаге процесса зависят от $j(\mathfrak{r})$.

Предположим, что для некоторого k мы уже установили, что при отыскании оптимального управления можно ограничиться решающими функциями, которые на траекториях $\mathfrak{r}$, $lgh(\mathfrak{r}) \geqslant n - k$ принимают значения, зависящие только от $j(\mathfrak{r})$ и $n - lgh(\mathfrak{r})$.

Обозначим через $v_k(i)$ максимум математического ожидания дохода на k-шаговой траектории, начинающейся из

$i \in M$, и определим $\bar{u}_k(i)$ как управление, на котором достигается максимум,

$$\max \{c_u + \mathsf{E}_u v_k(x) \mid u \in N_i\}. \tag{36}$$

Траекторию r можно представить в виде $(n - k)$-шагового начала $\mathrm{r}^{(k)}$ и k-шагового продолжения $\mathrm{r}_{(k)}$. По предположению,

$$\mathsf{E}_u c(\mathrm{r}) \leqslant \mathsf{E}_u c(\mathrm{r}^{(k)}) + v_k(j(\mathrm{r}^{(k)})).$$

Отщепляя, как в (34), последний элемент траектории $\mathrm{r}^{(k)}$, мы получим

$$\mathsf{E}_u c(\mathrm{r}) \leqslant \mathsf{E}_u c(\mathrm{r}^{k+1}) + v_{k+1}(j(\mathrm{r}^{k+1})),$$

и, следовательно, максимум дохода можно искать на решающих функциях, которые на траекториях с $lgh(\mathrm{r}) = n - k - 1$ совпадают с $\bar{u}_k(i)$.

Это завершает индуктивный переход. ▲

Такими же рассуждениями можно показать, что при суммарном ограничении на продолжительность процесса марковской достаточной статистикой будет пара (i, t), где t — остаточная продолжительность, а при движении до попадания в поглощающее состояние марковская достаточная статистика состоит из одного лишь последнего состояния траектории i, и, следовательно, можно искать оптимальную решающую функцию в классе стационарных политик.

§ 5. Функциональные уравнения и рекуррентные соотношения Беллмана для марковских процессов решения

Введенные при доказательстве теоремы 4 функции $v_k(i)$ являются аналогами потенциалов, которые встречались нам при изучении детерминированных процессов динамического программирования. Из (36) легко следует, что $v(i_k)$ удовлетворяют рекуррентному соотношению Беллмана

$$v_k(i) = \max \{c_u + \mathsf{E}_u v_{k-1}(x) \mid u \in N_i\}. \tag{37}$$

Мы будем здесь рассматривать случай бесконечных траекторий, предполагая, что будут сделаны ограничения, при которых доход на траектории остается конечным. Если обозначить максимум математического ожидания дохода на траекториях, начинающихся из i, через $v(i)$, естественно

ожидать, что $v(i)$ будет удовлетворять уравнению Беллмана, являющемуся пределом соотношений (37)

$$v(i) = \max \{c_u + \mathsf{E}_u v(\varkappa) \mid u \in N_i\}. \tag{38}$$

Теорема 5. *Если существует* $v(M) = \lim\limits_{k\to\infty} v_k(M)$, *то* $v(M)$ *удовлетворяет уравнению Беллмана* (38).

Доказательство. Так как решающих функций $u[M]$ имеется лишь конечное число, то найдется решающая функция $u[M]$, на которой максимум в (37) достигается при бесконечном числе значений k. Переходя к пределу по этим k, получаем

$$v(i) = c_{\bar{u}[i]} + \mathsf{E}_{\bar{u}[i]} v(\varkappa) < \max \{c_u + \mathsf{E}_u v(\varkappa) \mid u \in N_i\}.$$

Вместе с тем, прямо переходя в (37) к пределу, имеем

$$v(i) \geqslant \max \{c_u + \mathsf{E}_u v(\varkappa) \mid u \in N_i\},$$

откуда и следует утверждение теоремы. ▲

В этом параграфе мы будем рассматривать вопрос о существовании и единственности решения уравнения (37). Однако прежде, чем будет сформулирован основной результат, мы обсудим, какие условия следует наложить на процесс, чтобы математическое ожидание дохода было конечным.

Очевидно, что ни при каком управлении не должны появляться эргодические классы с положительным средним доходом. Действительно, при таком управлении система уравнений для потенциалов не имеет решения, а уравнение (37) и подавно. Вместе с тем должна быть возможность перейти из любого состояния в эргодический класс с нулевым средним доходом.

Эти условия проще формулировать, если привести нахождение эргодических классов с максимальным средним доходом к экстремальной задаче. Введем предварительно еще одно определение.

Назовем *подпроцессом* рассматриваемого марковского процесса такое множество состояний $M' \subset M$, что для любого $u \in N(M') = \bigcup_{i\in M'} N_i$ с вероятностью 1 состояние $\varkappa \in M'$.

Оптимальный стационарный режим для подпроцесса.

Пусть задан подпроцесс $M' - M$. Положим $N' = N(M')$, $p[u, i] = \mathsf{E}_u \delta(\varkappa = i)$. Требуется найти вектор,

удовлетворяющий условиям

$$x[N'] \geqslant 0\,[N], \quad \mathbf{1}[N'] \times x[N'] = 1, \tag{39}$$

$$x[N_i] \times \mathbf{1}[N_i] = x[N'] \times p[N', i] \tag{40}$$

и максимизирующий

$$c[N'] \times x[N']. \tag{41}$$

Прежде всего опишем в естественных для марковских цепей терминах, что представляет собой базисное решение этой задачи. Пусть $x[N']$ — допустимое решение задачи, т. е. вероятностный вектор удовлетворяющий условию (40). Поставим в соответствие этому решению граф переходов $\langle M^+, N^+\rangle$, где

$$\begin{aligned} M^+ &= \{i \mid i \in M', \ \mathbf{1}[N_i] \times x[N_i] > 0\}, \\ N^+ &= \{(i, j) \mid i, j \in M', \ x[N_i] \times p[N_i, j] > 0\}. \end{aligned} \tag{42}$$

Назовем допустимое решение *элементарным*, если этот граф вполне связен и для любого $i \in M^+$ ровно одна из компонент $x[N_i]$ положительна.

Элементарному решению естественно сопоставляется стохастическая матрица $q[M^+ M^+]$, где

$$q[i, M^+] = p[\bar{u}_i, M^+], \tag{43}$$

а $\bar{u}_i$ — то управление из N_i, для которого $x[\bar{u}_i] > 0$. Из свойств элементарного решения следует, что этой матрице соответствует единственный эргодический класс $M' \subset M^+$ и, следовательно, существует однозначно определяемый вектор стационарных вероятностей

$$\pi[M] > 0\,[M'].$$

Отметим, что условие (40) принимает для элементарного решения вид

$$x[\bar{u}_i] = \sum_{j \in M'} x[\bar{u}_j] \times p[\bar{u}_j, i], \tag{44}$$

и, следовательно, векторы $\pi[M']$ и $x[N']$ связаны равенством

$$\pi[i] = x[\bar{u}_i], \quad i \in M'. \tag{45}$$

Таким образом, каждое элементарное решение определяет некоторый эргодический класс некоторой марковской цепи, причем средний доход в этом эргодическом классе равен

значению целевой функции (41) на соответствующем элементарном решении.

Теорема 6. *Пусть* $x\,[N']$ — *допустимое решение задачи* (39)—(41). *Для того чтобы оно было базисным, необходимо и достаточно, чтобы оно было элементарным.*

Доказательство. Достаточность. Пусть элементарное допустимое решение $x\,[N']$ представимо в виде выпуклой комбинации двух допустимых решений

$$x\,[N'] = \frac{1}{2} \times x_1\,[N'] + \frac{1}{2} \times x_2\,[N'].$$

Граф, соответствующий решению $x_i\,[N]$, обозначим через $\langle M_i, N_i\rangle$. Очевидно, $N_i^+ \subset N^+$, $i = 1, 2$, $M_1^+ \cup M_2^+ = M^+$. Предположим, что нашлась вершина $\bar{i} \in M^+ \setminus M_i^+$. Не умаляя общности, можно считать, что в графе $\langle M^+, N^+\rangle$ найдется дуга с концом $\bar{i}$ и началом $i_1 \in M_i^+$.

В векторе $x\,[N_{i_1}]$ была отлична от нуля лишь компонента $x\,[\bar{u}_{i_1}]$, которая в векторе $x_{i_1}\,[N_{i_1}]$ равна нулю, так как $\bar{i} \notin M_1^+$. Следовательно, $x_1\,[N_{i_1}] \times 1\,[N_{i_1}] = 0$, что, однако, противоречит определению M_1^+.

Таким образом, $M_1^+ = M_2^+ = M^+$, а значит, $N_1^+ = N_2^+ = N^+$. Множества M^+ и N^+ однозначно определяют решение $x\,[N']$ в силу отмеченной связи элементарных решений с эргодическими классами, и, следовательно, решения $x_1\,[N']$ и $x_2\,[N']$ совпадают с $x\,[N']$.

Необходимость. Пусть $x\,[N']$ — допустимое решение. Если граф $\langle M^+, N^+\rangle$ не является вполне связным, то, как легко видеть, он разбивается на несколько компонент связности. В этом случае можно каждой из компонент связности $\langle M_\alpha, N_\alpha\rangle$ сопоставить допустимое решение $x_\alpha\,[N']$

$$x_\alpha\,[N'] = c_\alpha \times \delta\,(u \in N_\alpha) \times x\,[N'],$$

и исходное решение $x\,[N']$ будет выпуклой комбинацией решений $x_\alpha\,[N']$.

Предположим теперь, что граф $\langle M^+, N^+\rangle$ вполне связен, но в некоторых из векторов $x\,[N_i]$ больше одной компоненты отлично от нуля. Зафиксируем для каждого $i \in M^+$ какое-то управление $\tilde{u}\,(i)$ такое, что $x\,[\tilde{u}\,(i)] > 0$. Рассмотрим стохастическую матрицу $q\,[M^+, M^+]$, где

$$q\,[i, j] = p\,[\tilde{u}\,(i), j].$$

Возьмем какой-либо из эргодических классов получившейся цепи M_α, найдем соответствующий ему вектор ста-

ционарных вероятностей $\pi_\alpha[M^+]$ и определим вектор $x_\alpha[N']$, полагая

$$x_\alpha[\tilde{u}(i)] = \pi_\alpha[i], \quad i \in M_\alpha,$$

а остальные компоненты считая нулевыми. Легко видеть, что

$$x_\alpha[u] > 0 \Rightarrow x[u] > 0$$

и вектор $x_\alpha[N']$ допустим. Следовательно, $x[N']$ представим в виде выпуклой комбинации $x_\alpha[N']$ и некоторого другого допустимого вектора. ▲

Обозначим теперь для простоты максимум целевой функции в задаче (39)—(41) через $\gamma(M')$.

Т е о р е м а 7. *Для того чтобы уравнение* (37) *имело решение, необходимо и достаточно, чтобы* $\gamma(M') = 0$ *для любого процесса* M'.

Д о к а з а т е л ь с т в о. Н е о б х о д и м о с т ь. Пусть система имеет решение $v[M]$ и $\bar{u}[M]$ — политика, составленная из управлений, на которых достигается максимум в (37). Рассмотрим какой-либо подпроцесс M'. По определению подпроцесса матрица $\bar{p}[M', M']$, составленная для политики $\bar{u}[M]$ по формуле (43), является стохастической. Пусть $\pi[M']$ — вектор стационарных вероятностей для какого-либо эргодического класса этой матрицы. Умножая систему (37) на $\pi[M']$, получим

$$\pi[M'] \times v[M'] = \sum_{i \in M'} \pi[i] \times c_{\bar{u}[i]} + \sum_{i \in M'} \pi[i] \times p[\bar{u}[i], M'] \times v[M']$$

или

$$\sum_{i \in M'} \pi[i] \times c_{\bar{u}[i]} = 0.$$

Вектору $\pi[M']$ соответствует (ср. (45)) допустимое решение $x[N']$ для подпроцесса M' и, следовательно, $\gamma[M'] \geqslant 0$.

Рассмотрим теперь оптимальное базисное (и, следовательно, элементарное) решение $\tilde{x}[N']$ задачи (39)—(45) для подпроцесса M' и соответствующую ему «частичную политику» $\tilde{u}[M']$. Из (37) мы имеем

$$v[i] \geqslant c_{\tilde{u}[i]} + p[\tilde{u}[i], M'] \times v[M'],$$

и, следовательно,

$$v[M'] \times \tilde{x}[\tilde{u}[M']] \geqslant \gamma(M') + v[M'] \times \tilde{x}[\tilde{u}[M']],$$

откуда следует $\gamma(M') \leqslant 0$.

Достаточность. Прежде всего отметим, что система

$$g[i] \geqslant c_u + p[u, M'] \times g[M'], \quad u \in N_i, \quad i \in M'$$

имеет решение. Действительно, если бы она не имела решений, то, используя теорему отделимости (теорема 3 гл. 2), мы убедились бы в существовании неотрицательного вектора $z[N']$, для которого

$$z[N_i] \times 1[N_i] - z[N'] \times p[N', i] = 0,$$
$$u \in N_i, \quad i \in M',$$

и

$$c[N'] \times z[N'] > 0,$$

что противоречит предположению теоремы.

Последовательность функций $\{f_k[M]\}$, в которой

$$f_0[M] = g[M],$$
$$f_k[i] = \max\{c[u] + p[u, M] \times f_{k-1}[M] \mid u \in N_i\}, \tag{46}$$

очевидно, невозрастающие (ср. рассуждения § 4 гл. 6). Покажем, что эта последовательность сходится к решению уравнения (37). Достаточно убедиться в том, что последовательность имеет нижнюю границу.

Мы построим эту границу, последовательно решая задачи (39)—(41). Прежде всего найдем оптимальное базисное решение этой задачи для $M' = M$. Это решение определяется множеством «активных состояний» T и набором управлений $\bar{u}[T]$.

Если в множестве $M \setminus T$ найдется подпроцесс M', повторим для него решение задачи (39)—(41), найдем новое множество активных состояний $T' \subset M'$ и $\bar{u}[T']$, добавим T' к T и повторим действия, перечисленные в этом абзаце.

В конце концов мы придем к некоторому $T \subset M$, на котором определен набор управлений $\bar{u}[T]$, причем множество $M \setminus T$ подпроцессов уже не содержит. Доопределим набор управлений $\bar{u}[T]$ до политики $\bar{u}[M]$, выбрав на $M \setminus T$ произвольные управления, при которых с вероятностью 1 процесс попадал бы в T. Политика $\bar{u}[M]$ определяет стохастическую матрицу $q[M, M]$, для которой T — объединение эргодических классов (в каждом из которых средний доход равен 0). Система

$$h[M] = c[\bar{u}[M]] + q[M, M] \times h[M] \tag{47}$$

разрешима. (Она разрешима с точностью до произвольного слагаемого в каждом эргодическом классе. Для переходных состояний при известных решениях на эргодических классах решение вычисляется однозначно.)

Выберем $h[M]$ так, чтобы $g[M] \geqslant h[M]$. Тогда для любого k мы получаем $f_k[M] \geqslant h[M]$. ▲

Следствие 1. *Если на множестве M задан квазипорядок и из каждого состояния i возможен переход лишь в состояния, строго следующие за i, причем последним является поглощающее состояние i_+, то уравнение* (37) *имеет решение.*

Следствие 2. *Если существует такое $\beta > 0$, что для каждого $u \in N_i$, $i \in M$, справедливо неравенство $p[u, t_+] \geqslant \beta$, то уравнение* (37) *имеет решение.*

Легко видеть, что в каждом из этих случаев $\{i_+\}$ представляет собой единственно возможный эргодический класс.

§ 6. Асимптотика рекуррентных соотношений

В этом параграфе мы вкратце опишем, что происходит, когда условие теоремы 7 не выполняется и функции (37) не сходятся к пределу. Последнее вполне естественно, — когда средний доход за один шаг отличен от нуля, ожидаемый доход за n шагов должен линейно (или приблизительно линейно) изменяться с ростом n. Коэффициент этого линейного роста в случае, когда процесс может состоять из нескольких подпроцессов, может быть различным для различных состояний.

Если зафиксировать какую-либо политику $\bar{u}[M]$, то этой политике будет соответствовать стохастическая матрица $q[M, M]$ и вектор доходов $\bar{c}[M]$, которым по формуле (12) будет сопоставлен вектор математических ожиданий среднего дохода на траектории $\gamma_{\bar{u}}[M]$. Для каждого состояния $i \in M$ можно ставить задачу нахождения наилучшей политики — той, при которой достигается максимум $\gamma_u[i]$. Оказывается, что существует политика $\tilde{u}[M]$, которая доставляет максимум одновременно всем $\gamma_u[i]$, $i \in M$.

Построение такой политики аналогично тому, как строилась политика $\bar{u}$ при доказательстве достаточности в теореме 7, хотя здесь и появляются некоторые дополнительные сложности.

На каждом шаге процесса мы будем иметь множество состояний T, для которых выбрано управление $\bar{u}$ (хотя и не окончательно) и множество состояний $M \setminus T$, для которых управление предстоит выбрать. Предположим, что в множестве $M \setminus T$ найдутся подпроцессы. Отбрасывая у всех состояний $M \setminus T$ те управления, которые с положительной вероятностью переводят в T, мы получим максимальный подпроцесс, содержащийся в $M \setminus T$ и сможем найти для него оптимальную стационарную политику на одном из его эргодических классов T'. Положив $S = T' \cup T$ и определив на T' политику $\bar{u}$, мы не можем еще закончить один шаг построения политики, так как добавление T'' может разрушить построенные ранее эргодические классы. На множестве S определена функция $\gamma[S]$ максимальных ожидаемых средних доходов на траекториях.

Доопределим вектор γ, положив $\gamma[i] = -\infty$ для $i \in M \setminus S$, и рассмотрим последовательность $\gamma^k[M]$

$$\begin{aligned} \gamma^k[i] &= \max\{\mathsf{E}_u \gamma^{k-1}[x] \mid u \in N_i\}, \\ \gamma^0[M] &= \gamma[M]. \end{aligned} \tag{48}$$

Наличие бесконечных значений не мешает нашим рассмотрениям — каждое γ можно считать парой, состоящей из коэффициента при $-\infty$ и обычного вещественного числа, как мы это уже делали многократно раньше.

Последовательность γ^k, очевидно, — неубывающая и ограниченная. Поэтому она сходится к пределу $\bar{\gamma}$. Обозначим через T множество индексов, для которых $\bar{\gamma}[i]$ принимает конечные значения, а через $\bar{u}[T]$ — управления, на которых достигается максимум в уравнении

$$\bar{\gamma}[i] = \max\{\mathsf{E}_u \bar{\gamma}[x] \mid u \in N_i\}, \quad i \in T. \tag{49}$$

Очевидно, что $T \supset S$. Так определенные T и $\bar{u}[T]$ и принимаются при $T \neq M$ в качестве исходной информации следующего шага.

При начальном $T = \emptyset$ мы придем к $T = M$ за конечное число шагов, так как множество T будет на каждом шаге монотонно увеличиваться. Существование подпроцессов в $M \setminus T$ следует из того, что в этом множестве есть состояния, из которых ни при каком управлении нельзя добраться

до T, так как иначе мы смогли бы добиться для этих состояний конечных значений $\bar{\gamma}[i]$.

Теорема 8. *Политика $\bar{u}[M]$, получаемая при описанном выше процессе, доставляет максимум одновременно всем $\gamma_u[i]$, $i \in M$.*

Доказательство. Обозначим через $\bar{\gamma}[M]$ вектор, соответствующий $\bar{u}$, а через $\tilde{\gamma}[M]$ — вектор максимальных средних доходов. Очевидно, что $\bar{\gamma}[M] \leqslant \tilde{\gamma}[M]$.

Предположим, что $\bar{\gamma}[M] \neq \tilde{\gamma}[M]$. Пусть i_0 — состояние, на котором достигается максимум $\tilde{\gamma}[i] - \bar{\gamma}[i] = \lambda$. Обозначим через $u_0[M]$ политику, при которой достигается максимум $\gamma_u[i_0]$. Для любого состояния i мы имеем

$$\gamma_{u_0}[i] - \bar{\gamma}[i] = \mathsf{E}_{u_0(i)}\gamma_{u_0}[x] - \max\{\mathsf{E}_u\bar{\gamma}[x] \mid u \in N_i\} \leqslant$$
$$\leqslant \mathsf{E}_{u_0(i)}(\gamma_{u_0}[x] - \bar{\gamma}[x]) \leqslant \mathsf{E}_{u_0(i)}(\tilde{\gamma}[x] - \bar{\gamma}[x]) \leqslant$$
$$\leqslant \max\{\tilde{\gamma}[j] - \bar{\gamma}[j] \mid p[u_0(i), j] > 0\}.$$

Следовательно, из i_0 можно попасть в другое состояние, на котором достигается максимум разности $\tilde{\gamma}[i] - \bar{\gamma}[i]$ и $\gamma_{u_0}[i] = \tilde{\gamma}[i]$. Отсюда легко следует, что множество состояний, на которых достигается максимум разности $\tilde{\gamma} - \bar{\gamma}$, содержит эргодический класс. В этом эргодическом классе S, очевидно, $\tilde{\gamma}[i] = \text{const}$ и, следовательно, $\bar{\gamma}[i] = \text{const}$.

Теперь мы легко придем к противоречию. При построении политики $\bar{u}$ в тот момент, когда хотя бы одно из состояний S должно было включаться в T, эргодический класс S обладал бо́льшим доходом, чем класс T', добавляемый к T, вопреки предположению об оптимальности добавляемого класса. ▲

Свойство политики $\bar{u}$, доставляющей одновременно максимум всем переменным $\gamma_u[i]$, можно использовать для того, чтобы в единой задаче линейного программирования находить одновременно векторы $\hat{\gamma}[M]$ и потенциалы $v[M]$.

Зададимся произвольным положительным вектором $d[M]$ (не умаляя общности, будем считать его вероятностным) и рассмотрим задачу линейного программирования.

Задача об оптимальной системе показателей.

Найти векторы $g[M]$ и $v[M]$, удовлетворяющие условиям

$$g[i] \geqslant p[u, M] \times g[M], \quad u \in N_i, \quad i \in M, \tag{50}$$

$$v[i] \geqslant c[u] - g[i] + p[u, M] \times v[M], \quad u \in N_i, \quad i \in M, \tag{51}$$

и минимизирующие $d[M] \times g[M]$.

Здесь $v[M]$ — система потенциалов, $g[i]$ — ожидаемый доход на единицу времени от траектории, начинающейся в состоянии i, $d[i]$ можно считать вероятностью того, что началом траектории будет i, тогда целевая функция приобретает смысл оценки ожидаемого дохода.

Двойственной к этой задаче является

Задача об оптимальной стационарной политике.

Найти векторы $x[N]$ и $y[N]$, удовлетворяющие условиям

$$x[N] \geqslant 0[N], \quad y[N] \geqslant 0[N],$$

$$x[N_i] \times 1[N_i] = x[N] \times p[N, i], \quad i \in M, \tag{52}$$

$$(x[N_i] + y[N_i]) \times 1[N_i] = y[N] \times p[N, i] + d[i], \quad i \in M, \tag{53}$$

и максимизирующие $c[N] \times x[N]$.

Так как, просуммировав условия (53) по $i \in M$ и воспользовавшись тем, что $p[N, M] \times 1[M] = 1[N]$, мы получим условие $x[N] \times 1[N] = 1$, то вектор $x[N]$ удовлетворяет всем условиям задачи (39)—(41). В базисном решении вектор $x[N]$ задает в соответствии с формулой (45) вектор $\pi[M]$ стационарных распределений эргодических классов стохастической матрицы (43). Вектор $y[N]$ является аналогом вектора $y[M]$, введенного в § 1 и описывающего перевод начального распределения $d[M]$ в стационарное распределение $\pi[M]$. Как отмечалось в § 1, каждому $d[M]$ можно сопоставить векторы $\pi[M]$ и $\rho[M]$, что при фиксированной политике задает векторы $x[N]$ и $y[N]$, образующие допустимое решение задачи (52), (53). Из этого факта следует, что оптимальное решение задачи (50), (51) является оптимальным одновременно для всех вероятностных векторов $d[M]$.

Теорема 9. *Существует такое оптимальное решение* $g^*[M]$, $v^*[M]$ *задачи об оптимальной системе показателей, что* $g^*[M]$ *удовлетворяет уравнению* (49), *а* $v^*[M]$ — *уравнению*

$$v^*[i] = \max\{c[u] + \mathsf{E}_u v^*[\varkappa] \mid u \in N_i\} - g^*[i], \quad (54)$$

причем максимум в (54) *и в* (49) *достигается для каждого* $i \in M$ *на одном и том же* u_i.

Поскольку о марковских процессах решения мы здесь говорим лишь для единства картины, доказательство этой теоремы не приводится.

Решение $v^*[M]$ уравнения (54) может быть использовано для изучения асимптотического поведения функций $v_k[M]$ из рекуррентного соотношения (37).

Теорема 10. *Пусть* $v^*[M]$ — *решение уравнения* (54). *Существуют такие постоянные* k_1 *и* k_2, *что одновременно для всех* $n > 0$ *справедливы неравенства*

$$k_1 \leqslant v_n[i] - v^*[i] - n \times g^*[i] \leqslant k_2, \quad i \in M. \quad (55)$$

Доказательство. Выберем k_1 и k_2 так, чтобы

$$k_1 \leqslant v_0[i] - v^*[i] \leqslant k_2. \quad (56)$$

Так как (54) можно переписать в виде

$$v^*[i] + n \times g^*[i] + k = \max\{c[u] + \mathsf{E}_u (v^*[\varkappa] + (n-1) \times g^*[\varkappa] + k) \mid u \in N_i\},$$

то функции $v^*[i] + n \times g^*[i] + k$ при любом k удовлетворяют рекуррентному соотношению (37). Теперь осталось воспользоваться монотонностью оператора Беллмана (она была доказана для детерминированных процессов в гл. 6, перенос на случай марковских процессов решения труда не представляет). ▲

Неравенство (55) дает возможность доказать теорему о магистрали и для вероятностных процессов. Разумеется, здесь утверждение теоремы существенно видоизменится.

Зафиксируем решение $g^*[M]$, $v^*[M]$ задачи (50), (51), о котором говорится в теореме 9, и определим

$$\Psi[u] = v^*[i] + g^*[i] - c[u] - \mathsf{E}_u v^*[\varkappa], \quad u \in N_i, \quad i \in M. \quad (57)$$

Рассмотрим аддитивную функцию от траектории процесса $\Psi(\mathrm{r})$—сумму $\Psi[u]$ по выбранным на траектории управлениям.

Теорема 11. *Существует такая постоянная k, что для любого $n > 0$ для оптимальной n-шаговой политики справедливо неравенство*

$$\mathsf{E}_r\ \Psi(r) < k. \qquad (58)$$

Доказательство. Действительно,

$$v_k[i] = \mathsf{E}_r\,(\Sigma_{l\in 1:n} c[u_l]).$$

Пользуясь (57) и (49), после несложных преобразований имеем

$$v_k[i] = \mathsf{E}_r\,(\Sigma_{l\in 1:n}\,(v^*[i_l] - \mathsf{E}_{u_l} v^*[x] + g^*[i_l] - \Psi(u_l))) =$$
$$= \mathsf{E}_r \Sigma_{l\in 1:n}\,(v^*[i_l] - \mathsf{E}_{u_l} v^*[x]) + \mathsf{E}_r \Sigma_{l\in 1:n} g^*[i_l] -$$
$$- \mathsf{E}\Psi(r) \leqslant v^*[i] + n \times g^*[i] - \mathsf{E}\Psi(x),$$

что вместе с (55) и дает утверждение теоремы. ▲

§ 7. Методы нахождения оптимальной стационарной политики

Здесь будет описано несколько методов, имеющих принципиальное значение. Самым естественным подходом считается «метод дисконтирования» — замена рекуррентного соотношения (49) соотношением

$$v_k^\beta[i] = \max\{c[u] + \beta \times \mathsf{E}_u v_{k-1}^\beta[x] \mid u \in N_i\}, \qquad i \in M, \quad (59)$$

где $\beta \in (0, 1)$ достаточно близко к единице. Последовательность $\{v_k^\beta[M]\}$, очевидно, сходится к функции $\bar{v}^\beta$, удовлетворяющей уравнению

$$\bar{v}^\beta[i] = \max\{c[u] + \beta \times \mathsf{E}_u \bar{v}^\beta[x] \mid u \in N_i\}. \qquad (60)$$

Для $\bar{v}^\beta$ справедливо следующее асимптотическое представление.

Теорема 12 (Д. Блекуэлл). *При $\beta \to 1$ имеет место соотношение*

$$\bar{v}^\beta[i] = \frac{g^*[i]}{1-\beta} + v^*[i] + O(1-\beta), \qquad (61)$$

где $v^*[M]$ — решение уравнения (59), *математическое ожидание которого на каждом эргодическом классе стохастической матрицы, соответствующей оптимальной политике, равно* 0.

Доказательство этой теоремы требует иной техники и поэтому не приводится.

При дополнительных предположениях об изучаемом процессе могут быть использованы и другие методы. Интересный метод был предложен Р. Ховардом. Предположим, что при любой политике изучаемый марковский процесс решения имеет один эргодический класс. В этом случае $g^*[M] = \lambda \times \mathbf{1}[M]$.

Метод Ховарда заключается в последовательном пересчете политики $u[M]$ и может быть описан так:

0. Выбрать (произвольно) политику $u[M]$.

1. Построить по политике $u[M]$ стохастическую матрицу (43), вектор стационарных вероятностей ее эргодического класса, средний доход λ и систему потенциалов $v[M]$.

2. Для каждого $i \in M$ найти $\bar{u}[i] \in N_i$, на котором достигается

$$\max \{c[u] + \mathsf{E}_u v[\varkappa] - \lambda - v[i] \mid u \in N_i\} = \Delta[i]$$

(если максимум достигается на $u[i]$, выбрать $\bar{u}[i] = u[i]$. Отметим, что слагаемое $-\lambda - v[i]$ не влияет на выбор максимизирующего u, оно добавлено, чтобы $\Delta[i]$ было неотрицательным).

3. Если $\Delta[M] \neq \mathbf{0}[M]$ (и, следовательно, $\bar{u}[M] \neq u[M]$), положить $\bar{u}[M] := u[M]$ и перейти к 1. В противном случае закончить процесс.

Теорема 13 (Р. Ховард). *Описанный процесс сходится за конечное число шагов к оптимальной стационарной политике.*

Доказательство. Покажем, что на каждом шаге процесса не происходит уменьшения λ. Действительно, пусть $\bar{\lambda}$ и $\bar{\pi}[M]$ — средний доход и стационарные вероятности, соответствующие политике $\bar{u}[M]$. Тогда

$$\bar{\lambda} - \lambda = \sum_{i \in M} \bar{\pi}[i] \times (c[\bar{u}[i]] - \lambda) =$$

$$= \sum_{i \in M} \bar{\pi}[i] \times (v[i] - \mathsf{E}_{\bar{u}[i]} v[\varkappa] + \Delta[i]) = \bar{\pi}[M] \times \Delta[M].$$

Так как на каждом шаге $\Delta[M] \geqslant \mathbf{0}[M]$, то возможно лишь конечное число шагов, на которых значение λ увеличивается. Для тех шагов, на которых λ не увеличивается, можно доказать увеличение вектора потенциалов $v[M]$.

Действительно, если $\bar{\lambda} = \lambda$, то

$$\bar{v}[i] - v[i] = c[\bar{u}[i]] - \lambda + \mathsf{E}_{\bar{u}[i]}\bar{v}[\varkappa] - v[i] = \\ = \Delta[i] + \mathsf{E}_{\bar{u}[i]}(\bar{v}[\varkappa] - v[\varkappa]),$$

т. е. разность $\bar{v}[M] - v[M]$ является потенциалом заряда $\Delta[M]$ и, следовательно, положительна: $\bar{v}[M] \leqslant v[M]$. Поэтому возврат процесса к встречавшейся ранее политике $u[M]$ невозможен, и процесс сходится за конечное число шагов Результатом является политика $\tilde{u}[M]$, для которой существуют такие $\tilde{\lambda}$ и $\tilde{v}[M]$, что

$$\tilde{v}[i] + \tilde{\lambda} = \max\{c[u] + \mathsf{E}_u\tilde{v}[\varkappa] \mid u \in N_i\} = \\ = c[\tilde{u}[i]] + \mathsf{E}_{\tilde{u}[i]}\tilde{v}[\varkappa], \qquad (62)$$

и, следовательно, $\tilde{u}[M]$ — оптимальная стационарная политика. ▲

Метод Ховарда можно рассматривать как разновидность блочного метода для решения задачи линейного программирования (39)—(41), в которой (39) является соединяющей полосой, а (40) — своеобразные блоки.

С другой стороны, его можно рассматривать и как вариант метода приближений в пространстве поведений, описанного в § 4 гл. 6, в котором уравнение (35) решается заново для каждой следующей политики. Нужно, однако, иметь в виду, что этот интересный метод при большом числе состояний процесса оказывается весьма трудоемким, так как требует на каждой итерации решения двух линейных систем размерности $|M|$.

При еще более жестких ограничениях на процесс можно обойтись следующим более простым вариантом метода последовательных приближений. Пусть

$$\omega_k = \min\{v_k[i] \mid i \in M\}, \qquad w_k[M] = v_k[M] - \omega_k \times 1[M]. \qquad (63)$$

Очевидно, можно вести вычисления прямо для $\{w_k\}$, каждый раз «опуская» $w_k[M]$ до нуля, т. е. вычитая $\lambda_k = \omega_k - \omega_{k-1}$ из $w_k[i]$.

Теорема 14. *Если существует такое $j \in M$ и такое $\varepsilon > 0$, что для всех $u \in N$ справедливо неравенство $p_u[j] \geqslant \varepsilon$, то последовательность $\{\lambda_k\}$ сходится к λ, а последовательность $\{w_k\}$ — к функции $v[M]$, удовлетворяющей уравнению* (58).

Доказательство. Пусть

$$\rho(x[M], y[M]) = \max\{x[i] - y[i] \mid i \in M\} - \min\{x[i] - y[i] \mid i \in M\}.$$

Функция $\rho(x, y)$ обладает тем свойством, что если ко всем компонентам $x[M]$ или $y[M]$ прибавить одно и то же число, значение ρ не изменится. Из $\rho(x, y) = 0$ следует $x[i] - y[i] = \text{const}$. Так как

$$v_{k+1}[i] - v_k[i] \leqslant \varepsilon \times (v_k[j] - v_{k-1}[j]) + (1 - \varepsilon) \times \min\{v_k[r] - v_{k-1}[r]\}$$

и

$$v_{k+1}[i] - v_k[i] \geqslant \varepsilon \times (v_k[j] - v_{k-1}[j]) + (1 - \varepsilon) \times \max\{v_k[r] - v_{k-1}[r]\},$$

то

$$\rho(v_{k+1}[M], v_k[M]) \leqslant (1 - \varepsilon) \times \rho(v_k[M], v_{k-1}[M])$$

и, следовательно,

$$\rho(v_{k+1}, v_k) \leqslant (1 - \varepsilon)^k \times \rho(v_1, v_0).$$

Отсюда уже без труда следуют утверждения теоремы. ▲

БИБЛИОГРАФИЧЕСКИЕ УКАЗАНИЯ

К предисловию

Назовем здесь литературу по смежным вопросам, не вошедшим в эту книгу. По выпуклому программированию накопилась уже целая библиотека — книги А. Д. Иоффе и В. М. Тихомирова [70], И. В. Гирсанова [41], Б. Н. Пшеничного [128], В. Г. Карманова [80], У. И. Зангвилла [64] и др. Геометрические вопросы выпуклого программирования излагаются в книгах Р. Т. Рокафеллара [130], Х. Никайдо [116], С. С. Кутателадзе и А. М. Рубинова [92].

По теории оптимальных процессов могут быть названы книги Л. С. Понтрягина и др. [125], В. Г. Болтянского [20], Л. Янга [167].

Численным методом для непрерывных задач и оптимальных процессов посвящены книги Н. Н. Моисеева [112], В. Ф. Демьянова и А. М. Рубинова [56], Б. Н. Пшеничного и Ю. М. Данилина [129], Э. Полака [122].

По экономическим моделям можно назвать лишь книги, посвященные различным проблемам моделирования. Так, общие принципы моделирования и использования линейного программирования в экономических моделях освещены в книге Л. В. Канторовича [75]. Математические вопросы анализа моделей рассматриваются в книгах Д. Гейла [40], М. Моришимы [113]. Обширный круг производственных моделей рассмотрен А. А. Первозванским [118]. Интересна и разнообразна по тематике книга С. Карлина [79]. Другие ссылки будут даны позднее при рассмотрении отдельных задач.

По стохастическому программированию можно рекомендовать книги Ю. М. Ермольева [62] и Д. Б. Юдина [164]. Задачи линейного программирования большого объема рассматривались в книге Л. С. Лэсдона [101]. Вопросы регуляризации оптимизационных процедур хорошо освещаются в книге А. Н. Тихонова и В. Я. Арсенина [147]. О методах отсечения достаточно полная информация содержится в книге А. А. Корбута и Ю. Ю. Финкельштейна [86] и в недавно переведенной книге Т. С. Ху [159].

Некоторые информационные задачи и методы работы со сложной информацией описаны С. С. Лавровым и Л. И. Гончаровой [94].

К главе 1

§ 1. Описанную здесь систему обозначений мы систематически используем, начиная с 1970 г. Гораздо раньше подобные обозначения встречались у К. Бержа [175], но Берж использовал традиционное написание индексов, которое представляется неудобным при многократной индексации.

§ 2. По языку алгол-60 имеется уже много первоклассных руководств. Назовем здесь книги С. С. Лаврова [93] и А. Л. Брудно [24]. При некотором навыке работы с этим языком обязательным руководством ставится его официальное описание [5]. Кстати, в книге [24] детально описано «бумажное» выполнение алгольных программ.

Идею выделения состояния вычислительного процесса мы почерпнули у А. А. Маркова [107].

§ 3. Языки, использующие структурные записи, — это прежде всего Кобол, Симула [53], из более современных ПЛ/1 [57], Паскаль [232] и используемый нами здесь алгол-68 [6, 98, 119]. Расширение алгола-60, позволяющее работать с записями, предложено К. Хоором [158].

Применительно к фортрану работа со структурированными данными описана в книге А. Берзтисса [19].

§ 4. Недавно была переведена небольшая и простая книга Дж. Фостера [153], в которой описываются основные операции над списками и связанные с ними понятия. Дж. Айлиф [4] обсуждает эти операции в связи с рассмотрением подходов к построению вычислительных машин.

Что касается алгола-68, то мы отсылаем читателя к книге [98], хотя имеющийся в ней материал несколько устарел (в книге есть предупреждение о возможных изменениях). Мы следуем изменениям основного текста [119].

К главе 2

§ 1. Из книг — руководств по линейному программированию следует предже всего назвать книги Дж. Данцига [54], Д. Б. Юдина и Е. Г. Гольштейна [165], М. Симонара [229]. Четкое сжатое изложение необходимых сведений имеется в первой части книги К. Бержа и А. Гуйла-Ури [177]. Вычислительному аспекту линейного программирования посвящена книга В. Орчард-Хейса [221].

§ 2. Термины «стандартная» и «каноническая» применительно к задачам линейного программирования систематически перепутываются. Так, в русском переводе книги Т. Ху [159] и в ее английском оригинале они введены в точности противоположным образом.

Рассмотрение эквивалентных преобразований часто встречается в руководствах по оптимальному программированию, начиная с обзорной статьи А. Дж. Голдмана и А. У. Таккера [42]. Нам представляется очень важным систематическое использование понятия «расширение экстремальной задачи».

§ 3. В сборнике [99] имеется несколько превосходных статей, посвященных истолкованию двойственности с точки зрения геометрии и линейной алгебры.

Интересное простое доказательство теоремы двойственности есть в учебнике А. Г. Пинскера и Э. Ф. Брызжиной [121].

§ 4. Вопрос о крайних точках выпуклых множеств, — естественно, геометрический. Мы отсылаем читателя за большими подробностями к сборнику [99], к книгам С. Карлина [79] и Рокафеллара [130].

Чрезвычайно интересен вопрос о крайних точках выпуклых множеств в функциональных пространствах. По этому поводу, кроме [70, 116], можно рекомендовать книги М. Г. Крейна и А. А. Нудельмана [90], К. Иосиды [69], Р. Фелпса [151], В. Н. Судакова [143] и обзорную статью В. И. Аркина и В. Л. Левина [12].

Принятая сейчас схема изложения метода последовательных улучшений в терминах линейных систем восходит к работе Л. В. Канторовича [72].

Из методов, не использующих базисных решений, можно рекомендовать методы В. А. Булавского [28], Л. М. Брэгмана [26], Э. П. Борисовой [21], Б. И. Коробочкина [88], Б. Т. Поляка и Н. В. Третьякова [123] Альтернатива методам линейного программирования предлагается и в уже называвшейся книге Н. Н. Моисеева [112].

§ 5. Подробная библиография по устранению зацикливания и примеры зацикливающихся задач имеются у Данцига [54].

§ 6. К настоящему времени издано уже много алгоритмов линейного программирования [13, 49, 87, 105, 127], обзоров по программам [106, 210, 212], описаний пакетов и специальных языков для решения задач линейного программирования [27, 45].

Сравнительно редко в руководствах встречается мультипликативный метод. Следует назвать здесь книгу Г. Зойтендейка [65], в которой этот метод был изложен впервые, а также Л. Лэсдона [101].

LU-представление и его использование описаны в работах Р. Х. Бартельса и Г. Х. Голуба [172], Дж. Форреста и Дж. Томлина [193, 231].

LT-представление предложено В. А. Булавским [30]. Программа на алголе-60 для этого метода опубликована в [142].

Методы, использующие особую структуру обратной матрицы в блочных задачах, развивались новосибирской школой. Подробную библиографию можно найти в [31].

Метод генерирования столбцов уже давно культивируется в советских работах. Отметим работу Л. В. Канторовича и В. А. Залгаллера ([77], 1-е изд. 1951), в которой даны различные постановки задачи раскроя и обширная литература по этому вопросу. В работе [9] тот же метод был предложен для решения задачи маршрутизации автомобильных перевозок (см. также [10]). По существу методом генерирования столбцов является метод Келли для задач выпуклого программирования ([207], см. также [80]). Метод разложения Данцига — Вулфа изложен в [54] (см. также [44]). Некоторые новые приложения метода генерирования столбцов описаны в [8].

К главе 3

§ 1. В этой главе и последующем изложении существенно использованы книги К. Бержа [18] и [177]. Для более подробного ознакомления с теорией графов и различными ее приложениями можно рекомендовать книги [15, 66, 84, 117].

Имеется несколько книг специально по задачам транспортного типа, из которых мы отметим особо книги Е. Г. Гольштейна и Д. Б Юдина [44], Ю. М. Ермольева и И. М. Мельника [63], Е. П. Нестерова [115], Е. Б. Триуса [149].

§ 3 и 4. Здесь мы следуем [18]. Много интересных задач, связанных с перечислениями деревьев, можно найти, например, у Басакера и Саати [15].

§ 5. Первая известная постановка транспортной задачи принадлежит советскому экономисту А. Толстому [148]. Первые математические исследования Л. В. Канторовича и М. К. Гавурина [76] касались именно сетевой постановки задачи и интересного непрерывного ее аналога

[73], который был предложен еще Г. Монжем [217] и исследовался П. Аппелем [168] и Л. В. Канторовичем [74]. Дальнейшие исследования, связанные с непрерывным аналогом транспортной задачи, см. в книге В. Фридриха [195]. Интересный метод приближенного решения транспортной задачи был предложен Ю. Шацким и Г. В. Шелейховским. Математическое изложение этого метода и доказательство сходимости имеется в [25].

§ 6. Здесь мы также почти полностью следуем [18].

§ 7 и 8. Название «потенциал» в данном контексте было предложено Л. В. Канторовичем [76, 73]. Идея использования подходящей нумерации элементов базиса предлагалась А. Л. Брудно [22], а затем В. А. Булавским [29]. Дальнейшее совершенствование метода потенциалов проводилось, например, в работах К. В. Кима [85] и В. И. Шмырева [162].

§ 9. О задаче с двусторонними ограничениями см. [44, 138, 131]. Задача о замкнутом потоке рассматривалась в [177] и в [135].

Двухкомпонентной задачей и ее вариантом — так называемой λ-задачей (или обобщенной транспортной задачей) — занимались многие [39, 85, 104, 138, 162, 166].

К главе 4

§ 1. Поиск контура в графе — обычная процедура сетевого планирования [32, 120]. Интересное применение имеет эта процедура в задачах кодирования [58].

§ 2. Алгоритм Беллмана — Шимбела описан в их статьях [174, 228], алгоритм Флойда в работе [192]. Другой вариант матричного метода, принадлежащий Г. Данцигу [184], приведен в интересном обзоре С. Дрейфуса [188]. См. также недавнюю статью [202].

Задачей о пути с максимальной пропускной способностью занимался М. Поллак [223].

§ 3. По-видимому, первая работа о кратчайших путях в лабиринте была выполнена Э. Муром [218]. Обзоры работ в этой области см. в [188, 224]. Метод Дейкстры [186], следуя обзору [224], часто называли методом Минти; некоторые другие эффективные реализации этого метода см. в [10]. Отметим еще метод Левита [97], который также оказался весьма эффективным в вычислительном отношении.

§ 4. Кроме работ [32, 120], можно назвать еще руководство [89]. Специфическим вопросам решения больших задач посвящены работы [160, 111].

В работе [135] рассмотрен циклический вариант модели сетевого графика.

§ 5. Исходными здесь являются работы Краскала [209] и Прима [126], см. также [186].

Задача о кратчайшем иерархическом дереве рассматривалась в [197] (там же указано еще несколько предшествовавших работ).

Задача о кратчайших частичных деревьях для некоторых специальных случаев рассматривалась в книге И. Моцкуса [114], где имеется обширная библиография, и в работах С. Ю. Рудермана [139] и др.

§ 6. Многочисленные работы Л. Форда и Д. Фалкерсона в этой области собраны в их книге [152]. Из более поздних работ см. [2, 201].

Многопродуктовыми потоками на сети занимается Т. Ху [159]. См. также [2].

§ 7. В качестве первоисточника здесь следует назвать работы Д. Фалкерсона [196] и Дж. Келли [82].

Основные сведения о параметрических задачах линейного программирования можно найти в [43], см. также недавно вышедшую монографию [220].

Как вариант схемы распределения ресурсов можно упомянуть еще нелинейную задачу распределения ресурсов [183, 103]. Задача распределения ресурсов в циклическом варианте модели сетевого графика рассмотрена в [136].

§ 8. Исходная статья Дилворта [187], первое применение линейного программирования, по-видимому, в [55]. Другое изложение в [157].

§ 9. По-видимому, наилучшее изложение теории паросочетаний — в книге Бержа [18], где имеется вся необходимая библиография.

Многомерные обобщения задачи о назначениях рассматривались в [222, 68].

Имеется интересный обзор проблематики, связанной с бистохастическими матрицами, принадлежащий Л. Мирскому [216] (см. также [108]).

§ 10. Название «венгерский метод» появилось в статье Г. Куна [91], который ссылался на статью Эгервари [189]. Дальнейшее развитие метода проводилось в работах [219, 48].

§ 11. По поводу «систем различных представителей» имеется обширная литература [46, 225]. Основные факты, относящиеся к гиперграфам, собраны в книге К. Бержа [176] и в обзоре А. А. Зыкова [67], топологические эквиваленты этих понятий вводятся в курсах по комбинаторной топологии [7, 124].

Задачи линейного программирования с матрицами из нулей и единиц рассматривались В. А. Трубиным [150], а также Э. Балашем [169].

Задача о порядке исключения переменных в связи с вычислительной реализацией мультипликативного метода рассматривалась в [11, 145, 227, 230].

Интересные задачи редукции комплекса рассматривал Н. Н. Воробьев в связи с вопросом о продолжимости меры с данным набором проекций [36, 37].

К главе 5

§ 1. Из многочисленных характеристических чисел графа мы выбрали наиболее интересные, следуя [18]. Интересные экстремальные задачи другого типа возникают в связи с проблемами нумерации вершин графа [191].

§ 2. В настоящее время литература по задаче о коммивояжере так велика, что целесообразно упомянуть лишь обзор [38] и последние работы [182, 200].

§ 3. Метод ветвей и границ был описан А. Лэнд и А. Дойг [211] и А. Л. Брудно [23] и получил широкое распространение после появления работы [100]. См. обзоры [133, 154, 170, 198, 213].

Принцип сравнения ростков дерева развивался киевской школой, см. обзор [110].

§ 4. Задача о размыкании контуров излагается по [132]. Постановку задачи о круговом расположении станков см. в [1], другие варианты в [146].

§ 6. См. [77].

§ 7. Многие ссылки на методы решения бивалентных задач имеются в обзорах [169] и [199]. См. также [50, 96, 135, 180].

§ 8. Задачи размещения производства — важный и трудный класс многоэкстремальных задач. Здесь приведен лишь простейший вариант этой задачи, чтобы продемонстрировать еще одну эффективную вычислительную схему (обзор белорусской школы в этом направлении имеется у В. А. Емеличева [60]). Другие постановки задач размещения и обзор методов см. в [52], библиографию иностранных работ в [194].

§ 9. См. [81, 137, 208].

К главе 6

§ 3. Исходные варианты схемы динамического программирования были разработаны Р. Беллманом [16]. Затем существенное развитие эти схемы получили в связи с теорией автоматов, см. [163, 190, 204, 206].

§ 4. См. [16].

§ 5. Здесь мы следуем работе [133]. По поводу теорем о магистрали для экономических процессов см. книги [113] и [102] и статью [214].

§ 7. См. [14].

§ 8. По поводу задач поиска неисправности см. [95]. Приводимая здесь программа написана М. И. Фрейманом.

§ 9. Мы следуем схеме, разработанной А. Б. Грибовым [47].

§ 11. Многочисленные работы по этой тематике собраны в монографии Бертеле и Бриоши [178].

К главе 7

§ 1. Литература по марковским цепям многочисленна и разнообразна. В качестве достаточно простого источника назовем книгу [83].

§ 2. Теория потенциалов для конечных марковских цепей излагается по статье Г. Шоке и Ж. Дени [181].

Современное изложение теории потенциала в ее теоретико-вероятностной трактовке см. в [109].

§ 3. Управление запасами — тема, чрезвычайно популярная. Из недавней литературы можно рекомендовать книги [155, 141], а также [34, т. II].

§ 4. Детальное изложение теории марковских и полумарковских процессов решения имеется в книгах [203, 185, 215]. Для начального ознакомления можно рекомендовать [156, 17, 34].

Понятие «решающая функция» восходит к А. Вальду [35], современное изложение статистического последовательного анализа, имеющего много общего с теорией управляемых марковских процессов см. в книге А. Н. Ширяева [161].

§ 5. См. [226].

§ 6. См. [133].

Двойные системы этого параграфа вводились в [185, 215].

§ 7. Книги Ховарда упоминались [156, 203]. Теорема Д. Блекуэлла доказана в [179], ссылки на другие работы Блекуэлла имеются в [59].

ЛИТЕРАТУРА

1. Абрамов Л. М., Голова В. П., Горькова К. А., Алгоритм решения задачи об оптимальном планировании оборудования.— В кн.: «Прим. матем. методов в экономике», вып. 4, Изд-во ЛГУ, 1967, 10—20.
2. Адельсон-Вельский Г. М., Диниц Е. А., Карзанов А. В., Потоковые алгоритмы. М., «Наука», 1975, 119 с.
3. Адельсон-Вельский Г. М., Фидлер Ф. М., Программа вычисления сетевых графиков. — ЖВМ и МФ, 1965, 5, 1, 144—148.
4. Айлиф Дж., Принципы построения базовой машины. М., «Мир», 1973, 120 с.
5. Алгоритмический язык Алгол-60, пересмотренное сообщение, пер. с англ. М., «Мир», 1965, 79 с.
6. Алгоритмический язык Алгол-68. — Кибернетика, 1969, 6, 17—144; 1970, 1, 13—160.
7. Александров П. С., Комбинаторная топология. М.— Л., 1947, 660 с.
8. Алексеев А. М., Волконский В. А., Шапиро А. Д., Методы оптимизации планов путей автоматического формирования плановых вариантов и их применение. — Экономика и матем. методы, 1973, 9, 1, 3—18.
9. Аникеич А. А., Грибов А. Б., Романовский И. В., Об одной задаче маршрутизации перевозок. — Кибернетика, 1965, 1.
10. Аникеич А. А., Грибов А. Б., Сурин С. С., Сменно-суточное планирование работы грузовых автомобилей на ЭВМ. М., «Транспорт», 1976, 152 с.
11. Анисимов-Спиридонов Д. Д., Метод линейных ветвлений и некоторые его применения в оптимальном планировании. М., «Наука», 1964, 119 с.
12. Аркин В. И., Левин В. Л., Выпуклость значений векторных интегралов, теоремы измеримого выбора и вариационные задачи. — УМН, 27, 3, 1972, 21—77.
13. Ахметов П. А., Программа мультипликативного алгоритма симплексного метода для решения общих задач линейного программирования. — Стандартные программы. М., 1967 (ЦЭМИ).
14. Барлоу Р., Прошан Ф., Математическая теория надежности. М., «Сов. радио», 1969, 488 с.
15. Басакер Р., Саати Г., Конечно графы и сети. М., «Наука», 1974 368 с.
16. Беллман Р., Динамическое программирование. М., ИЛ, 1960.
17. Беллман Р., Дрейфус С., Прикладные задачи динамического программирования. М., «Наука», 1965, 458 с.
18. Берж К., Теория графов и ее применения. М., ИЛ, 1962, 319 с.

19. Берзтисс А. Т., Структуры данных. М., «Статистика», 1974.
20. Болтянский В. Г., Оптимальное управление дискретными системами. М., «Наука», 1973, 446 с.
21. Борисова Э. П., Итеративные программы системы «Симплекс». — Программы и алгоритмы, вып. 38, ЦЭМИ, М., 1971, 99 с.
22. Брудно А. Л., Решение транспортной задачи методом вычеркивающей нумерации. — В кн. :«Применение ЦВМ в экономике». М., Изд-во АН СССР, 1962, 17—38.
23. Брудно А. Л., Грани и оценки для сокращения перебора вариантов. — Проблемы кибернетики, 1963, вып. 10, 141—150.
24. Брудно А. Л., Алгол, изд. 2-е. М., «Наука», 1971, 80 с.
25. Брэгман Л. М., Доказательство сходимости метода Г. В. Шелейховского для задачи с транспортными ограничениями. — ЖВМ и МФ, 1967, 7, 1, 147—156.
26. Брэгман Л. М., Релаксационный метод нахождения общей точки выпуклых множеств и его применение для решения задач выпуклого программирования. — ЖВМ и МФ, 1967, 7, 3, 620—631.
27. Брэгман Л. М., и др., Пакет ЛП-ЛГУ для решения задач линейного программирования. — В кн.: «Исследование операций и статистическое моделирование, 3». Л., Изд-во ЛГУ, 1975, 3—64.
28. Булавский В. А., Итеративный метод решения задачи линейного программирования. — Сиб. матем. ж., 1962, 3, 3, 313—332.
29. Булавский В. А., Об одном алгоритме решения транспортной задачи. — Оптимальное планирование, 1964, 2, 41—49.
30. Булавский В. А., Метод ортогонализации в линейном программировании. — Оптимизация, 1971, 1 (18), 134—157.
31. Булавский В. А., Звягина Р. А., Каплан А. А., Шмырев В. И., Об исследованиях по численным методам математического программирования. — Оптимизация, 1974, 13 (30).
32. Бурков В. Н., Ланда Б. Д., Ловецкий С. Б. и др., Сетевые модели и задачи управления. М., «Сов. радио», 1967, 144 с.
33. Бурков В. Н., Ловецкий С. Б., Методы решения экстремальных комбинаторных задач. — Изв. АН СССР. Техн. кибернетика, 1968, 4, 82—93.
34. Вагнер Г., Основы исследования операций, т. 1—3. М., «Мир», 1973.
35. Вальд А., Статистические решающие функции. — В кн.: «Позиционные игры». М., «Наука», 1967, 300—544.
36. Воробьев Н. Н., Согласованные семейства мер и их продолжения. — Теор. вероятн. и ее прим., 1962, 7, 2, 153—169.
37. Воробьев Н. Н., Марковские меры и марковские продолжения. Теор. вероятн. и ее прим., 1963, 8, 4, 451—462.
38. Габович К., Задача коммивояжера, I. — Тр. ВЦ Тартуского ун-та, 1970, вып. 19, 52—96.
39. Гавурин М. К., Рубинштейн Г. Ш., Сурин С. С., Об оптимальном использовании средств при выполнении нескольких видов работ. — Сиб. матем. ж., 1962, 3, 4, 481—499.
40. Гейл Д., Теория линейных экономических моделей. М., ИЛ, 1963, 418 с.
41. Гирсанов И. В., Математическая теория экстремальных задач. М., Изд-во МГУ, 1970, 118 с.
42. Голдман А. Дж., Таккер А. У., Теория линейного программирования. — В кн. [99], 172—213.

43. Гольштейн Е. Г., Юдин Д. Б., Новые направления в линейном программировании. М., «Сов. радио», 1966, 524 с.
44. Гольштейн Е. Г., Юдин Д. Б., Залачи линейного программирования транспортного типа. М., «Наука», 1969, 384 с.
45. Гончарова Л. И., Станевичус А.-И. А. ЛП-язык. — Алгоритмы и алгоритмические языки, вып. 5, М., Изд-во ВЦ АН СССР, 1971, 94—104.
46. Гофман А. Дж., Кун Г. У., О системах различных представителей. — В кн. [99], 302—310.
47. Грибов А. Б., Алгоритм решения задачи плоского раскроя. — Кибернетика, 1973, **6**, 110—115.
48. Грибов А. Б., Программа для решения задачи о назначениях. — Алгол-процедуры, Изд-во ЛГУ, вып. 14, 10—11.
49. Грибов А. Б., Романовский И. В., Программирование симплекс-метода и его вариантов на Алголе. — Оптимальное планирование, 1969, вып. 12, 5—27.
50. Гришухин В. П., Оценка сложности алгоритма Балаша.— В кн.: «Математические методы решения экстремальных задач», М., «Наука», 1972, 93—105.
51. Гурин Л. С., Дымарский Я. С., Меркулов А. Д., Задачи и методы оптимального распределения ресурсов, М., «Сов. радио», 1968, 458 с.
52. Давыдова И. М., Задачи размещения. — В кн.: «Исследование операций и статистическое моделирование, 4», Л., Изд-во ЛГУ, 1977.
53. Дал У.-И., Мюрхауг Б., Нюгорд К., Симула 67, универсальный язык программирования. М., «Мир», 1969, 100 с.
54. Данциг Дж. Б., Линейное программирование, его обобщение и применение. М., «Прогресс», 1966, 600 с.
55. Данциг Дж. Б., Гофман А. Дж., Теорема Дилворта о частично упорядоченных множествах. — В кн. [99], 311—317.
56. Демьянов В. Ф., Рубинов А. М., Приближенные методы решения экстремальных задач. Л., Изд-во ЛГУ, 1968, 180 с.
57. Джермейн К., Программирование на IBM/360. М., «Мир», 1971, 870 с.
58. Дискретная математика и математические вопросы кибернетики т. I, под общ. ред. С. В. Яблонского и О. Б. Лупанова. М., «Наука», 1974, 312 с.
59. Еремин И. И., Астафьев Н. Н., Введение в теорию линейногои выпуклого програмирования. М., «Наука», 1976, 192 с.
60. Емеличев В. А., Дискретная оптимизация. Последовательные схемы решения, I—II. — Кибернетика **6**, 1971, 109—129; **2**, 1972.
61. Еремин И. И., Мазуров Вл. Д., О нестационарных процессах математического программирования, — В кн.: «Нестационарные процессы математического программирования», Труды Ин-та математики и механики УНЦ АН СССР, вып. 14, Свердловск, 1974.
62. Ермольев Ю. М., Методы стохастического программирования. М., «Наука», 1976, 240 с.
63. Ермольев Ю. М., Мельник И. М., Экстремальные задачи на графах. Киев, «Наукова думка», 1968, 176 с.
64. Зангвилл У. И., Нелинейное программирование. Единый подход. М., «Сов. радио», 1973, 311 с.

65. Зойтендейк Г., Методы возможных направлений. М., ИЛ, 1963, 176 с.
66. Зыков А. А., Теория конечных графов, I. Новосибирск, «Наука», 1969, 543 с.
67. Зыков А. А., Гиперграфы. — УМН, 1974, **29**, 6, 89—154.
68. Иванов В. В., О трехиндексной задаче о назначениях. — Экономика и матем. методы, 1974, **10**, 2, 336.
69. Иосида К., Функциональный анализ. М., «Мир», 1967, 619 с.
70. Иоффе А. Д., Тихомиров В. М., Теория экстремальных задач. М., «Наука», 1974, 479 с.
71. Исследования по дискретной оптимизации. Сб. статей под ред. А. А. Фридмана. М., «Наука», 1976, 446 с.
72. Канторович Л. В., Математические методы в организации и планировании производства. Л., Изд-во ЛГУ, 1939, 67 с.
73. Канторович Л. В., О перемещении масс. — ДАН СССР, 1942, **37**, 7, 227—229.
74. Канторович Л. В., Об одной задаче Монжа. УМН, 1948, **3**, 2.
75. Канторович Л. В., Экономический расчет наилучшего использования ресурсов. М., Изд-во АН СССР, 1959, 343 с.
76. Канторович Л. В., Гавурин М. К., Применение математических методов в вопросах анализа грузопотоков. — В кн.: «Проблемы повышения эффективности работы транспорта», М., Изд-во АН СССР, 1949, 110—138.
77. Канторович Л. В., Залгаллер В. А., Рациональный раскрой промышленных материалов. Новосибирск, «Наука», 1971.
78. Канторович Л. В., Рубинштейн Г. Ш., Об одном функциональном пространстве и некоторых экстремальных задачах. — ДАН СССР, 1957, **115**, 6, 1058—1061.
79. Карлин С., Математические методы в теории игр, программировании и экономике. М., «Мир», 1964, 839 с.
80. Карманов В. Г., Математическое программирование. М., «Наука», 1975, 272 с.
81. Кацев С. Б., Романовский И. В., Две задачи целочисленного программирования. Задача о распределении заданий. — Алгол-процедуры. Изд-во ЛГУ, 1975, 13, 18—23.
82. Келли Дж., Метод критического пути: распределение ресурсов и составление календарного плана работ: — В кн.: «Календарное планирование». М., 1963, 398—420.
83. Кемени Дж., Снелл Дж., Конечные цепи Маркова. М., «Наука», 1970, 272 с.
84. Кемени Дж., Снелл Дж., Томпсон Дж., Введение в конечную математику. М., ИЛ, 1963, 486 с.
85. Ким К. В., Об эффективности алгоритмов решения двухкомпонентных задач линейного программирования. — Экономика и матем. методы, 1974, **10**, 3, 621—631.
86. Корбут А. А., Финкельштейн Ю. Ю., Дискретное программирование. М., «Наука», 1969, 368 с.
87. Корнилова Г. Ф., Глезер Н. Н., Программное обеспечение экономико-математического анализа. — Программы и алгоритмы, вып. 11, ЦЭМИ, М., 1968.
88. Коробочкин Б. И., Обобщенные функции Лагранжа и некоторые их свойства. — Оптимизация, 1973, 10 (27), 120—127.

89. Кофман А., Дебазей Г., Сетевые методы планирования и их применение. М., «Прогресс», 1968, 182 с.
90. Крейн М. Г., Нудельман А. А., Проблема моментов Маркова и экстремальные задачи. М., «Наука», 1973, 546 с.
91. Кун Г., Венгерский метод решения задачи о назначениях. — В кн.: «Методы и алгоритмы решения транспортной задачи», М., Госстатиздат, 1963.
92. Кутателадзе С. С., Рубинов А. М., Двойственность Минковского и ее приложения. Новосибирск, «Наука», 1976, 254 с.
93. Лавров С. С., Универсальный язык программирования (Алгол-60), изд. 3-е. М., «Наука», 1972, 184 с.
94. Лавров С. С., Гончарова Л. И., Автоматическая обработка данных, хранение информации в памяти ЭВМ. М., «Наука», 1971, 160 с.
95. Латинский С. М., Шарапов В. И., Ксёнз С. П., Афанасьев С. С., Теория и практика эксплуатации радиолокационных систем. М., «Сов. радио», 1970, 432 с.
96. Лебедев С. С., Целочисленное программирование и множители Лагранжа. — Экономика и матем. методы, 1974, **10**, 3, 592—610.
97. Левит Б. Ю., Лившиц В. Н., Нелинейные сетевые транспортные задачи. М., «Транспорт», 1972, 144 с.
98. Линдси Ч., Мюйлен С. ван дер, Неформальное введение в Алгол-68. М., «Мир», 1973, 407 с.
99. Линейные неравенства и смежные вопросы. Сб. статей под ред. Г. У. Куна и А. У. Таккера. М., ИЛ, 1959, 469 с.
100. Литл Дж., Мурти К. Г., Суини Д., Кэрел К., Алгоритм для решения задачи о коммивояжере. — Экономика и матем. методы, 1965, **1**, 1, 94—107.
101. Лэсдон Л. С., Оптимизация больших систем. М., «Наука», 1975, 432 с.
102. Макаров В. Л., Рубинов А. М., Математическая теория экономической динамики и равновесия. М., «Наука», 1973, 336 с.
103. Малинников В. В., Поварков а Л. Д., Соболев А. И., Об одной экстремальной задаче, связанной с моделью PERT. — Тр. Ленинградского инж. эконом. ин-та, 1966, вып. 58, 229—232.
104. Малков У. Х., Алгоритм решения распределительной задачи. — ЖВМ и МФ, 1962, **2**, 2, 358—366.
105. Малков У. Х., Алгоритмы для решения задач линейного программирования на Альфа-языке. Сб. статей. М., 1968, 141 с. (ЦЭМИ АН СССР).
106. Малков У. Х., Обзор программ решения общей задачи линейного программирования. — Экономика и матем. методы, 1969, **5**, 4, 594—597.
107. Марков А. А., Теория алгорифмов. Тр. МИАН, 1954, 42, 376 с.
108. Маркус М., Минк Х., Обзор по теории матриц и матричных неравенств. М., «Наука», 1972, 232 с.
109. Мейер П.-А., Вероятность и потенциалы. М., «Мир», 1973, 324 с.
110. Михалевич В. С., Ермольев Ю. М., Шкурба В. В., Шор Н. З., Сложные системы и решение экстремальных задач. — Кибернетика, 1967, 5, 29—39.
111. Михельсон В. С., Козус В. М., Климкович И. И., Фрагментарный метод исследования сетевых моделей. — В кн.: «Частные задачи автоматики и систем управления», Минск, 1970.

112. Моисеев Н. Н., Численные методы в теории оптимальных систем. М., «Наука», 1971, 424 с.
113. Мориш има М., Равновесие, устойчивость, рост. М., «Наука», 1972, 280 с.
114. Моцкус И. Б., Многоэкстремальные задачи в проектировании. М., «Наука», 1967, 215 с.
115. Нестеров Е. П., Транспортные задачи линейного программирования, 2-е изд. М., «Транспорт», 1971, 216 с.
116. Никайдо Х., Выпуклые структуры и математическая экономика. М., «Мир», 1972, 517.
117. Оре О., Теория графов. М., «Наука», 1968, 352 с.
118. Первозванский А. А., Математические модели в управлении производством. М., «Наука», 1975, 616 с.
119. Пересмотренное сообщение об Алголе-68. М., «Мир», 1976.
120. Петрова Л. Т., Введение в сетевое планирование. Новосибирск, Изд-во НГУ, 1969, 152 с.
121. Пинскер А. Г., Брызжина Э. Ф., Элементы оптимального программирования. Л., Изд-во ЛГУ, 1974, 188 с.
122. Полак Э., Численные методы оптимизации. Единый подход. М., «Мир», 1974, 374 с.
123. Поляк Б. Т., Третьяков Н. В., Метод штрафных оценок для задач на условный экстремум. — ЖВМ и МФ, 1973, **13**, 1.
124. Понтрягин Л. С., Основы комбинаторной топологии. М.—Л., 1947, 142 с.
125. Понтрягин Л. С., Болтянский В. Г., Гамкрелидзе Р. В., Мищенко Е. Ф., Математическая теория оптимальных процессов. М., Физматгиз, 1961, 391 с.
126. Прим Р. К., Кратчайшие связывающие сети и некоторые обобщения. — Киберн. сб., 1961, 2, 95—107.
127. Программы симплекс-метода на языке фортран (метод, разработка). М., Изд-во ВЦ МГУ, 1976, 54 с.
128. Пшеничный Б. Н., Необходимые условия экстремума. М., «Наука», 1969, 152 с.
129. Пшеничный Б. Н., Данилин Ю. М., Численные методы в экстремальных задачах. М., «Наука», 1974, 320 с.
130. Рокафеллар Р. Т., Выпуклый анализ. «Мир», 1973,469 с.
131. Романовский И. В., Об эквивалентности различных постановок транспортной задачи. — УМН, 1962, **17**, 3, 193—195.
132. Романовский И. В., Задача о наивыгоднейшей круговой расстановке станков. — Экономика и матем. методы, 1966, **2**, 4, 578—581.
133. Романовский И. В., Магистральные теоремы для полумарковских процессов решения. Тр. МИАН, 1970, 111, 208—223.
134. Романовский И. В., Методы неявного перебора для решения задач целочисленного программирования с бивалентными переменными. — Изв. вузов, Математика, 1970, 4, 17—29.
135. Романовский И. В., Оптимизация стационарного управления дискретным детерминированным процессом. — Кибернетика, 1967, 2, 66—78.
136. Романовский И. В., Циклические варианты моделей сетевого графика. — В кн.: «Исследование операций и статистическое моделирование, 1», Л., Изд-во ЛГУ, 1972, 145—152.

137. Романовский И. В., Христова Н. П., Решение дискретных минимаксных задач методом дихотомии. — ЖВМ и МФ, 1973, 13, 5, 1200—1209.
138. Рубинштейн Г. Ш., О решении задач линейного программирования большого объема. — Оптим. планирование, 1964, 2, 3—22.
139. Рудерман С. Ю., Алгоритм ускоренного поиска оптимальной трассы трубопровода с отводами при аддитивном показателе качества. Транспорт и хранение нефти и нефтепродуктов, 1972, 8.
140. Рыжиков Ю. И., Управление запасами. М., «Наука», 1969, 344 с.
141. Рыков В. В., Управление системы массового обслуживания. — В кн.: «Итоги науки и техники. Теория вероятностей. Матем. статистика. Теор. кибернетика, 12», М., ВИНИТИ, 1975, 43—154.
142. Сорокина М. Г., Симплекс-метод В. А. Булавского. — Алгол-процедуры, Изд-во ЛГУ, 1970, 4, 1—5.
143. Судаков В. Н., Геометрические проблемы теории бесконечномерных вероятностных распределений. — Тр. МИАН, 1976, 141, Л., «Наука», 192 с.
144. Супруненко Д. А., Айзенштат В. С., Лепешинский Н. А., Экстремальные значения функций на множествах перестановок. — Тезисы докл. на I Всесоюзн. конф. по исслед. операций. Минск, 1972, 61—64.
145. Сурин С. С., Решение задач линейного программирования с большим числом нулевых элементов в исходной симплексной таблице. — Оптимальное планирование, 1968, 10, 69—80.
146. Танаев В. С., Шкурба В. В., Введение в теорию расписаний. М., «Наука», 1975, 256 с.
147. Тихонов А. Н., Арсенин В. Я., Методы решения некорректных задач. М., «Наука», 1974, 224 с.
148. Толстой А., Методы устранения нерациональных перевозок при планировании. — Соц. транспорт, 1939, 9, 28—51.
149. Триус Е. Б., Задачи математического программирования транспортного типа. М., «Сов. радио», 1967, 208 с.
150. Трубин В. А., О методе решения задач целочисленного линейного программирования специального вида. — ДАН СССР, 1969, 189, 5, 552—554.
151. Фелпс Р., Лекции о теоремах Шоке. М., «Мир», 1968, 112 с.
152. Форд Л. Р., Фалкерсон Д. Р., Потоки в сетях. М., «Мир», 1966 276 с.
153. Фостер Дж., Обработка списков. М., «Мир», 1974, 72 с.
154. Фридман А. А., Войтяков А. А., Дискретные задачи и метод ветвей и границ. Экономика и матем. методы, 1974, 10, 3.
155. Хедли Дж., Уайтин Т., Анализ систем управления запасами. М., «Наука», 1969, 512 с.
156. Ховард Р., Динамическое программирование и марковские процессы. М., «Сов. радио», 1965, 190 с.
157. Холл М., Комбинаторика. М., «Мир», 1970, 420 с.
158. Хоор К., Обработка записей. — В кн.: «Языки программирования», М., «Мир», 1972, 278—343.
159. Ху Т., Целочисленное программирование и потоки в сетях. М., «Мир», 1974, 520 с.

160. Чудновский Э. А., Блочный алгоритм расчета крупных сетевых графиков. — Вычисл. и организ. техника в строительстве и проектировании, 1966, вып. 8, 26—33.
161. Ширяев А. Н., Статистический последовательный анализ, М., «Наука», 1976, 272 с.
162. Шмырев В. И., Алгоритм решения одного класса задач линейного программирования большого объема. — Оптимальное планирование, 1969, 11, 88—116.
163. Шрейдер Ю. А., Задачи динамического программирования и автоматы. — Проблемы кибернетики, 1961, вып. 5, 31—48.
164. Юдин Д. Б., Математические методы управления в условиях неполной информации. М., «Сов. радио», 1974, 400 с.
165. Юдин Д. Б., Гольштейн Е. Г., Линейное программирование (теория, методы и приложения). М., «Наука», 1969, 424 с.
166. Яковлева М. А., Двухкомпонентная задача линейного программирования. — Оптимальное планирование, 1964, 2.
167. Янг Л., Лекции по вариационному исчислению и теории оптимального управления. М., «Мир», 1974, 484 с.
168. Appel P., La problème géométrique des deblais et remblais. — Mem. Sci. Math., 1928, 27, 1—34.
169. Balas E., Padberg M. W., On the set covering problem. — Operat. Res., 1972, **20**, 6, 1152—1161.
170. Balinski M. L., Integer programming: methods, uses, computation. — Manag. Sci., 1965, **12**, 3, 253—313.
171. Balinski M., Hellerman E., Computational practice in Mathematical Programming, Math. Program. Study, 1975, **4**.
172. Bartels R. H., Golub G. H., The simplex method of linear programming using LU decomposition. — Comm. ACM, 1969, 12, 266—268, 275—278.
173. Bellman R. E., A Markovian decision process. — J. Math. and Mech., 1957, **6**.
174. Bellman R. E., On a routing problem. — Quart. Appl. Math., 1958, **16**, 1, 87—90.
175. Berge C., Espace topologique et fonctions multivoque, Paris, Dunod, 1963, 272 p.
176. Berge C., Graphes et hyper-graphes, Paris, Dunod, 1970, 502 p.
177. Berge C., Ghouila-Houri A., Programmes, Jeux et Reseaux du Transport, Paris, Dunod, 1962, 254 p.
178. Bertele U., Brioshi F., Nonserial dynamic programming, N. Y., Acad. Press. 1972, 235 p.
179. Blackwell D., Discounted dynamic programming. — Ann. Math. Stat., 1965, 36, 226—235.
180. Bradley G. H., Transformation of integer programs to knapsack problems. — Discrete math., 1971, **1**, 1, 29—45.
181. Choquet G., Deny J., Modelès finis en theorie du potentiel. — J. d'Analyse math., 1956/57, 5, 77—135.
182. Christofides N., Eilon S., Algorithms for large scale travelling salesman problems. — Operat. Res. Q. 1972, **23**,4, 511—518.
183. Clark C. E., The optimum allocation of resources among the activities of a network. — J. industr. engineering, 1962, **13**, 2.
184. Dantzig G. B. All shortest routes in a graph. — In: «Theorie des Graphes», Paris, Dunod, 1967, 91—92.

185. Derman C., Finite state Markovian decision processes, N. Y., Acad. Press, 1970, 159 p.
186. Dijkstra E. W., A note on two problems in connection with graphes. — Numerische Math., 1959, 1, 269—271.
187. Dilworth R. P., A decomposition theorem for partially ordered sets. — Ann. of Math., 1950, 51, 161—166.
188. Dreyfus S. E., An appraisal of some shortest-path algorithms. — Operat. Res., 1969, 17, 3, 395—412.
189. Egerváry J., Matrixok kombinatorikus tulajdonságairol. — Mat. es Fis. Lapok, 1931, 38, 16—28.
190. Elmaghraby S., The concept of state in discrete dynamic programming. — J. of Math. Anal. and Appl., 1970, 29, 523—557.
191. Fiedler M., Some applications of the theory of graphes in matrix theory and geometry, — In: «Theory of graphes and its applications, Proceedings of the Symposium held in Smolenice», Prague, 1963.
192. Floyd R. W., Algorithm 97, Shortest path. — Comm. of ACM, 1962, 5, p. 345.
193. Forrest J. J. H., Tomlin J. A., Updating triangular factors of the basic to maintain sparsity in the product form simplex method.— Math. Progr., 1972, 2, 3, 263—278.
194. Fransis R. L., Goldstein J. M., Location theory: A selective bibliography. — Operat. Res., 1974, 22, 2, 400—410.
195. Friedrich V., Stetige Transportoptimierung. Berlin, VEB Deutsh. V. der Wissenschaften, 1972, S. 175.
196. Fulkerson D. R., A network flow computation for project cost curves. — Manag. Sci., 1959, 5, 2, 472—483.
197. Fulkerson D. R., Packing rooted directed cuts in a weighted directed graph. — Math. Program., 1974, 6, 1, 1—13.
198. Geoffrion A. M., Marsten R. E., Integer programming algorithms: a framework and state-of-the art survey. — Manag. Sci., 1972, 18, 9, 465—491.
199. Hammer (Ivanescu) P. L., Rudeanu S., Boolean Methods in Operations Research and Related Areas, Springer, 1968, 329 p.
200. Held M., Karp R. M., The travelling salesman problem and minimum spanning trees. — Math. Program., 1971, 1, 1, 6—25.
201. Hoffman A. J., A generalization of max flow-min cut. — Math. Program., 1974, 6, 3, 352—359.
202. Hoffman A. J., Winograd S., Finding all shortest distances in a directed network. — IBM J. Res. Devel., 1972, 16, 414—416.
203. Howard R., Dynamic probabilistic systems, N. Y., 1971.
204. Ibaraki T., Solvable classes of discrete dynamic programming.— J. of Math. Anal. and Appl., 1973, 43, 642—693.
205. Kao E. P. C., Optimal replacement rules when changes of state are semi-Markovian. — Operat. Res., 1973, 21, 6, 1231—1249.
206. Karp R. M., Held M., Finite-state processes and dynamic programming. — SIAM J. Appl. Math., 1967, 15, 693—718.
207. Kelley J. E., jr., The cutting plane method for solving convex programs. — J. of SIAM, 1960, 8, 4, 703—712.
208. Kosten L., Heuristische Methoden zur Arbeitsangleichung bei Fliessbandern. In: Proc. Operat. Res. 1, Wurzburg — Wien, 1972.
209. Kruskal J. B., On the shortest spanning tree of a graph and the travelling salesman problem. — Proc. 1956, 7, 48—50.

210. Künzi H. P., Tchach H. G., Zehnder C. A., Numerische Methoden der mathematischen Optimierung, Stuttgart, B. G. Taubner. 1967, S. 151.
211. Land Ailsa H. Doig Alison, An automatic method of solving discrete programming problems. — Econometrica, 1960, **28**, 3.
212. Land Ailsa H., Powell Susan, Fortran codes for mathematical programming, linear, quadratic and discrete, Wiley, 1973, 249 p.
213. Lawler E. R., Wood D. E., Branch-and-bound methods: a survey. — Operat. Res., 1966, **14**, 4, 699—719.
214. McKenzie L, Turnpike theory. — Esonometrica, 1976, **44**, 5.
215. Mine H., Osaki S., Markovian decision processes, N. Y., Elsevier, 1971, 142 p.
216. Mirsky L., Results and problems in the theory of doubly stochastic matrices. — Z. Wahrscheinlichkeitstheorie, 1963, 1, 319—334.
217. Monge G., Déblais et remblais. Mem. Ac. Sci., Paris, 1781.
218. Moore E., The shortest path through a maze. In «Proc. Internat. Symp. on the Theory of Switching», p. 2, Harvard, 1959, 285—292.
219. Munkres J., Algorithms for the assignment and transportation problems. — J. of SIAM, 1957, 5, 32—38.
220. Nožička F., Guddat J., Hollatz H., Bank B., Theorie der linearen parametrishen Optimierung, Berlin, 1974.
221. Orchard-Hays W., Advanced linear programming computing technique, N. Y., McGraw-Hill, 1968, 355 p.
222. Pierskalla W. P., The multidimentional assignment problem. — Operat. Res., 1968, **16**, 422—431.
223. Pollack M., The maximum capacity route through a network, — Operat. Res., 1960, **8**, 1, 35—63.
224. Pollack M., Wiebenson W., Solution of the shortest-route problem — A review. — Operat. Res., 1960, **8**, 224—230.
225. Ray-Chandhuri D. K., An algorithm for a maximum cover of an abstract complex. — Canad. J. Math., 1963, **15**, 11—24.
226. Romanovsky I. V., On the solvability of Bellman's functional equation for a Markovian decision process. — J. of Math. Anal. and Appl., 1973, **42**, 2, 485—498.
227. Rose D. J., Triangulated graphs and the elimination process. — J. of Math. Anal. and Appl., 1970, **32**, 3, 597—609.
228. Shimbal A., Structure in communications nets. — In: «Proc. Symp. Information Networks», Polytech. Inst. of Brooklyn, 1954.
229. Simmonard M., Programmation linéare, Paris, Dunod, 1962, (2 ed., t. 1 — Fondaments, 1972, 287 p.).
230. Tinney W. F., Walker J. W., Directed solution of sparse network equations by optimally ordered triangular factorization. — Proc. IEEE, 1967, 55, 1801—1809.
231. Tomlin J. A., On pricing and backward transformation in linear programming. — Math. Program., 1974, **6**, 1, 42—47.
232. Wirth N., An algorithmic language «Pascal». — Acta Informatica, 1971, **1**, 1, 37—63.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ