ГОСУДАРСТВЕННЫЙ КОМИТЕТ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПО ВЫСШЕМУ ОБРАЗОВАНИЮ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А.В.Косточка

ДИСКРЕТНАЯ МАТЕМАТИКА

Учебное пособие

Часть 2

Новосибирск
1996

Пособие является второй частью конспекта лекций по курсу "Дискретная математика". Рассматриваются дискретные алгоритмические задачи (включая основы теории матроидов) и задачи теории кодирования. Пособие предназначено для студентов технического факультета ВКИ НГУ (специальнсть "компьютерная техника") и физического факультета НГУ (специальность "информатика").

1 АЛГОРИТМЫ И ИХ СЛОЖНОСТЬ

1.1 Понятие о сложности алгоритмов

ЭВМ выполняют (пока) лишь корректно поставленные задачи. В частности, они выполняют *алгоритмы* — точные и однозначно понимаемые последовательности команд, при помощи которых решаются определенные вычислительные задачи. Существует ряд разных определений понятия "алгоритм" (рекурсивные функции, машины Тьюринга-Поста (одноленточные и многоленточные), нормальные алгоритмы Маркова, машины с произвольным доступом к памяти). Оказалось, что множество функций, вычислимых с помощью определенных по-разному алгоритмов, почти совпадают. Это одначает, что определения были вполне естественны.

Введем несколько понятий. Под *массовой задачей* (далее просто *задачей*) мы понимаем общий вопрос, на который следует дать ответ. Задача содержит некоторые параметры, значения которых неопределены. Нам нужны:

1. Общий список параметров;

2. Формулировка свойств, которым должен удовлетворять *ответ*.

Индивидуальная задача получается из массовой, когда всем параметрам массовой задачи присвоены конкретные значения.

Пример.

Массовая задача: 1) параметры a и b;
2) найти $a + b$.

Индивидуальная задача: найти $2 + 3$.

Понятно, что говорить об алгоритмах имеет смысл лишь для массовых задач. Поскольку упомянутые выше определения алгоритмов были строгими, оказалось, что в терминах любого из них существуют неразрешимые задачи, то есть такие, что для них не существует алгоритма, который за конечное время для любых исходных данных давал бы решение. Тьюринг показал, что не существует алгоритма, который по любой программе для так называемой машины Тьюринга может за конечное время указать, будет ли машина работать по этой программе бесконечно.

На практике более актуален другой вопрос. Нас интересует не просто конечное время работы алгоритма, а возможность получить ответ за не слишком большое время.

Пример. Пусть в полном n-вершинном графе каждому ребру сопоставлен вес. Требуется найти каркас (т.е. остовное дерево), сумма весов ребер которого минимальна. Это так называемая задача о кратчайшей связывающей сети. Она может возникнуть, например, при построении самой дешевой сети дорог, связывающей данные n населенных пунктов. Очевидно, конечный алгоритм решения этой задачи состоит в просмотре всех каркасов и выборе среди них каркаса минимального веса. Но при этом потребуется рассмотреть, как мы знаем, n^{n-2} вариантов, что уже при $n \geq 40$ больше 10^{60}.

Поэтому естественной мерой качества работы алгоритма является отрезок времени, затрачиваемый ЭВМ для окончательного ответа. Мы будет оценивать время

работы алгоритма в терминах числа элементарных шагов (арифметических операций, сравнений и т.д.), необходимых для выполнения работы алгоритма на гипотетической ЭВМ. Для простоты будем считать, что элементарные операции требуют одинаковое время. Понятно, что малые объемы входной информации ЭВМ должна обрабатывать быстрее, чем большие. Мы будем оценивать $f_A(n)$ — максимальное время работы алгоритма A по всем входам длины не более n.

Как измерять длину входа? Длиной целого числа m естественно считать $\log m$. Если граф (орграф) G задан матрицей смежности, то длина записи есть $O(n^2)$, где n — число вершин G; если списком смежностей — то $O(m \log n+n)$, где m — число ребер G. Если дана целочисленная матрица $A = (a_{ij})_{i,j=1}^{n}$, то длина ее записи есть $O(mn + \Sigma_{i,j} \log(a_{i,j} + 1))$. Но поскольку многие современные ЭВМ отводят разным числам примерно одинаковое место, мы будем считать, что длина записи информации о графе G, заданном списком смежностей, есть $O(m + n)$, где m — число ребер, а n — число вершин графа G.

Нас будет интересовать порядок роста функции $f_A(n)$, а не точные ее значения. Будем говорить, что алгоритм эффективен, если для некоторого фиксированного k

$$f_A(n) = O(n^k).$$

Рассмотрим на примере, чем это обусловлено.

Предположим, что $f_{A_1}(n) = 2^n, f_{A_2}(n) = 10^6 n^2$. Во-первых, при достаточно больших n (а именно, при $n \geq 30$) A_2 работает быстрее чем A_1. Во-вторых, предположим, что некоторая ЭВМ за время t может решать с помощью A_1 все задачи с длиной входа не более n_1, а с помощью A_2 — все задачи с длиной входа не более n_2. Пусть, далее, быстродействие нашей ЭВМ увеличиловь в 16 раз. Тогда она за время t сможет решать с помощью A_2 все задачи с длиной входа не более $4n_2$, а с помощью A_1 — все задачи с длиной входа не более $n_1 + 4$.

1.2 Поиск по графу

Существует ряд алгоритмов на графах, основанных на таком систематическом переборе, при котором каждая вершина просматривается ограниченное число раз. Поэтому важно знать хорошие методы поиска вершин в графе.

Пусть граф $G = (V, E)$ задан списком смежности $A(v)(v \in V)$ своих вершин. Алгоритм ПОИСК работает следующим образом.

Вход. Граф G; вершина v.

Выход. Тот же граф, в котором вершины, достижимые из v, помечены.

```
Begin
      Q := {v}, пометить v
   While Q ≠ ∅ do
    Begin Пусть x ∈ Q; удалить x из Q;
      for all y ∈ A(X) do
       if y не помечена then
      добавить y в Q и пометить y;
    End
End
```

Утверждение 1.1 *Алгоритм ПОИСК помечает все вершины графа G, достижимые из v, за время $O(|E|)$.*

ДОКАЗАТЕЛЬСТВО. Если на каком-то шаге в Q были лишь вершины, достижимые из v, то и на следующем шаге в Q будут лишь вершины, достижимые из v. И наоборот, индукцией по расстоянию от v легко доказать, что любая достижимая из v вершина рано или поздно окажется в Q.

Для оценки трудоемкости заметим, что работа алгоритма разбивается на три части.

1. Начальный шаг. Требуется время $O(1)$ (для записи v).

2. Работа с множеством Q. Добавление и удаление элементов множества Q производится $2|V_1|$ раз, где V_1 — множество достижимых из v вершин. Заметим, что $|V_1| \leq |E|+1$. Если организовать Q очередью или стеком, то каждое включение требует не более трех элементарных операций с числами.

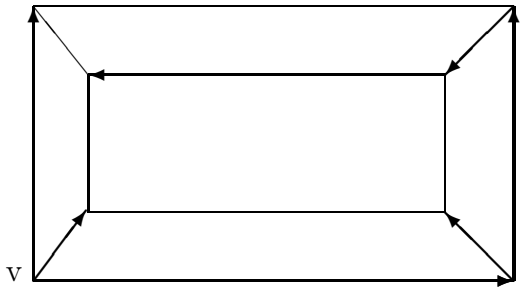
3. Поиск по спискам смежностей. При этом любой элемент списков просматривается один раз, а общее число элементов равно $2|E|$.

Суммарно получаем оценку $O(|E|)$. ∇[1]

Пример. Алгоритм ПОИСК можно непосредственно использовать для проверки графа на связность и для нахождения компонент связности.

Когда Q организована в виде стека, имеем ПОИСК В ГЛУБИНУ (ПГ) (*depth-first search*). При ПГ в стеке все время хранится путь от v до рассматриваемой вершины x. Можно выделять *дерево поиска,* образующееся при ПГ.

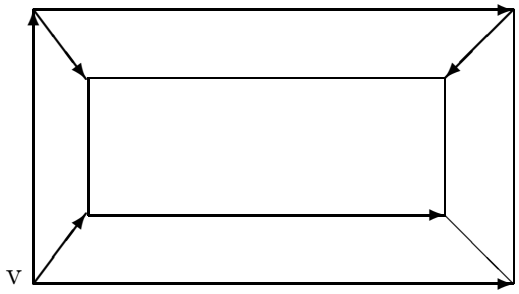
Пример.



Если Q организована в виде очереди, то получаем ПОИСК В ШИРИНУ (ПШ) *(breadth-first search).* При ПШ просматриваются сначала вершины на расстоянии 1 от v, а затем на расстоянии 2 и т.д. ПШ удобен для нахождения кратчайшего (по числу ребер или дуг, если граф ориентированный) пути из v до заданной вершины. Более того, дерево поиска при ПШ является деревом кратчайших (по числу ребер) расстояний от v до остальных вершин из той же самой компоненты связности.

Пример.

[1]Везде в тексте знак ∇ читается "окончание доказательства".



1.3 Быстрая сортировка

Сортировкой называют упорядочение множества объектов по неубыванию или невозрастанию какого-нибудь параметра. Существует много эвристических алгоритмов сортировки. Но многие из них требуют в худших случаях выполнения $O(n^2)$ операций сравнения параметров, где n — число объектов в рассматриваемом множестве. Мы приведем два алгоритма, которые даже в худшем случае требуют выполнения $O(n \log n)$ операций сравнения параметров.

Первый из них иногда называют алгоритмом Фон-Неймана.

На вход подается последовательность чисел $a(1), \ldots, a(n)$. Алгоритм работает $\lceil \log_2 n \rceil$ итераций. Перед началом итерации с номером $(k = 1, 2, \ldots, \lceil \log_2 n \rceil)$ имеется последовательность $a(i(1)), \ldots, a(i(n))$ тех же чисел, разбитая на группы по 2^{k-1} элементов (последняя группа может быть неполной), и внутри каждой группы элементы упорядочены по неубыванию. Итерация состоит в том, что эти группы разбиваются на пары соседних групп, и элементы упорядочиваются внутри этих новых в два раза больших групп. При этом используется то, что внутри исходных групп элементы уже упорядочены. Пусть, например, элементы групп $x_1, \ldots, x_m$ и $y_1, \ldots, y_m$ уже упорядочены. Последовательность $z_1, \ldots, z_{2m}$ образована из них по следующим правилам.

Полагаем $z_1 = \min\{x_1, y_1\}$. Если этот минимум достигается на x_1, то x_2 сравниваем с y_1 и минимум обозначаем через z_2 и т.д. После каждого сравнения одно из сравниваемых чисел вносится в список z_i, пока не кончатся элементы одной из групп. Тогда далее просто выписываются оставшиеся элементы еще не истощившейся группы.

Упражнения.

1.Доказать, что приведенный выше алгоритм упорядочивает числа $a(1), \ldots, a(n)$ по неубыванию.

2. Доказать, что приведенный выше алгоритм использует не более $n \cdot \lceil \log_2 n \rceil$ операций сравнения.

Следующий алгоритм более сложен. Будем называть его пирамидальным алгоритмом. Он состоит из двух этапов.

На первом этапе за линейное (от количества элементов) число шагов мы добиваемся, чтобы для каждого $i (1 \le i \le n)$ элемент с номером i был не меньше, чем элемент с номером $\lfloor i/2 \rfloor$. На втором этапе происходит окончательное упорядочивание.

(i, n)-операцией называется следующая последовательность действий. Сравниваем a_{2i} и a_{2i+1}. Пусть x — меньшее из этих чисел (если $2i = n$ или $a_{2i} = a_{2i+1}$, то

$x = a_{2i}$). Сравниваем x с a_i; если $a_i > x$ то меняем местами a_i и x.

Так называемая (j, n)-процедура состоит в следующем. Выполняем (j, n)-операцию. Если a_j переместилось вниз (скажем, стало a_{2j+1}), то производим $(2j+1, n)$-операцию и так далее, пока либо в результате очередной операции наше бывшее a_j не перестанет опускаться, либо его новый номер не станет больше $n/2$.

На первом этапе последовательно для $j = \lceil n/2 \rceil, \lceil n/2 \rceil) - 1, \ldots, 1$ выполняем (j, n)-процедуру.

Упражнения.

1. Доказать, что после первого этапа $a_i \leq \min\{a_{2i}, a_{2i+1}\}$ для любого $1 \leq i \leq n/2$.

2. Доказать, что если $n = 2^k - 1$, то для выполнения первого этапа достаточно выполнить $n - k$ (i, n)-операций.

3. Доказать, что описанный первый этап алгоритма использует не более $2n$ (i, n)-операций.

Перед началом j-й итерации второго этапа $(j-1)$ самых малых чисел уже стоят на местах $n, n-1, \ldots, n-j+2$ а для каждого $i (1 \leq i \leq n-j+1)$ элемент с номером i не меньше, чем с номером $\lfloor i/2 \rfloor$. Из-за этого самый малый из первых $n-j+1$ элементов стоит на первом месте. На j-й итерации первый и $(n-j+1)$-й элементы меняются местами, а затем выполняется $(1, n-j)$-процедура. После этого упорядочены уже j элементов, и выполнены условия для $(j+1)$-й итерации. После n-й итерации все числа будут упорядочены по невозрастанию. Поскольку $(1, n-j)$-процедура при любом j и любых наборах чисел требует не более чем $\log_2 n + 1$ $(i, n-j)$-операций, общая трудоемкость алгоритма не более $O(n \log n)$ элементарных операций.

Замечания

1. Можно запрограммировать работу алгоритма так, что почти не потребуется дополнительной памяти.

2. Если требуется найти только k самых малых элементов, то на втором этапе достаточно k итераций.

1.4 Идея динамического программирования на примере распределительной и обратной к ней задач

Метод динамического программирования (ДП), созданный Р. Беллманом, можно трактовать как алгоритмическую версию рассуждений по индукции. Он применим к процессам, имеющим этапы, где любой отрезок оптимальной последовательности решений сам является оптимальным. Например, если кратчайший путь между x и y проходит через v, то часть этого пути от v до y сама является кратчайшим путем от v до y.

Рассмотрим следующую задачу. Пусть имеется n предприятий и Y единиц некоторого ресурса. Известно количество $f_k(x)$ продукции, которое будет произведено на k-м предприятии, если в него будет вложено x единиц ресурса. Считаем, что $f_k(x)$ монотонно неубывающая функция при любом k. Требуется максимизировать объем произведенной продукции. Сформулируем задачу математически.

$$f_1(x_1) + \cdots + f_n(x_n) \longrightarrow \max, \tag{1}$$

$$x_1 + \cdots + x_n \leq Y, \tag{2}$$

$$x_i \geq 0, \ , \ i = \overline{1,n}. \tag{3}$$

Идея ДП состоит в разбиении большой задачи на много малых, которые решаются просто. В данном случае пусть $s_k(y)$ для $1 \leq k \leq n$, $0 \leq y \leq Y$ есть значение целевой функции задачи (1)–(3), где n заменено на k, Y — на y. Наша цель — найти $s_n(Y)$ и набор переменных, на котором достигается это значение.

Теорема 1.2. *Если функции $f_1, \ldots, f_n$ монотонно неубывающие, то справедливы следующие рекуррентные соотношения:*

$$s_1(y) = f_1(y), \ 0 \leq y \leq Y; \tag{4}$$

$$s_k(y) = \max\{s_{k-1}(y-x) + f_k(x) \mid 0 \leq x \leq y\}, \ 2 \leq k \leq n, \ 0 \leq y \leq Y. \tag{5}$$

ДОКАЗАТЕЛЬСТВО. Соотношение (4) очевидно. По определению,

$$s_k(y) \geq \max\{s_{k-1}(y-x) + f_k(x) \mid 0 \leq x \leq y\}.$$

Пусть теперь $(x_1^*, \cdots, x_k^*)$ — такой вектор, что $x_1^* + \cdots + x_k^* \leq y$ и $s_k(y) = f_1(x_1^*) + \cdots + f_k(x_k^*)$. Поскольку

$$s_{k-1}(y - x_k^*) \geq f_1(x_1^*) + \cdots + f_{k-1}(x_{k-1}^*),$$

имеем $s_k(y) = f_1(x_1^*) + \cdots + f_k(x_k^*) \leq s_{k-1}(y - x_k^*) + f_k(x_k)^*$. $\quad \nabla$

Алгоритм ДП для этой задачи вычисляет множества $S_k = \{s_k(y) | 0 \leq y \leq Y\}$, $k = \overline{1,n}$ с помощью соотношений (4) и (5), где на каждом шаге оптимизируется ровно одна переменная. Процесс вычисления $S_1, \ldots, S_n$ называется *прямым ходом* алгоритма. При *обратном ходе* алгоритма вычисляются $x_n^*, \ldots, x_1^*$ с учетом того, что уже известны S_k. Так, x_n^* определяется из уравнения $s_n(Y) = f_n(x_n^*) + s_{n-1}(Y - x_n^*)$ и так далее.

Это алгоритм с одним прямым и одним обратным ходом.

При *релаксационном варианте* запоминаются только последние значения S_k и в конце находятся $s_n(Y)$ и x_n^*. Затем работа алгоритма начинается сначала и продолжается до нахождения $s_{n-1}(Y - x_n^*)$ и x_{n-1}^*. Далее аналогично находятся $x_{n-2}^*, \ldots, x_1^*$.

Оценим трудоемкость алгоритма ДП с одним прямым и одним обратным ходом. Пусть

$$M = \max_{1 \leq k \leq n} \{f_k(Y)\}.$$

Нам требуется вычислять nY величин $s_k(y)$ и для каждого вычисления надо подсчитать и сравнить $y + 1$ выражений вида $s_{k-1}(y - x) + f_k(x)$. Так как $s_{k-1}(y - x) + f_k(x) \leq nM$, то общее количество затраченных элементарных операций есть $O(YnY \log nM) = O(nY^2 \log nM)$. На обратном ходе нужно вычислить только Y величин $s_k(y)$. Требуемая память — $O(nY \log nM)$.

Время работы релаксационного варианта алгоритма составляет
$O(n^2Y^2 \log nM)$, зато требуемая память — $O(Y \log nM)$.

Обобщим задачу (1)–(3) следующим образом:

$$f_1(x_1) + \cdots + f_n(x_n) \longrightarrow \max, \tag{1'}$$

$$h_1(x_1) + \ldots + h_n(x_n) \leq Y, \tag{2'}$$

$$a_i \geq x_i \geq 0, \ , \ i = \overline{1,n}. \tag{3'}$$

Содержательные интерпретации этой задачи понятны.

Если $h_i(x)$ — целочисленные монотонно неубывающие функции, то вместо (4)–(5) можно использовать следующие рекуррентные соотношения:

$$s_1(y) = f_1(x^*), \ \text{где}\, x^* = \max\{x \leq a_1 | h_1(x) \leq y\},$$
$$0 \leq y \leq Y; \tag{4'}$$

$$s_k(y) = \max_{\{x \leq a_k | h_k(x) \leq y\}} (f_k(x) + s_{k-1}(y - h_k(x))\},$$
$$2 \leq k \leq n, 0 \leq y \leq Y. \tag{5'}$$

В целом задачу можно трактовать как поиск способа получения наибольшего количества продукции при ограниченных ресурсах. Естественной постановкой является также задача поиска наименьших затрат на получение заданного количества продукции:

$$h_1(x_1) + \cdots + h_n(x_n) \longrightarrow \min, \tag{6}$$

$$f_1(x_1) + \cdots + f_n(x_n) \geq D, \tag{7}$$

$$a_i \geq x_i \geq 0, \ , \ i = 1, \ldots, n. \tag{8}$$

Задача (6)–(8) называется обратной к задаче (1')–(3'). Если $f_k(x)$ — целочисленные монотонно неубывающие функции, то для решения задачи (6)–(8) тоже можно использовать динамическое программирование. Обозначим $f_i^{-1}(d) = \min\{0 \leq x \leq a_i | f_i(x) \geq d\}$. Пусть $t_k(d)$ для $1 \leq k \leq n, \ 0 \leq d \leq D$ есть решение задачи (6)–(8), в которой n заменено на k, а D — на d. Наша цель — найти $t_n(D)$. Справедливы следующие рекуррентные соотношения:

$$t_1(d) = \begin{cases} \infty, & f_1(a_1) < d, \\ h_1(f_1^{-1}(d)), & f_1(a_1) \geq d, \end{cases} \tag{9}$$

$$t_k(d) = \min\{t_{k-1}(d - f_k(x)) + h_k(x) | 0 \leq x \leq a_k, \tag{10}$$
$$x \leq f_k^{-1}(d)\} \qquad (k \geq 2).$$

Теорема 1.3 *Предположим, что D — наибольшее число, для которого оптимальное значение целевой функции задачи (6)–(8) не превосходит Y. Тогда оптимальное значение целевой функции задачи (1') –(3') равно D.*

ДОКАЗАТЕЛЬСТВО. Пусть D удовлетворяет условию теоремы и $(x_1^*, \ldots, x_n^*)$ — соответствующее решение задачи (6) –(8). Это значит, что $f_1(x_1^*) + \cdots + f_n(x_n^*) \geq D$ и $h_1(x_1^*) + \cdots + h_n(x_n^*) \leq Y$. Следовательно, D не превосходит оптимального решения D_1 задачи (1')–(3'). Если бы D_1 было больше чем D, то решение задачи (6)–(8) , в которой D заменено на D_1, тоже не превышало бы Y, что противоречит максимальности D. ∇

Таким образом, если, например, h_i не целочисленны, а f_i — целочисленны, то задачу (1')–(3') можно решать переходом к обратной.

1.5 Задача о кратчайшем пути

Как мы уже видели, алгоритм поиска в ширину позволяет за $O(|E|)$ шагов для любой вершины графа с невзвешенными ребрами найти кратчайшие пути от нее до всех остальных вершин. В настоящем параграфе приведены алгоритмы нахождения кратчайших путей в графах (орграфах), ребра (дуги) которых могут иметь разную длину.

Алгоритм Дейкстры

ВХОД. Орграф $G = (V, A)$ с длинами $c_{uv} \geq 0$ его дуг; вершина $s \in V$.
ВЫХОД. Вектор ρ кратчайших расстояний от s до всех $v \in V$ и вектор p предшественников на кратчайших путях из s в v.

```
Begin положить W := s, ρ(s) := 0, p(s) := ∅;
      For all y ∈ V\{s} do ρ(y) := c_sy;
      While W ≠ V do
       Begin найти min{ρ(y)|y ∈ V\W}; пусть это ρ(x);
            W := W ∪ {x};
            For all y ∈ V\W do
             Begin z := ρ(y), ρ(y) := min{ρ(y), ρ(x) + c_xy};
                  If ρ(y) < z then p(y) := x
            end
       end
end
```

Утверждение 1.4. *Алгоритм Дейкстры находит кратчайшие пути из вершины s до каждой из остальных вершин за $O(|V|^2)$ операций сравнения и сложения.*

ДОКАЗАТЕЛЬСТВО. Покажем сначала, что на каждой итерации справедливо:

$$\left.\begin{array}{l} \text{а) Для любой вершины } x \in V \text{ величина } \rho(x) \text{ равна длине кратчайшего из путей от } s \text{ до } x, \text{ все промежуточные вершины которых принадлежат } W; \\ \\ \text{б) Для любой вершины } w \in W \text{ величина } \rho(w) \text{ есть длина кратчайшего пути от } s \text{ до } w. \end{array}\right\} \quad (11)$$

Поскольку в конце работы алгоритма $W = V$, то из (11) будет следовать, что на выходе ρ станет вектором кратчайших расстояний от s до остальных вершин. Итак, когда $W = \{s\}$, то (11) верно. Предположим, что на некотором шаге (11) верно и $\rho(x) = \min\{\rho(y)|y \in V\backslash W\}$. Допустим, что длина некоторого пути $(s, v_1, v_2, \ldots, v_t, x)$ меньше $\rho(x)$. Тогда согласно (11а) имеем $\{v_1, v_2, \ldots, v_t\} \cap W \neq \emptyset$. Пусть i — наименьший номер, для которого $v_i \notin W$. По выбору, $\rho(v_i) \leq$ длина $(s, v_1, v_2, \ldots, v_i) \leq$ длина $(s, v_1, v_2, \ldots, v_t, x) < \rho(x)$. Это противоречит минимальности $\rho(x)$. Значит, $\rho(x)$ есть длина кратчайшего пути от s до x и (11б) будет продолжать выполняться после

добавления x к W. После пересчета $\rho(y) := \min\{\rho(y), \rho(x) + c_{xy}\}$ утверждение (11a) тоже будет выполняться. Таким образом, (11) доказано.

На каждой итерации не более $|V|$ раз выбирается минимум из двух чисел. Число итераций равно $|V| - 1$. ∇

В алгоритме Дейкстры существенно, что длины всех дуг неотрицательны. Для нахождения самих кратчайших путей служит величина $p(y)$, указывающая последнюю вершину на кратчайшем пути из s в y. Весьма эффективным и просто программируемым является алгоритм Флойда-Уоршелла.

Определение. Пусть дана матрица (d_{ij}) размеров $n \times n$. *Операцией треугольника относительно* j называется пересчет $d_{ik} := \min\{d_{ik}, d_{ij} + d_{jk}\}$ по всем $i, k = 1, ..., n$ таким, что $j \neq i, k$.

Эта операция заменяет d_{ik} на $d_{ij} + d_{jk}$, если $d_{ij} + d_{jk} \leq d_{ik}$.

Теорема 1.5. *Пусть* c_{ij} *– длины дуг орграфа* $G = (V, A)$,

$$a \qquad d_{ij} = \begin{cases} c_{ij} & \text{при} \quad i \neq j, \\ 0 & \text{при} \quad i = j. \end{cases}$$

Если выполнить над матрицей (d_{ij}) *операцию треугольника последовательно для* $j = 1, 2, ..., n$, *то в полученной матрице каждый элемент* d_{ik} *равен длине кратчайшего пути из* i *в* k.

ДОКАЗАТЕЛЬСТВО. Покажем сначала, что для каждого j после выполнения операции треугольника для вершины j элемент d_{ik} для любых i и k равен наименьшей длине среди путей из i в k, все промежуточные вершины которых имеют номера, не меньшие j.

Для $j = 1$ это утверждение очевидно. Пусть оно верно для $j = t - 1$ и проведена операция треугольника для вершины t:

$$d_{ik} := \min\{d_{ik}, d_{ii} + d_{ik}\}. \tag{11}$$

Если кратчайший путь из i в k в подграфе орграфа G на вершинах $\{1, 2, \ldots, t, i, k\}$ не проходит через t, то минимум в (12) достигнется на первом аргументе и утверждение остается верным. Если же этот путь проходит через t, то $d_{ii} + d_{ik} \leq d_{ik}$, а по индукционному предположению d_{ii} и d_{ik} — длины кратчайших путей из i в t и из t в k по вершинам с номерами $\leq t - 1$. ∇

Алгоритм Флойда-Уоршелла

ВХОД. Набор неотрицательных чисел $_{ij}$ (длин дуг G).
ВЫХОД. Матрица (d_{ij}) кратчайших расстояний между вершинами G, матрица (e_{ij}) промежуточных вершин с наибольшим номером на кратчайших путях.

```
Begin For all i ≠ j do d_ij = c_ij;
      For all i, j do e_ij = 0;
      For i = 1, ..., n do d_ii = 0;
      For j = 1, ..., n do
          For i = 1, ..., n, i ≠ j do
              For k = 1, ..., n, k ≠ j do
      Begin z := d_ik; d_ik := min{d_ik, d_ij + d_jk};
          if d_ik < z then e_ik := j
```

```
    end
end
```

Замечания.

1. Требуется $n(n-1)^2$ сравнений.

2. Алгоритм правильно работает и в случае, когда в орграфе есть дуги отрицательной длины, но нет контуров отрицательной длины. Если же в G есть контур отрицательной длины, то на одном из шагов для некоторого i значение d_{ii} станет отрицательным.

3. Если вначале положить $d_{ii} = \infty$, то в конце d_{ii} будет равно длине кратчайшего контура, проходящего через i.

1.6 Метод ветвей и границ на примере задачи коммивояжера

Для ряда важных в практическом и теоретическом отношении задач до сих пор неизвестны эффективные алгоритмы их решения. Более того, есть основания полагать, что таких алгоритмов вообще нет. Возможными выходами здесь являются использование приближенных алгоритмов и построение алгоритмов направленного перебора. Обсуждаемая ниже схема *метода ветвей и границ* (МВГ) принадлежит ко второму типу. Она позволяет решать задачи как на максимум, так и на минимум.

Опишем идею МВГ для задач на минимум. Нам требуется найти минимум функции $f(x)$ на "большом" множестве D. Например, среди всех n^{n-2} каркасов n-вершинного графа найти каркас минимального веса.

На каждом шаге МВГ имеются:

1. рекорд x^* — элемент с наименьшим значением $f(x)$ среди уже просмотренных;
2. "просмотренная" часть P множества D, про которую уже известно, что $f(x) \geq f(x^*)\ \forall\ x \in P$;
3. разбиение $\pi = (d_1, ..., d_m)$ остального множества;
4. функция $H : \mathcal{P}(D) \to R$, называемая нижней границей, которая обладает свойствами:
   (a) $H(\{x\}) = f(x)\ \forall\ x \in D$;
   (b) $H(d) \leq \min\{f(x) | x \in d\} \quad \forall d \subset D$.

Шаг МВГ начинается с выбора (по некоторому правилу) какого-то d_i (например, d_m). Затем вычисляется $H(d_m)$. Возможны следующие варианты:

(a) $H(d_m) \geq f(x^*)$. Тогда всё d_m добавляем к "просмотренной" части D и переходим к следующему шагу.

(b) $H(d_m) < f(x^*)$ и известен $x_m \in d_m$ с $f(x_m) = H(d_m)$. Тогда полагаем $x^* := x_m$, всё d_m добавляем к "просмотренной" части D и переходим к следующему шагу.

(c) $H(d_m) < f(x^*)$ и б) не имеет места. Тогда по определенному правилу разбиваем d_m на $\tilde{d}_m$, $\tilde{d}_{m+1}$, ..., $\tilde{d}_{m+s}$, и переходим к следующему шагу, имея вместо разбиения π разбиение $\tilde{\pi} = (d_1, \ldots, d_{m-1}, \tilde{d}_m, \tilde{d}_{m+1}, \ldots, \tilde{d}_{m+s})$.

Поскольку D конечно, то через конечное время минимум функции $f(x)$ на множестве D будет найден.

Важны следующие моменты:

1. выбор функции $H(d)$ (с одной стороны, ее вычисление должно быть не слишком трудоемким, с другой — желательно, чтобы ее значения были не слишком далеки от $\min\{f(x)|x \in d\}$);

2. выбор перспективного множества d_m (здесь часто удобно применять просмотр в глубину);

3. способ разбиения d_m, если его не удалось отбросить;

4. выбор первого рекорда.

Применим схему МВГ для решения задачи коммивояжера. Пусть дана матрица (c_{ij}) неотрицательных стоимостей переезда из города i в город j размеров $n \times n$. Напомним, что задача коммивояжера состоит в нахождении наименьшего по суммарной стоимости пройденных дуг замкнутого обхода всех городов, при котором каждый город посещается ровно один раз.

Пусть матрица $d_{ij}(k, \alpha)$ определена так:

$$d_{ij}(k,a) = \begin{cases} c_{ij} & i \neq k; \\ c_{ik} - \alpha & i = k. \end{cases}$$

Тогда длина каждого обхода в задаче с матрицей $d_ij(k, \alpha)$ отличается от длины этого же обхода в задаче с матрицей c_{ij} ровно на α. Следовательно, обход $i_1, i_2, \ldots, i_n$ в задаче с матрицей c_{ij} будет оптимальным, если и только если он будет оптимальным в задаче с матрицей $d_{ij}(k, \alpha)$.

Мы используем это свойство в следующем алгоритме типа ветвей и границ для задачи коммивояжера.

1. Множество D — это множество замкнутых обходов всех городов, при которых каждый город посещается ровно один раз, $f(x)$ — стоимость обхода x в матрице c_{ij}.

2. Выбираем первый рекорд x^* для МВГ (например, по принципу “иди в ближайший из еще не пройденных городов”).

3. Подмножества d_i, которые используются в алгоритме, будут иметь специальный вид. Пара $d = (\{l_1, \ldots, l_k\}, \{q_1, \ldots, q_s\})$, где $l_1 = 1$ и $|\{l_1, \ldots, l_k, q_1, \ldots, q_s\}| = k + s$, определяет множество тех замкнутых обходов, в которых после l_1 посещается l_2, затем $l_3, \ldots, l_k$, а следующим после l_k не является ни один из городов $q_1, \ldots, q_s$.

4. По заданным $\{l_1, \ldots, l_k\}, \{q_1, \ldots, q_s\}$ построим матрицу (d_{ij}), где

$$d_{ij} = \begin{cases} c_{ij} & \text{если} \{i,j\} \cap \{l_1, \ldots, l_k\} = \emptyset \text{ или} \\ & i \notin \{l_1, \ldots, l_k\}, j = l_1 \text{ или} \\ & i = l_k, j \notin \{l_1, \ldots, l_k, q_1, \ldots, q_s\} \text{ или} \\ & i = l_r, j = l_{r+1} (1 \leq r \leq k-1); \\ \infty, & \text{иначе.} \end{cases}$$

Пусть

$$\alpha_i = \min\{d_{ij} |\ 1 \le j \le n\} \qquad i \notin \{l_1, \ldots, l_{k-1}\},$$
$$\beta_j = \min\{d_{ij} - \alpha_i |\ 1 \le i \le n\} \qquad j \notin \{l_2, \ldots, l_k\}.$$

5. Положим

$$H(d) = \sum_{r=1}^{k-1} c_{l_r, l_{r+1}} + \sum_{i \notin \{l_1, \ldots, l_{k-1}\}} \alpha_i + \sum_{j \notin \{l_2, \ldots, l_k\}} \beta_j$$

.

6. Выбор перспективного множества будем осуществлять как вариант просмотра в глубину:

   (a) множества $d_m, \ldots, d_1$ хранятся в стеке;

   (b) если после вычисления $H(d_m)$ множество d_m просмотрено, то переходим к d_{m-1};

   (c) если после вычисления $H(d_m)$ не просмотрено множество $d_m = (\{l_1, \ldots, l_k\}, \{q_1, \ldots, q_s\})$ и $s + k < n - 1$, то выберем индекс t, на котором достигается минимум по j выражения $d_{l_k, j} - \alpha_{l_k} - \beta_j$, затем разобьем d_m на два подмножества:

$$\tilde{d}_m = (\{l_1, \ldots, l_k\}, \{q_1, \ldots, q_s, t\})$$
$$\tilde{d}_{m+1} = (\{l_1, \ldots, l_k, t\}, \emptyset)$$

   и, поместив $\tilde{d}_m$ в стек, рассматриваем $\tilde{d}_{m+1}$;

   (d) если после вычисления $H(d_m)$ не просмотрено множество $d_m = (\{l_1, \ldots, l_k\}, \{q_1, \ldots, q_s\})$, $s+k = n-1$ и $k < n-1$, то вместо d_m рассматриваем $\tilde{d}_m = (\{l_1, \ldots, l_k, t\}, \emptyset)$, где

$$\{t\} = \{1, ..., n\} \backslash \{l_1, \ldots, l_k, q_1, \ldots, q_s\}$$

   (понятно, что $\tilde{d}_m = d_m$);

   (e) если $k = n - 1$, то d_m содержит только обход (l_1, …, l_{n-1}, t), где $\{t\} = \{1, \ldots, n\} \backslash \{l_1, \ldots, l_{n-1}\}$.

Описанный алгоритм неплохо решает задачу коммивояжера.

2 ПОТОКИ В СЕТЯХ

2.1 Сети и потоки в них

Сетью будем называть орграф G, некоторые вершины которого отмечены. Отмеченные вершины назовем *полюсами*, а остальные вершины — *внутренними*. Мы будем рассматривать классические сети с двумя полюсами: источником s и стоком t.

Функция $f : E(G) \longrightarrow R$ называется *потоком*, если для любой внутренней вершины v в G

$$div_f(v) = \sum_{e \in E^-(v)} f(e) - \sum_{e \in E^+(v)} f(e) = 0, \tag{1}$$

где $E^+(v)$ — множество дуг сети G, заходящих в v,
$E^-(v)$ — множество дуг сети G, выходящих из v.
Рассмотрим

$$\sum_{v \in V(G)} div_f(v). \tag{2}$$

Ввиду (1), эта величина равна $div_f(s) + div_f(t)$. С другой стороны, для любой дуги $e = (v, u)$ величина $f(e)$ входит в сумму (2) с плюсом в слагаемом $div_f(v)$ и с минусом — в слагаемом $div_f(u)$. Следовательно,

$$div_f(s) + div_f(t) = 0.$$

Величина $M(f) = div_f(s) = -div_f(t)$ называется *мощностью* потока f. Поток нулевой мощности называется *циркуляцией*.

Мощность — линейный функционал на множестве потоков. То есть для любых потоков f и g и действительных чисел λ и μ

$$M(\lambda f + \mu g) = \lambda M(f) + \mu M(g).$$

Поток f с $f(e) = 0$ для каждого $e \in E(G)$ назовем *нулевым потоком*.
Простейшими ненулевыми потоками являются:

1. Поток φ_L вдоль ориентированного пути L, соединяющего s с t или t с s : значения φ_L ненулевые лишь на дугах L;

2. Поток φ_C вдоль контура C : значения φ_C ненулевые лишь на дугах C.

Нетрудно убедиться (проделайте это в качестве упражнения), что значения потока φ_L (соответственно потока φ_C) одинаковы на всех дугах L (соответственно на всех дугах C). Описанные потоки вдоль путей и циклов будем называть *элементарными*. Элементарный поток, при котором значение потока на каждой дуге выбранного пути L (или цикла C) равно ρ, будем обозначать через $\varphi_L(\rho)$ (соответственно через $\varphi_C(\rho)$).
Скажем, что поток f в сети G *положителен*, если

(a) $f(e) \geq 0 \quad \forall\, e \in E(G)$;
(b) $\exists\, e \in E(G) : f(e) > 0$.

Лемма 2.1. *Произвольную положительную циркуляцию f в сети G можно представить в виде суммы не более чем $|E(G)| - 1$ положительных потоков вдоль контуров.*

ДОКАЗАТЕЛЬСТВО. Допустим, что утверждение леммы неверно, G — наименьшая по числу дуг сеть, для которой утверждение леммы неверно, а f — некоторая циркуляция в G.

Если $f(e_0) = 0$ для некоторого $e_0 \in E(G)$, то рассмотрим $G = G_0 \backslash e_0$ и $f_0 = f|_{G_0}$. Ввиду минимальности G, поток f_0 можно представить в виде суммы не более чем $|E(G_0)| - 1$ положительных потоков вдоль контуров. Но это представление и будет требуемым представлением для f. Поэтому в дальнейшем считаем, что

$$f(e) > 0 \qquad \forall\ e \in E(G).$$

Рассмотрим произвольную дугу $e_1 \in E(G), e_1 = (v_0, v_1)$. Поскольку f — циркуляция, то существует дуга $e_2 \in E(G)$, выходящая из $v_1, e_2 = (v_1, v_2)$. Аналогично, существует дуга $e_3 \in E(G)$, выходящая из $v_2, e_3 = (v_2, v_3)$, и т.д. Предположим,что v_k — первая из вершин, встретившаяся в последовательности $v_0, v_1, v_2, \ldots$ второй раз (такая вершина найдется, так как сеть конечна). Пусть $v_k = v_m, m < k$. Тогда в G имеется контур $C = (v_m, v_{m+1}, \ldots, v_k = v_m)$. Обозначим $\rho = \min\{f(e)|\ e \in E(C)\}, \varphi_C(\rho)$ — поток вдоль контура C величины $\rho, \tilde{f} = f\varphi_C$. Если $\tilde{f} \equiv 0$, то лемма доказана.

Если $\tilde{f}$ не является тождественно нулевой функцией, то

(a) поток $\tilde{f}$ положителен, и

(b) $\exists\ e_o \in E(C) : \tilde{f}(e_o) = 0.$

Для орграфа $G_0 = G\backslash e_0$ из-за минимальности G, поток $\tilde{f}_0 = \tilde{f}|_{G_0}$ можно представить в виде суммы не более чем $|E(G_0)| - 1$ $(= |E(G)| - 2)$ положительных потоков вдоль контуров. Но тогда утверждение леммы справедливо для орграфа G, что противоречит его выбору. ∇

Теорема 2.1. *Произвольный положительный поток f в сети G можно представить в виде суммы не более чем $|E(G)|$ положительных элементарных потоков.*

ДОКАЗАТЕЛЬСТВО. Случай 1. $M(f) \geq 0$. Добавим к G новую дугу e_o, ведущую из t в s, и в получившейся сети G_o положим

$$f_o(e) = \begin{cases} f(e), & e \in E(G) \\ M(f), & e = e_0. \end{cases}$$

Поток f_o будет циркуляцией в G_o, и по лемме 2.1 f_o можно представить в виде суммы не более чем $|E(G_o)| - 1 (= |E(G)|)$ положительных потоков вдоль контуров. Контурам сети G_o, содержащим дугу e_o, в G соответствуют пути из s в t. Отсюда получаем требуемое представление для f.

Случай 2. $M(f) < 0$. Тогда добавим к G новую дугу $\tilde{e}$, ведущую из s в t, и в получившейся сети $\tilde{G}$ положим $\tilde{f}(\tilde{e}) = -M(f)$. Далее действуем аналогично случаю 1. ∇

В дальнейшем под *сетью* будем понимать орграф G с полюсами s и t, на дугах которого заданы неотрицательные числа $c(e)$, называемые *пропускными способностями дуг*. То есть $G = (V, E, s, t, \{c(e)|e \in E\})$.

Будут строиться такие потоки f, что

$$0 \leq f(e) \leq c(e) \qquad \forall\ e \in E(G). \tag{3}$$

Если ранее допускались кратные дуги, то ниже будет удобнее каждый набор $\{e_1, \ldots, e_r\}$ кратных дуг заменить одной дугой $\tilde{e}$ с $c(\tilde{e}) = c(e_1) + \cdots + c(e_r)$.

В дальнейшем для каждой дуги $e \in E(G)$ через $\bar{e}$ будем обозначать обратную к e дугу (которая может как принадлежать, так и не принадлежать $E(G)$), а через $\overline{E}$ — множество $E \cup \{\bar{e} | e \in E\}$. Будем полагать $c(e) = 0$ для $e \in \overline{E} \backslash E$.

Пусть G — некоторая сеть и g — поток в G, удовлетворяющий условию (3). Построим сеть G_g по следующим правилам. Сначала добавим к G все дуги из $\overline{E} \backslash E$. Затем для каждой дуги $e \in \overline{E}$ положим

$$c_g(e) = c(e) - g(e) + g(\bar{e}). \tag{4}$$

Наконец, удаляем дуги e с $c_g(e) = 0$. Сеть G_g построена.

Пусть f и g — потоки в сети G. Определим функцию $f \ominus g$ на $\overline{E}$ следующим образом. Для каждой дуги $e \in \overline{E}$ положим

$$(f \ominus g)(e) = max\{0, f(e) - f(\bar{e}) - g(e) + g(\bar{e})\}. \tag{5}$$

Покажем, что $f \ominus g$ удовлетворяет ограничениям (3) для сети G_g. Другими словами, мы хотим доказать, что

$$(f \ominus g)(e) \leq c_g(e) \qquad \text{e} \in \overline{\text{E}}.$$

Действительно, согласно (4),

$$f(e) - f(\bar{e}) - g(e) + g(\bar{e}) = f(e) - f(\bar{e}) - (c(e) - c_g(e)) =$$
$$= (f(e) - c(e)) - f(\bar{e}) + c_g(e) \leq c_g(e).$$

Пусть теперь h — поток в сети G_g. Определим функцию $h \oplus g$ на $\overline{E}$ следующим образом. Для каждой дуги $e \in \overline{E}$ положим

$$(h \oplus g)(e) = \max\{0, h(e) - h(\bar{e}) + g(e) - g(\bar{e})\}.$$

Аналогично предыдущему легко доказывается, что $h \oplus g$ удовлетворяет ограничениям (3) для сети G.

Теорема 2.2. *Для каждой вершины $v \in V$*

$$a) \quad div_{f \ominus g}(v) = div_f(v) - div_g(v);$$
$$b) \quad div_{h \oplus g}(v) = div_h(v) + div_g(v).$$

ДОКАЗАТЕЛЬСТВО. Согласно (5), для любой дуги $e \in \overline{E}$

$$(f \ominus g)(e) - (f \ominus g)(\bar{e}) = f(e) - f(\bar{e}) - g(e) + g(\bar{e}).$$

Следовательно,

$$div_{f \ominus g}(v) = \sum_{(v,u) \in \overline{E}} \Big(f \ominus g)((v,u)) - (f \ominus g)((u,v)) \Big) =$$

$$= \sum_{(v,u) \in \overline{E}} \Big(f((v,u)) - f((u,v)) - g((v,u)) + g((u,v)) \Big) =$$

$$= div_f(v) - div_g(v).$$

Аналогично доказывается утверждение b) нашей теоремы. ∇

Следствие 2.1. *Функции $f \ominus g$ и $h \oplus g$ являются потоками.*

Следствие 2.2. $M(f \ominus g) = M(f) - M(g);\ M(h \oplus g) = M(h) + M(g)$.

2.2 Теорема о максимальном потоке и минимальном разрезе

Постановка задачи о максимальном потоке. Дана сеть G с полюсами s и t и набором неотрицательных чисел $\{c(e)|e \in E\}$. Среди потоков f в сети G, удовлетворяющих ограничениям (3), требуется найти поток наибольшей мощности $M(G)$.

Покажем, что максимальный поток существует. В евклидовом замкнутом $|E|$-мерном пространстве функций $f : E \to R^+$, удовлетворяющих условиям (3), подпространство потоков $\mathcal{M}$ образуется наложением ограничений

$$\sum_{(v,u)\in E} f(e) - \sum_{(u,v)\in E} f(e) = 0, \quad v \in V\backslash\{s,t\}.$$

Значит, при конечных $c(e)$ множество потоков $\mathcal{M}$ является замкнутым и ограниченным подмножеством конечномерного евклидова пространства. Но тогда линейный функционал $M(f)$ достигает на M своего максимума.

Из теоремы 2.2 и следствия 2.1 вытекает

Лемма 2.2. *Если $f \ominus g$ — максимальный поток в сети G_g, то f — максимальный поток в сети G. Обратно, если $h \oplus g$ — максимальный поток в сети G, то h — максимальный поток в сети G_g.*

Разрезом $R = (X, \overline{X})$ сети G назовем любое такое разбиение множества V на подмножества X и $\overline{X}$, что $s \in X, t \in \overline{X}$. При этом дуги с началом в X и концом в $\overline{X}$ называются *выходящими* из X и множество таких дуг обозначается $E^-(R)$. Аналогично дуги с началом в $\overline{X}$ и концом в X называются *входящими* в X и множество таких дуг обозначается $E^+(R)$. *Пропускной способностью* разреза R называется величина

$$c(R) = \sum_{e\in E^-(R)} c(e).$$

Отметим, что для любого разреза R и любого потока f в G

$$M(f) = div_f(s) = \sum_{v\in X} div_f(v) = \sum_{e\in E^-(R)} f(e) - \sum_{e\in E^+(R)} f(e) \leq$$

$$\leq \sum_{e\in E^-(R)} f(e) \leq \sum_{e\in E^-(R)} c(e) = c(R). \qquad (6)$$

Лемма 2.3. *Если f — максимальный поток в сети G, то в сети G_f сток t недостижим из источника s.*

ДОКАЗАТЕЛЬСТВО. Допустим, что в G_f есть путь L из s в t. Поскольку пропускные способности всех дуг сети G_f положительны, то число $\rho = \min\{c_f(e)|e \in E(L)\}$ положительно. Пусть φ_L — элементарный поток вдоль пути L мощности ρ. Тогда, согласно следствию 2.1,

$$M(\varphi_L \oplus f) = M(\varphi_L) + M(f) > M(f),$$

что противоречит выбору f. ∇

Теорема 2.3 (о максимальном потоке и минимальном разрезе). *Величина максимального потока в сети G равна минимальной из пропускных способностей всех разрезов в G.*

ДОКАЗАТЕЛЬСТВО. Ввиду (6), нам достаточно построить разрез $R = (X, \overline{X})$ с $c(R) = M(f)$, где f — максимальный поток в сети G. Обозначим через X множество вершин, достижимых из s в сети $G_f, \overline{X} = V \backslash X$. По определению $s \in X$. По лемме 2.3, $t \in \overline{X}$. Значит, разбиение $R = (X, \overline{X})$ является разрезом.

Если бы в G_f нашлась дуга (v, u) с $v \in X, u \in \overline{X}$, то u тоже была бы достижима из s в сети G_f, что противоречит определению $\overline{X}$. Следовательно, для каждой дуги $e \in \overline{E}$, ведущей из X в $\overline{X}$,

$$c_f(e) = c(e) - f(e) + f(\bar{e}) = 0.$$

Но $c(e) - f(e) \geq 0$. Значит, $f(e) = c(e), f(\bar{e}) = 0$. Таким образом,

$$c(R) = \sum_{e \in E^-(R)} c(e) = \sum_{e \in E^-(R)} f(e) - \sum_{e \in E^+(R)} f(e) =$$

$$\sum_{v \in X} div_f(v) = M(f).$$

2.3 Алгоритмы для нахождения максимального потока

Лемма 2.4. *Пусть поток f не максимален в G. Тогда в сети G_f существует путь из источника s в сток t.*

ДОКАЗАТЕЛЬСТВО. Пусть поток f не максимален в G и g — максимальный поток в G_f. По следствию 2.1, $M(g) > 0$. По теореме 2.1 поток g можно представить в виде суммы положительных элементарных потоков. Хотя бы один из этих потоков должен быть потоком вдоль пути. ∇

Алгоритм Форда-Фалкерсона (АФФ)

ВХОД. Сеть $G = (V, E, s, t, \{c(e) | e \in E\})$.
ВЫХОД. Поток f в сети G наибольшей мощности.

B egin $f = (0, ..., 0); H = G;$
W hile в H есть путь L из s в t do
begin $\rho = \min\{c(e) | e \in E(L)\}$;
$\varphi_L(r) = \{$ поток вдоль L мощности $\rho\}$;
$H = H_{\varphi_L(\rho)}$; $f = \varphi_L(\rho) \oplus f$ $\left(H = H_{\varphi_L(\rho)} \text{ означает и пересчет } c(e) \right)$.
end
End

Если все $c(e)$ рациональны, то алгоритм работает конечное время. Но в общем случае это не так. Хотя для некоторых простых модификаций АФФ достаточно полиномиального от размеров сети количества элементарных операций.

Назовем *рангом* вершины v в сети G расстояние (по числу дуг) в G от s до v.

Лемма 2.5. *Пусть f — поток в сети G, L — кратчайший по числу дуг путь из s в t в сети G_f, φ_L — поток вдоль пути φ_L в сети G_f, $g = \varphi_L \oplus f$. Тогда ранг $r_g(v)$ любой вершины v в сети G_g не меньше ее ранга в сети G_f.*

ДОКАЗАТЕЛЬСТВО. Если $r_g(v) = \infty$, то для v утверждение леммы выполнено. Пусть $r_g(v) = k$ и $L_v = (v_o, v_1, \ldots, v_k)$ — кратчайший по числу дуг путь из $v_o = s$ в $v_k = v$ в сети G_g.

Рассмотрим произвольную дугу $e = (v_i, v_{i+1})$ пути L_v. Если $e \in E(G_f)$, то $r_f(v_{i+1}) - r_f(v_i)) \leq 1$. Если $e \notin E(G_f)$, то $\bar{e} = (v_{i+1}, v_i) \in E(G_f) \cap E(L)$. Поскольку L — кратчайший по числу дуг путь из s в t в сети G_f и $\bar{e} \in E(L)$, то $r_f(v_{i+1}) + 1 = r_f(v_i)$ и $r_f(v_{i+1}) - r_f(v_i) = -1$. Следовательно,

$$r_f(v_k) = \sum_{i=0}^{k-1} \Big(r_f(v_{i+1}) - r_f(v_i) \Big) \leq k. \qquad \nabla$$

Назовем *t-рангом* вершины v в сети G расстояние (по числу дуг) в G от v до t. Аналогично лемме 2.5 доказывается следующий факт.

Лемма 2.5′. *Пусть f — поток в сети G, L — кратчайший по числу дуг путь из s в t в сети G_f, φ_L — поток вдоль пути φ_L в сети $G_f, g = f \oplus \varphi_L$. Тогда t-ранг $tr_g(v)$ любой вершины v в сети G_g не меньше ее t-ранга в сети G_f.*

Алгоритм кратчайших путей (АКП) отличается от алгоритма Форда-Фалкерсона лишь тем, что увеличивающий путь L ищется с помощью поиска в ширину (и тем самым оказывается кратчайшим в G_f по числу дуг).

Оценим трудоемкость АКП. Как отмечалось в главе 1, поиск в ширину требует $O(|E|)$ элементарных операций (множитель $\log |V|$ возникал из-за того, что длина номера вершины может быть порядка $\log |V|$). Для вычисления ρ и пересчета f и H достаточно $O(|V|)$ элементарных операций, поскольку пересчет происходит лишь для дуг L и обратных к ним. Значит, каждая итерация требует не более $O(|E|)$ элементарных операций.

Чтобы оценить число итераций, разобьем их на группы. В k-ю группу попадут те итерации, на которых ранг вершины t равен k. По лемме 2.5 все итерации из группы k идут подряд. Предположим, что перед началом итераций k-й группы мы имели сеть G_g.

Лемма 2.6. *Все дуги увеличивающих путей k-й группы итераций АКП принадлежат G_g и для каждой используемой дуги ранг в G_g ее конца на единицу больше ранга ее начала.*

ДОКАЗАТЕЛЬСТВО. Обозначим через $V_j, j = 0, 1, ...|V|$ множество вершин ранга j в G_g. Покажем, что любая дуга любого из увеличивающих путей k-й группы итераций ведет из V_j в V_{j+1} для некоторого $j \in \{0, 1, \ldots, k-1\}$. По леммам 2.5 и 2.5′ для каждой вершины v, лежащей на каком-нибудь увеличивающем пути, имеем $r(v) + tr(v) = k$. Пусть L — первый из увеличивающих путей k-й группы итераций, в котором есть такая дуга $(v, u), v \in V_i, u \in V_j$, что $j \neq i + 1$. Поскольку L — первый такой путь, то "новые" (по сравнению с G_g) дуги могут вести лишь из V_i в V_{i-1} для некоторого $l \in \{1, ..., k\}$. Значит, $j \leq i$. Но тогда L должен содержать более чем k дуг. ∇

Поскольку на каждой итерации из k-й группы хотя бы одна из "старых" дуг исчезает (поворачивается), то эта группа состоит из менее чем $|E|$ итераций. Отсюда

следует оценка $O(|E|^2|V|)$ элементарных операций для АКП.

Заметим еще раз, что на итерациях k-й группы участвовали лишь такие вершины ранга $j, j \in \{0, 1, \ldots, k\}$ в G_g, расстояние от которых до t равно $k-j$. А дуги, использованные на этих итерациях, ведут из V_j в V_{j+1} для некоторого $j \in \{0, 1, \ldots, k-1\}$. Построить такую подсеть можно за $O(|E|)$ элементарных операций: сначала поиском в ширину найти множества $V_j, j = 0, 1, \ldots k$, затем обратным поиском в ширину из t выделить те из них, расстояние от которых до t равно $k-j$, и, наконец, оставить только нужные дуги.

Применим модификацию АКП для построения путей длины k в полученной k-*слойной сети*.

На каждой итерации последовательно для $i = 0, 1, \ldots, k-1$ проделываем следующее: уже имеется путь $(v_0, v_1, \ldots, v_i)$ от $v_o = s$ до v_i и мы, если $E^-(v_i) = \emptyset$, удаляем v_i из сети вместе со всеми инцидентными ей дугами и начинаем новую итерацию; а если $E^-(v_i) \neq \emptyset$, то добавляем к пути $(v_o, v_1, \ldots, v_i)$ первую дугу (v_i, v_{i+1}) из списка $E^-(v_i)$. Если мы дошли до t, то понижаем пропускные способности дуг (v_i, v_{i+1}) для $i = 0, 1, \ldots, k-1$ на величину $\rho = \min\{c(v_i, v_{i+1})|\ i \in \{0, 1, ..., k-1\}\}$ и удаляем дуги с новыми пропускными способностями, равными нулю. Получается, что на удаление из нашей подсети одной дуги тратится $O(k)$ элементарных операций, на k-ю группу — $O(|E||V|)$, а общая трудоемкость модифицированного алгоритма — $O(|E||V|^2)$ элементарных операций.

Существуют и более быстрые алгоритмы для нахождения максимального потока. Их трудоемкости — $O(|V|^3)$ и $O(|E||V|log|V|)$.

2.4 Использование сетевых моделей для нахождения связности графов. Теорема Менгера и теорема Уитни

Пусть $G = (V, E)$ — орграф, $B, C \subset V,\ B \cap C = \emptyset$. Подмножество X дуг (вершин, не принадлежащих $B \cup C$) в G называется (B, C)-*разделяющим*, если в графе $G - X$ все вершины C недостижимы из B.

Утверждение 2.4. *Для любого орграфа $G = (V, E)$ и любых $B, C \subset V, B \cap C = \emptyset$ мощность наименьшего (B, C)-разделяющего множества дуг равна наибольшему количеству попарно непересекающихся по дугам путей, ведущих из B в C.*

ДОКАЗАТЕЛЬСТВО. Построим сеть H по следующим правилам. Добавим к G вершины s и t и дуги $(s, b), b \in B$ и $(c, t), c \in C$. Положим пропускные способности дуг $(s, b), b \in B$ и $(c, t), c \in C$ равными $|E| + 1$, а пропускные способности остальных дуг равными 1. Поскольку число дуг, выходящих из B, не больше $|E|$, то минимальный разрез $R = (X, \overline{X})$ в H не содержит дуг вида $(s, b), b \in B$ и $(c, t), c \in C$. Отметим, что тогда пропускная способность R равна числу дуг, ведущих из X в $\overline{X}$, а множество A этих дуг является (B, C)-разделяющим. По теореме о максимальном потоке и минимальном разрезе в H существует поток f мощности $|A|$ из s в t.

Так как пропускные способности всех дуг H целочисленны, то, следуя АФФ, можно выбрать f целочисленным. Тогда по теореме 2.1 f можно представить в виде суммы целочисленных положительных потоков вдоль путей и циков. Удалив потоки вдоль циклов, получим поток той же величины, являющийся суммой потоков вдоль путей. По определению пропускных способностей дуг из G, поток вдоль каждого из

этих путей равен 1 и, следовательно, их число равно $|A|$. По той же причине эти пути не имеют общих дуг кроме инцидентных s или t. Тогда части этих путей, полученные после удаления вершин s и t, удовлетворяют требованиям нашего утверждения. ∇

Утверждение 2.5. *Для любого орграфа $G = (V, E)$ и любых $B, C \subset V$, $B \cap C = \emptyset$ таких, что нет дуг, ведущих из B в C, мощность наименьшего (B, C)-разделяющего множества вершин равна наибольшему количеству попарно непересекающихся по вершинам путей, ведущих из B в C.*

ДОКАЗАТЕЛЬСТВО. Построим по следующим правилам вспомогательный орграф G'. Пусть V' - множество копий (дублей) элементов множества V, то есть $V' = \{v' | v \in V\}$. Положим $V(G') = V \cup V', E(G') = \{(v, v') | v \in V\} \cup \{(v', u) |\ (v, u) \in E\}$. Обозначим $B' = \{v' | v \in B\}$. По утверждению 2.4 мощность наименьшего (B', C)-разделяющего множества дуг равна наибольшему количеству попарно непересекающихся по дугам путей, ведущих из B' в C. Заметим, что в G' пути длины два и более, непересекающиеся по дугам, не пересекаются и по вершинам, а каждому пути в G' из B' в C взаимнооднозначно соответствует путь в G из B в C. ∇

Пусть $G = (V, E)$ — граф, $B, C \subset V, B \cap C = \emptyset$. Подмножество X ребер (вершин, не принадлежащих $B \cup C$) в G называется (B, C)-*разделяющим*, если в графе $G - X$ все вершины C недостижимы из B.

Теорема 2.6(Менгера). *Пусть $G = (V, E)$ — граф, $\{v, u\} \subset V, (v, u) \notin E$. Тогда мощность наименьшего $(\{v\}, \{u\})$-разделяющего множества вершин в G равна наибольшему количеству попарно непересекающихся по внутренним вершинам путей, ведущих из v в u.*

ДОКАЗАТЕЛЬСТВО. Построим вспомогательный орграф G' по следующим правилам:

$$V(G') = V, \quad E(G') = \bigcup_{(v,u) \in E} \{(v, u), (u, v)\}$$

.

Применение утверждения 2.5 с $B = \{v\}, C = \{u\}$ к орграфу G' завершает доказательство. ∇

Аналогично доказывается реберный вариант теоремы Менгера.

Утверждение 2.7. *Пусть $G = (V, E)$ — граф, $v, u \subset V$. Тогда мощность наименьшего $(\{v\}, \{u\})$-разделяющего множества ребер в G равна наибольшему количеству попарно непересекающихся по ребрам путей, ведущих из v в u.*

Реберной (вершинной) связностью $\lambda(G)(\kappa(G))$ графа G называется наименьшее число ребер (вершин), после удаления которых граф становится несвязным или одновершинным.

Говорят, что граф G k-связен, если $\kappa(G) \geq k$.

Из теремы Менгера легко следует теорема Уитни.

Теорема 2.8 (Уитни). *Граф G является k-связным тогда и только тогда, когда любые две его вершины соединяют не менее k непересекающихся по внутренним вершинам путей.*

Отметим также, что алгоритмы нахождения максимального потока позволяют за полиномиальное время находить минимальные вершинные и реберные разделяющие множества. Более того, оценки трудоемкости алгоритмов в случае используемых здесь сетей могут быть еще улучшены.

2.5 Задача о наибольшем паросочетании в двудольном графе как задача о максимальном потоке. Теоремы Кёнига и Дилворта

Пусть $G = (V, E)$ — двудольный граф с долями B и C. Построим сеть H по следующим правилам. Ориентируем каждое ребро графа G из B в C. Добавим к G вершины s и t и дуги $(s, b), b \in B$, и $(c, t), c \in C$. Положим пропускные способности всех дуг равными 1. Тогда любой целочисленный поток в сети H взаимнооднозначно соответствует некоторому паросочетанию в G. Действительно, как и в утверждении 2.4, в H можно выбрать максимальный поток f так, что

$$f(e) \in \{0, 1\} \quad \forall\, e \in E(G).$$

Поскольку в каждую вершину из B заходит только одна дуга, а из каждой вершины из C выходит только одна дуга, то пути из s в t, на дугах которых поток равен 1, не имеют общих вершин кроме s и t. Значит, дуги e с $f(e) = 1$, не инцидентные ни s, ни t, образуют паросочетание. Следовательно, $M(f) \leq \pi(G)$. Обратно, каждому паросочетанию мощности k в G легко сопоставить поток величины k в H, протекающий по этим ребрам.

Снова алгоритмы нахождения максимального потока помогают нам. Теперь они дают возможность за полиномиальное время находить наибольшее паросочетание в двудольном графе. Кроме того, теорема о максимальном потоке и минимальном разрезе позволяет дать новое доказательство теоремы Кёнига.

Теорема Кёнига. *Для любого двудольного мультиграфа G мощность $\pi(G)$ наибольшего паросочетания равна мощности $\tau(G)$ наименьшего множества вершин, покрывающего все ребра.*

ДОКАЗАТЕЛЬСТВО. На паросочетания и покрытия наличие кратных ребер не влияет. Поэтому можно считать, что G — двудольный граф с долями B и C. Построим сеть H и поток f как в начале параграфа. Имеем $M(f) \leq \pi(G)$.

Пусть $R = (X, \overline{X})$ — минимальный разрез в сети H, $Y = \overline{X} \cap B, Z = X \cap C$. Можно считать, что среди всех минимальных разрезов в сети H на разрезе R минимизируется $|X|$. Допустим, что в H есть дуга $(b, c), b \in B \backslash Y = X \cap B, c \in C \backslash Z = \overline{X} \cap C$. Тогда, положив $X' = X \backslash \{b\}$, получим разрез R' не большей пропускной способности с $|X'| = |X| - 1$. Это противоречит выбору R.

Следовательно, $Y \cup Z$ покрывает все ребра G, и $\tau(G) \leq |Y \cup Z|$. С другой стороны, по определению,

$$E^-(R) \supset \{(s, y) |\ y \in Y\} \ \cup \{(z, t) |\ z \in Z\},$$

откуда $|Y \cup Z| \leq |E^-(R)|$. Но $|E^-(R)| = M(f)$. $\quad \nabla$

Выведем из теоремы Кёнига теорему Дилворта. Сначала напомним некоторые понятия.

Частично упорядоченным множеством (ЧУМ) называется пара $(P, <)$, где P — конечное множество, а *отношение порядка* $<$ — некоторое антирефлексивное, симметричное и транзитивное бинарное отношение на P. *Цепью* в ЧУМ называется подмножество, любые два элемента которого сравнимы между собой. *Антицепью* в ЧУМ называется подмножество, любые два различных элемента которого несравнимы между собой.

Теорема Дилворта. *Для любого конечного ЧУМ $(P, <)$ наименьшее число непересекающихся цепей, на которое может быть разложено P, равно мощности наибольшей антицепи в P.*

Замечание. Любой двудольный граф G с долями B и C можно представить как ЧУМ $(B \cup C, <)$, где

$$x < y \iff x \in B, y \in C, (x, y) \in E(G).$$

Каждая цепь в этом ЧУМ состоит из одной или двух вершин. Следовательно, наименьшее число непересекающихся цепей $\gamma(B \cup C, <)$, на которое может быть разложено $(B \cup C, <)$, равно $|B \cup C| - \pi(G)$. Антицепями в $(B \cup C, <)$ являются независимые множества графа G. Значит, мощность $\alpha(B \cup C, <)$ наибольшей антицепи в $(B \cup C, <)$ равна $|B \cup C| - \tau(G)$. Таким образом, теорема Кёнига является частным случаем теоремы Дилворта.

ДОКАЗАТЕЛЬСТВО теоремы Дилворта. Пусть $(P, <)$ — конечное ЧУМ, а G — двудольный граф с долями P и P' (где P' — копия P) и пара $(x, y'), x \in P, y' \in P'$ является ребром в G, если $x < y$.

По теореме Кёнига, $\pi(G) = \tau(G)$.

По паросочетанию П в G построим орграф H на множестве вершин $P : (x, y) \in E(H) \iff (x, y') \in$ П. Тогда компоненты связности H соответствуют цепям в P. Число этих компонент равно $|P| - |\text{П}|$. Значит, $\pi(G) \leq |P| - \gamma(P)$.

С другой стороны, если $\mathcal{N}$ — независимое множество вершин в графе G, $\{x, x', y, y'\} \subseteq \mathcal{N}$, то элементы x и y несравнимы в P. Следовательно, $\alpha(P) \geq \Big(|V(G)| - \tau(G)\Big) - |P| = |P| - \tau(G)$.

Таким образом, $|P| - \gamma(P) \geq \pi(G) = \tau(G) \geq |P| - \alpha(P)$, и $\alpha(P) \geq \gamma(P)$. Обратное неравенство очевидно.

2.6 Симплекс-метод решения задач линейного программирования

Задачами линейного программирования называются задачи поиска экстремумов линейных функций на множествах, заданных системами линейных уравнений и неравенств. Такие задачи естественно возникают при исследовании многих экономических, химических и технических проблем. Задача нахождения максимального потока в сети тоже относится к этому классу: требуется найти максимум линейной функции $M(f)$ — потока на множестве $|E|$-мерных векторов f, удовлетворяющих набору линейных неравенств $(0 \leq f(e) \leq c(e), e \in E)$ и набору линейных уравнений (дивергенция f в каждой внутренней вершине равна 0). Способы решения задач линейного программирования были найдены независимо в СССР Л. В. Канторовичем и в США. Автором предлагаемого ниже метода является Г.Данциг.

Симплекс-метод применяется для решения задачи, поставленной в канонической форме:

Найти

$$\min z = \sum_{j=1}^{n} c_j x_j$$

при условиях

$$\begin{cases} \sum_{j=1}^{n} a_{ij}x_j = b_i, & i = 1, \ldots, m, \\ x_j \geq 0, & i = 1, \ldots, m. \end{cases} \tag{7}$$

Из алгебры известно, что множество решений системы линейных уравнений $Ax = b$ не меняется при выполнении следующих операций:
(а) умножение какого-либо уравнения на число $r \neq 0$;
(б) прибавление к одному из уравнений другого, умноженного на любое число.

Применяя метод исключения Гаусса и переименовывая при необходимости переменные, получим эквивалентную (7) систему

$$\begin{array}{cccccc} x_1 & & +a'_{1,m'+1}x_{m'+1}+ & \cdots & +a'_{1,n}x_n & = b'_1 \\ & x_2 & +a'_{2,m'+1}x_{m'+1}+ & \cdots & +a'_{2,n}x_n & = b'_2 \\ & \cdots & \cdots & \cdots & \cdots & = \cdots \\ & x_{m'} & +a'_{m',m'+1}x_{m'+1}+ & \cdots & +a'_{m',n}x_n & = b'_{m'} \end{array} \tag{8}$$

$$x_j \geq 0, \quad j = 1, \ldots, n.$$

Здесь m' — число линейно независимых уравнений в (7).

Случай 1. $b_i \geq 0, i = 1, ..., m'$.

Подставив в

$$z = \sum_{j=1}^{n} c_j x_j$$

выражения $x_1, \ldots, x_{m'}$ из (8), получим

$$z = \sum_{j=m'+1}^{n} c_j x_j - b'_0, \quad \text{т.е.} \quad -z + \sum_{m'+1}^{n} c_j x_j = b'_o.$$

Перепишем систему, взяв $x_0 = -z$ и убрав штрихи:

$$\begin{array}{cccccc} a_{o,o} = & x_0 & & +a_{0,m+1}x_{m+1}+ & \cdots & +a_{0,n}x_n \\ a_{1,o} = & x_1 & & +a_{1,m+1}x_{m+1}+ & \cdots & +a_{1,n}x_n \\ a_{2,o} = & x_2 & & +a_{2,m+1}x_{m+1}+ & \cdots & +a_{2,n}x_n \\ \cdots & \cdots & & \cdots & \cdots & \cdots \\ a_{m,o} = & x_m & & +a_{m,m+1}x_{m+1}+ & \cdots & +a_{m,n}x_n \end{array} \tag{9}$$

$$x_j \geq 0, \quad j = 1, \ldots, n.$$

$$x_0 \longrightarrow max.$$

Взяв

$$\begin{cases} x_i := a_{i,0} & 1 \leq i \leq m, \\ x_j := 0 & m+1 \leq j \leq n, \end{cases}$$

имеем допустимое решение. При этом $x_0 = {}_{00}$. Если все ${}_{0j} \geq 0 (m+1 \leq j \leq n)$, то $x_0 = {}_{00}$ оптимально (почему?).

Система (9) обычно записывается симплексной таблицей:

	1	x_1	x_2	$\cdots$	x_m	x_{m+1}	$\cdots$	x_n
x_0	$a_{o,o}$	0	0	$\cdots$	0	$a_{0,m+1}$	$\cdots$	$a_{0,n}$
x_1	$a_{1,o}$	1	0	$\cdots$	0	$a_{1,m+1}$	$\cdots$	$a_{1,n}$
x_2	$a_{2,o}$	0	1	$\cdots$	0	$a_{2,m+1}$	$\cdots$	$a_{2,n}$
	$\cdots$			$\cdots$		$\cdots$		$\cdots$
x_m	$a_{m,o}$	0	0	$\cdots$	1	$a_{m,m+1}$	$\cdots$	$a_{m,n}$

Здесь 0-я строка читается:

$$x_0 + \sum_{j=m+1}^{n} a_{0,j}x_j = a_{0,0}.$$

Остальные строки таблицы — уравнения системы (9). Переменные слева от таблицы указывают какие x_j выражены через остальные. Эти переменные называются *базисными*.

Введем несколько понятий. Говорят, что вектор-строка *лексикографически положительна*, если первая из ненулевых ее компонент положительна.

Говорят, что вектор-строка x *лексикографически больше* вектор-строки y, если вектор-строка $x - y$ лексикографически положительна.

Описание работы симплекс-метода

Шаг 0. Начинаем с таблицы с $a_{i0} \geq 0$, $i = 1, \ldots, m$, и все строки (кроме, может быть, нулевой) лексикографически положительны.

Шаг 1. Пусть $a_{0s} = \min\{a_{0,j} | 1 \leq j \leq n\}$. Если $a_{0s} \geq 0$, то $x_0 = a_{00}$ оптимально. Иначе s назовем ведущим столбцом.

Шаг 2. Если $a_{i,s} \leq 0$ для любого i, то x_0 не ограничено сверху. Действительно, для любого $t > 0$ можно взять решение:

$$\begin{cases} x_s = t, & i = 1, ..., m, \\ x_i = a_{i,0} - a_{ij}t \quad (\geq a_{i,0} \geq 0), & j = m+1, \ldots, n. \end{cases}$$

Тогда $x_0 = a_{0,0} - ta_{0s} \quad \longrightarrow_{t\to\infty} \infty$. Иначе среди вектор-строк A_i с $a_{i,s} > 0$ выберем такую, чтобы $\frac{1}{a_{i,s}}A_i$ была лексикографически минимальна. Пусть это строка A_r. Тогда она называется *ведущей строкой*, а $a_{r,s}$ — *ведущим элементом.*

Шаг 3. Использовать процедуру исключения Гаусса, чтобы $a_{r,s}$ стало равно 1, а все остальные коэффициенты в s-м столбце стали нулевыми. Заменить базисную переменную слева от r-й строки на x_s. Вернуться к шагу 1.

Покажем, что после 3-го шага

(а) все строки кроме может быть нулевой останутся лексикографически положительны,

(б) 0-я строка лексикографически возрастет.

Действительно, при $i \neq r$ i-й строкой в новой таблице будет $A_i - \frac{a_{i,s}}{a_{r,s}}A_r$. Если $a_{r,s} \leq 0$, то имеем сумму лексикографически положительной и лексикографически неотрицательной строк. Если же $a_{i,s} > 0$, то по выбору r имеем $\frac{1}{a_{i,s}}A_i > \frac{1}{a_{r,s}}A_r$. Значит и в этом случае получим лексикографически положительную строку. Для доказательства б) заметим, что к 0-й строке прибавляется r-я, умноженная на $\left|\frac{a_{0,s}}{a_{r,s}}\right|$.

Из свойства а) следует, что после 3-го шага снова можно выполнять первый уже с другим базисом, а из б) — что один и тот же базис не встретится дважды.

Так как число возможных базисов не более $\binom{n}{m}$, то этим гарантирована конечность алгоритма. Из него следует, что в конце получим либо неотрицательную (кроме, может быть, $a_{o,o}$) 0-ю строку, либо неположительный столбец. В частности, из алгоритма следует, что среди оптимальных решений есть такие, в которых ненулевые значения принимают лишь x_j, соответствующие линейно независимым столбцам.

Случай 2. $b'_1 \geq 0, \ldots, b'_k \geq 0, b'_{k+1} < 0, \ldots, b'_m < 0.$

Умножим последние $m' - k$ уравнений на -1. Введем новые неотрицательные переменные $u_{k+1}, \ldots, u_m$. Рассмотрим систему

$$
\begin{array}{lllllll}
 & x_1 & & & +a'_{1,m'+1}x_{m'+1} + \cdots + a'_{1,n}x_n & = & b'_1 \\
 & \cdots & & & \cdots & & \\
 & & x_k & & +a'_{k,m'+1}x_{m'+1} + \cdots + a'_{k,n}x_n & = & b'_k \\
u_{k+1} & & & -x_{k+1} & -a'_{k+1,m'+1}x_{m'+1} - \cdots - a'_{k+1,n}x_n & = & -b'_{k+1} \\
 & & & \cdots & \cdots & & \\
u_{m'} & & & -x_{m'} & -a'_{m',m'+1}x_{m'+1} - \cdots - a'_{m',n}x_n & = & -b'_{m'}
\end{array}
$$

$$x_j \geq 0, \quad j = 1, \ldots, n.$$

$$u_i \geq 0, \quad i = k+1, \ldots, m'.$$

Здесь правые части всех уравнений неотрицательны, базисными переменными являются $x_1, \ldots, x_k, u_{k+1}, \cdots, u_{m'}$. В этих условиях ищем минимум функции $\xi = u_{k+1} + \cdots + u_{m'}$. Допустимое решение имеется, минимизируемая функция ограничена снизу нулем. Значит, через конечное число шагов найдем решение нашей вспомогательной задачи. Запишем симплекс-таблицу следующим образом:

	1	x_1	x_2	$\cdots x_k$	x_{k+1}	$\cdots x_n$	u_{k+1}		$\cdots u_m$
x_0	0	0	0	$\cdots 0$	0	$\cdots 0$	1		$\cdots 1$
x_1	$a_{1,0}$	1	0	$\cdots 0$	$a_{1,k+1}$	$\cdots a_{1,n}$	0		$\cdots 0$
x_2	$a_{2,0}$	0	1	$\cdots 0$	$a_{2,k+1}$	$\cdots a_{2,n}$	0		$\cdots 0$
	$\cdots$			$\cdots$	$\cdots$	$\cdots\cdots$			
x_k	$a_{k,0}$	0	0	$\cdots 1$	$a_{k,k+1}$	$\cdots a_{k,n}$	0		$\cdots 0$
x_{k+1}	$-a_{k+1,0}$	0	0	$\cdots 0$	$-a_{k+1,k+1}$	$\cdots -a_{k+1,n}$	1	0	$\cdots 0$
x_{k+2}	$-a_{k+2,0}$	0	0	$\cdots 0$	$-a_{k+2,k+2}$	$\cdots -a_{k+2,n}$	0	1	$\cdots 0$
	$\cdots$			$\cdots$	$\cdots$	$\cdots\cdots$			
x_m	$-a_{m,0}$	0	0	$\cdots 0$	$-a_{m,k+1}$	$\cdots -a_{m,n}$	0	$\cdots 0$	1

Отметим, что последние $m - k$ строк являются лексикографически положительными, поскольку $-a_{m,0} > 0$. Если в основной задаче есть допустимый базис, то решение ξ вспомогательной задачи равно 0. Следовательно, если оптимальное решение вспомогательной задачи положительно, то допустимых базисов нет. Пусть $\xi = 0$. Занулим все u_i. Поскольку все u_i стояли в концах строк и строки были лексикографически положительными, то после зануления строки должны остаться лексикографически положительными либо стать нулевыми. Последнее невозможно, т.к. в исходной задаче мы оставили лишь линейно независимые уравнения. Итак, мы получили неотрицательное решение исходной задачи. Но может оказаться так, что некоторым уравнениям (которым раньше соответствовали u_i, входящие в базис) ничего

не соответствует. Предположим, что это последние $m - l$ уравнений. Перенумеруем переменные так, чтобы базисные переменные стояли первыми. Легко видеть, что строки останутся лексикографически положительными. Поскольку $\xi = 0$, система имеет следующий вид:

	1	x_1	x_2	$\cdots$	x_i	x_{i+1}	$\cdots$	x_n
x_1	$a_{1,0}$	1	0	$\cdots$	0	$a_{1,m+1}$	$\cdots$	$a_{1,n}$
x_2	$a_{2,0}$	0	1	$\cdots$	0	$a_{2,m+1}$	$\cdots$	$a_{2,n}$
	$\cdots$		$\cdots$			$\cdots$	$\cdots$	
x_l	$a_{l,0}$	0	0	$\cdots$	1	$a_{l,l+1}$	$\cdots$	$a_{l,n}$
	0	0	0	$\cdots$	0	$a_{l+1,l+1}$	$\cdots$	$a_{l+1,n}$
	$\cdots$		$\cdots$			$\cdots$	$\cdots$	
	0	0	0	$\cdots$	0	$a_{m,i+1}$	$\cdots$	$a_{m,n}$

Теперь, когда ведущими строками будут последние, они не повлияют на лексикографическую положительность первых l строк. Поэтому действуем так: находим первое $j > l$, для которого $a_{i,j} \neq 0$ при некотором $i > l$. В силу лексикографической положительности всех строк имеем $a_{i,j} > 0$. Среди строк i с $i > l$, $a_{i,j} > 0$ выбираем лексикографически минимальную строку k и делаем $a_{k,j}$ ведущим элементом. После этого число элементов базиса увеличивается. Повторяем эту процедуру, пока не построим базис.

3 МАТРОИДЫ

Матроидом будем называть произвольную пару $M := [E, \mathcal{I}]$, где E — конечное множество, а $\mathcal{I}$ — непустое подсемейство подмножеств множества E, удовлетворяющее условиям:

(M1) из $(A \in \mathcal{I}, B \subset A)$ следует, что $B \in \mathcal{I}$;

(M2) $\forall A, B \in \mathcal{I}$ таких, что $|A| < |B|, \exists\, e \in B \backslash A : A \cup \{e\} \in \mathcal{I}$.

Множества семейства $\mathcal{I}$ назовем *независимыми множествами*, а все другие подмножества E — *зависимыми множествами* матроида M.

Из $(M1)$ и непустоты $\mathcal{I}$ следует, что $\emptyset \in \mathcal{I}$ в любом матроиде $[E, \mathcal{I}]$.

База матроида — это любое максимальное по включению независимое множество. Из аксиомы (M2) следует, что все базы матроида имеют одну и ту же мощность.

Цикл матроида — это любое минимальное по включению зависимое множество.

Примеры.

1. **Матричный матроид.** Элементами множества E являются столбцы некоторой матрицы A, и подмножество столбцов считается независимым, если оно линейно независимо (в обычном смысле).

2. **Графический матроид.** Элементами множества E являются ребра некоторого графа G, и подмножество множества ребер G считается независимым, если оно не содержит цикла графа G.

3. **Матроид разбиений.** Элементами множества E являются дуги некоторого орграфа G, и подмножество множества дуг G считается независимым, если никакие две дуги из этого подмножества не заходят в одну и ту же вершину.

4. **Матроид Фано.** Здесь $E = \{e_1, \ldots, e_7\}$ — множество элементов проективной плоскости порядка 2 (так называемой плоскости Фано). В плоскости Фано прямыми являются множества $\{e_1, e_2, e_3\}$, $\{e_3, e_4, e_5\}$, $\{e_5, e_6, e_1\}$, $\{e_1, e_4, e_7\}$, $\{e_2, e_5, e_7\}$, $\{e_3, e_6, e_7\}$, $\{e_2, e_4, e_6\}$. Подмножество A множества E считается независимым, если $|A| \leq 2$ или $|A| = 3$ и A не является прямой в плоскости Фано.

5. **k-матроид.** Здесь E — произвольное конечное множество, и подмножество A множества E считается независимым, если и только если $|A| \leq k$.

Упражнение. Доказать, что каждый из перечисленных примеров является матроидом. Описать базы и циклы этих матроидов.

6. Матроид трансверсалей. Пусть $G = (V, W; R)$ — двудольный граф с долями V и W. Подмножество A множества V назовем (частичной) трансверсалью, если в G существует паросочетание, покрывающее все вершины множества A. По теореме Кенига-Оре множество A является трансверсалью, если и только если

$$|N(C)| \geq |C| \quad \forall\, C \subseteq A,$$

где $N(C) = \{w \in W : (v, w) \in R\}$.

Теорема 3.1 (Эдмондс и Фалкерсон). *Пусть $G = (V, W; R)$ — двудольный граф с долями V и W. Тогда пара $M = [V, \mathcal{I}]$, где $\mathcal{I} = \{A \subseteq V \mid A$— трансверсаль в $G\}$, является матроидом.*

ДОКАЗАТЕЛЬСТВО. Выполнение (M1) очевидно. Пусть $A, B \in \mathcal{I}$ и $|A| < |B|$. Выберем содержащее $|A|$ ребер паросочетание π_A, покрывающее A и содержащее $|B|$ ребер паросочетание π_B, покрывающее B, так, чтобы они имели максимально возможное число общих ребер. Для каждого $x \in A \cup B$ обозначим через $\pi_A(x)$ (соответственно через $\pi_B(x)$) вершину, соединенную с x ребром, принадлежащим $\pi_A(x)$ (соответственно, $\pi_B(x)$). Для $C \subseteq A \cup B$ определим $\pi_A(C) = \{\pi_A(x) \mid x \in C\}$ и $\pi_B(x) = \{\pi_B(x) \mid x \in C\}$). Если $\pi_B(b') \notin \pi_A(A)$ хотя бы для одного $b' \in B \backslash A$, то $A \cup \{b'\} \in \mathcal{I}$ и теорема доказана.

Пусть

$$\pi_B(\{B \backslash A\}) \subseteq \pi_A(A).$$

Поскольку $|B| > |A|$, найдется $b \in B$ с $\pi_B(b) \notin \pi_A(A)$. Согласно доказанному выше $b \in A \cap B$. Пусть $c = \pi_A(b)$. Тогда паросочетание $\pi'_A = (\pi_A \backslash \{(b, c)\}) \cup \{(b, \pi_B(b))\}$ имеет больше общих ребер с π_B чем π_A в противоречие с выбором π_A. ∇

Замечание. Изложенное здесь доказательство принадлежит В.Ню.

Теорема 3.2. *Пусть E — конечное множество, а $\mathcal{I}$ — непустое подсемейство подмножеств множества E, удовлетворяющее условию (M1). При этих предположениях $M = [E, \mathcal{I}]$ является матроидом, если и только если выполняется условие* **(M3)** Для каждого $C \subset E$ любые два максимальных (по включению) независимых подмножества множества C имеют одну и ту же мощность.

ДОКАЗАТЕЛЬСТВО. Допустим сначала, что не выполняется (M3). Это значит, что для некоторого $C \subset E$ найдутся максимальные (по включению) независимые

$A, B \subset E$ с $|A| < |B|$. Согласно (M2), $\exists\ e \in B\backslash A : A \cup \{e\} \in \mathcal{I}$. Это противоречит максимальности A.

Допустим теперь, что не выполняется (M2). Тогда найдутся такие $A, B \in \mathcal{I}$ с $|A| < |B|$, что $A \cup \{e\} \notin \mathcal{I}$ для любого $e \in B\backslash A$. Значит, для $C := A \cup B$ не выполняется (M3): A — максимальное по включению независимое подмножество множества C, но $|A| < |B|$. ∇

Следствие. В любом матроиде все базы имеют одну и ту же мощность.

Ввиду теоремы 3.2 на подмножествах множества E матроида M определена *ранговая функция*, сопоставляющая каждому $C \subset E$ мощность максимальных (по включению) независимых подмножеств C.

Теорема 3.3. *Пусть $M = [E, \mathcal{I}]$ является матроидом, а ρ — ранговая функция на 2^E. Тогда*

(R1) $0 \le \rho(A) \le |A|$;

(R2) $A \subset B \Rightarrow \rho(A) \le \rho(B)$;

(R3) $\rho(A \cup B) + \rho(A \cap B) \le \rho(A) + \rho(B)$;

(R4) $\rho(\emptyset) = 0$;

(R5) $\rho(A) \le \rho(A \cup \{e\}) \le \rho(A) + 1$;

(R6) если $\rho(A \cup \{e\}) = \rho(A \cup \{f\}) = \rho(A)$, то $\rho(A \cup \{e, f\}) = \rho(A)$.

ДОКАЗАТЕЛЬСТВО. Выполнение (R1), (R2), (R4) и (R5) очевидно. Допустим, что $\rho(A \cup \{e\}) = \rho(A \cup \{f\}) = \rho(A)$, но $\rho(A \cup \{e, f\}) = \rho(A)$. Пусть $I \subseteq A, I \in \mathcal{I}, |I| = \rho(A)$. По теореме 3.2 существует $F \subseteq A \cup \{e, f\}, F \in \mathcal{I}$ такое, что $|F| > |I|, F \supset I$. Но это противоречит допущению $\rho(A \cup \{e\}) = \rho(A \cup \{f\}) = \rho(A)$. Значит, (R6) верно.

Докажем (R3). Пусть $I \subseteq A \cap B, I \in \mathcal{I}, |I| = \rho(A \cap B)$. По теореме 3.2 существуют $F \subseteq A\backslash B$ и $H \subseteq B\backslash A$ такие, что $F \cup I \in \mathcal{I}, F \cup I \cap H \in \mathcal{I}, |F \cup I| = \rho(A), |F \cup I \cup H| = \rho(A \cup B)$. Но тогда $|H \cup I| \le \rho(B)$ и $\rho(A \cup B) + \rho(A \cap B) = |F| + |H| + 2|I| \le \rho(A) + \rho(B)$. ∇

Теорема 3.4. *Функция $\rho : 2^E \longrightarrow Z$ является ранговой функцией некоторого матроида на E тогда и только тогда, когда она удовлетворяет (R4), (R5) и (R6).*

ДОКАЗАТЕЛЬСТВО. Ввиду теоремы 3.3 достаточно доказать импликацию "только тогда". Положим $\mathcal{I} = \{A \subseteq E : \rho(A) = |A|\}$. Из-за (R4) имеем $\mathcal{I} \neq \emptyset$. Допустим, что $A \subset B, \rho(A) < |A|, B\backslash A = \{x_1, \ldots, x_k\}$. Согласно (R5), $\rho(B) \le 1 + \rho(B\backslash\{x_1\}) \le 2 + \rho(B\backslash\{x_1, x_2\}) \le \ldots \le k + \rho(B\backslash\{x_1, \ldots, x_k\}) = k + \rho(A) < |B|$. Таким образом, (M1) верно.

Допустим теперь, что (M2) неверно. Тогда для некоторых $A, B \in \mathcal{I}, A = \{x_1, \ldots, x_m, a_1, \ldots, a_k\}, B = \{x_1, \ldots, x_m, b_1, \ldots, b_p\}, p > k$ имеем $\rho(A \cup \{b_i\}) = \rho(A)$ при любом $1 \le i \le p$. Из (R6) получаем $\rho(A \cup \{b_1, b_i\}) = \rho(A \cup \{b_1\}) = \rho(A)$ при любом $2 \le i \le p$. Снова по (R6) $\rho(A \cup \{b_1, b_2, b_i\}) = \rho(A)$ при любом $3 \le i \le p$ и т.д. Через p шагов получаем $\rho(A \cup B) = \rho(A \cup \{b_1, b_2, \ldots, b_p\}) = \rho(A)$. Противоречие. ∇

Теорема 3.5. *Функция $\rho : 2^E \longrightarrow Z$ является ранговой функцией некоторого матроида на E тогда и только тогда, когда она удовлетворяет (R1), (R2) и (R3).*

ДОКАЗАТЕЛЬСТВО. Ввиду теорем 3.3 и 3.4 достаточно доказать, что выполнение (R1),(R2) и (R3) влечет выполнение (R4),(R5) и (R6).

Очевидно, (R4) следует из (R1), а левое неравенство в (R5) следует из (R2). Правое неравенство в (R5) и свойство (R6) следуют из (R3). ∇

Замечание. Для множества $\mathcal{B}$ баз произвольного матроида $M = [E, \mathcal{I}]$ верно

(B1) Для любых $B_1 \neq B_2$ и любого $x \in B_1 \backslash B_2$ найдется такой $y \in B_2 \backslash B_1$, что

$$(B_1 \backslash \{x\}) \cup \{y\} \in \mathcal{B}.$$

Теорема 3.6. *Пусть E — конечное множество, $\mathcal{B}$ — непустое подсемейство подмножеств множества E, удовлетворяющее условию (B1). Тогда множество $\mathcal{I} = \{A \subset E \mid \exists\, B \in \mathcal{B} : A \subset B\}$ удовлетворяет аксиомам (M1) и (M2).*

ДОКАЗАТЕЛЬСТВО. Выполнение (M1) очевидно.

Покажем, что все члены $\mathcal{B}$ равномощны. Допустим, что это не так. Выберем множества $B_1, B_2 \in \mathcal{B}$ разной мощности так, чтобы максимизировать мощность пересечения. Можно считать, что $|B_1| > |B_2|$. Выберем произвольно $x \in B_1 \backslash B_2$. Согласно (B1), найдется такой $y \in B_2 \backslash B_1$, что $B := (B_1 \backslash \{x\}) \cup \{y\} \in \mathcal{B}$. Но $|B \cap B_2| > |B_2 \cap B_1|, |B| = |B_1|$. Это противоречит выбору B_1 и B_2.

Итак, предположим, что (M2) неверно. Это значит, что для некоторых $A_1, A_2 \in \mathcal{I}$ с $|A_1| < |A_2|$,

$$A_1 \cup \{e\} \notin \mathcal{I} \qquad \forall\, e \in A_2 \backslash A_1. \tag{1}$$

Выберем $B_1, B_2 \in \mathcal{B}, B_1 \supset A_1, B_2 \supset A_2$ так, чтобы максимизировать $|B_2 \cap B_1|$. В силу (1),

$$B_1 \cap (A_2 \backslash A_1) = \emptyset \tag{2}$$

Если $B_1 \backslash A_1 \subset B_2$, то в силу (2) имеем $|B_2| \geq |B_1 \backslash A_1| + |A_2| > |B_1 \backslash A_1| + |A_1| = |B_1|$. Значит, найдется $x \in B_1 \backslash (A_1 \cup B_2)$. Согласно (B1), найдется такой $y \in B_2 \backslash B_1$, что $B := (B_1 \backslash \{x\}) \cup \{y\} \in \mathcal{B}$. Но $|B \cap B_2| > |B_2 \cap B_1|, |B \supset A_1|$. Это противоречит выбору B_1 и B_2. ∇

Из теоремы 3.6 следует, что для задания матроида достаточно задать набор баз, удовлетворяющий аксиоме (B1).

Теорема 3.7. *Пусть $M = [E, \mathcal{I}]$ — матроид. Тогда для любых членов C и D семейства $\mathcal{C}$ циклов в M выполняются условия:*

(C1) *если $C \subset D$, то $C = D$;*

(C2) *если $C \neq D$ и $e \in C \cap D$, то найдется цикл $F \subset (C \cup D) \backslash \{e\}$.*

И наоборот, если для любых множеств C и D из некоторого семейства $\mathcal{C}$ подмножеств E выполнены условия (C1) и (C2), то $\mathcal{C}$ является семейством циклов некоторого матроида на E.

ДОКАЗАТЕЛЬСТВО. "$\Rightarrow$"Выполнение (C1) очевидно. Докажем выполнение (C2). Пусть $C \neq D$ и $e \in C \cap D$. Обозначим $A = C \cap D$. По условию, $A \in \mathcal{I}$. Допустим, что $(C \cup D) \backslash \{e\}$ независимо. Тогда по теореме 3.2 найдется независимое множество $B \subset C \cup D$ с $|B| = |C \cup D| - 1$ такое, что $B \supset A$. Но в этом случае $B \supset C$ или $B \supset D$.

"$\Leftarrow$"Пусть для любых множеств C и D из некоторого семейства $\mathcal{C}$ подмножеств E выполнены условия (C1) и (C2). Обозначим $\mathcal{I} = \{A \subseteq E : \forall\, C \in \mathcal{C}\ C \backslash A \neq \emptyset\}$. Выполнение (M1) очевидно. Предположим, что условие (M2) не выполняется. Выберем пару (A, B) множеств из $\mathcal{I}$, противоречащих (M2), с максимальной мощностью пересечения, а среди таких — пару с минимальной мощностью A. Пусть $A =$

$\{x_1, \ldots, x_m, a_1, \ldots, a_k\}$, $B = \{x_1, \ldots, x_m, b_1, \ldots, b_p\}$, $p > k, A \cup \{b_i\} \notin \mathcal{I}$ при любом $1 \le i \le p$. Это означает, что

$$\forall\, i, 1 \le i \le p\ \exists\, C_i \in \mathcal{C} : C_i \subseteq A \cup \{b_i\}.$$

По выбору (A, B), для пары $(A \backslash \{a_1\}, B)$ найдется b_1 с $F = (A \backslash \{a_1\}) \cup \{b_1\} \in \mathcal{I}$. Снова по выбору (A, B) существует b_2 с $D = F \cup \{b_2\} \in \mathcal{I}$. Но по (C2) найдется $C \in \mathcal{C},\ C \subseteq (C_1 \cup C_2) \backslash \{a_1\} = D \backslash \{a_1\}$. ∇

Укажем на связь матроидов со структурами, на которых хорошо работают "жадные"алгоритмы.

Пусть E — конечное множество, $\mathcal{I}$ — подсемейство подмножеств множества E и $w : E \to R^+$ — функция, сопоставляющая каждому $e \in E$ его "вес"$w(e)$. Требуется найти

$$\max\{\sum_{e \in S} w(e) \mid S \in \mathcal{I}\} \tag{3}$$

и множество S, на котором максимум достигается.

"Жадный"алгоритм

```
Begin
    Упорядочить E так, чтобы w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_n);
    S := ∅;
    for i := 1 to n do
        if S ∪ {e_i} ∈ I then S := S ∪ {e_i}
end
```

Теорема Радо-Эдмондса. *Если $M = [E, \mathcal{I}]$ — матроид, то множество S, найденное "жадным"алгоритмом, является решением задачи (??). Напротив, если $M = [E, \mathcal{I}]$ — не матроид, то найдется функция $w : E \to R^+$, для которой это S не будет оптимальным.*

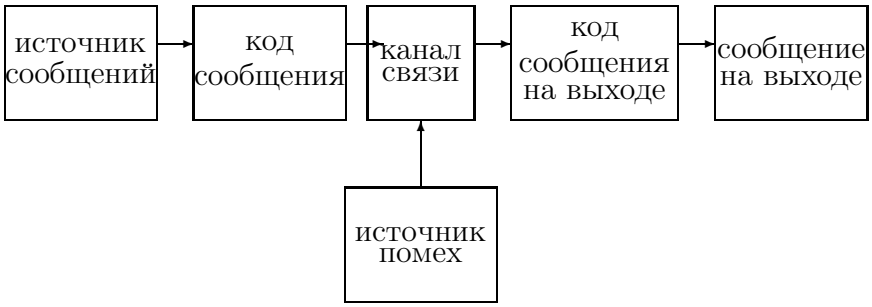
ДОКАЗАТЕЛЬСТВО. Пусть $M = [E, \mathcal{I}]$ — матроид, $S = \{s_1, \ldots, s_k\}$ — множество, найденное "жадным"алгоритмом; причем $w(s_1) \ge w(s_2) \ge \ldots \ge w(s_k)$. Отметим сначала, что S — база матроида M, поскольку отбрасывались лишь элементы, зависимые от S. Пусть $T = \{t_1, \ldots, t_k\}$ — другая база M и $w(t_1) \ge w(t_2) \ge \ldots, w(t_k)$. Предположим, что $w(t_i) > w(s_i)$ для некоторого i. Рассмотрим независимые множества $A := \{s_1, \ldots, s_{i-1}\}$ и $B := \{t_1, \ldots, t_i\}$. Согласно (M2) найдется такое $j, 1 \le j \le i$, что $A \cup \{t_j\} \in \mathcal{I}$. Но $w(t_j) \ge w(t_i) > w(s_i)$. Значит, $w(t_i) \le w(s_i)$ для каждого i, и S — решение (??).

Предположим теперь, что $M = [E, \mathcal{I}]$ не является матроидом.

Случай 1. Найдутся $B \in \mathcal{I}$ и $A \notin \mathcal{I}$ такие, что $A \subset B$. Положим

$$w(e) = \begin{cases} 2, & e \in A, \\ 1, & e \in B \setminus A, \\ 0, & e \in E \setminus B. \end{cases}$$

Понятно, что искомым множеством в задаче (??) является B и $\sum\limits_{e \in B} w(e) = |B| + |A|$. Но при работе "жадного"алгоритма кандидатами в S вначале будут рассматриваться элементы множества A, и хотя бы один из них не будет включен в S.



Случай 2. Аксиома (M1) выполняется для $\mathcal{I}$, но найдутся $B \in \mathcal{I}$ и $A \in \mathcal{I}$ такие, что $|A| < |B|$ и $A \cup \{e\} \notin \mathcal{I}$ для любого $e \in B \setminus A$. Обозначим $m = |B|$. Положим

$$w(e) = \begin{cases} 1 + 1/m, & e \in A, \\ 1, & e \in B \setminus A, \\ 0, & e \in E \setminus B. \end{cases}$$

Тогда "жадный"алгоритм сначала включит в S все элементы множества A, а затем ни один элемент множества $B \setminus A$ не добавится к S. Следовательно, суммарный вес элементов множества S будет равен $|A|(m+1)/m \leq (m-1)(m+1)/m < m$. Но оптимальное решение задачи (**??**) не меньше чем $|B| = m$. ∇

4 КОДИРОВАНИЕ

4.1 Задачи и понятия теории кодирования

За основу этой главы взята часть III книги С.В.Яблонского “Введение в дискретную математику”.

Кодирование (в широком смысле) — один из самых распространенных видов человеческой деятельности: мы кодируем наши мысли, намерения и чувства словами (разных языков), кодируем нашу речь буквами, иероглифами и рисунками, тексты кодируются для записи в ЭВМ и т.д. Много вопросов о кодировании возникает при передаче сообщений. Они могут быть проиллюстрированы приводимой здесь схемой.

В этой схеме источник сообщений хочет передать по каналу связи некоторый набор *слов* — то есть конечных последовательностей символов из заданного конечного алфавита $\mathcal{A} = \{a_1, \ldots, a_r\}$. Для передачи ему нужно (или он хочет) *закодировать* это сообщение — переписать его словами во вспомогательном алфавите $\mathcal{B} = \{b_1, \ldots, b_q\}$. После получения сообщения (возможно искаженного помехами) его нужно снова записать словами в алфавите $\mathcal{A}$ (возможно исправив возникшие ошибки).

Выбор кодов связан с различными обстоятельствами, а именно:

- с удобством передачи кодов,

- со стремлением увеличить пропускную способность канала,
- с удобством обработки кодов,
- с обеспечением помехоустойчивости,
- с удобством декодирования,
- с необходимостью однозначного декодирования,
- с другими возможными требованиями к кодам.

Ниже будут рассматриваться два вида кодирования:

(а) *Алфавитное кодирование.* Каждой букве a_i из $\mathcal{A} = \{a_1, \ldots, a_r\}$ ставится в соответствие некоторое слово B_i из алфавита $\mathcal{B} = \{b_1, \ldots, b_q\}$. Схема кодирования, сопоставляющая эти слова, будет обозначаться буквой Σ.

(б) *Равномерное кодирование.* Некоторое слово B_i из алфавита $\mathcal{B}$ ставится в соответствие не букве, а какому-то слову A_i фиксированной длины в алфавите $\mathcal{A}$.

Конечно, одно из первых требований к используемому коду — требование однозначности восстановления сообщения по его коду.

4.2 Проверка однозначности декодирования

Рассмотрим алфавитные коды. Каждое из слов $B_i, i = 1, \ldots, r$, называется *элементарным кодом*. Слово в алфавите $\mathcal{B}$ назовем кодовым, если его можно *расшифровать*, т.е. разбить на элементарные коды. Одна из трудностей проверки однозначности декодирования состоит в том, что формально надо проверять бесконечное число кодовых слов. Оказывается, этой бесконечности можно избежать. Введем несколько обозначений.

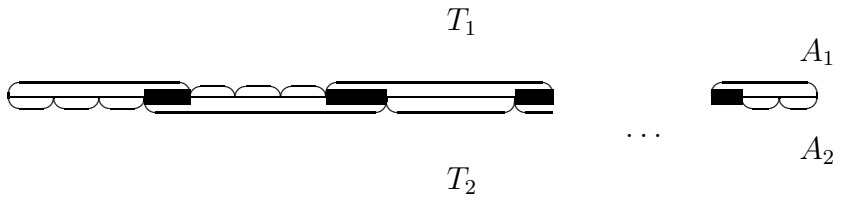
Пусть дана схема кодирования Σ, l_i — длина слова B_i, $L = l_1 + \ldots + l_q$. Назовем *нетривиальным разложением* слова B_i его представление в виде $B_i = \beta' B_{j_1} \ldots B_{j_w} \beta''$, где $B_{j_1} \neq B_i$, β'' является началом какого-нибудь элементарного кода, а β' является концом какого-нибудь элементарного кода. Слова β' и β'' могут быть пустыми.

Очевидно, что для каждого i число нетривиальных разложений слова B_i конечно. Обозначим через W максимум чисел w, взятый по всем нетривиальным разложениям всех слов $B_i, i = 1, \ldots, r$.

Теорема 4.1 *Для любой схемы кодирования Σ найдется такое $N = N(\Sigma)$, что для проверки однозначности декодирования в Σ достаточно проверить коды слов из $\mathcal{A}$ длины не более N, и*

$$N \leq \lfloor (W+1)(L-r+2)/2 \rfloor.$$

Доказательство. Допустим, что теорема неверна. Выберем самое короткое слово B в алфавите $\mathcal{B}$, допускающее две различные расшифровки A_1 и A_2. С ними связаны два разбиения слова B на элементарные коды T_1 и T_2 (см. рис.):



Обозначим через T произведение разбиений T_1 и T_2, т. е. разбиение, полученное после "разрезания" B там, где его "разрезало" хотя бы одно из разбиений T_1 и T_2. Части разбиения T разделим на два класса: к первому отнесем части, являющиеся элементарными кодами, ко второму — все остальные. Заметим, что каждая часть β, принадлежащая второму классу, является концом одного из элементарных кодов и началом другого. Причем если β оканчивает некоторое элементарное кодовое слово в T_1, то оно начинает какое-то элементарное кодовое слово в T_2 и наоборот (см. рис.). Более точно, если $B = B'\beta B''$, то либо $B'\beta$ и B'' являются кодовыми словами в T_1, а B' и $\beta B''$ являются кодовыми словами в T_2, либо наоборот.

Покажем, что все части из второго класса различны. Допустим, что $B = B'\beta B''\beta B'''$. Тогда слово $B'\beta B'''$ имеет две расшифровки в противоречие с выбором B. Чтобы убедиться в этом, заметим, что согласно вышесказанному, слова $B'\beta$, B', $\beta B'''$ и B''' являются кодовыми.

Число частей из второго класса не превосходит числа непустых начал элементарных кодов, т.е. $(l_1 - 1) + \ldots + (l_r - 1) = L - r$. Каждый из кусков, на которые разбивается B после выбрасывания всех частей из второго класса, является кодовым словом, входящим в одно из разбиений T_i, и частью некоторого элементарного кода, входящего в T_{3-i}. Причем соседние куски являются частями элементарных кодов, входящих в различные T_i.

Таким образом, имеем не более $L - r + 1$ кусков, и в каждом из T_i как минимум $\lfloor (L-r+1)/2 \rfloor$ кусков вместе с соседними частями из второго класса образуют элементарный код в расшифровке A_i. Следовательно, длина каждого из A_i не превосходит

$$W \cdot \lceil (L - r + 1)/2 \rceil + 1 \cdot \lfloor (L - r + 1)/2 \rfloor \leq \lfloor (W + 1)(L - r + 2)/2 \rfloor. \; quad\nabla$$

Итак, проблема верификации однозначности декодирования заданной схемы является конечной. Но если, например, $r = 6, W = 3, L = 20$, то $\lfloor (W+1)(L-r+2)/2 \rfloor = \lfloor 4 \cdot 16/2 \rfloor = 32$ и требуется проверить 6^{32} слов. Это очень много. К счастью, из доказательства теоремы можно извлечь существенно более эффективный алгоритм.

Пусть дана схема кодирования Σ. Для каждого элементарного кода B_i рассмотрим все его нетривиальные разложения

$$B_i = \beta' B_{j_1} \ldots B_{j_w} \beta''. \tag{1}$$

Обозначим через $V = V(\Sigma)$ множество, содержащее пустое слово Λ и слова β, встречающиеся в разложениях вида (**??**) как в виде начал, так и в виде окончаний. Построим далее помеченный ориентированный граф $\Gamma = \Gamma(\Sigma)$ по следующим

правилам. Множеством вершин графа Γ является $V = V(\Sigma)$. Проводим дугу из вершины $\beta' \in V$ в вершину $\beta'' \in V$, если и только если в некотором разложении вида (**??**) β' является началом, а β'' — концом. При этом дуга (β', β'') помечается словом $B_{j_1} \dots B_{j_w}$.

Теорема 4.2 *Схема кодирования Σ не обладает свойством однозначности декодирования тогда и только тогда, когда граф $\Gamma(\Sigma)$ содержит контур, проходящий через вершину Λ.*

Доказательство. Допустим, что Σ не обладает свойством однозначности декодирования. Тогда, как следует из доказательства теоремы 4.1, кратчайшее слово, имеющее две расшифровки в схеме Σ, имеет вид

$$B = B_{i_{1,1}} \dots B_{i_{1,k(1)}} \beta_1 B_{i_{2,1}} \dots B_{i_{2,k(2)}} \beta_2 \dots \beta_{s-1} B_{i_{s,1}} \dots B_{i_{s,k(s)}},$$

где все β_i различны и слова $B_{i_{1,1}} \dots B_{i_{1,k(1)}} \beta_1$, $\beta_1 B_{i_{2,1}} \dots B_{i_{2,k(2)}} \beta_2$, …, $\beta_{s-1} B_{i_{s,1}} \dots B_{i_{s,k(s)}}$ являются элементарными кодами. Это значит, что в $\Gamma(\Sigma)$ есть контур, проходящий через вершины $\Lambda, \beta_1, \dots, \beta_{s-1}$.

Обратно, пусть в $\Gamma(\Sigma)$ существует контур, проходящий через вершины β_0, β_1, …, β_{s-1}, где $\beta_0 = \Lambda$ и дуга (β_j, β_{j+1}), $j = 0, 1, \dots, s-1$ $((s-1)+1 = 0)$, помечена словом $B_{i_{j+1,1}} \dots B_{i_{j+1,k(j+1)}}$. Тогда слово

$$B = B_{i_{1,1}} \dots B_{i_{1,k(1)}} \beta_1 B_{i_{2,1}} \dots B_{i_{2,k(2)}} \beta_2 \dots \beta_{s-1} B_{i_{s,1}} \dots B_{i_{s,k(s)}}$$

имеет две различные расшифровки. ∇

4.3 Префиксные коды

Важным классом однозначно декодируемых кодов являются *префиксные коды* — такие алфавитные коды, где ни один элементарный код не является *префиксом* (т.е. началом) другого элементарного кода.

Упражнение. Доказать, что любой префиксный код является однозначно декодируемым.

Нам потребуется следующий факт.

Теорема 4.3 (Неравенство Макмиллана) *Если схема кодирования Σ обладает свойством однозначности декодирования, то*

$$\sum_{i=1}^{r} q^{-l_i} \leq 1. \tag{2}$$

ДОКАЗАТЕЛЬСТВО. Выберем произвольное n. Рассмотрим коды всех r^n слов длины n в алфавите $\mathcal{A}$, полученные с помощью Σ. Поскольку Σ обладает свойством однозначности декодирования, все r^n кодовых слов различны. Обозначим через $l(B)$ длину слова B и рассмотрим величину $S = \sum\limits_{B} q^{-l(B)}$, где суммирование ведется по всем кодам B слов длины n в алфавите $\mathcal{A}$, полученных с помощью Σ. Пусть $l = \max\{l_i \mid 1 \leq i \leq r\}$ и $\nu(n,t)$ — число кодовых слов длины t. Тогда $\nu(n,t) \leq q^t$ и длина каждого из наших кодовых слов не превосходит nl. Следовательно,

$$S = \sum_{t=1}^{nl} \frac{\nu(n,t)}{q^t} \leq nl. \tag{3}$$

С другой стороны, поскольку каждое кодовое слово состоит из n элементарных кодов,

$$S = \sum_{(i_1,\ldots,i_n)} q^{-l_{i_1}-\ldots-l_{i_n}},$$

где суммирование ведется по всем r^n наборам длины n чисел из диапазона $\{1,\ldots,r\}$. Но последняя сумма равна $(q^{-l_1}+\ldots+q^{-l_r})^n$. Сравнивая с (**??**), получаем

$$q^{-l_1}+\ldots+q^{-l_r} \le (nl)^{1/n}.$$

Это неравенство справедливо для любого n, а его правая часть стремится к 1 при $n \to \infty$. Поскольку его левая часть не зависит от n, необходимо, чтобы

$$q^{-l_1}+\ldots+q^{-l_r} \le 1. \quad \nabla$$

Следующий факт характеризует префиксные коды с положительной стороны.

Теорема 4.4 *Если схема кодирования Σ обладает свойством однозначности декодирования, то существует такая префиксная схема кодирования Σ', что для каждого $i, i = 1,\ldots,s$ длина l'_i элементарного кода B'_i в Σ' равна длине l_i элементарного кода B_i в Σ.*

ДОКАЗАТЕЛЬСТВО. Можно считать, что элементарные коды B_i занумерованы в порядке неубывания их длин. Пусть длинами элементарных кодов в Σ являются числа $\lambda_1,\ldots,\lambda_s$, $\lambda_1 < \lambda_2 \ldots < \lambda_s$ и число элементарных кодов длины λ_i, $i = 1,\ldots,s$ равно ν_i. Тогда неравенство Макмиллана можно переписать в виде

$$S = \sum_{t=1}^{s} \frac{\nu_t}{q^{\lambda_t}} \le 1. \tag{4}$$

В частности, $\nu_1/q^{\lambda_1} \le 1$, откуда $\nu_1 \le q^{\lambda_1}$. Выберем среди q^{λ_1} слов длины λ_1 в алфавите $\mathcal{B}$ произвольные ν_1 слов в качестве элементарных кодов $B'_1,\ldots,B'_{\nu_1}$.

Перейдем к словам длины λ_2. Из (**??**) получаем

$$\frac{\nu_1}{q^{\lambda_1}} + \frac{\nu_2}{q^{\lambda_2}} \le 1,$$

$$\nu_2 \le q^{\lambda_2} - \nu_1 q^{\lambda_2-\lambda_1}. \tag{5}$$

Рассмотрим множество слов длины λ_2 в алфавите $\mathcal{B}$, не начинающихся с $B'_1,\ldots,B'_{\nu_1}$. В силу (**??**) из этого множества можно выбрать ν_2 каких-нибудь слов в качестве элементарных кодов $B'_{\nu_1+1},\ldots,B'_{\nu_1+\nu_2}$.

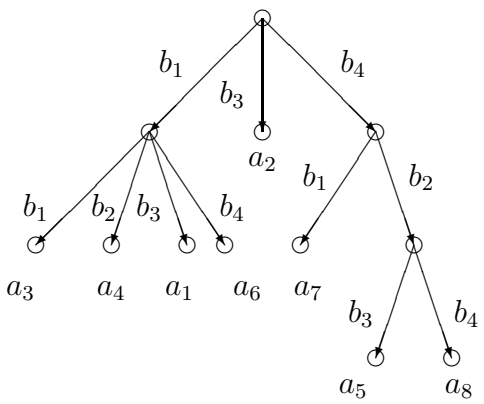
Далее из (**??**) получаем

$$\nu_3 \le q^{\lambda_3} - \nu_1 q^{\lambda_3-\lambda_1} - \nu_2 q^{\lambda_3--\lambda_2}$$

и строим ν_3 слов длины λ_3, не начинающихся с $B'_1,\ldots,B'_{\nu_1+\nu_2}$ и т.д.

Через конечное число шагов построим нужное количество слов нужной длины. По построению новый код будет префиксным. ∇

a_1 – b_1b_3
a_2 – b_3
a_3 – b_1b_1
a_4 – b_1b_2
a_5 – $b_4b_2b_3$
a_6 – b_1b_4
a_7 – b_4b_1
a_8 – $b_4b_2b_4$



4.4 Коды с минимальной избыточностью

При выборе схемы кодирования естественно учитывать экономичность, т.е. средние затраты времени на передачу и прием сообщений.

Предположим, что задан алфавит $\mathcal{A} = \{a_1, \ldots, a_r\}, r \geq 2$, и набор вероятностей $(p_1, \ldots, p_r), \quad p_1 + \ldots + p_r = 1$ появлений букв $a_1, \ldots, a_r$. Тогда *избыточностью кодирования схемой* Σ называется величина

$$l_{\text{ср}} = l_{\text{ср}}(\Sigma) = a_1p_1 + \ldots + a_rp_r,$$

т.е. математическое ожидание длины элементарного кода.

Понятно, что чем меньше $l_{\text{ср}}$, тем экономнее в среднем схема Σ. Пусть $l_* = l_*(a_1, \ldots, a_r, p_1, \ldots, p_r) = \inf l_{\text{ср}}$, где инфимум взят по всем однозначно декодируемым схемам.

Пусть $k = \lceil \log_q r \rceil$. Тогда все a_i можно закодировать разными словами длины k в алфавите $\mathcal{B}$. Очевидно, такое кодирование будет префиксным (а, следовательно, и взаимно однозначным). Отсюда $l_* \leq k$. Таким образом, при поиске схем с $l_{\text{ср}} < k$ достаточно рассматривать для i с $p_i > 0$ элементарные коды B_i только длины, меньшей k/p_i. Поскольку длины элементарных кодов B_i для i с $p_i = 0$ не влияют на $l_{\text{ср}}$, достаточно рассмотреть конечное число вариантов. Это значит, что значение l_* достигается на некоторой схеме.

Коды, определяемые схемами Σ с $l_{\text{ср}} = l_*$, называются *кодами с минимальной избыточностью*. Согласно теореме 4.4 существуют префиксные коды с минмальной избыточностью. Изучим структуру префиксных кодов с минмальной избыточностью.

Каждому префиксному коду (со схемой Σ) поставим в соответствие *кодовое дерево* — ориентированное корневое дерево $T = T(\Sigma)$ по следующим правилам. Множество вершин $V(T)$ дерева T состоит из элементарных кодов и всех их префиксов, включая пустое слово. Дуга в T ведет из C в D, если C является префиксом D и короче D ровно на одну букву (см. пример).

Элементарные коды соответствуют висячим вершинам в T.

Итак, пусть T — кодовое дерево префиксного кода с минимальной избыточностью (со схемой Σ). Можно считать, что $p_1 \geq \ldots \geq p_r$. Тогда можно преобразовать Σ таким образом, чтобы

(а) $i < j \Longrightarrow l_i \le l_j$;
(б) порядки ветвления всех его вершин, за исключением быть может одной, лежащей в предпоследнем ярусе, равны или 0, или q;
(в) порядок ветвления q_0 исключительной вершины (если она есть) не равен 1.

Доказательство этих свойств оставляется в качестве упражнения.

Если порядки ветвления всех вершин T равны или 0, или q, то положим $q_0 = q$. Ввиду (б), по индукции легко видеть, что для некоторого целого t имеем $r = t(q-1) + q_0$. Следовательно, если h — остаток от деления r на $q-1$, то

$$q_0 = \begin{cases} h, & \text{если } h \ge 2, \\ q, & \text{если } h = 1, \\ q-1, & \text{если } h = 0. \end{cases} \qquad (6)$$

Нетрудно видеть, что можно выбрать такой префиксный код с минимальной избыточностью, кодовое дерево которого кроме (а)–(в) обладает свойством
(г) для некоторой вершины v, лежащей в предпоследнем ярусе, порядок ветвления вершины v равен q_0, а потомками v являются a_r, a_{r-1}, …, a_{r-q_0+1}.

Теорема 4.5 *Пусть схема кодирования Σ задает код с минимальной избыточностью для алфавита $\mathcal{A} = \{a_1, \ldots, a_r\}$ и набора вероятностей $(p_1, \ldots, p_r)$, а ее кодовое дерево T удовлетворяет свойствам (а)-(г). Обозначим $p'_{r-q_0+1} = p_r + p_{r-1} + \ldots + p_{r-q_0+1}$, а через T' — кодовое дерево, полученное из T удалением вершин $a_r, a_{r-1}, \ldots, a_{r-q_0+1}$ и сопоставлением образовавшейся висячей вершине v буквы a'_{r-q_0+1}. Тогда T' является кодовым деревом кода с минимальной избыточностью для алфавита $\mathcal{A}' = \{a_1, a_2, \ldots, a_{r-q_0}, a'_{r-q_0+1}\}$ и набора вероятностей $(p_1, p_2, \ldots, p_{r-q_0}, p'_{r-q_0+1})$.*

ДОКАЗАТЕЛЬСТВО. Обозначим схему, которой соответствует T', через Σ', номер уровня вершины v через m. Тогда $l_{\text{ср}}(\Sigma') = l_{\text{ср}}(\Sigma) - (m+1)(p_r + p_{r-1} + \ldots + p_{r-q_0+1}) + mp'_{r-q_0+1} = l_{\text{ср}}(\Sigma) - p'_{r-q_0+1}$.

Если бы для алфавита $\mathcal{A}' = \{a_1, \ldots, a_{r-q_0}, a'_{r-q_0+1}\}$ и набора вероятностей $(p_1, \ldots, p_{r-q_0}, p'_{r-q_0+1})$ нашлась схема Θ' префиксного кодирования с меньшей избыточностью чем $l_{\text{ср}}(\Sigma) - p'_{r-q_0+1}$, то подвесив в кодовом дереве для Θ' к вершине v вершины $a_r, a_{r-1}, \ldots, a_{r-q_0+1}$, получили бы схему Θ префиксного кодирования для алфавита $\mathcal{A} = \{a_1, \ldots, a_r\}$ с $l_{\text{ср}}(\Theta) = l_{\text{ср}}(\Theta) + p'_{r-q_0+1} < l_{\text{ср}}(\Sigma)$. Противоречие с выбором Σ завершает доказательство теоремы. ∇

Данная теорема в сочетании с предыдущими леммами дает следующий алгоритм построения кодов с минимальной избыточностью.

ПРЯМОЙ ХОД

1. Если $r = 1$, то переходим к обратному ходу.
2. Упорядочим вероятности так, чтобы $p_1 \ge \ldots \ge p_r$.
3. Выберем q_0 по правилу (**??**), удалим из списка вероятностей p_r, p_{r-1}, …, p_{r-q_0+1} и добавим $p'_{r-q_0+1} = p_r + p_{r-1} + \ldots + p_{r-q_0+1}$. Положим $r = r - q_0 + 1$, уберем штрих с p'_r и перейдем к шагу 1.

ОБРАТНЫЙ ХОД

Кодовым деревом для одной буквы является одна вершина. В порядке, обратном к тому, в котором склеивались вероятности, расклеиваем вершины кодового дерева.

4.5 Самокорректирующиеся коды

Рассмотрим одну из простейших ситуаций, когда сообщение может искажаться в канале связи. Пусть $\mathcal{A} = \{0, 1\}$ — алфавит, содержащий два символа. Предположим, что в канале связи действует источник помех, который в слове длины l искажает не более p символов. Возникает вопрос: для какого m можно все m-буквенные слова в алфавите $\mathcal{A}$ закодировать l-буквенными словами так, чтобы по коду на выходе закодированные слова однозначно восстанавливались? И как это сделать?

Если $l > 2p$, то $m \geq \lfloor l/(2p+1) \rfloor$. Действительно, каждую букву можно писать $2p+1$ раз подряд. Но такое кодирование не является самым экономным. Ниже будет построен *код Хэмминга* для $p = 1$.

1.Кодирование

При передаче слова $\beta_1\beta_2 \dots \beta_l$ и не более чем одном искажении на выходе может оказаться одно из $l+1$ различных слов. Поскольку для разных кодовых слов эти группы по $l+1$ слов не должны пересекаться, необходимо, чтобы

$$(l+1)2^m \leq 2^l, \quad \text{т.е.} \quad m \leq l - \log_2 l. \tag{7}$$

Выберем наибольшее m, удовлетворяющее (**??**). Обозначим $k = l - m$. Слову $\alpha_1\alpha_2 \dots \alpha_m$ ставится в соответствие слово $\beta_1\beta_2 \dots \beta_l$ по следующим правилам.

Буквы β_i слова $\beta_1\beta_2 \dots \beta_l$, у которых индекс i принадлежит $\{1, 2, 2^2, \dots, 2^{k-1}\}$, будут *контрольными символами*. Остальные m символов будут *информационными*: просто поставим символы $\alpha_1, \alpha_2, \dots, \alpha_m$ на эти места. Затем переходим к определению контрольных символов. Пусть V_i — множество таких натуральных чисел от 1 до l, в двоичной записи которых на i-м месте стоит 1. Для $i = 1, \dots, k$ положим

$$\beta_{2^{i-1}} = \sum_{j \in V_i \setminus \{2^{i-1}\}} \beta_j \pmod 2.$$

Тогда для $i = 1, \dots, k$ имеем

$$\sum_{j \in V_i} \beta_j = 0 \pmod 2.$$

2.Обнаружение и исправление ошибок

Допустим, что мы передали закодированное так слово $\beta_1\beta_2 \dots \beta_l$ и получили слово $\beta'_1\beta'_2 \dots \beta'_l = \beta_1\beta_2 \dots \bar{\beta}_s \dots \beta_l$. Пусть $x_k \dots x_1$ — двоичная запись числа s. Рассмотрим, чему будут равны $\sum_{j \in V_i} \beta'_j$ для $i = 1, \dots, k$ в зависимости от s. Пусть $x_k \dots x_1$ — двоичная запись числа s.

Случай 1. $x_i = 0$. Тогда

$$\sum_{j \in V_i} \beta'_j = \sum_{j \in V_i} \beta_j = 0 (\mathrm{mod}\ 2).$$

Случай 2. $x_i = 1$. Тогда $\sum_{j \in V_i} \beta'_j = \sum_{j \in V_i} \beta'_j - \sum_{j \in V_i} \beta_j = \beta'_i - \beta_i = 1 \pmod 2$.

Это значит, что выписав подряд справа налево значения $\sum_{j \in V_i} \beta'_j$ для $i = 1, \dots, k$, мы получим последовательность $x_k \dots x_1$, т.е. двоичную запись числа s — номера символа, который передан с ошибкой. Заменяем неправильный символ на правильный. Если ошибок не было, то все суммы $\sum_{j \in V_i} \beta'_j$ равны нулю.

3. Декодирование

Удаляем проверочные символы и оставляем информационные.

Содержание