



Санкт-Петербургский
государственный
университет
www.spbu.ru

Билеты по дискретной математике

1-2 семестр, преподаватель Григорьева Н. С.¹
Записали Костин П.А. и Щукин И.В.²

¹Также были использованы учебник Романовский И.В. "Дискретный анализ" и интернет

²Данный документ неидеальный, прошу сообщать о найденных недочетах [в контакте](#) (можно присыпать скрины неточностей с указанием билетов)

Содержание

1 Некоторые определения из теории множеств. Прямое произведение, разбиение множеств. Мощность объединения	6
2 Вектора из нулей и единиц	10
3 Алгоритм перебора 0-1 векторов. Коды Грея	11
4 Перебор элементов прямого произведения множеств	12
5 Размещения, сочетания, перестановки без повторений	13
6 Размещения, сочетания, перестановки с повторениями	14
7 Два алгоритма перебора перестановок. Нумерация перестановок	15
8 Задача о минимуме скалярного произведения	18
9 Числа Фибоначчи. Теорема о представлении	19
10 Перебор сочетаний. Нумерация сочетаний	21
11 Бином Ньютона и его комбинаторное использование	22
12 Свойства биномиальных коэффициентов	23
13 Основные определения теории вероятностей	24
14 Условные вероятности и формула Байеса	26
15 Математическое ожидание и дисперсия случайной величины	28
16 Схема Бернулли	30
17 Случайные числа. Схема Уолкера	31
18 Двоичный поиск и неравенство Крафта	33
19 Энтропия. 2 леммы	36
20 Теорема об энтропии	38

21 Операции над строками переменной длины	39
22 Поиск образца в строке (Карпа-Рабина, Бойера-Мура)	42
23 Сuffixное дерево	44
24 Задача о максимальном совпадении двух строк	45
25 Код Шеннона-Фано. Алгоритм Хаффмена. 3 леммы	46
26 Сжатие информации по методу Зива-Лемпеля	48
27 Метод Барроуза-Уилера	49
28 Избыточное кодирование. Коды Хэмминга	51
29 Шифрование с открытым ключом	52
30 Сортировки (5 методов)	54
31 Информационный поиск и организация информации	57
32 Хеширование	58
33 АВЛ-деревья	59
34 В-деревья	62
35 Биномиальные кучи	65
36 Основные определения теории графов	67
37 Построение транзитивного замыкания	71
38 Обходы графа в ширину и глубину. Топологическая сортировка	72
39 Связность. Компоненты связности и сильной связности	76
40 Алгоритм поиска контура и построение диаграммы порядка	78
41 Теорема о связном подграфе	80

42 Деревья. Теорема о шести эквивалентных определениях дерева	81
43 Задача о кратчайшем остовном дереве. Алгоритм Прима с.208	82
44 Алгоритм Краскала	84
45 Задача о кратчайшем пути. Алгоритм Дейкстры	86
46 Алгоритм Левита	89
47 Задача о кратчайшем дереве путей	90
48 Сетевой график и критические пути. Нахождение резервов работ	94
49 Задача о максимальном паросочетании в графе. Алгоритм построения	102
50 Теорема Кенига	111
51 Алгоритм построения контролирующего множества	113
52 Задача о назначениях. Венгерский метод	116
53 Задача коммивояжера. Метод ветвей и границ	119
54 Метод динамического программирования. Задача линейного раскроя	122
55 Приближенные методы решения дискретных задач. Жадные алгоритмы	124
56 Алгоритмы с гарантированной оценкой точности. Алгоритм Эйлера	125
57 Жадные алгоритмы. Задача о системе различных представителей	126
58 Приближенные методы решения дискретных задач	127
59 Конечные автоматы	130

60 Числа Фибоначчи. Производящие функции 137

61 Числа Каталана 139

1 Некоторые определения из теории множеств. Прямое произведение, разбиение множеств. Мощность объединения

Опр

Пустое множество (\emptyset) - мн-во, которому \notin ни один элемент

Опр

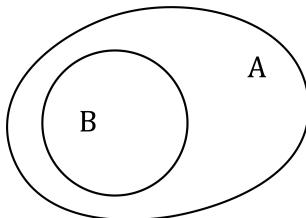
Число элементов мн-ва A - мощность $|A|$

Опр

Множество чисел от k до l обозначается $k : l$

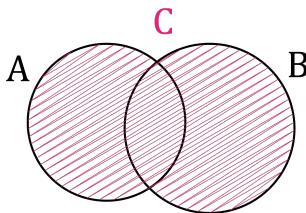
Опр

Мн-во A - подмн-во мн-ва B ($A \subset B$), если каждый элемент из A принадлежит B



Опр

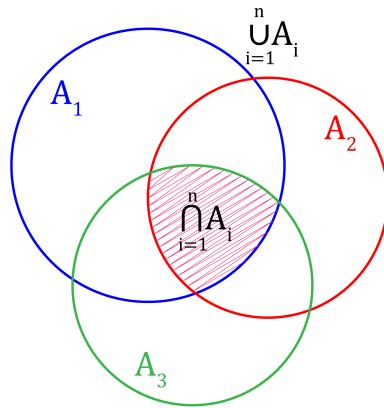
C - объединение A и B ($A \cup B$), если оно состоит из всех элементов A и B ($C = \{x | x \in A \text{ и } x \in B\}$)



Опр

$\bigcup_{i=1}^n A_i$, $\bigcap_{i=1}^n A_i$ - объединение и пересечение конечного числа мн-в

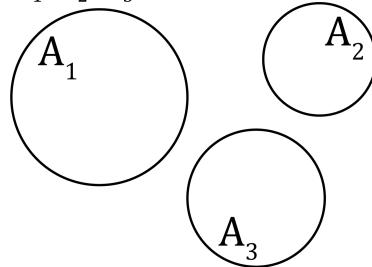
$(\bigcup_{i \in I} A_i, \bigcap_{i \in I} A_i)$ - аналогично



Опр

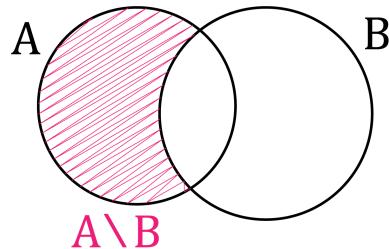
Если пересечение мн-в пусто, то они называются дизъюнктивными

A_1, A_2, A_3 - дизъюнктивны



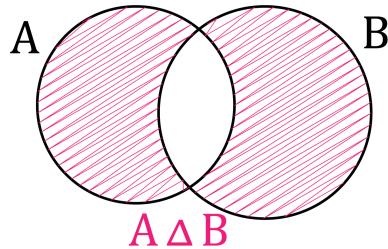
Опр

Мн-во С называется разностью мн-в А и В ($C = A \setminus B$), если оно состоит из всех эл-в, принадлежащих А и не принадлежащих В



Опр

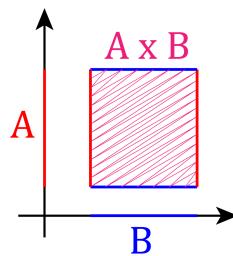
$A \Delta B = A \setminus B \cup B \setminus A$ - симметрическая разность



Опр

Мн-во упорядоченных пар (i, j) , где $i \in A$, $j \in B$ называется прямым произведением мн-в А и В

$$A \times B = \{(i, j) \mid i \in A, j \in B\}$$



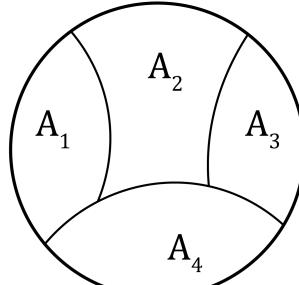
Замечание

Мощность прямого произведения $|A \times B| = |A| \cdot |B|$. Аналогично произведение \forall конечного числа множеств

Опр

Пусть A_1, \dots, A_k - ненулевые и попарно дизъюнктивные, $M = A_1 \cup \dots \cup A_k$, тогда мн-во $\{A_1, \dots, A_k\}$ называется разбиением M
(если они попарно не дизъюнктивны, тогда это покрытие)

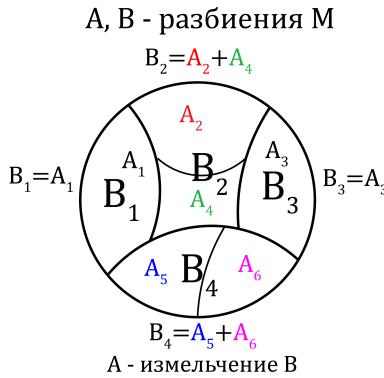
$$M = A_1 \cup A_2 \cup A_3 \cup A_4$$



A_1, A_2, A_3, A_4 - дизъюнктивны

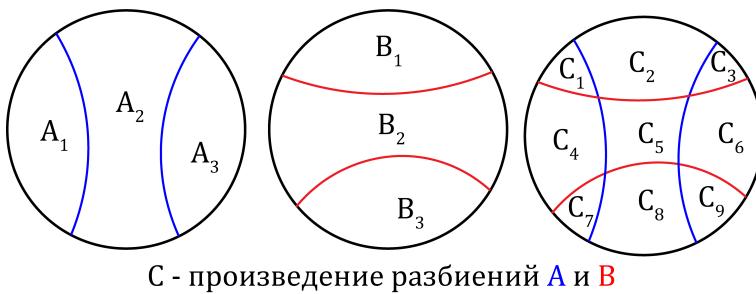
Опр

Разбиение А мн-ва М называется измельчением В, если $\forall A_i \in A$ содержится в некотором $B_j \in B$



Опр

Пусть А, В - размельчения мн-ва М, разбиение С называется произведением А и В, если оно является измельчением, причем самым крупным $C = A \cdot B$



Теорема

Произведение двух разбиений существует

Док-во

Предъявим разбиение, которое будет пересечением $A = \{A_1, \dots, A_k\}$ и $B = \{B_1, \dots, B_l\}$, точнее $D_{ij} = A_i \cap B_j, \quad i \leq k, \quad j \leq l$

$$\Rightarrow \mathcal{P} = \cup D_{ij} \text{ (т.е. без пустых строк)}$$

Покажем, что тогда оно самое крупное

Пусть $\exists F = \{F_1, \dots, F_t\}$ - измельчение А и В, тогда:

$$\forall F_k \quad \exists A_{i_k}, B_{i_k} : F_k \subset A_{i_k}, B_{i_k} \Rightarrow F_k \subset (A_{i_k} \cup B_{i_k}) = D_{i_k j_k} \Rightarrow \text{мельче } F$$

2 Вектора из нулей и единиц

Пусть мн-во B состоит из двух элементов которые отождествляются с 0 и 1, т.е. $B = 0 : 1$

Произведение m экземпляров такого мн-ва обозначим за $B^m = (0 : 1)^m$, состоит из 2^m эл-ов

Опр

Вектор из нулей и единиц - упорядоченный набор из фиксированного числа нулей и единиц, т.е. эл-т мн-ва B^m

Упорядоченный набор из чисел обычно называется вектором, m - раз мерностью вектора, каждый отдельный элемент набора - компонента вектора

Замечание

Модели, в которых используются наборы из 0 и 1:

1. Геометрическая интерпретация

Точкой в m -мерном пространстве является m -мерный вектор, каждая его компонента - одна из декартовых координат точки. Набор из 0 и 1, рассматриваемый как точка в пространстве, определяет вершину куба, построенного на ортах (единичных отрезках) координатных вероятностей

2. Логическая интерпретация

Операции над векторами выполняются покомпонентно, т.е. независимо над соотв. компонентами векторов-операндов

Пример

x	0	0	0	1	1
y	1	1	1	0	1
$x \wedge y$	0	0	0	0	1
$x \vee y$	1	1	1	1	1
$x \equiv y$	0	0	0	0	1
$x \neq y$	1	1	1	1	0

3. Двоичное представление (натуральные числа)

Число представляется в виде суммы степеней 2

4. Состояние памяти компьютера

5. Сообщение, передаваемое по каналу связи

6. Можно задавать подмножества мн-ва $1 : n$

3 Алгоритм перебора 0-1 векторов. Коды Грэя

Опр

Код Грэя — такое упорядочение k -ичных (обычно двоичных) векторов, что соседние вектора отличаются только в одном разряде

Алгоритм

it - номер итерации, k_{it} - номер обновляемой компоненты

x_4	x_3	x_2	x_1	it	k_{it}
0	0	0	0	0	1
0	0	<u>0</u>	1	1	2
0	0	1	1	2	1
0	<u>0</u>	1	0	3	3
0	1	1	0	4	1
0	1	<u>1</u>	1	5	2
0	1	0	1	6	1
0	1	0	0	7	4
...		...			

Суть алгоритма: зафиксируем нулевое значение у m -й компоненты и переберем все наборы длины $m - 1$ для ост. компонент. Перебрав их меняем значение m -й компоненты на 1 и перебираем набор длины $m - 1$ в обратном порядке

Замечание*

Явная формула для проверки $G_i = i \oplus (\lfloor i/2 \rfloor)$

4 Перебор элементов прямого произведения множеств

$$M(1:k) = M_1 \times M_2 \times \dots \times M_k$$

$$|M_1 \times M_2 \times \dots \times M_k| = \prod_{i \in 1:k} m_i, \text{ где } m_i = |M_i|$$

Пусть каждое M_i состоит из целых чисел от 0 до $m_i - 1$, тогда каждый элемент $M(1:k)$ - последовательность неотрицательных чисел r_1, \dots, r_k , причем $r_i < m_i$

$$\text{num}(r_1, \dots, r_k) = \sum_{i=1}^k r_i \cdot (\prod_{j=1}^{i-1} m_j) = r_1 + r_2 m_1 + \dots + r_k m_1 \cdot \dots \cdot m_{k-1}$$

5 Размещения, сочетания, перестановки без повторений

Опр

Перестановка из n без повторений - упорядоченный набор из n неповторяющихся элементов, каждый из которых берется из диапазона $1 : n$

$$P_k = n!$$

Опр

Размещение - упорядоченный набор из k неповторяющихся элементов из диапазона $1 : n$

$$A_n^k = \frac{n!}{(n-k)!} = n(n-1)(n-k+1)$$

Опр

Сочетание - набор из k неповторяющихся элементов из диапазона $1 : n$ (порядок не важен)

$$C_n^k = \frac{A_n^k}{P_k} = \frac{n!}{(n-k)!k!}$$

Перестановки
3 разл. эл.

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.

Размещения
3 разл. эл. по 2 поз.

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.

Сочетания
2 из 3 разл. эл.

- 1.
- 2.
- 3.

6 Размещения, сочетания, перестановки с повторениями

Опр

Перестановка - последовательность длины n , составленная из k разных символов, i -ый из которых повторяется n_i раз ($n_1 + n_2 + \dots + n_k = n$)

$$P(n_1, n_2, \dots, n_k) = \frac{n!}{n_1! \cdot \dots \cdot n_k!}$$

Пример (Перестановки с повторениями, aabc)

Перестановки:

$abac, baac, aabc, aacb, abca, baca, acba, acab, bcaa, cbaa, caba, caab$

Опр (размещения с повторениями)

Аналогичное определение

$$\bar{A}_n^k = n^k$$

Опр (сочетания с повторениями)

Аналогичное определение

$$|\bar{C}_n^k| = C_{n+k-1}^k = C_{n+k-1}^{n-1}$$

7 Два алгоритма перебора перестановок. Нумерация перестановок

$$|P_k| = k! = |T_k|$$

P_k - мн-во всех перестановок

$$T_k = \prod_{i=1}^k M_i = \{0\} \times \{0, 1\} \times \dots \times \{0, 1, \dots, k-1\}$$

Построим взаимно однозначное соответствие между P_k и T_k . Возьмем перестановку (t_1, \dots, t_k) и сопоставим ей перестановку (r_1, \dots, r_k) следующим образом: для любого $i \in 1 : k$ найдем число значений, меньше r_i среди r_{i+1}, \dots, r_k - это число мы и примем в качестве t_i

В соответствии с таким определением чисел t_i в мн-ве T_k будет естественно сделать значения t_i не возрастающими, а убывающими до единицы

Разберем $r = (4, 8, 1, 5, 7, 2, 3, 6)$

i	1	2	3	4	5	6	7	8
r_i	4	8	1	5	7	2	3	6
t_i	3	6	0	2	3	0	0	0
m_i	8	7	6	5	4	3	2	1

$$t_1 = |\{1, 2, 3\}| \quad t_2 = |\{1, 5, 7, 2, 3, 6\}|, \quad t_3 = |\{\}|, \quad \dots$$

По (t_1, \dots, t_k) легко восстановить исходную перестановку. Для этого меняя i от 1 до k нужно проверить мн-во значений S_i , которые могут быть в перестановке на i месте.

В нашем примере для $i = 1$ $S_1 = 1 : 8$, $t_1 = 3 \Rightarrow r_1 = 4$, далее $S_2 = 1 : 3 \cap 5 : 8$, $t_2 = 6 \Rightarrow r_2 = 8$.

Замечание

Если использовать это отображение при переборе, то перестановки будут перебираться в лексикографическом порядке

Опр

(r_1, \dots, r_k) предшествует (R_1, \dots, R_k) , если начала перестановок совпадают до индекса i , а дальше $r_i < R_i$

Алгоритм (1)

Если перестановки перебираются в лексикографическом порядке, можно вывести правило получения следующего:

1. В перестановке (r_1, \dots, r_k) найти наибольший суффикс (r_t, \dots, r_k) , в котором элементы расположены по убыванию $r_t > \dots > r_k$; ($r_{t-1} < r_t$ - суффикс максимальн)
2. Выбрать (r_t, \dots, r_k) элемент следующий по величине после r_{t-1} и поставить его на место $t - 1$. Оставшиеся элементы, включая r_{t-1} расположить в порядке возрастания

num	t _k				p _k			
0	0	0	0	0	1	2	3	<u>4</u>
1	0	0	1	0	1	2	<u>4</u>	<u>3</u>
2	0	1	0	0	1	3	2	<u>4</u>
3	0	1	1	0	1	3	<u>4</u>	<u>2</u>
4	0	2	0	0	1	4	2	<u>3</u>
5	0	2	1	0	1	<u>4</u>	3	2

Замечание

Чтобы найти номер перестановки используем факториальную запись:

$$\begin{array}{ccccc} p & 3 & 4 & 2 & 1 \\ t & 2 & 2 & 1 & 0 \\ & 3 & 2 & 1 & 0 \end{array}$$

$$\text{num} = 2 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! = 7$$

Так можем, например, найти перестановку через n шагов от данной: сначала ищем номер исходной, прибавляем n , затем выполняя поэтапное деление в столбик на значение факториалов, восстанавливаем перестановку

Алгоритм (2)

$r = (1, 2, \dots, k)$ - рабочая перестановка

$t = (0, 0, \dots, 0)$ - номер r в факториальной системе счисления
(младший разряд последний)

$d = (-1, -1, \dots, -1)$ - направление движения элементов

$p = (1, 2, \dots, k)$ - сопоставление каждому i места, в котором он в r

1. Увеличить t на 1. При этом несколько младших разрядов получат нулевые значения, в j -м значении ув-ся на 1. При $j = 1$ процесс заканчивается

2. Сменить направление движения всех элементов младше j-го ($d_i = -d_i$ для $i > j$). Поменять местами j и соседний с ним (если $d_j = -1$ - левый, d_j - правый)

i	t	d	p	r	j	Комментарий
1	0000	- - - -	1234	1234	-	
2	0001	- - - -	1243	1243	4	Нач-ся движение эл-та 4
4	0003	- - - -	2341	4123	4	
5	0010	- - - +	2431	4132	3	Шаг эл-та 3, у 4 смена направления
6	0011	- - - +	1432	1432	4	
7	0012	- - - +	1423	1342	4	
8	0013	- - - +	1324	1324	4	
9	0020	- - - -	2314	3124	3	Второй шаг эл-та 3

8 Задача о минимуме скалярного произведения

Пусть заданы числа x_1, \dots, x_m и y_1, \dots, y_m . Составим пары (x, y) , включив каждое x_i и y_i ровно в одну пару. Затем перемножим числа каждой пары и сложим полученное произведение. Требуется найти \min такое разбиение чисел на пары S

Теорема

$$\bar{x} = (x_1, \dots, x_n) \quad x_1 \geq x_2 \dots \geq x_n$$

$$\bar{y} = (y_1, \dots, y_n) \quad y_1 \leq y_2 \dots \leq y_n$$

$$S = \sum_{i=1}^n x_i y_i \rightarrow \min$$

Док-во

Покажем, что если найдутся пары чисел (x_i, y_i) и (x_j, y_j) : $x_i < x_j$, $y_i < y_j$, то S можно уменьшить, заменив парами (x_i, y_j) и (x_j, y_i)

Действительно, так как $(x_j - x_i)(y_j - y_i) > 0$, то, раскрывая скобки, получим после переноса $x_i y_i + x_j y_j > x_i y_j + x_j y_i$

Поскольку число возможных расположений равно $m!$, т.е. конечное число, то начиная с любого расположения за конечное число шагов мы закончим процесс улучшений на расположении, которое дальше улучшить невозможно. На нем и достигается минимум

9 Числа Фибоначчи. Теорема о представлении

Опр

Последовательность чисел Фибоначчи F:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n > 1$$

Утв

$$\varphi_n = \frac{F_{n+1}}{F_n} - \text{сходится}$$

Следствие

$$\varphi_n = \frac{F_{n+1}}{F_n} = \frac{F_{n-1} + F_n}{F_n} = 1 + \frac{1}{\varphi}$$

$$\Rightarrow \varphi = \frac{\sqrt{5} + 1}{2}$$

Лемма

При $n > 1$ выполнено $\varphi^{n+2} = \varphi^{n+1} + \varphi^n$

Док-во *здесь когда-нибудь будет док-во*

Лемма

При $k > 2$ выполнено:

$$F_{2k} = F_{2k-1} + F_{2k-3} + \dots + F_1$$

$$F_{2k+1} = 1 + F_{2k} + F_{2k-2} + \dots + F_0$$

Док-во (по индукции)

($k = 3$):

$$F_6 = 8 = 5 + 2 + 1$$

$$F_7 = 13 = 1 + 8 + 3 + 1 + 0$$

($k \rightarrow k + 1$):

$$F_{2(k+1)} = F_{2k+2} = F_{2k+1} + F_{2k} = F_{2k+1} + F_{2k-1} + \dots + F_1 = ???$$

Теорема

Любое натуральное число можно однозначно представить в виде суммы чисел Фибоначчи

$$s = F_{i_0} + F_{i_1} + \dots + F_{i+r}, \text{ где } i_{k-1} + 1 < i_k, \quad k \in 1 : r \quad i_0 = 0$$

Док-во

Существование:

Пусть $j(s)$ - номер максимального числа Фибоначчи, не превосходящего s . Положим $s' = s - F_{j(s)}$. Из определения $j(s)$ следует, что $s' < F_{j(s)-1}$, иначе число Фибоначчи не было бы максимальным. Теперь мы получим искомое представление для s как представление s' , дополненное слагаемым $F_{j(s)}$

Единственность:

Пусть есть ещё одно представление $s = F_{j_0} + \dots + F_{j_q}$. Н.У.О. считаем, что $j_q < j(s)$. Если мы заменим F_{j_q} на $F_{j(q)-1}$, то правая часть разве что лишь увеличится. Аналогично заменим с возможным увеличением предпоследнее слагаемое на $F_{j(s)-3}$. ???

10 Перебор сочетаний. Нумерация сочетаний

Состояние вычислительного процесса. Массив (x_1, \dots, x_m) номеров, включенных в сочетание. Начальное состояние: принять $x_i = i \quad \forall i \in 1 : m$. Стандартный шаг: просматривать компоненты вектора x , начиная с x_m и искать первую компоненту, которую можно увеличить (нельзя $x_m = n, x_{m-1} = n - 1$ и т.д.). Если такой нет, то закончить процесс. В противном случае пусть k - наибольшее число, для которого $x_k < n - m + k$, тогда увеличить x на единицу, а для всех следующих за k -ый продолжаем, но ряд от значения x_k , т.е. $x_i = x_k + (i - k)$

num	Сочетание					k
1	1	2	3	4	5	5
2	1	2	3	4	6	5
3	1	2	3	4	7	5
4	1	2	3	5	6	4
5	1	2	3	5	7	5

Удобно использовать вектора из 0 и 1, чтобы перенумеровать. С каждым сочетанием из n по m можно связать вектор из n нулей и единиц, в котором единиц ровно m - числа, входящие в данное сочетание просто задают номера этих единиц

$$\text{num}(b[1 : n], m) = \begin{cases} \text{num}(b[1 : n - 1], m) & b[n] = 0 \\ C_{n-1}^m + \text{num}(b[1 : n - 1], m - 1) & b[n] = 1 \end{cases}$$

Пример

$$\begin{aligned} \text{num}((0, 1, 0, 1, 0, 0, 1), 3) &= \\ &= C_6^3 + \text{num}((0, 1, 0, 1, 0, 0), 2) = C_6^3 + C_3^2 + \text{num}((0, 1, 0), 1) = \\ &= C_6^3 + C_3^2 + \text{num}((0, 1), 1) = C_6^3 + C_3^2 + C_1^1 + \text{num}((0), 0) = 24 \end{aligned}$$

11 Бином Ньютона и его комбинаторное использование

Треугольник Паскаля (в узлах C_n^k):

	1						
$n = 0$		1	1				
$n = 1$		1	2	1			
$n = 2$		1	3	3	1		
$n = 3$		1	4	6	4	1	
$n = 4$		1	5	10	10	5	1
$n = 5$		1	6	15	20	15	6
$n = 6$		0	1	2	3	4	5
		1	6	15	20	15	6

Опр

Бином Ньютона: $(a + b)^n = \sum_{k=0}^n C_n^k a^k b^{n-k}$

Лемма

$$C_{n-1}^{k-1} + C_{n-1}^k = C_n^k$$

Док-во (по индукции)

$$\begin{aligned} (a+b)^n &= a(a+b)^{n-1} + b(a+b)^{n-1} = \\ &= \sum_{k=0}^{n-1} C_{n-1}^k a^{k+1} b^{(n-1)-k} + \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{1+(n-1)-k} = \\ &= \sum_{k=1}^n C_{n-1}^{k-1} a^k b^{n-k} + \sum_{k=0}^{n-1} C_{n-1}^k a^k b^{n-k} = \sum_{k=0}^n (C_{n-1}^{k-1} + C_{n-1}^k) a^k b^{n-k} \end{aligned}$$

Следствие

$a = 1, \quad b = 1:$

$$\sum_{k=0}^n C_n^k = 2^n$$

$a = 1, \quad b = -1:$

$$\sum_{k=0}^n C_n^k (-1)^k = 0$$

(благодаря $C_n^k = C_n^{n-k}$)

$a = 1, \quad b = i:$

$$\sum_{k=0}^n C_n^k i^k = (1+i)^n = (\sqrt{2} \cdot (\cos \frac{\pi}{4} + i \sin \frac{\pi}{4}))^n = 2^{\frac{n}{2}} \cdot e^{in\frac{\pi}{4}}$$

12 Свойства биномиальных коэффициентов

Определения см. в прошлом билете

1. $C_n^k = C_n^{n-k}$
2. $C_{n-1}^{k-1} + C_{n-1}^k = C_n^k$
3. $C_n^m C_m^k = C_n^k C_{n-k}^{m-k}$

13 Основные определения теории вероятностей

Опр

А - событие, $p(A) \subset [0; 1]$ - характеристика события, р - вероятность события А

Опр

S - мн-во элементарных событий (мн-во исходов), если его элементы равноправны

Опр

Пусть А - событие, $A \subset S$, тогда:

$$\frac{|A|}{|S|} = p(A)$$

Событие с вероятностью 1 называется достоверным, событие с вероятностью 0 - невозможным

Опр (совмещение событий)

Событие, которое составлено из всех элементарных событий (исходов), входящих и в А, и в В, называется совмещением А и В ($A \cup B$)

$$P(A \cup B) \leq P(A), P(B)$$

Опр

Событие, состоящее из всех эл. событий, входящих или в А, или в В, называется объединением событий А и В ($A \cap B$)

Опр

А, В - события, $P(A \cup B) = 0$ (события, которые не могут вместе выполниться) =
А, В - несовместные события

$$P(A) + P(B) = P(A \cap B)$$

Опр

События А и В называются независимыми, если $P(A \cup B) = P(A) \cdot P(B)$

Опр

Разбиение мн-ва S на несовместные события S_1, \dots, S_n называется полной системой событий

$$P(A) = \sum_{i=1}^m P(A \cup S_i) - \text{ф-ла полной вероятности}$$

Опр

A_1, \dots, A_k - независимы в совокупности, если:

$$\forall I \subset 1 : k \quad P\left(\bigcup_{i \in I} A_i\right) = \prod_{i \in I} P(A_i)$$

Замечание

Независимость в совокупности отличается от попарной независимости, первое жестче

Пример (С.Н.Бернштейн)

Рассмотрим игральную кость в форме правильного тэтраэдра. Одна грань этой кости окрашена в белый цвет W, вторая - в черный B, третья - в красный R, а окраска четвертой - смешанная M, в ней есть все три цвета. При каждом бросании кость ложится какой-то стороной вниз. Вероятность того что на нижней грани окажется белый цвет - очевидно $\frac{1}{2}$ ($2/4$). Аналогично для любого другого цвета. Вероятность выпадения двух цветов сразу - $\frac{1}{4}$ (M)

$$P(R \cup B \cup W) = \frac{1}{4} \neq P(R) \cdot P(B) \cdot P(W) = \frac{1}{8}$$

14 Условные вероятности и формула Байеса

Опр (Условная вероятность)

Пусть A - событие, $P(A) > 0$, тогда:

$$P(B | A) = \frac{P(B \cap A)}{P(A)}$$

Вероятность события B, если произошло A
(все исходы, при которых произошли B и A на исходы с A)

Замечание

События независимы, если $P(B | A) = P(B)$ (очевидно)

Напоминание (Формула полной вероятности)

$$B_1, \dots, B_n \quad B_i \cap B_j = \emptyset$$

$$P(A) = \sum_{i=1}^n P(A \cap B_i) = \sum_{i=1}^n P(A | B_i) \cdot P(B_i)$$

Пример

Старая линия завода выпускает в 2 раза меньше продукции, чем новая ($2P(S) = P(N)$). А доля брака у нее в 4 раза больше ($P(Br|S) = 4P(Br|N)$). Что можно сказать о доле брака ($P(Br)$) в продукции?

Док-во

По ф-ле полной вероятности:

$$\begin{aligned} P(Br) &= P(Br|S) P(S) + P(Br|N) P(N) = \\ &4P(Br|N)P(S) + 2P(Br|N)P(S) = 2P(Br|N) \end{aligned}$$

Теорема (Формула Байеса)

$$P(B_i | A) = \frac{P(A | B_i)P(B_i)}{\sum_{i=1}^n P(A | B_i) \cdot P(B_i)} \quad \text{по Григорьевой}$$

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)} \quad \text{по Интернету}$$

Док-во

Подставим $P(B \cup A) = P(B | A)P(A)$ в формулу полной вероятности:

$$P(B) = \sum_i P(B | A_i)P(A_i)$$

$$P(B \cup A) = P(B | A)P(A) = P(A | B)P(B)$$

$$\Rightarrow P(A_i | B) = \frac{P(B | A_i)P(A_i)}{\sum_j P(B | A_j)P(A_j)}$$

Пример

Пусть у нас есть две колоды: 36 и 52 карты. Выбираем с равной вероятностью одну из колод. Достаем из нее карту и хотим угадать, какая это из колод. Пусть $T\Diamond$

$$P(B_{36}|T\Diamond) = \frac{P(T\Diamond|B_{36})P(B_{36})}{P(T\Diamond|B_{36})P(B_{36}) + P(T\Diamond|B_{52})(B_{52})} = \frac{\frac{1}{36}\frac{1}{2}}{\frac{1}{36}\frac{1}{2} + \frac{1}{52}\frac{1}{2}} = \frac{52}{88}$$

15 Математическое ожидание и дисперсия случайной величины

Опр

Пусть $\Omega = \{\omega\}$ - множество элементарных событий (произвольное непустое множество, элементы которого - элементарные события), p_ω - вероятность элементарного события ω , тогда функция $f: \Omega \rightarrow \mathbb{R}$ называется случайной величиной.

Опр

Случайные величины ξ и η называют независимыми, если независимы события $\{\omega \mid \xi(\omega) = a\}$ и $\{\omega \mid \eta(\omega) = b\}$ для любых значений a и b

Пример

Пусть мы подбрасываем монеты с Васей по очереди. $\Omega = \{\text{ор}, \text{ре}\}$. Пусть мы выигрываем соответственно 10, -33, а Вася 10, -30.

Тогда $\xi(\text{ор.}) = 5$, $\xi(\text{ре.}) = -10$, а у Васи $\eta(\text{ор.}) = 10$, $\xi(\text{ре.}) = -30$. Мы знаем, что ξ , η - независимы, то есть:

$$P(\xi = a, \eta = b) = P(\xi = a) \cdot P(\eta = b) \quad \forall a, b$$

Где a, b - 5, 10, -33, -30, ..., а P - вероятность события

Опр

$E(\xi) = \sum_{\omega \in \Omega} \xi(\omega) p_\omega = \sum_a a \cdot P(\xi = a)$ называется мат. ожиданием случайной величины ξ

Свойства

1. Если $P(\xi = a) = 1 \Rightarrow E(\xi) = a$
2. ξ, η - случ. вел. $E(\xi + \eta) = E(\xi) + E(\eta)$ (Линейность)
3. Если $\xi = c\eta$, где $c = \text{const}$ $E(\xi) = cE(\eta)$
4. $E(\xi \cdot \eta) = E(\xi) \cdot E(\eta)$ если ξ и η нез.

Док-во

$$\begin{aligned} \text{Линейность} \quad E(\xi + \eta) &= \sum_{\omega \in \Omega} (\xi(\omega) + \eta(\omega)) p_\omega = \sum_{\omega \in \Omega} \xi(\omega) p_\omega + \\ &+ \sum_{\omega \in \Omega} \eta(\omega) p_\omega = E(\xi) + E(\eta) \end{aligned}$$

Опр

Мат. ожидание квадрата отклонения случайной величины от ее мат. ожидания называется дисперсией этой случайно величины

$$D(\xi) = E(\xi - E(\xi))^2$$

Дисперсия характеризует разброс случайной величины вокруг ее мат. ожидания

$$\begin{aligned} D(\xi) &= E(\xi - E(\xi))^2 = E(\xi^2) - E(2\xi E(\xi)) + E(E^2(\xi)) = \\ &= E(\xi^2) - 2E(\xi)E(E(\xi)) + E^2(\xi) = E(\xi^2) - E^2(\xi) \end{aligned}$$

Свойства (Дисперсии)

1. Дисперсия неотрицательна. Если $P(\xi = a) = 1$, то $D(\xi) = 0$
2. $\xi = \eta + c, \quad c = const, \quad \Rightarrow D(\xi) = D(\eta)$
3. $\xi = c\eta, \quad c = const, \quad \Rightarrow D(\xi) = c^2 D(\eta)$
4. ξ и η нез. с.в. $\Rightarrow D(\xi + \eta) = D(\xi) + D(\eta)$

Док-во 1. (очевидно)

2. $xi = \eta + c$

$$\begin{aligned} E(\xi) &= \sum_{\omega \in \Omega} \xi(\omega) \rho_\omega = \sum_{\omega \in \Omega} (\eta(\omega) + c) \rho_\omega = \sum_{\omega \in \Omega} \eta(\omega) \rho_\omega + \sum_{\omega \in \Omega} c \rho_\omega = \\ &= E(\eta) + c \\ D(\xi) &= E(\xi - E(\xi))^2 = E(\eta + c - E(\eta) - c)^2 = D(\eta) \end{aligned}$$

3. $D(\xi) = E(c\eta - E(c\eta))^2 = E(c\eta - cE(\eta))^2 = c^2 E(\eta - E(\eta))^2 = c^2 D(\eta)$
4. $D(\xi + \eta) = E(\xi + \eta - E(\xi + \eta))^2 =$
 $= E(\xi - E(\xi) + \eta - E(\eta))^2 = E(\xi - E(\xi))^2 + 2E((\xi - E(\xi)(\eta - E(\eta)))) +$
 $+ E(\eta - E(\eta))^2 = D(\xi) + D(\eta) + 2(E(\xi\eta - \xi E(\eta) - \eta E(\xi) + E(\xi)E(\eta))) =$
 $= D(\xi) + D(\eta) + 2(E(\xi\eta) - E(\xi)E(\eta)) - E(\eta)E(\xi) + E(\xi)E(\eta) = D(\xi) + D(\eta)$

Опр

Корень из дисперсии называется средним квадратичным отклонением

Опр

$$\rho(\xi, \eta) \stackrel{\text{def}}{=} \frac{m(\xi, \eta)}{\sqrt{D(\xi)D(\eta)}} = \frac{E(\xi\eta) - E(\xi)E(\eta)}{\sqrt{D(\xi)D(\eta)}}$$

16 Схема Бернулли

Задача

Стрелок делает 5 выстрелов, какова вероятность того, что он попадет не меньше 4 раз?

p - вероятность попасть в мишень

$$P_5(A) = P_5(4) + P_5(5) = P_5(4) + p^5$$

$$P_5(A) = C_5^4 \cdot p^4(1-p) + p^5$$

hint: мы выбираем, когда стрелок промахнется из всех выстрелов

Опр

$$P_n(m) = C_n^m p^m (1-p)^{n-m} \quad \text{формула Бернулли}$$

n - число попыток m - удачных событий

Док-во

Рассмотрим последовательность независимых, случайных величин $\delta_1, \dots, \delta_n, \dots$, каждая из которых принимает два значения: 1 с вероятностью p и 0 с вероятностью $q = 1 - p$. Такая вероятностная схема называется схемой Бернулли.

Случайная величина ξ_n , полученная при сложении n таких случайных величин δ_i имеет распределение, называемое биноминальным распределением. Чтобы $\xi_n = k$, нужно чтобы ровно k из случайных величин $\delta_1, \dots, \delta_n$ принимали значение 1, а остальные должны равняться нулю. Вероятность этого события при фиксированных местах единиц и нулей равна $p^k q^{n-k}$, и если учесть все возможные C_n^k расположений этих мест, то получим $P(\xi_n = k) = C_n^k p^k q^{n-k}$ - k -ый член биноминального распределения

Замечание

$$E(\xi_n) = np, \quad D(\xi_n) = npq$$

это когда-нибудь будет дополнено

17 Случайные числа. Схема Уолкера

Замечание

Зачем все это нужно? Мы хотим провести серию неких экспериментов, для этого мы можем использовать метод статистического моделирование. На компьютере можно имитировать случайные эксперименты. В качестве источника случайности мы можем взять генератор случайных чисел (при каждом обращении генератор дает нам какое-то число). В Романовском написано, что эти величины имеют равномерное распределение, но это зависит только от того, какой генератор взять (Если коротко, то равномерное распределение - это, когда у нас нет "перекосов" в сторону каких-то значений).

Пример

Для вычисления площади $sq(A)$ плоской ограниченной фигуры A можно построить содержащий фигуру прямоугольник R (стороны которого \parallel осям коорд.) Будем бить случайными точками в этот прямоугольник. Отношение числа точек, попавших в A, к общему числу точек будет хорошей оценкой площади.

При равномерном распределении $P(\text{попасть в } A) = sq(A)/sq(R)$

Опр (схема Уолкера)

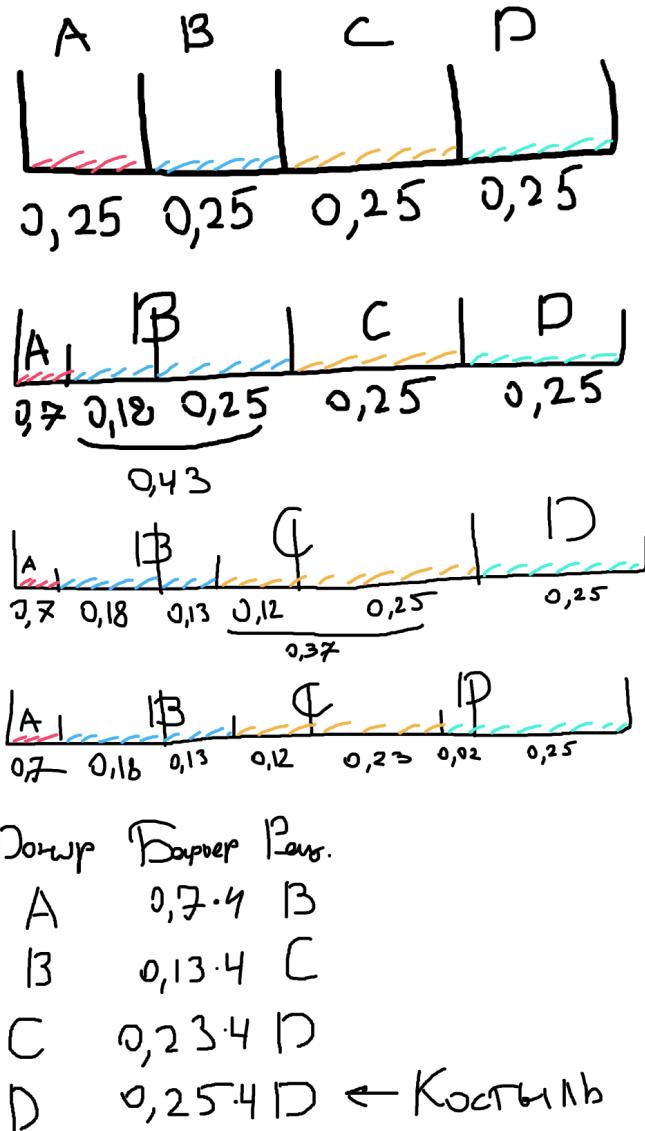
(По-сущи это своеобразный генератор случайных чисел, но с нашим распределением)

Мы хотим создать некую схему, которая сможет нам отвечать, куда мы попали нашей точкой на отрезке $[1, 0]$, какой мы получили исход. Изначально нам даны вероятности этих событий, всего их n. Мы строим таблицу, для каждого исхода запишем $\frac{1}{n}$. То есть, мы изначально предполагаем, что у нас равномерное распределение. Потом мы начинаем это корректировать. Берем событие, которое получило "вероятностной массы" больше, чем остальные. Назовем его донором. Возьмем событие, у которого вероятность ниже, чем мы хотели изначально. Назовем его рецепциентом. Отрежем от донора и отдадим рецепциенту. (То есть, когда мы будем бить точками в этот отрезанный кусок, то он будет относиться уже к рецепциенту, и наш генератор вернет нам другое число). В таблицу нужно еще записать барьеры, которые помогут нам определять, куда попала наша точка. Барьер = остаток донора в $\frac{1}{n}$ отрезка $\cdot 4$. (то есть, если точка правее, то она попала в рецепциента, а иначе в донора) Если у рецепциента стало слишком много массы, то он сам станет донором для другого события. (Резать нужно от его исходного куска в $\frac{1}{n}$ - ую). После того, как все исходы получили нужную вероятность, нам остается научиться быстро определять, куда попала наша точка $x \in [0, 1]$. Мы

можем быстро понять, в какую $\frac{1}{n}$ -ую попала значение. Умножим x на n и возьмем целую часть. Дальше мы смотрим на барьер, если значение x больше барьера, то выбираем реципиента, иначе донора.

Пример

$$P_a = 0,07 \quad P_b = 0,31 \quad P_c = 0,35 \quad P_d = 0,27$$



18 Двоичный поиск и неравенство Крафта

Опр (схема дихотомического (двоичного) поиска)

Разыскиваем на прямой линии отрезок, соответствующий нужному значению индекса, разбиваем область поиска на две части и устанавливаем, в какой лежит интересующий нас индекс.

Затем повторяем действие для оставшейся части, пока не найдем часть, содержащую только одно значение индекса.

Теорема

Для того чтобы набор из целых чисел s_1, \dots, s_m мог быть набором длин путей в схеме с m исходами необходимо и достаточно, чтобы:

$$\sum_{i \in 1:m} 2^{-S_i} \leq 1, \quad S_i - \text{числа из набора}$$

Док-во

(Необходимость \Rightarrow):

Рассмотрим поисковую схему - двоичное дерево T с m листьями, сопоставим каждой вершине k , находящейся на расстоянии t от корня, $a_k = 2^{-t}$. То есть если r_0 - корень $\Rightarrow a_{r_0} = 1$. Значит мы должны доказать:

$$a_{r_0} \geq \sum_{k \in F} a_k, \quad \text{где } F - \text{мн-во листьев}$$

Для каждого не-листа $k \in M \setminus F$ следует:

$$(*) \quad a_k \geq \sum_{r \in \text{next}(k)} a_r, \quad \text{где } \text{next}(k) - \text{мн-во прямых потомков } k$$

В случае двух вершин неравенство выполняется как равенство. В случае одной - как строгое неравенство. Суммируя неравенства (1) по $M \setminus F$ получаем:

$$\sum_{k \in M \setminus F} a_k \geq \sum_{r \in \text{Im } M \setminus \{r_0\}} a_r$$

Сокращение обзих слагаемых (промежуточных частях неравенств) и дает искомое

(Достаточность \Leftarrow):

Взяв m чисел s_k , удовлетворяющих условию теоремы, расположим их в порядке возрастания. Определим, как и раньше, числа $a_k = 2^{-s_k}$ и положим:

$$b_1 = 0; \quad b_{k+1} = b_k + a_k, \quad k > 1$$

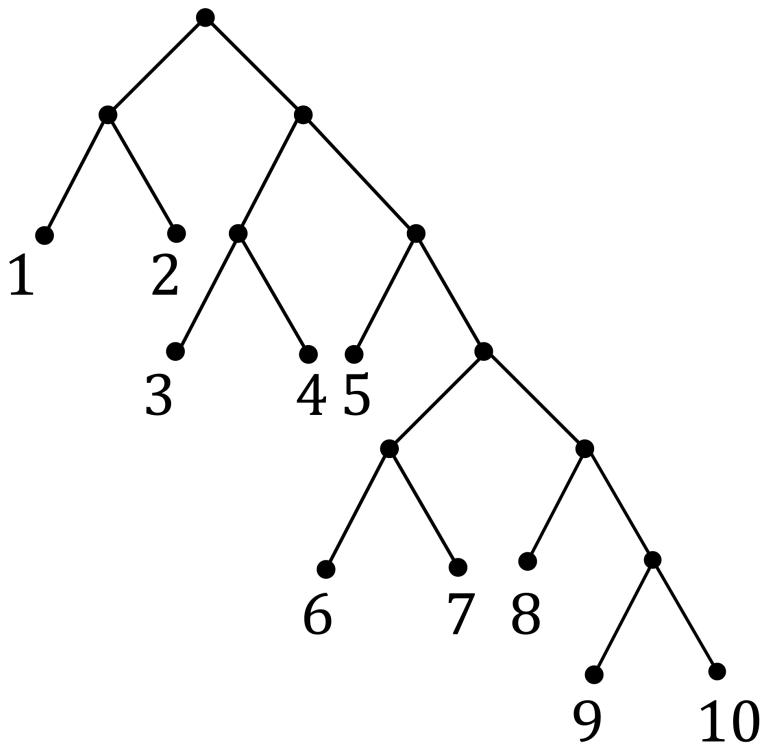
$$P_a = 0,07 \quad P_b = 0,31 \quad P_c = 0,35 \quad P_d = 27$$

Рассмотрим последовательности из нулей и единиц t_k , являющиеся двоичными представлениями дробей b_k , в каждой такой дроби b_k возьмем первые s_k знаков. Покажем, что никакая из последовательностей t_k не является началом другой такой последовательности.

Так как длины последовательностей t_k не убывают с ростом k , нам достаточно показать, что никакая последовательность t_k не является началом последовательности с большим номером. Так как дроби b_i возрастают, то $b_k < b_{k+1} < \dots < b_m \leq 1$. Обозначим через $b_r^{(k)}$ число, получающееся из b_r , отсечением первых s_k цифр. Очевидно, что для таких "урезаний" сохраняется то же неравенство, хотя и нестрогое $b_k < b_{k+1}^{(k)} \leq \dots \leq b_m^{(k)}$. При этом первое из неравенств осталось строгим, так как $b_k + a_k = b_{k+1} = b_{k+1}^{(k)}$. Следовательно, начало $t_k^{(k)}$ никакой дроби b_r не совпадает с t_k при $r > k$.

Далее, любой набор последовательностей t_k , из которых ни одна не является началом другой, задает некоторую процедуру поиска. Действительно, рассмотрим полное двоичное дерево с достаточно большим числом этажей и наложим на это дерево все последовательности t_k , трактуя каждую из них как путь от корня, который идет влево, если очередной элемент последовательности равен 0, и вправо - для единицы. В силу сделанного предположения ни одна из вершин, соответствующих концам последовательностей, не лежит на какой-либо другой последовательности как промежуточная вершина.

	a_i	b_i		
2	0.010..0	0.000000	00	
2	0.010..	0.01000	01	
3	0.001..	0.100 00	100	
<u>Пример</u>	0.001..	0.101 00	101	
	3	0.001..	0.110 000	110
	5	0.00001	0.111000	11100
	5	0.00001	0.11101	11101
	5	0.00001	0.11110	11110
	6	0.000001	0.111110	111110
	6	0.000001	0.111111	111111



19 Энтропия. 2 леммы

Опр

Энтропия случайной схемы - мера содержания в этой схеме неопределенности. ρ - вероятностная схема с m исходами, вероятности которых равны p_1, \dots, p_m

$$H(\{p_1, \dots, p_m\}) = \sum_{i=1}^m p_i \log_2 \frac{1}{p_i}$$

Свойства

1. Энтропия непрерывно зависит от вероятности при фиксированном m
2. При перестановке в наборе $\{p_1, \dots, p_m\}$ энтропия не меняется
3. Нужно ввести шкалу для измерения неопределенности. Примем за 1 схему с двумя равновероятными исходами

$$(H(\{\frac{1}{2}, \frac{1}{2}\}) = 1)$$

4. При фиксированном m наибольшей неопределенностью обладает схема, в которой все события равновероятны

$$(H(\{\frac{1}{m}, \dots, \frac{1}{m}\}) = h(m))$$

5. $h(m)$ возрастает с ростом m
6. Рассмотрим схему P_m с m исходами и вероятностями $\{p_1, \dots, p_m\}$ и схему R_k с $\{r_1, \dots, r_k\}$. Образуем комбинированную схему с $m+k-1$ исходами следующим образом: выбирается случайнм образом один из исходов схемы P_m и если произошел m -й исход, выбирается случайно один из исходов схемы R_k , а остальные $m-1$ исходов схемы P_m считаются окончательно. В комбинированной схеме PR получаем исходы:

$$1, 2, \dots, m-1, (m, 1), (m, 2), \dots, (m, k)$$

с вероятностями:

$$p_1, p_2, \dots, p_{m-1}, p_m q_1, \dots, p_m q_k$$

Легко видеть, что $H(PR) = H(P_m) + p_m H(R_k)$

Теорема

Единственная функция на множестве всех вероятностных схем, удовлетворяющая условиям 1-6 - это функция H

Лемма (1)

$$g(m) = \log_2 m$$

Док-во

Q_k, Q_l - две схемы с равновероятными исходами

$$Q_{kl} \quad (1, 1), (1, 2), \dots, (k, l)$$

$$g(kl) = g(k) + g(l)$$

$$g(m^k) = kg(m)$$

$$g(2^k) = 2g(k)$$

$$S = [\log_2 m^k]$$

$$g(2^s) \leq g(m^k) \leq g(2^{s+1}) \Rightarrow s \leq kg(m) \leq s+1$$

$$\Rightarrow 0 \leq g(m) - \frac{\lfloor k \log_2 m \rfloor}{k} = g(m) - \log_2 m + \frac{\{k \log_2 m\}}{k} \leq \frac{1}{k}$$

При $k \rightarrow \infty \quad g(m) = \log_2 m$

Если набор вероятностей $\{p_1, \dots, p_m\}$ состоит из рациональных чисел, то $G(\{p_1, \dots, p_m\}) = H(\{p_1, \dots, p_m\})$

Док-во

здесь когда-нибудь будет док-во

Док-во (теоремы)

здесь когда-нибудь будет док-во

20 Теорема об энтропии

Теорема

Единственная функция, удовлетворяющая 6-ти св-ам энтропии - это функция $H(\{p_1, \dots, p_m\}) = \sum_{i=1}^m p_i \log_2 \frac{1}{p_i}$

Док-во

Обозначим $y_i = 2^{-s_i}$, так что $s_i = \log_2 \frac{1}{y_i}$

В этих обозначениях условие 1 преобразуется к виду:

$$\sum_{i \in 1:m} y_i \leq 1, \quad (@)$$

а целевая функция к виду:

$$T(p, y) = \sum_{i \in 1:m} p_i \log_2 \frac{1}{y_i}$$

Условие 2 перейдет в условие $0 < y_i < 1$

Второе неравенство из этой пары следует только из полученного ограничения на сумму переменных y_i , так что останется только условие положительности.

Целевая функция Т состоит из отдельных слагаемых, соответствующих отдельным переменным. Каждое слагаемое убывает с ростом аргумента. Если бы неравенство (@) выполнялось как строгое, то увеличение любой из переменных уменьшило бы значение целевой функции. Поэтому в точке минимума условие 1 выполняется как равенство. Выразим переменную y_m через остальные:

$$y_m = 1 - \sum_{i \in 1:m-1} y_i$$

Подставим её в целевую функцию и приравняем к нулю производные целевой функции $T(p, y)$ по всем оставшимся переменным:

$$\frac{\partial T}{\partial y_i} = -p_i \cdot \log_2 e \cdot \frac{1}{y_i} + p_m \cdot \log_2 e \cdot \frac{1}{y_m}$$

Откуда

$$\frac{p_1}{y_1} = \frac{p_2}{y_2} = \dots = \frac{p_m}{y_m}$$

Так как обе суммы (p_i и y_i) равны 1, то $y_i = p_i$. Это единственная точка, в которой возможен экстремум функции $T(p, y)$, и следовательно, $\min_y T(p, y) = H(p)$

21 Операции над строками переменной длины

Опр

А - конечное мн-во, называется алфавитом. Его элементы буквами. Программная конечная последовательность букв называется строкой. Кол-во букв в этой последовательности - длина строки

1. Нахождение длины строки
2. Выделение подстроки. Операция выделяет подстроку с заданным числом букв, начиная с заданного места

Опр

Начальная подстрока строки - префикс. Конечная - суффикс. Голова строки (head) - префикс из одной буквы. Остальная часть - хвост (tail)

3. Конкатенация строк (спецификация)
4. Обращение строк (переворот)
5. Сравнение строк по предшествованию, обычно используется лексикографическое сравнение.

Пусть буквы из А можно сравнивать

Тогда для строк а и b результат лексикографического сравнения равен

- (a) $a = b$, если а и b - пусты
 - (b) $a < b$, если а пустая, а b - нет
 - (c) $a > b$, если b пустая, а а - нет
 - (d) $a < b$, если head $a <$ head b
 - (e) $a > b$, если head $a >$ head b
 - (f) Результат сравнения tail a и tail b , если head $a =$ head b
6. Поиск образца в строке
 7. Подстановка (вместо заданного образца нужно вписать другой)
 8. Преобразование

S - строка, $A' = \{a \mid a \in S\}$, $\varphi : A' \Rightarrow V$, $\varphi(B)$ - последовательность элементов из В

9. Фильтрация (все символы делятся на подмн-ва А и В). Если $s_i \in A$, то остается в строке, если $\in B$ - удаляется

10. Слияние

Пусть на алфавите задан порядок. s_1, s_2 - строки из А. На каждом шаге к результату приписывается $m = \min(\text{head } s_1, \text{head } s_2)$ и удаляется из старого списка

11. Поиск максимального совпадения

Опр

Функция $f(s)$, заданная на множестве всех строк S называется аддитивной, если $\exists \varphi : A \rightarrow \mathbb{R}$:

$$f(s) = \varphi(\text{head } s) + f(\text{tail } s),$$

когда строка непустая и $f(s) = 0$ для пустой строки

Пример

Длина строки - аддитивная функция

Опр

Функция $f(s)$, заданная на множестве всех строк S называется мультипликативной, если $\exists \psi : A \rightarrow \mathbb{R}$:

$$f(s) = \psi(\text{head } s) \cdot f(\text{tail } s),$$

когда строка непустая и $f(s) = 1$ для пустой строки

Пример

Вероятность совмещения независимых в совокупности событий, заданных списком s

Опр

Функция $f(s)$, заданная на множестве всех строк S называется мультипликативной, если $\exists \varphi, \psi : A \rightarrow \mathbb{R}$:

$$f(s) = \varphi(\text{head } s) + \psi(\text{head } s) \cdot f(\text{tail } s),$$

когда строка непустая и $f(s)$ как-то определена для пустой строки

Пример

Схема Горнера: Полином представляется в виде

$$P(x; \{a_0, \dots, a_n\}) = a_0 + x \cdot P(x; \{a_1, \dots, a_n\})$$

a_0 - голова $\{a_1, \dots, a_n\}$ - хвост

Опр

Функция $f : S \rightarrow D$, где D - некоторое множество, называется марковской, если $\exists \theta : A \times D \rightarrow D$:

$$f(s) = \theta(\text{head } s, f(\text{tail } s))$$

и задано начальное значение для пустой строки $f(s) = r_0 \in D$

Замечание

Все предыдущие функции являются марковскими

22 Поиск образца в строке (Карпа-Рабина, Бойера-Мура)

Опр

Пусть заданы две строки: t (text) и p (pattern). Говорят, что образец входит точно в текст с позиции j , если $t[j : j + m - 1] = p[1 : m]$

Наивный метод: ходим по строке, ищем первый символ, затем второй... Сложность $m \cdot n$

Опр

Пусть задана числовая последовательность p_1, \dots, p_n , определим скользящую сумму как $s_i = p_i + p_{i+1} + \dots + p_{r+i}$, где r - некоторое фиксированное число

Замечание

Нетрудно заметить, что $s_{k+1} = s_k - p_k + p_{k+r}$

Замечание

Аналогично можно считать полиномы $\sigma_k(x) = \sum_{i=0:r-1} p_{k+i}x^{r-1-i}$, аналогично $\sigma_{k+1} = x\sigma_k - p_kx^r + p_{k+r}$. Эта функция быстро становится большей из-за x и r , но можно считать $p_k(x)$ как остаток деления σ_k на d . Тогда $p_{k+1} = (x\sigma_k - p_kR(x, r) + p_{k+r}) \bmod d$, где $R(x, r) = x^r \bmod d$

Алгоритм (метод Карпа-Раббина)

Выберем подходящие x и d , вычислить σ для строки длины m и для всех подстрок изначальной строки длины m и сравнить.

Алгоритм (метод Бойера-Мура)

Сравнение начинается с последнего символа образца, который совмещается с началом. Если совпал \Rightarrow нашли. Если нет, пользуемся эвристиками:

1. Эвристика стоп-символа.

(a) Если не совпало и текущего символа нет в строке, то следующую проверку можно сдвинуть на длину образа

строка: П

шаблон: К О Л О К О Л

next step: К О Л О К О Л

(b) Если такой символ есть, сдвигаемся до последнего вхождения в образце

строка: К . . .

шаблон: К О Л О К О Л

next step: К О Л О К О Л

2. Правило хорошего окончания

строка: ... К КОЛ ...

шаблон: КОЛ О КОЛ

next step: КОЛ ОЛОКОЛ

Пример для суффиксов и сдвигов: $\emptyset - 1$, Л – 4, ОЛ – 4, КОЛ – 4

Алгоритм* (Кнута-Морриса-Пратта)

$\forall i \in 2 : l = m + n + 1$ вычислить:

1. z_i - наибольшее k т.ч. $p[1 : k] = p[i : (i + k - 1)]$
2. r_i - наибольшее l т.ч. $\exists b \leq i : p[b : e]$ совпадает с $p[1 : (e - b + 1)]$
3. l_i - какое-либо b из определения r_i

Определяемая этими числами подстрока $p[l_i : r_i]$ совпадает с префиксом р и идет дальше всего по строке. По ней и определяется $z_i = l_i - 0 + 1$

Зная $z[2 : i - 1]$, $L = l_{i-1}$ и $R = r_{i-1}$ мы можем вычислить z_i и пересчитать L и R. Возможны несколько случаев. Если $i > R$, то подстрока, найденная для $i - 1$, для i уже не годится и её нужно вычислить заново. Непосредственно сравниваем строки $p[i : m]$ и $p[1 : m]$, находим длину совпадений части s и полагаем $z_i = s$, $L = l_i = i$, $R = r_i = i + s - 1$

Если $i \leq R$, то символ $p[i]$ лежит в подстроке $p[L : R]$, совпадающей с префиксом, и следовательно, $p[i : R]$ совпадает с суффиксом этого префикса. Длина суффикса равна $S = R - i + 1$, а начальная позиция $i' = i - L + 1$

Если $z_{i'} < S$, то нужно принять $z_i = z_{i'}$, а L и R не изменятся

Если $z_{i'} \geq S$, то вся подстрока $p[i : R]$ совпадает с префиксом. Возможно, что это совпадение идет и дальше, так что нужно продолжить, сравнивая строки $p[R+1 : m]$ и $p[S+1 : m]$. Установив совпадение вплоть до позиций $R + k$ и $S + k$, полагаем $L := i$, $R := R + k$, $z_i = k + 1$

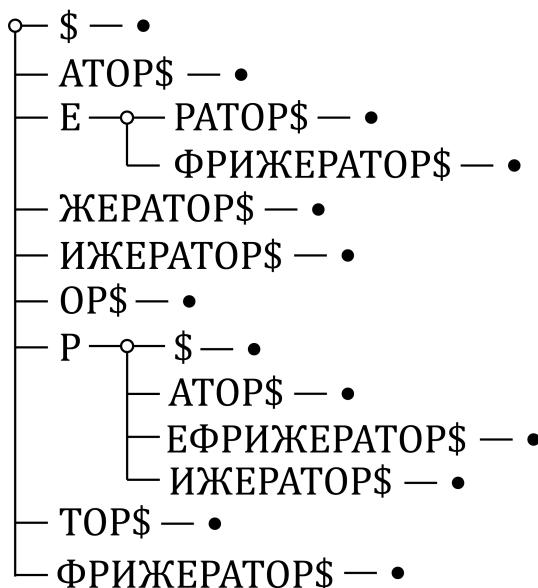
23 Суффиксное дерево

Алгоритм

Пример: Стока РЕФРИЖЕРАТОР, приписываем уникальный индекс в конец, который при лексикографическом сравнении считаем предшествующим всем символам (напр, \$). Получаем РЕФРИЖЕРАТОР\$

Теперь просматриваем все символы строки в лексикографическом порядке и ищем максимальный уникальный префикс в оставшейся части. Записываем вместе с символом в дереве от корня по алфовиту. Повторяем рекурсивно, записывая от нашей части

Пример (РЕФРИЖЕРАТОР)



Замечание

Размер дерева и память связаны линейно, а также существует алгоритм построения за линию.

Можно по дереву искать подстроку в строке сравнением посимвольно и наибольшую повторяющуюся строку поискам в потомках суффикса (P)

24 Задача о максимальном совпадении двух строк

Задача

Пусть заданы две последовательности (строки) $a[1 : m]$ и $b[1 : n]$, требуется найти их максимальное совпадение. Будем также решать подзадачи с меньшими строками $a[1 : k]$ и $b[1 : l]$, где $k \in 1 : m$ и $l \in 1 : n$. Задачу с такими параметрами будем называть задачей (k, l) а максимальную длину последовательности в ней обозначим через $v(k, l)$ (искомая задача становится задачей (m, n) и требуется найти $v(m, n)$)

Сформируем из искомых значений матрицу $v[1 : m, 1 : n]$. Начнем с того, что решения при $k = 1$ и при $l = 1$ находятся сразу:

$$v[1, 1] = \begin{cases} 1, & \text{если } a[1] = b[1] \\ 0 & \text{иначе} \end{cases}$$

$$v[k, 1] = \begin{cases} 1, & \text{если } v[k - 1, 1] = 1 \text{ или } a[k] = b[1] \\ 0, & \text{иначе} \end{cases}$$

$$v[1, l] = \begin{cases} 1, & \text{если } v[1, l - 1] = 1 \text{ или } a[1] = b[l] \\ 0 & \text{иначе} \end{cases}$$

Для произвольного элемента матрицы, "отодвинутого" от левого верхнего края, получаем:

$$v[k, l] = \begin{cases} 1 + v[k - 1, l - 1], & \text{если } a[k] = b[l] \\ \max\{v[k - 1, l], v[k, l - 1]\} & \text{иначе} \end{cases}$$

Пример (bcaccd и abaccbd)

		1	2	3	4	5	6	7	8
		a	b	a	c	c	b	c	d
1	b	0	1	1	1	1	1	1	1
2	c	0	1	1	2	2	2	2	2
3	a	1	1	2	2	2	2	2	2
4	c	1	1	2	3	3	3	3	3
5	c	1	1	2	3	4	4	4	4
6	d	1	1	2	3	4	4	4	5

25 Код Шеннона-Фано. Алгоритм Хаффмена. 3 леммы

Опр

Код называется префиксным, если никакая последовательность не является началом другой (код Шеннона-Фано). Такое свойство называется свойством префикса.

Задача

Пусть в тексте n символов, частота появления каждого $p_i = \frac{n_i}{n}$. Можно кодировать символы последовательностью 0 и 1. Пусть соответствующая полученная длина s_i , $\sigma_i = \sum_i p_i s_i$. Задача $\sigma \rightarrow \min$ по всем наборам длин $\{s_i\}$, удовлетворяющая неравенству Крафта (необходимое и достаточное условие префиксного кода в r -ичном алфавите для данных символов) $\sum r^{-s_i} \leq 1$

Опр

Набор, для которого σ минимальна называется оптимальным

Решение (Шеннона-Фано)

Разбить все символы на две группы с примерно одинаковыми суммарными вероятностями, коды первой начать с 0, второй с 1. Внутри каждой группы сделать то же самое. Полученный код будет близок к минимальному

Решение (решение Хаффмена)

Основывается на трех леммах о наборе $\{s_i\}$:

Лемма (1)

Пусть $\{p_i\}$ - набор вероятностей символов и $\{s_i\}$ - длины оптимальных кодовых комбинаций. Если $p_1 \geq \dots \geq p_n$, то $s_1 \leq \dots \leq s_n$

Док-во

По задаче о перестановке, минимизирующей скалярное произведение двух векторов

Лемма (2)

Длины двух самых длинных символов равны. То есть $s_n = s_{n-1}$

Док-во

Пусть $s_n > s_{n-1}$ и, следовательно, n -я кодовая комбинация - самая длинная. Так как никакая комбинация не является началом другой, то сократив эту на один символ мы снова получим уникальную. Значит начальная кодировка была неоптимальной, противоречие.

Лемма (3)

Рассмотрим наравне с исходной задачей P сокращенную P' , которая получается объединением двух самых редких символов в один. В предположении леммы 1 - это два символа с суммарной вероятностью $p'_{n-1} = p_{n-1} + p_n$. Минимальное значение целевой функции в задаче P' отличается от значения в задаче P на p'_{n-1} , а оптимальный кодовый набор для задачи P получается из решения для задачи P' удлинением на один бит кодовой последовательности для объединенного символа

Док-во

Действительно, каждому кодовому набору для задачи P' можно так, как сказано в утверждении леммы, сопоставить коловий набор для задачи P с равными ждинами самых редких символов. Это удлинение приводит к приращению функции на p'_{n-1}

Теперь можно рассуждать так: в задаче P мы ищем минимум на множестве кодов, которые мы обозначим через C_n , а в задаче P' - аналогично на множестве C_{n-1} . В соответствии с леммой 2 в задаче P можно искать минимум на множестве C'_n , собственном подмножестве C_n , состоящем из таких наборов, в которых две самые длинные кодовые последовательности имеют одинаковую длину. Имеется взаимоднозначное соответствие между C_{n-1} и C'_n и значения целевых функций на соответствующих элементах двух множеств отличаются на постоянное слагаемое, так что минимуму в задаче P' соответствует минимум на C'_n , а он будет минимальным и для задачи P

Алгоритм, который основывается на этих леммах: если в алфавите два символа, то нужно закодировать их 0 и 1, а если больше, то обединить два самых редких символа в один новый символ, решить получившуюся задачу и вновь разделить этот новый символ, приписав 0 и 1 к его кодовой последовательности

Пример

$aaaadbbcccccffff \rightarrow \overbrace{10}^a \overbrace{10}^a \overbrace{10}^a \overbrace{10}^a \overbrace{1100}^d \overbrace{1101}^b \overbrace{1101}^b \overbrace{00000000}^{8c} \overbrace{111}^f \overbrace{111}^f \overbrace{111}^f$

26 Сжатие информации по методу Зива-Лемпеля

Алгоритм (кодирования по Зива-Лемпелю)

X - Входная фраза (некая строка, которую мы строим)

Точкой обозначена конкатенация

1. Занести все возможные символы в словарь. (им всем будет присвоен код)
2. Считаем один символ из сообщения и добавим его в X
3. Считаем символ Y, если это символ конца сообщения, то вернем код X (он уже лежит в словаре), иначе:
 - (a) Если X.Y уже есть в словаре, то присвоим X = X.Y, перейдем к шагу 3
 - (b) Иначе вернем код для X, добавим X.Y в словарь и присвоим входной фразе значение Y X = Y, перейдем к шагу 3

Пример

Пример плохой. На самом деле, если не добавить все возможные символы в словарь, то декодировать строку будет невозможно, либо нам придется передать вместе со строкой еще и словарь, состоящий из односимвольных фраз

abrakadabra

a - 1 b - 2 r - 3 k - 4 d - 5

ab - 6 br - 7 ra - 8 ak - 9 ka - 10

ad - 11 ab - 12 bra - 13

Output: $\begin{smallmatrix} 1 & 2 & 3 & 1 & 4 & 1 & 5 & 1 & 7 & 1 \\ a & b & r & & & & & & & \end{smallmatrix}$

Алгоритм (декодирование)

1. Занести все возможные символы в словарь.
2. В X считать первый код сообщения
3. Считать очередной код Y из сообщения, если Y - конец сообщения, то выдать символ, соответствующий коду X, иначе:
 - (a) если фразы под кодом X.Y нет в словаре, то вывести фразу, соответствующую коду X, а фразу с кодом X.Y занести в словарь (! но присвоить ей другой код, а именно: кол-во элементов в словаре + 1)
 - (b) Иначе присвоить фразе код X.Y и перейти к шагу 3

27 Метод Барроуза-Уилера

Алгоритм

- Составляется таблица всех циклических сдвигов строки.
- Производится лексикографическая сортировка строк таблицы.
- В качестве выходной строки выбирается последний столбец таблицы преобразования и номер строки, совпадающей с исходной.

Пример

Вход	ц. сдвиги	сортировка	Выход
	<u>abacaba</u>	aabacab	
	bacabaa	abaabac	
	acabaab	<u>abacaba</u>	
abacaba	cabaaba	acabaab	bcabaaa, 3
	abaabac	baabaca	
	baabaca	bacabaa	
	aabacab	cabaaba	

$$\text{BWT}(\text{"abacaba"}) = (\text{"bcabaaa"}, 3)$$

Алгоритм (обратного преобразования)

Пусть нам дали $\text{BWT}(S) = (A, x)$

Тогда выпишем в столбик А, отсортируем, слева допишем А, снова отсортируем, так n раз, где n - длина строки A. После последней сортировки мы получим, что строка с номером x - S

Пример (по Григорьевой)

$$S_1 = \text{РЕФРИЖЕРАТОР\$}$$



Чтобы получить S_2 берем первый символ, стоящий перед каждым последним суффиксом:

$$S_2 = \text{ПРЖКРИРТОЕ\$ФАЕ}$$

Теперь составим таблицу, в которой:

1 столбец - строка S_2 2 столбец - номер символа в 3 столбце предыдущей строки (если символа не было, то ставим 0)

3 столбец - ставим символ на первую позицию и записываем оставшуюся часть строки

P	0	P
P	1	P
Ж	0	ЖР
Р	2	РЖ
И	0	ИРЖ
Р	2	РИЖ
Т	0	ТРИЖ
О	0	ОТРИЖ
Е	0	ЕОТРИЖ
\$	0	\\$ЕОТРИЖ
Ф	0	Ф\\$ЕОТРИЖ
А	0	АФ\\$ЕОТРИЖ
E	4	ЕАФ\\$ОТРИЖ

Результат: ЕАФ\\$ОТРИЖ + 010202000004

Теперь попробуем декодировать *каким-то образом строим такую же табличку*

???

28 Избыточное кодирование. Коды Хэмминга

Хорошая статья на [habr](#)

Решение (по Григорьевой)

Пусть наше изначальное сообщение 0110011

Вставим на позиции степени двойки x_i :

$$y = \begin{matrix} x_1^1 & x_1^2 & x_1^3 & x_3^4 & x_1^5 & x_1^6 & x_0^7 & x_4^8 & x_0^9 & x_1^{10} & x_1^{11} \end{matrix}$$

Матрица A (1,2,...11 преобразовали в двоичный код):

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Умножим матрицу A на вектор y:

$$Ay = B = \begin{pmatrix} x_1 + 2 \\ x_2 + 3 \\ x_3 + 2 \\ x_4 + 2 \end{pmatrix}$$

Выберем x так, чтобы сумма была четной:

$$x_1 = 0 \quad x_2 = 1 \quad x_3 = 0 \quad x_4 = 0$$

$$01001100011$$

Теперь допустим ошибку в n и попробуем её исправить:

$$n = \boxed{1}110011 \Rightarrow y = 01\boxed{1}01100011$$

Умножим y снова на матрицу A:

$$B^* = \begin{pmatrix} 3 \\ 5 \\ 2 \\ 2 \end{pmatrix} \stackrel{\text{mod } 2}{=} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Это третий столбец, значит ошибка в первом бите

29 Шифрование с открытым ключом

В системах с шифрованием с открытым ключом используется два ключа: один для шифрования, другой для дешифровки

У каждой стороны есть два ключа: публичный и секретный, который знает только его владелец. Для того чтобы А послал зашифрованное сообщение стороне В необходимо использовать публичный ключ В, потому что лишь В обладает секретным ключом для дешифровки этого сообщения

Примеры:

1. "Рюкзак" Меркла-Хеллмана
2. El Gamal
3. RSA - один из самых популярных

Алгоритм (RSA)

n - большое целое число, произведение двух простых сомножителей p и q

$$\varphi = p \cdot q - p - q + 1$$

Лемма (1)

а взаимно просто с $n \Rightarrow a^\varphi \equiv 1 \pmod{n}$

Док-во

$\Phi(n)$ - мн-во всех чисел, не превосходящих n и вз. простых с ним

$$|\Phi(n)| = \varphi$$

Если $ba \equiv ca \pmod{n}$, то $c - bn$. Таким образом

$$\prod_{b \in \Phi(n)} b = \prod_{b \in \Phi(n)} (ba) = a^\varphi \prod_{b \in \Phi(n)} b \pmod{n}$$

Лемма (2)

$\forall e$ вз. простого с $\varphi \exists! d \in 1 : n$, для которого $ed \equiv 1 \pmod{\varphi}$

Док-во

Если $\text{НОД}(e, \varphi) = 1$, то $\exists d, k: de + k\varphi = 1$ (по алгоритму Евклида). Верно \forall вз. простых e и φ . Однозначность определения d получается док-ом от противного

Алгоритм (RSA)

Получатель:

1. Выбирает два больших простых p и q

2. Перемножает $n = pq$

$$\varphi(n) = (p - 1)(q - 1)$$

3. Выбирает число e , в.п. с $\varphi(n)$

4. Определяет число d

5. Отправляет отправителю открытые ключи n и e

Отправитель:

1. Получает открытые ключи e и n

2. Делит текст на кусочки $< n$ (на в.п. с n длины), представляет их в двоичном виде

3. Каждый кусочек x возводит в степень e

$$x^e \mod n \rightarrow c$$

$$(\text{ex. } e = 11001001 \Leftrightarrow x \cdot x^8 \cdot x^{64} \cdot x^{128})$$

4. Отправляет результат получателю

Дешифрование (получатель):

1. Возводит полученное число в степень d по $\mod n$

$$\Rightarrow c^d = (x^e)^d = (x^{k\varphi(n)+1}) \mod n$$

$$\Leftrightarrow x^{k\varphi(n)}x \mod n = x$$

Пример

$$n = 1093709 = 997 \cdot 1097$$

$$\varphi = 1091616$$

$$e = 397 \quad d = 145777$$

n и e сообщаются отправителям

Кодируем текст:

$$x^397 = x^{1100011012} = x^{256}x^{128}x^8x^4x^1 = y$$

Отправляем:

Получатель вычисляет y^d по $\mod n = x$

30 Сортировки (5 методов)

- Сортировка слиянием (merge sort)

5	1		45	21		16	19		8	90
1	5		21	45		16	19		8	90
1	5	21	45		8	16	19	90		
1	5	8	16	19	21	45	90			

(делим, сортируем и объединяем группы)

- Сортировка вставками (Insertion sort)

5		1	45	21	16	19	8	90
1	5		45	21	16	19	8	90
1	5	45		21	16	19	8	90
.....								

(сортируем по 1 элементу)

- Сортировка Шелла (Shell sort)

5 1 45 21 16 19 8 90

(выбираем промежутки, например, 9,5,3,2,1 ($2^k + 1$) и сортируем внутри групп (рекомендуется $k = \frac{n}{2}$))

- Быстрая сортировка (quicksort)

16	1	45	21	5	19	8	90		
1	5	8		16		45	21	19	90

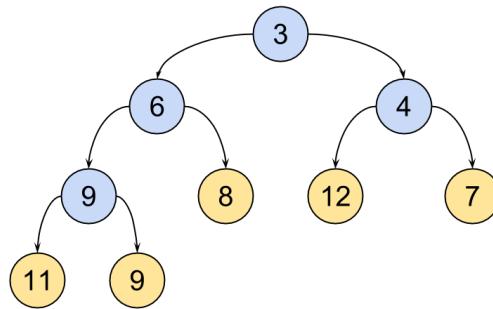
(выбираем опорный элемент и делим массив на меньшие и на большие элементы, рекурсивно повторяем там)

- Иерархическая сортировка (алгоритм сортировки кучей, heapsort)

Опр

Двоичная куча или пирамида — такое двоичное дерево, для которого выполнены следующие три условия:

- (a) Значение в любой вершине не больше (если куча для минимума), чем значения её потомков
- (b) На i -ом слое 2^i вершин, кроме последнего. Слои нумеруются с нуля
- (c) Последний слой заполнен слева направо (как показано на рисунке)



Удобнее всего двоичную кучу хранить в виде массива $> a[0..n-1]$, у которого нулевой элемент, $a[0]$ — элемент в корне, а потомками элемента $a[i]$ являются $a[2i+1]$ и $a[2i+2]$. Высота кучи определяется как высота двоичного дерева. То есть она равна количеству рёбер в самом длинном простом пути, соединяющем корень кучи с одним из её листьев. Высота кучи есть $O(\log n)$, где n — количество узлов дерева

Опр

`siftDown` Если значение измененного элемента увеличивается, то свойства кучи восстанавливаются функцией `siftDown`.

Работа процедуры: если i -й элемент меньше, чем его сыновья, всё поддерево уже является кучей, и делать ничего не надо. В противном случае меняем местами i -й элемент с наименьшим из его сыновей, после чего выполняем `siftDown` для этого сына. Процедура выполняется за время $O(\log n)$.

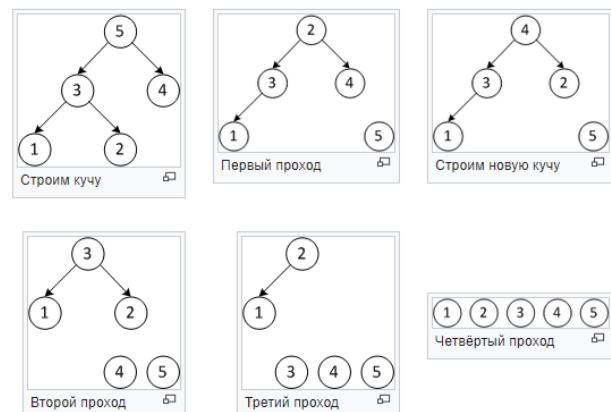
Алгоритм

Необходимо отсортировать массив A , размером n . Построим на базе этого массива за $O(n)$ кучу для максимума. Так как максимальный элемент находится в корне, то если поменять его местами с $A[n - 1]$, он встанет на своё место. Далее вызовем процедуру `siftDown(0)`, предварительно уменьшив `heapSize` на 1. Она за

$O(\log n)$ просеет $A[0]$ на нужное место и сформирует новую кучу (так как мы уменьшили её размер, то куча располагается с $A[0]$ по $A[n - 2]$, а элемент $A[n - 1]$ находится на своём месте). Повторим эту процедуру для новой кучи, только корень будет менять местами не с $A[n - 1]$, а с $A[n - 2]$. Делая аналогичные действия, пока `heapSize` не станет равен 1, мы будем ставить наибольшее из оставшихся чисел в конец не отсортированной части. Очевидно, что таким образом, мы получим отсортированный массив.

Пусть дана последовательность из 5 элементов 3, 2, 4, 1, 5.

Массив	Описание шага
5 3 4 1 2	Строим кучу из исходного массива
<i>Первый проход</i>	
2 3 4 1 5	Меняем местами первый и последний элементы
4 3 2 1 5	Строим кучу из первых четырёх элементов
<i>Второй проход</i>	
1 3 2 4 5	Меняем местами первый и четвёртый элементы
3 1 2 4 5	Строим кучу из первых трёх элементов
<i>Третий проход</i>	
2 1 3 4 5	Меняем местами первый и третий элементы
2 1 3 4 5	Строим кучу из двух элементов
<i>Четвёртый проход</i>	
1 2 3 4 5	Меняем местами первый и второй элементы
1 2 3 4 5	Массив отсортирован



31 Информационный поиск и организация информации

Опр

База данных - совокупность равноправных записей, каждая из которых имеет информационную и идентифицирующие части

Идентиф. часть = ключ записи (уникален для каждого)

Поиск ключа:

1. Дихотомия (деление поисковой области пополам)

Если ключ $\in \mathbb{Z}$, рассматриваем обл. его значений $(a : b)$ выыирается промежуточное значение d

Рассмотрим $a : (d - 1) \ d : b$. Смотрим в какой области искомый ключ и т.д. продолжается до тех пор, пока область не \emptyset (нет ключа) или не стала обозримой (одна запись или небольшой список)

Можно рассматривать как дерево

$\sim j \log_2 N$ (при N записях)

Критерии качества:

- (a) Затраты на поиск в худшем случае
- (b) Мат. ожидание, затрач. на поиск

32 Хеширование

Самый удобный метод. инф. поиска - массив (лучше всего одномерный)
элемент \rightarrow индекс

Элемент с индексом j разыскивается выисканием его адреса $A_j = A_0 + \Delta A \cdot j$

A_0 - адрес элемента с нулевым индексом, ΔA - шаг массива (расстояние между послед. записями)

Если диапазон индексов велик, то использовать уже трудно и затратно

Рассмотрим $\varphi : k \rightarrow I$, где k - мн-во ключей, I - приемл. диапазон индексов. Функция распределяет по индексам ключи почти равномерно, устраняя зависимости.

Пример

Для строк - сопоставим символу кода из ASCII $d = \{d_i\}$

$$A(d) = \sum_i r^i d_i \text{ - полиноминальная взв. сумма}$$

$$r \in \mathbb{Z} \quad (\Sigma) \text{ - ВП}$$

Производим вычисления в остатках по $\mod I$

Задача

На задано некоторое количество точек $S = \{s_i\}$, $i \in 1 : m$, и требуется найти те пары точек $i, j \in 1 : m$, расстояние между которыми меньше заданного порогового значения d . Попарное сравнение точек имеет сложность $O(m^2)$.

Для ускорения полезным оказывается алгоритм П.Элайеса. Найдем прямоугольник, содержащий все точки, и разобьем его на достаточно мелкие одинаковые квадраты. Каждый такой квадрат будет адресоваться двумя целыми числами i_x и i_y - порядковыми номерами по осям X и Y. Каждая точка попадает в какой-то квадрат и теперь для близких точек достаточно сравнивать между собой точки одного квадрата и точки близких квадратов.

Таких квадратов может оказаться слишком много, но многие из них могут оказаться пустыми. Чтобы не хранить это все, мы введем ассоциативный массив - специальную конструкцию, которая внешне будет выглядеть "почти как массив". При её создании нам и понадобится хеширование.

Каждому квадрату, т.к. определяющей его паре чисел (i_x, i_y) , сопоставим индекс $\psi(i_x, i_y) = (ai_x + bi_y) \mod D$, где коэффициенты a и b выбраны взаимно простыми между собой и с D - размером хеш-массива.

33 АВЛ-деревья

Опр

АВЛ-дерево - сбалансированное по высоте двоичное дерево поиска, для каждой его вершины высота ее двух поддеревьев различается не более чем на 1

Теорема

АВЛ-дерево с n ключами имеет высоту

$$h = O(\log N)$$

Опр

Баланс вершины - разница между высотами ее поддеревьев

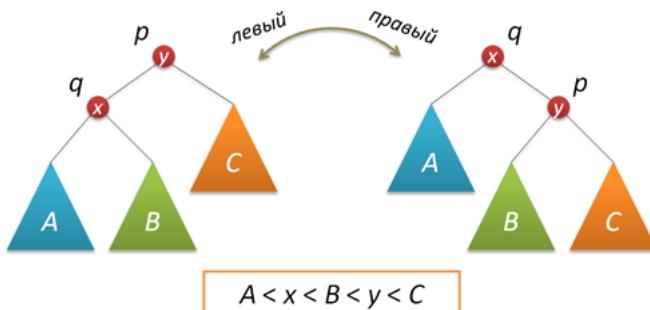
Опр (Балансировка)

Балансировка - операция, которая в случае разницы высот левого и правого поддеревьев $|h(L) - h(R)| = 2$, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится ≤ 1 , иначе ничего не меняет.

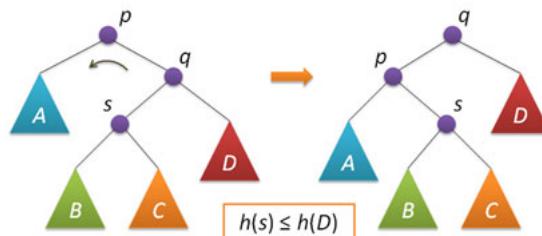
Восстановить баланс можно с помощью поворотов (всего их 4: левый простой, правый простой, левый большой, правый большой)

Простой поворот выполняется при условии $h(s) \leq h(D)$

Простой поворот вправо

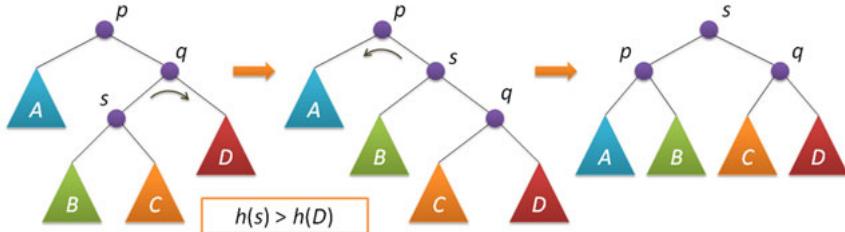


Применение простого поворота



Большой поворот выполняется при условии $h(s) > h(D)$ и сводится к двум простым поворотам

Большой поворот



hint: первым простым поворотом мы отменяем это условие

Опр (Добавление вершины)

Добавляем вершину как в бинарном дереве. Спускаемся вниз, как при поиске ключа t . Если мы стоим в вершине a и нам надо идти в поддерево, которого нет, то делаем ключ t листом, а вершину a его корнем. Дальше поднимаемся вверх и пересчитываем баланс у вершин. Если мы поднялись в вершину i из левого под дерева, то баланс i -ой вершины увеличивается на 1, иначе уменьшается.

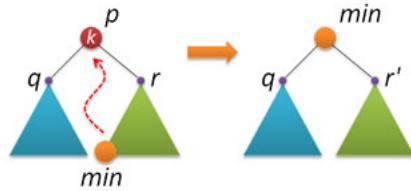
Если мы пришли в вершину и ее баланс = 0 после пересчета, то это значит, что высота под дерева с корнем в этой вершине не изменилась, можно остановить подъем.

Если баланс вершины после пересчета = -2 или 2, то нам необходимо сбалансиовать это поддерево.

Добавление работает за $O(\log n)$, т.к. мы рассмотрим не больше, чем $O(h)$ вершин

Опр (Удаление ключа)

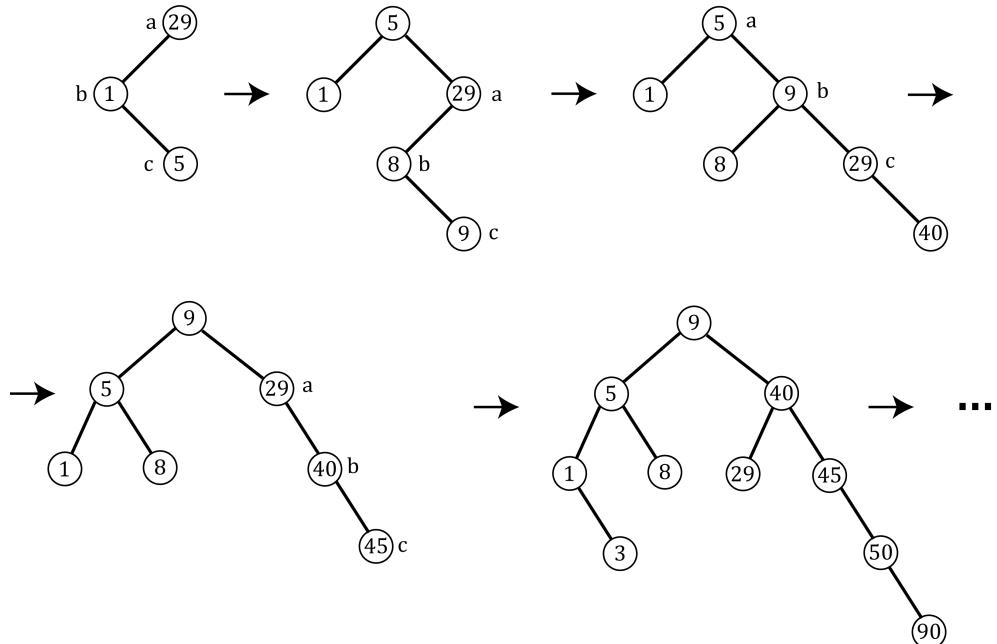
1. найдем удаляемый ключ p в дереве (если не нашли, то ничего не делаем)
2. в правом его поддереве найдем наименьший элемент \min
3. поменяем местами поменяем местами p и \min
4. удалим p
5. сбалансируем все, что выше p



При удалении возможна ситуация, когда вершина r не имеет правого поддерева, тогда по св-ву АЛВ-дерева либо эта вершина имеет слева единственный дочерний узел, либо она является листом. В обоих случаях мы просто удаляем r и возвращаем указатель на левое поддерево.

Пример

29 1 5 8 40 45 50 3 90



34 В-деревья

Опр

В-деревом называется дерево, удовлетворяющее следующим свойствам:

1. Ключи в каждом узле обычно упорядочены для быстрого доступа к ним. Корень содержит от 1 до $2t - 1$ ключей. Любой другой узел содержит от $t - 1$ до $2t - 1$ ключей. Листья не являются исключением из этого правила. Здесь t — параметр дерева, не меньший 2 (и обычно принимающий значения от 50 до 2000).
2. У листьев потомков нет. Любой другой узел, содержащий ключи K_1, \dots, K_n , содержит $n + 1$ потомков. При этом:
 - (a) Первый потомок и все его потомки содержат ключи из интервала $(-\infty, K_1)$
 - (b) Для $2 \leq i \leq n$, i -й потомок и все его потомки содержат ключи из интервала (K_{i-1}, K_i)
 - (c) $(n + 1)$ -й потомок и все его потомки содержат ключи из интервала (K_n, ∞)
3. Глубина всех листьев одинакова.

Свойство 2 можно сформулировать иначе: каждый узел В-дерева, кроме листьев, можно рассматривать как упорядоченный список, в котором чередуются ключи и указатели на потомков.

Алгоритм (поиск ключа)

Если ключ содержится в корне, он найден. Иначе определяем интервал и идём к соответствующему потомку. Повторяем.

Алгоритм (добавление ключа)

Будем называть "деревом потомков некоего узла" поддерево, состоящее из этого узла и его потомков.

Вначале определим функцию, которая добавляет ключ K к дереву потомков узла x . После выполнения функции во всех пройденных узлах, кроме, может быть, самого узла x , будет меньше $2t - 1$, но не меньше $t - 1$, ключей.

1. Если x — не лист,
 - (a) Определяем интервал, где должен находиться K . Пусть y — соответствующий потомок.

- (b) Рекурсивно добавляем К к дереву потомков у.
 - (c) Если узел у полон, то есть содержит $2t - 1$ ключей, расщепляем его на два. Узел y_1 получает первые $t - 1$ из ключей у и первые t его потомков, а узел y_2 — последние $t - 1$ из ключей у и последние t его потомков. Медианный из ключей узла у попадает в узел х, а указатель на у в узле х заменяется указателями на узлы y_1 и y_2 .
2. Если х — лист, просто добавляем туда ключ К.
- Теперь определим добавление ключа К ко всему дереву. Буквой R обозначается корневой узел.
3. Добавим К к дереву потомков R.
4. Если R содержит теперь $2t - 1$ ключей, расщепляем его на два. Узел R_1 получает первые $t - 1$ из ключей R и первые t его потомков, а узел R_2 — последние $t - 1$ из ключей R и последние t его потомков. Медианный из ключей узла R попадает во вновь созданный узел, который становится корневым. Узлы R_1 и R_2 становятся его потомками.

Алгоритм (удаление ключа)

Если корень одновременно является листом, то есть в дереве всего один узел, мы просто удаляем ключ из этого узла. В противном случае сначала находим узел, содержащий ключ, запоминая путь к нему. Пусть этот узел — x.

Если x — лист, удаляем оттуда ключ. Если в узле x осталось не меньше $t - 1$ ключей, мы на этом останавливаемся. Иначе мы смотрим на количество ключей в следующем, а потом в предыдущем узле. Если следующий узел есть, и в нём не менее t ключей, мы добавляем в x ключ-разделитель между ним и следующим узлом, а на его место ставим первый ключ следующего узла, после чего останавливаемся. Если это не так, но есть предыдущий узел, и в нём не менее t ключей, мы добавляем в x ключ-разделитель между ним и предыдущим узлом, а на его место ставим последний ключ предыдущего узла, после чего останавливаемся. Наконец, если и с предыдущим ключом не получилось, мы объединяем узел x со следующим или предыдущим узлом, и в объединённый узел перемещаем ключ, разделяющий два узла. При этом в родительском узле может остаться только $t - 2$ ключей. Тогда, если это не корень, мы выполняем аналогичную процедуру с ним. Если мы в результате дошли до корня, и в нём осталось от 1 до $t - 1$ ключей, делать

ничего не надо, потому что корень может иметь и меньше $t - 1$ ключей. Если же в корне не осталось ни одного ключа, исключаем корневой узел, а его единственный потомок делаем новым корнем дерева.

Если x — не лист, а K — его i -й ключ, удаляем самый правый ключ из поддерева потомков i -го потомка x , или, наоборот, самый левый ключ из поддерева потомков $i + 1$ -го потомка x . После этого заменяем ключ K удалённым ключом. Удаление ключа происходит так, как описано в предыдущем абзаце.

35 Биномиальные кучи

Опр (биноминальное дерево)

B_0 (б.д. порядка 0) - 1 вершина

B_1 - 1 вершина с одним сыном

B_2 - B_1 , к корню которого подвешано ещё одно B_1

...

B_k состоит из двух биномиальных деревьев B_{k1} , связанных вместе таким образом, что корень одного из них является дочерним узлом корня второго дерева

Свойства

1. Биномиальное дерево B_k с n вершинами имеет 2^k узлов.
2. Биномиальное дерево B_k с n вершинами имеет высоту k .
3. Биномиальное дерево B_k с n вершинами имеет высоту k .
4. Биномиальное дерево B_k с n вершинами имеет ровно $\binom{k}{i}$ узлов на высоте i
5. Биномиальное дерево B_k с n вершинами имеет корень степени k ; степень всех остальных вершин меньше степени корня биномиального дерева;
6. В биномиальном дереве B_k с n вершинами максимальная степень произвольного узла равна $\log n$

Док-во

1. База $k = 1$ - верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Так как в дереве порядка $k + 1$ вдвое больше узлов, чем в дереве порядка k , то дерево порядка $k + 1$ имеет $2^k \cdot 2 = 2^{k+1}$ узлов. Переход доказан, то биномиальное дерево B_k с n вершинами имеет 2^k узлов.

2. Докажем по индукции:

База $k = 1$ - верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Так как в дереве порядка $k + 1$ высота больше на 1 (так как мы подвешиваем к текущему дереву дерево того же порядка), чем в дереве порядка k , то дерево порядка $k + 1$ имеет высоту $k + 1$. Переход доказан, то биномиальное дерево B_k с n вершинами имеет высоту k .

3. Докажем по индукции:

База $k = 1$ - верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Рассмотрим i уровень дерева B_{k+1} . Дерево B_{k+1} было получено подвешиванием одного дерева порядка k к другому. Тогда на i уровне дерева B_{k+1} всего узлов $\binom{k}{i} + \binom{k}{i-1}$, так как от подвешенного дерева в дерево порядка $k + 1$ нам пришли узлы глубины $i - 1$. То для i -го уровня дерева B_{k+1} количество узлов $\binom{k}{i} + \binom{k}{i-1} = \binom{k+1}{i}$. Переход доказан, то биномиальное дерево B_k с n вершинами имеет ровно $\binom{k}{i}$ узлов на высоте i .

4. Так как в дереве порядка $k + 1$ степень корня больше на 1, чем в дереве порядка k , а в дереве нулевого порядка степень корня 0, то дерево порядка k имеет корень степени k . И так как при таком увеличении порядка (при переходе от дерева порядка k к $k + 1$) в полученном дереве лишь степень корня возрастает, то доказываемый инвариант, то есть степень корня больше степени остальных вершин, не будет нарушаться.
5. Докажем это утверждение для корня. Степень остальных вершин меньше по предыдущему свойству. Так как степень корня дерева порядка k равна k , а узлов в этом дереве $n = 2^k$, то прологарифмировав обе части получаем, что $k = O(\log n)$, то степень произвольного узла не более $\log n$.

Опр (биномиальная куча)

Представляет собой множество биномиальных деревьев, которые удовлетворяют следующим свойствам:

1. Каждое биномиальное дерево в куче подчиняется свойству неубывающей кучи: ключ узла не меньше ключа его родительского узла (упорядоченное в соответствии со свойством неубывающей кучи дерево),
2. Для любого неотрицательного целого k найдется не более одного биномиального дерева, чей корень имеет степень k .

Замечание

Григорьева ещё упорядочивала деревья

36 Основные определения теории графов

Опр

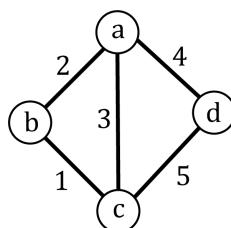
Граф $G = \langle M, N, T \rangle$, где M, N - конечные мн-ва,

$$T : N \rightarrow M \times M$$

M - мн-во вершин, N - мн-во дуг/ребер

Способы задания графа:

1. Схема (на ней просто нумерация)



2. Т-список:

- 1-(a,c)
- 2-(a,b)
- 3-(c,b)
- 4-(b,d)
- 5-(c,d)

3. Матрица смежности $r[M, M]$

$$r[i, j] = 1, \quad \exists(i, j)$$

$$r[i, j] = 0, \quad \text{нет}$$

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0

4. Матрица инденденций $a[M, N]$ (то есть что из вершин выходит)

	1	2	3	4	5
a	1	1	0	0	0
b	0	1	1	1	0
c	1	0	1	0	1
d	0	0	0	1	1

(если граф направленный, то ставить ± 1)

Опр (по Григорьевой)

1. Цепью звенности k называется последовательность $i_1 u_1 i_2 u_2 i_3 \dots u_{k-1} i_k$

$$u_t = (i_t, i_{t+1}), \quad \forall t \in 1, \dots, k-1$$

2. Путём то же самое, но $u_t = i_t \rightarrow i_{t+1}$
3. Дуга, выходящая из вершины и входящая в неё же называется петлёй $u_i = (i_i, i_i)$
4. Цикл - путь, у которой $i_1 = i_k$
5. Контур - путь, у которого $i_1 = i_k$
6. $G' = < M', N' >$ - частичный граф $< M, N >$, если $M' \subset M$, $N' \subset N$

Опр (по Романовскому)

1. Путем звенности k в графе $< M, N, T >$ называется пара отображений $A: 1 : k \rightarrow N$, $V: 0 : k \rightarrow M$, таких что $\forall i \in 1 : k$ выполняется: $\text{End } A(i) = V(i)$ и $\text{beg } A(i) = V(i-1)$
2. Тройка $< M, N', T >$, где $N' \subset N$ называется частичным графом графа $< M, N, T >$ (строго говоря тут T' , ограничение на отображение T)
3. Пусть $M' \subset M$, обозначим через $N(M')$ мн-во всех дуг, у которых начала и концы принадлежат M'
4. Граф $< M', N', T >$ называется подграфом графа $< M, N, T >$. Граф $< M', N', T >$, где $N' \subset N(M')$ называется частичным подграфом графа $< M, N, T >$
5. Частичный подграф $< M', N', T >$ называется простым путем, если:
 - (a) Число его дуг k на единицу меньше числа вершин
 - (b) Можно так перенумеровать M' числами от 0 до k и N' числами от 1 до k , что для любой другой дуги $u \in N'$

$$\text{num}(u) = \text{num}(\text{End } u) = \text{num}(\text{beg } u) + 1$$

То есть в определениях Григорьевой без второго условия это путь, а со всеми - это путь, у которого каждая дуга встречается не более одного раза

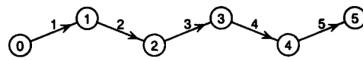


Рис. 8.3. Простой путь

6. Отношение транзитивно, если для любого пути в графе, соответствующему этому отношению, найдется дуга, идущая из начала пути в его конец (т.е. если любой путь "дублируется" соответствующий в графе дугой)
7. Частичный подграф $\langle M', N', T \rangle$ называется цепью, если:
 - (а) Число дуг k на единицу меньше числа вершин
 - (б) Можно так перенумеровать M' числами от 0 до k и N' числами от 1 до k , что для любой дуги $u \in N'$

$$\text{num}(u) = \text{num}(\text{End } u) = \text{num}(\text{beg } u) + 1$$

либо

$$\text{num}(u) = \text{num}(\text{End } u) + 1 = \text{num}(\text{beg } u)$$

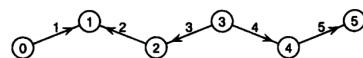


Рис. 8.4. Цепь

8. Дуги делятся на положительно ориентированные и отрицательно ориентированные
9. Частичный подграф $\langle M', N', T \rangle$ называется контуром, если:
 - (а) Число дуг k равно числу вершин
 - (б) Можно так перенумеровать M' и N' числами от 1 до k , что для любой дуги $u \in N'$

$$\text{num}(u) \stackrel{\text{mod } k}{=} \text{num}(\text{End } u) \stackrel{\text{mod } k}{=} \text{num}(\text{beg } u) + 1$$

Т.е. контур - это простой путь, в котором начало и конец совпадают

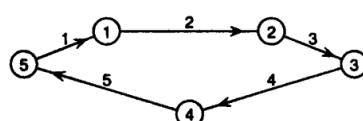


Рис. 8.5. Контур

10. Частичный подграф $\langle M', N', T \rangle$ называется циклом, если:

- (а) Число дуг k равно числу вершин
- (б) Можно так перенумеровать M' и N' числами от 1 до k , что для любой дуги $u \in N'$

$$\text{num}(u) \stackrel{\text{mod } k}{=} \text{num}(\text{End } u) \stackrel{\text{mod } k}{=} \text{num}(\text{beg } u) + 1$$

либо

$$\text{num}(u) \stackrel{\text{mod } k}{=} \text{num}(\text{End } u) + 1 \stackrel{\text{mod } k}{=} \text{num}(\text{beg } u)$$

Т.е. цикл - это цепь, в которой начало и конец совпадают

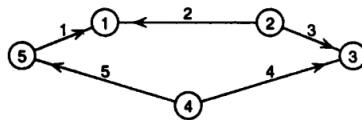


Рис. 8.6. Цикл

Опр

Граф называется связным, если любые его две вершины соединены цепью

Опр

Граф называется ациклическим, если в нем нет циклов

Теорема* (с.218)

37 Построение транзитивного замыкания

Напоминание (Романовский)

Отношение транзитивно, если для любого пути в графе, соответствующему этому отношению, найдется дуга, идущая из начала пути в его конец (т.е. если любой путь "дублируется" соответствующий в графе дугой)

Опр (Григорьева)

$G = \langle M, N \rangle$ - ориентированный называется транзитивно замкнутым, если $\forall i, j \in M$, т.ч. \exists путь из i в $j \Rightarrow \exists$ дуга u : $\text{beg}(u) = i, \text{end}(u) = j$

Задача

Хотим дополнить график так, чтобы он стал транзитивно замкнутым

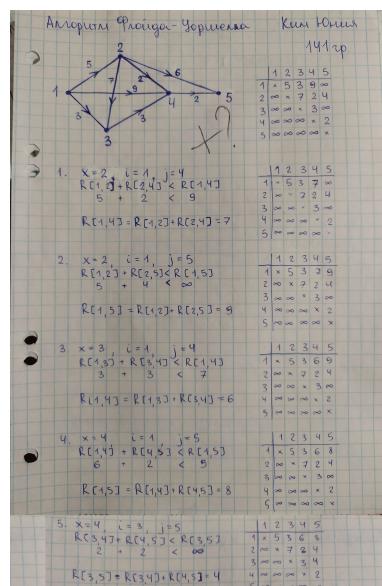
Алгоритм (Уоршела)

for $x = 1$ to m do

for $i = 1$ to m do

for $j = 1$ to m do

if $R[i, x] = 1 \quad R[x, y] = 1 \quad \text{then} \quad R[i, j] = 1$



38 Обходы графа в ширину и глубину. Топологическая сортировка

Алгоритм (обход в ширину)

$$M = M_0 \cap M_1 \cap M_2$$

x_0 - корень обхода (начальная вершина)

M_0 - пройденные вершины

M_1 - рассматриваемые

M_2 - ещё не пройденные

Начальное состояние:

$$M_0 = \emptyset \quad M_1 = \{x_0\} \quad M_2 = M \setminus M_1$$

while ($M_1 \neq \emptyset$)

1. Рассмотрим $x \in M_1$

2. $\forall u \in N_x$ (дуги, инцидентные с x):

Рассмотрим $\text{End}(u)$:

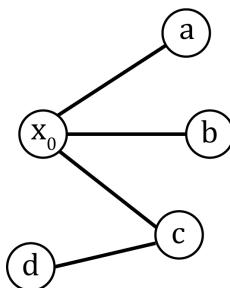
(a) если $\text{End } u \in M_0$: ничего не делаем

(b) если $\text{End } u \in M_1$: ничего не делаем

(c) если $\text{End } u \in M_2 \Rightarrow \text{End } u \rightarrow M_1$ и удаляем вершину из M_2

3. $x \rightarrow M_0$

Пример



1. $M_0 = \emptyset, M_1 = \{x_0\}, M_2 = \{a, b, c, d\}$

Рассмотрим x_0 :

$$U : a, b, c \rightarrow M_1$$

$$x_0 \rightarrow M_0$$

2. $M_0 = \{x_0\}, M_1 = \{a, b, c\}, M_2 = \{d\}$

Рассмотрим а:

$$u : x_0 \in M_0; a \rightarrow M_0$$

3. $M_0 = \{x_0, a\}, M_1 = \{b, c\}, M_2 = \{d\}$

Рассмотрим б:

$$u : x_0 \in M_0; b \rightarrow M_0$$

4. $M_0 = \{x_0, a, b\}, M_1 = \{c\}, M_2 = \{d\}$

Рассмотрим с:

$$u : x_0 \in M_0$$

$$d \in M_0 \Rightarrow d \rightarrow M_1$$

$$c \rightarrow M_0$$

5. $M_0 = \{x_0, a, b, c\}, M_1 = \{d\}, M_2 = \emptyset$

Рассмотрим д:

$$u : x_0 \in M_0, c \in M_0$$

$$d \rightarrow M_0$$

$$\Rightarrow M_0 = \{x_0, a, b, c, d\}$$

- перечисляем вершины от x_0 в порядке возрастания длин путей
(числа ребер в пути)

Алгоритм (алгоритм в глубину)

x_0 - корень

$R(x_0, x_1, \dots, x_k)$ - радиус, нач. сост: $R(x_0)$

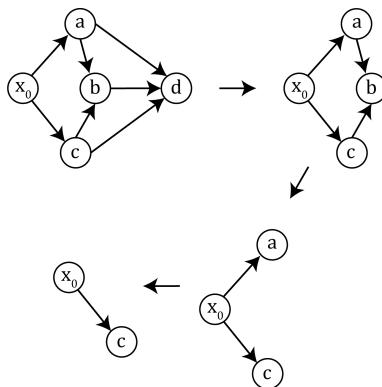
1. Рассмотрим $u \in N_{x_k}$

Если \exists неотмеченная $\frac{u}{\neq R(\dots)}$, то

$$\Rightarrow R(x_0, \dots, x_k, \text{End}(u))$$

Если \nexists неотмеченных \Rightarrow удаляем вершину x_k и все инцидентные ей вершины

Пример



$$R(x_0)$$

$$R(x_0, a)$$

$R(x_0, a, d)$: удаляем d и $(b, d), (a, d), (c, d) \Rightarrow R(x_0, a)$

$R(x_0, a, d)$: удаляем b и $(a, b), (x_0, b), (c, b) \Rightarrow R(x_0, a)$

$R(x_0, a)$: удаляем a и $(x_0, a) \Rightarrow R(x_0)$

$R(x_0, c)$: удаляем c и (x_0, c)

$\Rightarrow R(x_0)$: удаляем $x_0 \Rightarrow R(\emptyset)$

Алгоритм (топологическая сортировка)

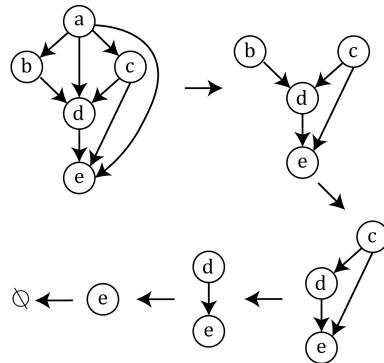
Применяется лишь к ориентированным графам без контуров

N_i^+ - мн-во дуг, входящих в i

N_i^- - мн-во дуг, выход из i

Рассмотрим вершину, которая $N_i^+ = \emptyset$

Удаляем её ($\rightarrow M_0$) и удаляем все её N_i^-



Пример

$$M_0 = \emptyset, \quad M_2 = \{a, b, c, d, e\}$$

1. $N_a^+ = \emptyset \Rightarrow a \rightarrow M_0$
2. $N_b^+ = \emptyset \Rightarrow b \rightarrow M_0$
3. $N_c^+ = \emptyset \Rightarrow c \rightarrow M_0$
4. $N_d^+ = \emptyset \Rightarrow d \rightarrow M_0$
5. $N_e^+ = \emptyset \Rightarrow e \rightarrow M_0$

39 Связность. Компоненты связности и сильной связности

Опр

Граф называется связным, если любые его две вершины соединены цепью

Опр

Компонента связности - max связный подграф

Опр

Пусть задан граф $< M, N, T >$. Определим на мн-ве M отношение C , относя к нему те пары вершин $(x, y) \in M \times M$, которые могут быть соединены цепью и все пары (x, x) . Это отношение, очевидно, рефлексивно и симметрично

Лемма

Отношение C - транзитивно, т.е. из того что цепями могут быть соединены все вершины пары (x, y) и вершины пары (y, z) , следует, что цепью можно соединить и вершину x с вершиной z

Док-во

Рассмотрим цепь $x = x_0, \dots, x_m = y$, соединяющую x и y и перенумерованную от x к y , и цепь $y = y_0, \dots, y_n = z$, соединяющую y и z и перенумерованную от y к z . Пусть k - наименьший номер вершины из первой цепи, содержащейся во второй цепи (такая, очевидно, существует, так как $x_m = y$ является вершиной из первой цепи, содержащейся во второй цепи). Пусть l - номер этой вершины второй цепи. Тогда последовательность вершин

$$v_0 = x_0, \dots, v_k = x_k = y_l, \quad v_{k+1} = y_{l+1}, \dots, v_{k+n-l} = y_n = z$$

"снабженная" соответствующей последовательностью дуг, и образует искомую цепь

Следствие

С задает отношение эквивалентности, а значит задает разбиение мн-ва M на классы эквивалентности

Опр

Граф называется сильно связанным, если любые две различные его вершины i_1 и i_2 можно соединить путем с началом i_1 и концом i_2

Замечание

В любом подграфе можно однозначно выделить сильно связанные подграфы, которые называются его компонентами сильной связности

Опр

Аналогично введем отношение

$$R(i_1, i_2) = \text{"существует путь из } i_1 \text{ в } i_2\text{"}$$

Замечание

Будем считать, что каждая вершина соединена сама с собой. Тогда это отношение рефлексивно. Кроме того, оно транзитивно, так как если взять путь из i_1 в i_2 и путь из i_2 в i_3 дают путь из i_1 в i_3 (цепи так просто не сцеплялись). Обратное отношение R^{-1} также транзитивно $S = R \cup R^{-1}$ и транзитивно и симметрично. Таким образом, мы снова имеем отношение эквивалентности S . Классы эквивалентности в этом отношении и являются компонентами сильной связности.

40 Алгоритм поиска контура и построение диаграммы порядка

Задача

Построить граф $\langle M_s, N_s \rangle$, где элементами M_s являются компоненты сильной связности графа $\langle M, N, T \rangle$, а дугами - те дуги графа $\langle M, N, T \rangle$, концы которых принадлежат разным компонентам. Кроме того, отождествим дуги с одинаковым началом и концом. Такой граф назовем диаграммой порядка (графом Герца) графа $\langle M, N, T \rangle$

Теорема*

Граф компонент сильной связности (диаграмма порядка) не имеет контуров

Док-во*

здесь когда-нибудь будет док-во

Алгоритм (поиска контура)

Состояние вычислительного процесса. Граф $\langle M_0, N_0 \rangle$ и простой путь $\langle M', N' \rangle$

Начальное состояние. $M_0 = M$, $N_0 = N$, путь состоит из любой вершины $i_0 \in M$

Стандартный шаг: Пусть k - длина пути, i_k - конец пути. Рассмотрим множество дуг из N_0 , для которых вершина i_k является началом:

$$N^-(i_k) = \{j \in N_0 | j = i_k\}$$

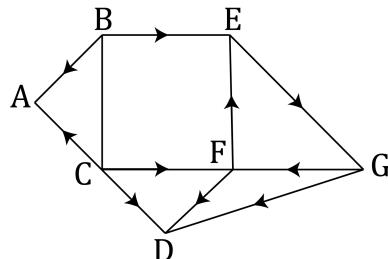
Если мн-во $N^-(i_k)$ не пусто, выберем в нем произвольную дугу, обозначим её через j_{k+1} , а её конец через i_{k+1} . Если нет, то увеличим k на единицу, нарастив путь $\langle M', N' \rangle$

Если мн-во $N^-(i_k)$ пусто, объявим вершину i_k тупиковой и удалим её из M_0 , а из N_0 удалим все дуги, для которых i_k является концом. Путь $\langle M', N' \rangle$, при этом следует уменьшить на одну дугу, если $k > 0$, а если $k = 0$, нужно искать очередную вершину, ещё оставшуюся в мн-ве M_0

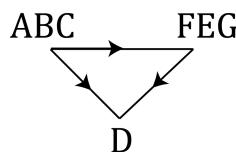
Алгоритм (построения диаграммы порядка)

Модифицируем предыдущий алгоритм. Заменим вершины графа на укрупненные, состав которых изменяется в ходе вычислений. Для каждой укрепленной вершины перебираются дуги, выходящие из вершин, составляющих эту укрепленную вершину. Дуги, привратившиеся из-за укрепления в петли игнорируются

Пример



1. $R(A)$
2. $R(A, B)$
3. $R(A, B, C)$
4. $R(A, B, C, A)$ - контур
5. $R((ABC), D)$ - тупик
6. $R((ABC), F)$
7. $R((ABC), F, D)$ - тупик
8. $R((ABC), F, E)$
9. $R((ABC), F, E, G)$
10. $R((ABC), F, E, G, D)$ - тупик
11. $R((ABC), F, E, G, F)$ - контур
12. $R((ABC), (F, E, G))$ - уде были (через BE)



41 Теорема о связном подграфе

Теорема

В связном графе $G = \langle M, N, T \rangle$ можно выделить связный подграф $T = \langle M, N', T \rangle$, где $|N'| = |M| - 1 = k$ и можно пронумеровать вершины из M числами $0 : k$, а ребра $1 : k$ т.ч. для любой дуги $u \in N'$:

$$\text{num}(u) = \max\{\text{num}(\text{beg } u), \text{num}(\text{End } u)\}$$

Док-во

Используем индукцию. Пусть строим частичный подграф графа $\langle M', N', T \rangle$, расширяя его мн-во вершин и мн-во дуг до тех пор, пока само мн-во M' не совпадет с самим M . При этом на каждом шаге построения число вершин будет на 1 больше числа дуг, частичный подграф будет связным, и на его вершинах и дугах будет определена требуемая нумерация.

Выберем произвольную вершину $i_0 \in M$ и примем первоначально $M' = \{i_0\}$, $N' = \emptyset$, $\text{num}(i_0) = 0$. То есть в начале $k = 0$

Предположим, что уже построен частичный подграф $\langle M', N', T \rangle$, обладающий перечисленными св-ми и такой, что $M \subset M' \neq \emptyset$

Выберем какую-либо вершину \bar{i} , не входящую ещё в M' . Так как граф $\langle M, N, T \rangle$ связан, существует цепь, соединяющая i_0 и \bar{i} . В этой цепи начальная вершина принадлежит M' , а конечная \bar{i} не принадлежит, и значит, в цепи найдутся две последовательно две нумерованные вершины i' и i'' , соединенные дугой u' , такие что $i' \in M'$, $i'' \notin M$. Добавим теперь вершину i'' в мн-во M' , а дугу u' в мн-во N' , увеличив число j на 1 и приписав вершине и дуге это новое значение k в качестве их номера в нумерации. Новые M' и N' обладают всеми перечисленными св-ми: дуг на 1 меньше чем вершин, граф связан, а така как все старые вершины были соединены между собой цепью, по предположению, i'' имеет соединение с i' , а через нее со всеми остальными вершинами. Нумерация определена, а на ней выполнено соотношение, в том числе и для новой дуги, номер которой совпадает с номером i'' и больше номера i' , который был введен раньше

Этот процесс можно продолжать, пока не множество M' не совпадет с M

Если $|N| < |M| - 1$, граф не может быть связным

Если $|N| > |M| - 1$, граф содержит циклы

42 Деревья. Теорема о шести эквивалентных определениях дерева

Теорема

1. Связный граф без циклов
2. Связный граф, в котором дуг на 1 меньше, чем вершин
3. Граф без циклов, в котором дуг на 1 меньше, чем вершин
4. Минимальный связный граф, т.е граф, который при удалении любого ребра перестает быть связным
5. Максимальный граф без циклов
6. Граф, в котором две любые вершины соединены цепью и притом только одной

Док-во

1)→ 5). Докажем, что если граф без циклов связен, то это максимальный граф без циклов. Действительно, при добавлении к графу любой дуги замыкается цепь, соединяющая ее начало и конец, и получается цикл.

5)→ 6). Если при добавлении дуги появляется цикл, значит, эта новая дуга замкнула цепь, соединяющую ее начало и конец. Значит, любые две вершины соединены цепью. Если бы какие-либо вершины были соединены двумя цепями, из них было бы легко построить цикл.

6)→ 4). То, что граф связен, очевидно. Так как для любых двух вершин соединяющая их цепь единственна, для начала и конца любой дуги этот путь — именно сама эта дуга. Ее удаление разрывает цепь между ними.

4)→ 2). Нужно доказать, что, если граф минимально связный, в нем число дуг n на 1 меньше, чем число вершин m . Воспользуемся следствиями из теоремы об оставном дереве. Очевидно, для связности нужно требовать выполнения неравенства $n \geq m - 1$. Но при $n > m - 1$ в графе должна найтись дуга, не входящая в оставное дерево, удаление которой не нарушает связности графа.

2)→ 3). Доказываем, что если $n = m - 1$, то связный граф не имеет циклов. Воспользуемся нумерацией, о которой говорится в теореме об оставном дереве. Ведь все вершины и дуги имеют номера. Пусть в графе есть цикл. Возьмем в нем вершину с наибольшим номером. Она инцидентна двум дугам, для каждой из которых этот номер — максимум из номера начала и номера конца и, следовательно, равен номеру дуги. Но две дуги иметь один и тот же номер не могут.

3)→ 1). Если в графе без циклов $n = m - 1$, то он связан. Действительно, если он не связен, в нем найдется компонента связности k , в которой $n_k > m_k - 1$, и значит, есть цикл. \square

43 Задача о кратчайшем оствовном дереве. Алгоритм Прима с.208

Опр

Дерево, являющееся частичным графом связного графа называется оствовным деревом

Задача

Пусть каждой дуге j графа $\langle M, N, T \rangle$ сопоставлено неотрицательное число $l[j]$, именуемое длиной этой дуги. Требуется построить такое оствовное дерево $\langle M, N', T' \rangle$, у которого сумма длин дуг $\sum_{j \in N'} l[j]$ была бы минимальна

Алгоритм (Прима)

Теорема

Док-во

13. Задача о кратчайшем деревянном дереве. Алгоритм Прима.

Задача о кратчайшем деревянном дереве.

Числовая строка $G = \langle M, N \rangle$, $\ell[N] \geq 0$ - длина всех дуг

Нужно найти T -дерево дерево, $T = \langle M, N' \rangle$, $N' \subset N$, чтобы

$$\sum_{u \in N} \ell(u) \rightarrow \min$$

Решение алгоритм линейное время задачи - алгоритм Прима.

M' - граница.

N' - множество дуг

Кратчайшее дерево.

$$P_0 = \{i_0\}$$

$$N' = \emptyset$$

$$T_0 = \{i_0\}, \emptyset\}$$

На $k+1$ -ом шаге

$$T_k = \langle M_k, N_k \rangle$$

$$U_k = \{u \in N \mid \begin{array}{l} \text{beg}(u) \in M_k \\ \text{end}(u) \notin M_k \end{array}\} - \text{нашёл}$$

Шаг: Выбрать $\bar{u} \in U_k$, так что $\ell(\bar{u}) = \min \{\ell(u) \mid u \in U_k\}$

$$M_{k+1} = M_k \cup \{\bar{u}\}, N_{k+1} = M_k \cup \{\text{end}(\bar{u})\}$$

Когда $M_k = M$, алгоритм завершён.

Теорема. Алгоритм Прима строит минимальное дерево дерево

Задача о кратчайшем дереве.

$T_D = \langle M, N_D \rangle$ - наше дерево (без i -го алгоритма Прима)

$T_{Opt} = \langle M, N_{Opt} \rangle$ - минимальное

$\langle M_k, N_k \rangle$ - текущее дерево, получаемое в алгоритме на k -ом шаге, $k = 0, \dots, |M|-1$.

$d(N_{Opt}) = \max \{k \mid N_k \subset N_{Opt}\}$ - максимальная длина пути к следующему шагу

нашего кратчайшего шага, на котором $N_k \subset N_{Opt}$ (а дальше стало отрываться).

$$S = d(N_{Opt}) + 1, \text{ тогда } u_k \in N_{Opt} \text{ для } k < S \text{ и } u_S \notin N_{Opt}$$

Рассмотрим $u_S \in N_{Opt}$; винеце с центром, соединяющим $\text{beg}(u_S)$ и $\text{end}(u_S)$, она содержит некий член, принадлежащий вершине номер S , а

и дуга $- \langle S, i \rangle$ в член, соединяющий их, находит нара соседних

вершин i' и i'' , у которых член $< S$, а и дуга $- \langle S \rangle$ шага.

Соединяющие их дуги $u' \in U_S$ - кратчайший шаг шага.

T, u_S - минимум, то $\ell[u_S] \leq \ell[u']$. Заменя u' на u_S в N_{Opt}

необходимо уменьшить суммарную длину дуг.

При этом $d(N_{Opt})$ увеличивается на шаг $< M, N_{Opt} \rangle$ оставшегося дерева.

• $d(N_{Opt})$ увеличивается, т.к. $N_S \subset N_{Opt}$,

• При оставшемся дереве, т.к.

1) кратчайшее дуга содержит члены

2) член содержит члены:

Помним, что каждая вершина содержит члены, например, с $\text{beg} u_S$:

- член, не содержащий i' , не является

- член, избыточный член i' , содержит член i' и i'' (затрагивает член u' и член i'' затрагивает член u_S)

- член от i' до $\text{beg} u_S$ можно заменить членом от i' до $\text{end} u_S$,

а затем дугой u_S .

44 Алгоритм Краскала

Алгоритм (Краскала)

Теорема

Док-во

Теорема

Док-во

М. Алгоритм Краскала. Доказательство.

Это второй алгоритм для решения задачи о минимальном дереве.

Сортируем ребра по возрастанию длины

$$l(u_1) \leq l(u_2) \leq \dots \leq l(u_n)$$

Начальное состояние - $\langle M, N_0 \rangle$, $N_0 = \emptyset$, $|M| = M_{\text{мин-чата}}$.
Граф из M ициализирован вершинами.

На k -ом шаге граф $\langle M, N_k \rangle$ - граф из исходных компонент связности, состоящих из концов - дерево.

Ребро u_k , начиная с конца, которого приближается ближайшим компонентам связности (то будет обрабатывать членов) и состоящим N_k из N_{k-1} и этой дру.

Когда число компонент связности равно 1, алгоритм завершён.

Теория. Если промежуточные длины в алгоритме краскала в порядке возрастания длины, дерево $\langle M, N_k \rangle$ имеет минимальную длину.

Доказательство.

$T_{kp} = \langle M, N_{kp} \rangle$ - дерево, полученные в k -ом шаге алгоритма.

$$\sum_{u \in N_{kp}} l(u) = f_{kp}.$$

Одно изображение. Раньше $T_{opt} = \langle M, N_{opt} \rangle$ - некое оптимальное дерево.

$$\sum_{u \in N_{opt}} l(u) = f_{opt} < f_{kp}.$$

"Переделка" оптимального дерева по алгоритму краскала.

$N_{opt} = (\bar{e}_1, \bar{e}_2, \dots, \bar{e}_{m-1})$ - набор ребер в порядке исключения

$N_{kp} = (u^1, u^2, \dots, u^{m-1})$

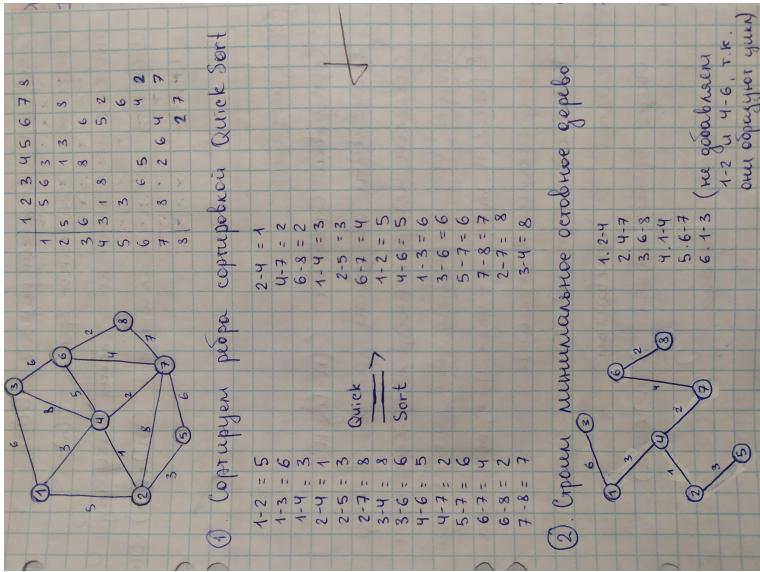
$e_i \neq u^i$ на каком-то шаге

Из добавивши к N_{opt} - получаем член; нужно удалить этот член, который будет дальше падать.

Утверждение: в цикле будет всего ℓ_p , $\ell(\ell_p) \geq \ell(u^t)$ (такое можно и хотят не делают в u^t одной комм. сдвигают первое (или второе) u^t, u^2, \dots, u^t не обнуляют цикл $\Rightarrow \ell_1, \ell_2, \dots, u^t$ не обнуляют цикла).

Значит ℓ_p не u^t и даёт необратимое сужение (значит не per^t , т.к. $\langle M, N_{\text{opt}} \rangle$ обратимо генерируется).

Пример



45 Задача о кратчайшем пути. Алгоритм Дейкстры

Задача

Алгоритм* (с.224)

Алгоритм (Дейкстры)

15. Задача о кратчайшем пути. Алгоритм Дейкстры. Доказательство.

Постановка задачи.
 Задан граф $G = \langle M, N \rangle$ и конфигурация α (расстояние от вершины i^+ до i^-)
 и целевое число $f(x_0)$ -функция. Найти кратчайший путь с начальной вершиной i^+ и конечной i^- и весом не превышающим $f(x_0)$.
 Несколько вариантов этой задачи:

1. Найти кратчайший путь из вершины i^+ в вершину i^- .
2. Найти кратчайший путь от i^+ до j .
3. Дана x_0 . Найти кратчайший путь от x_0 до всех вершин j .

Алгоритм Дейкстры для поиска кратчайшего пути

$G = \langle M, N \rangle$ - исх. граф. $M = M_0 \cup M_1 \cup M_2$.
 M_0 - вершина, близко до которой уже близко
 M_1 - вершина, близко до которой все еще
 M_2 - вершина, близко до которой еще не близко.

$M_0 = \emptyset$, $M_1 = \{x_0\}$, $M_2 = M \setminus \{x_0\}$.
 $f(x)$ - вес до вершины. $f(x_0) = 0$, $f(v) = \infty$. $g(v)$ - стоимость пути.

Безусловно: $f(x^*) = g(x^*)$ - стоимость кратчайшего пути.

While ($M_1 \neq \emptyset$):

1. $f(x^*) = \min_{x \in M_1} f(x)$ - берется вершина x^* из M_1 .
2. $\forall u \in N_{x^*}$ - просматриваются все ребра, которые выходят из нее

if $f(\text{end}(u)) > f(x^*) + l(u)$ then - если расстояние до конца - это
 вершина end(u) возрастает, то мы можем обновить
 $f(\text{end}(u)) := f(x^*) + l(u)$; $g(\text{end}(u)) := x^*$; $f(x^*) + l(u)$

$M_1 = M_1 \cup \{\text{end}(u)\}$;

3. $x^* \rightarrow M_0$ ($M_0 = M_0 \cup \{x^*\}$, $M_1 = M_1 \setminus \{x^*\}$)

end while;

Пример.

	1	2	3	4	5	6
$f(x)$	0	∞	∞	∞	∞	∞
$g(x)$	0	∞	∞	∞	∞	∞
1-2	2	2	∞	∞	∞	∞
1-3	8	∞	10	∞	∞	∞
2-3	2	2	10	∞	∞	∞
2-4	5	5	10	5	∞	∞
3-4	4	5	10	4	∞	∞
3-5	1	5	10	4	1	∞
4-5	8	5	10	4	1	8
4-6	7	5	10	4	1	7
5-6	3	5	10	4	1	3

Текущая.

- 1) $v \in M_0$, $f(v)$ - груз грузу края листа нутр., все вершины
крайних $\in M_0$.
- 2) $v \in M_0$, $f(v)$ - груз кр. нутр., со стороны боков, что все боковые
крайние несплошн., $\in M_0$.

Доказательство.

От противного. Рассмотрим \exists нутр. p : $f(p) < f(x^*)$

$$f(p) = f(x_0, y) + f(y, x^*) \\ \geq f(x^*) \geq 0$$

Пусть одинаково небольшой для вершин v, w - ей нутр. в краю листа
нужен нутр. из x_0 в v . Но приложением ресурса в w получается небольшое.

• нутр. $f(w) < f(v)$, тогда краинограничное ресурсо (w, v) . Ресурсение до
 w небольшое итого $\Rightarrow \exists$ нутр. из x_0 в v бесц. $f(w) + l(w, v) = f'(v) < f(v)$.

• нутр. $f(w) > f(v)$. Тогда краинограничное сопровождение вершины w .
В таком случае было задействовано ресурсо $(w, v) \Rightarrow$ текущий нутр.
ресурсение до v остался нутр. $f(v)$, а далее не могла уменьшиться
неравенство.

Доказательство.

Нутр. $g \in \Omega$, то где находится вершина из списка небольшого
бокового ресурсения. (находится не изнутри).

Небольшое, то где есть вершина из M_0 окончательно
использовано ресурс. и это H , будет дальнейшем тоже верно.

$f(v^*) = \min_i f(v_i)$. Такой, который уменьшит груз в M_0 ,

а груза небольшого сбрасывания

от v_0 сюда.

Одновременно. \exists наимен. нутр. неподалеку, но он тоже

использует ресурс, поэтому сократить который

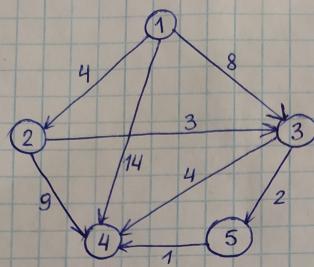
$$l(x_0, w) + l(w, v^*) \geq f(v^*) ?!$$

т.к. ~~боковидные нутр.~~ ^{небольшое} нутр.
но аналогично

Пример

Алгоритм Дейкстры

Ким Юния 141 гр.



X

	1	2	3	4	5
f(x)	0	∞	∞	∞	∞
g(x)	null	null	null	null	null
	4/1	8/1	14/1	∞	null
		7/2	13/2	∞	null
			11/3	9/3	
				10/5	

итоговый путь:

1 → 2 → 3 → 5 → 4

подходит!

обнула поставив

Сложность:

Хранить непосещенные вершины в двоичной куче, извлекать минимум и обновлять элемент можно за $O(\log n)$

Цикл выполняется порядка n раз

Количество смен меток не превосходит m ,
Общее время работы: $O(n \cdot \log n + m \cdot \log n) = O(m \cdot \log n)$

46 Алгоритм Левита

Опр (Алгоритм Левита)

M_0 - вершины, расстояние до которых уже вычислено, (но возможно не окончательно)

M_1 - очередь вершин, расстояние до которых вычисляется

M_2 - вершины, расстояние до которых еще не вычислено

$d[i]$ - расстояние до i -ой вершины

Изначально в M_1 лежит стартовая вершина, в M_2 все остальные.

M_0 - пусто

А $d[i] = +\infty$, кроме $d[start] = 0$

На каждом шаге берем вершину из M_1 (первый элемент в очереди). Пусть V - это выбранная вершина. Переводим эту вершину в M_0 . Программируем все ребра, выходящие из V . Пусть T - второй конец текущего ребра (то есть не равный V), а L - длины текущего ребра.

Тогда

1. Если T из M_2 , то переводим ее в M_1 (в конец очереди).

$$d[T] = d[V] + L$$

2. Если T из M_1 , то пытаемся улучшить значение $d[T]$

$$d[T] = \min(d[T], d[V] + L)$$

3. Если T из M_0 , и если $d[T]$ можно улучшить, то улучшаем $d[T]$, а вершину возвращаем в M_1

В Романовском используется дополнительная "срочная" очередь M''_1 , в которую возвращаются элементы из M_0 и на каждом шаге сначала берут вершину оттуда, а уже потом из основной очереди. Вероятно, такой подход ускорит работу алгоритма.

47 Задача о кратчайшем дереве путей

Задача

построить оствовное дерево на ориентированном графе с корнем в вершине v

При постановке математических задач нужно внимательно следить за деталями — очень близкие по формулировке задачи могут поразительно различаться по методу решения и по структуре получаемого ответа. В качестве примера рассмотрим здесь две задачи о наилучшем дереве путей, в каждом случае — наилучшем по своему показателю. Первая из них очень похожа на задачу о дереве кратчайших путей (и одновременно на задачу о кратчайшем оствовном дереве, от которой отличается тем, что искомое дерево должно быть ориентированным и задана корневая вершина, из которой дерево “растет”) и решается достаточно просто, вторая задача очень трудна.

Итак, первая задача.

Задача о кратчайшем дереве путей. Пусть задан граф $G = \langle M, N \rangle$, каждой дуге которого $j \in N$ сопоставлено положительное число $l[j]$. Пусть задана также вершина i_0 . Требуется построить ориентированное дерево $\langle M, N' \rangle$ с корнем i_0 (так что в нем должны существовать пути из i_0 во все остальные вершины), сумма длин дуг которого $\sum_{j \in N'} l[j]$ минимальна. \square

Пример. Различие между деревом кратчайших путей и кратчайшим деревом путей можно увидеть уже на очень простом примере (рис. 8.15). Для дерева кратчайших путей (левая картинка) нужно выбирать самый короткий путь до каждой вершины, и вершины 2 и 3 соединяются с вершиной 0 прямыми путями. Для кратчайшего дерева (правая картинка) нужно обеспечить

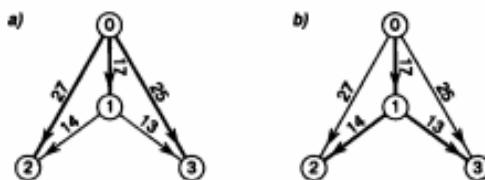


Рис. 8.15. Дерево кратчайших путей (a) и кратчайшее дерево путей (b)

Алгоритм (двух китайцев)

Пусть $M' = M \setminus \{i_0\}$. Прежде всего напомним, что для построения ориентированного дерева нужно для каждой вершины из M' выбрать одну входящую дугу, и если граф, составленный из этих дуг, будет связен, то получится ориентированное дерево. Действительно, это будет дерево, циклов и контуров в нем нет, и для каждой вершины $i \in M'$ есть входящая дуга. Стряя из вершины i путь в обратном направлении, мы должны где-то остановиться, и можем это сделать только в тот момент, когда путь попадет в корень i_0 .

Здесь и в дальнейшем нам понадобится следующая операция спуска до нуля: пусть на конечном множестве A задана функция $f : A \rightarrow \mathbb{R}$. Вычислим $\phi = \min\{f(a) \mid a \in A\}$ и положим $F(a) = f(a) - \phi$. Получившаяся функция неотрицательна на множестве A , и ее минимум равен 0. Переход от f к F называется спуском до нуля на множестве A .

Чтобы различать объекты разных итераций, обозначим исходный граф через $G_0 = \langle M_0, N_0 \rangle$.

Для каждой вершины $i \in M'$ спустим до нуля длины входящих в эту вершину дуг. Так как в искомое дерево должна быть включена ровно одна дуга, входящая в i , от этого спуска значение целевой функции уменьшится на вычитаемое для любого допустимого решения, и минимальное решение останется минимальным.

Построим частичный граф $P_0 = \langle M_0, Z_0 \rangle$, используя для множества Z_0 только дуги нулевой (после выполненного пересчета) длины и взяв из них по одной дуге, входящей в каждую вершину $i \in M'$. Если этот граф окажется ориентированным деревом с корнем в i_0 , оно автоматически будет оптимальным решением.

Если граф P_0 окажется не деревом, мы будем его перестраивать. Построим его диаграмму порядка $D_0 = \langle M_1, Z'_0 \rangle$. Каждая вершина диаграммы есть некоторое подмножество множества M_0 , так что M_1 образует разбиение множества M_0 , причем вершине i_0 соответствует однозначное множество. Заметим, что мощность множества M_1 всегда меньше, чем M_0 .

Образуем граф $G_1 = \langle M_1, N_1 \rangle$ таким образом: сначала у каждой дуги заменим начало и конец каждой вершины M_0 соответствующими вершинами из M_1 , а затем "склеим" все дуги с одинаковыми началом

и концом. Примем в качестве длины этой дуги минимальную из длин всех составляющих ее дуг. Будем строить тем же способом кратчайшее дерево путей в G_1 — создавать P_1 , затем, если потребуется, D_1 , по нему G_2 и т.д. В конечном счете на какой-то итерации k мы построим граф G_k , в котором найдется ориентированное дерево.

После этого пойдет обратный процесс: дерево в графе G_k (обозначим его через T_k) мы превратим в дерево T_{k-1} в графе G_{k-1} . Для этого каждую вершину в G_k , являющуюся классом эквивалентности в G_{k-1} , превратим во фрагмент дерева. Это делается очень просто: пусть M_α — множество вершин этого класса. В графе G_{k-1} оно является множеством дуг некоторого контура. Удаление из этого контура одной дуги превращает контур в путь. Само множество M_α является вершинной дерево T_k , и в него входит некая дуга j' , являющаяся дугой и в графе G_{k-1} . Удалим из цикла дугу j , для которой $\text{end } j = \text{end } j'$. Сделав это для каждого класса, мы и получим дерево T_{k-1} . Процесс закончится построением дерева T_0 для графа G_0 .

Для доказательства минимальности дерева нам достаточно проверить корректность одного перехода.

Лемма. Кратчайшее дерево путей T_0 в графе G_0 можно получить, найдя кратчайшее дерево путей T_1 в графе G_1 , а затем заменив в нем каждый класс деревом, построенным из дуг нулевой длины.

Доказательство. Зафиксируем любое дерево путей (M_0, N') и покажем, что в графе G_0 найдется дерево не большей длины, имеющее такую структуру, как сказано в лемме. Для такой структуры дерева необходимо и достаточно, чтобы в каждое из подмножеств входило только по одной дуге. Меньше быть не может, иначе получится отдельная компонента связности. Если же в какое-то подмножество входит больше чем одна дуга, то все дуги, кроме одной, можно заменить дугами нулевой длины, лежащими внутри подмножества, что разве лишь уменьшит длину дерева и не нарушит связности. Повторяя это преобразование нужное число раз, мы добьемся искомой структуры дерева. \square

Теперь мы можем выписать сам алгоритм.

Алгоритм построения кратчайшего дерева путей

1. Спустить до нуля длины дуг, входящих в каждую вершину M' .
2. Построить граф P_0 , взяв для каждой вершины M' по одной дуге нулевой длины.
3. Если этот граф является ориентированным деревом, завершить работу алгоритма.
4. В противном случае построить новый граф G_1 , вершинами которого являются сильно связные компоненты графа P_0 .
Этот пункт определяет рекурсивность алгоритма, и мы рассмотрим его более детально.
 - (a) Построить новый граф G_1 .
 - (b) Решить для графа G_1 задачу о кратчайшем дереве.
 - (c) Заменить каждую вершину полученного дерева деревом из нулевых дуг внутри соответствующей сильно связной компоненты.

48 Сетевой график и критические пути. Нахождение резервов работ

Задача

Алгоритм

Чтобы избежать таких неудобств, предложили другой подход: строить граф зависимостей между работами, считая работы дугами. Вариант так и называется графиком “работы-дуги”. В этом подходе важно разобраться в том, что представляют собой вершины получающегося графа. Они называются *событиями* и в руководствах по сетевым графикам туманно определяются как начала и концы работ. Четкого и короткого определения событий не дается, и это не случайно. Но так как важно, что у некоторых работ начала или концы становятся общими, то мы сейчас рассмотрим механизм этого объединения, чтобы прояснить, что же такое события.

Исходной информацией для построения графа служат множество работ W и заданное на нем отношение предшествования: каждой из работ $w \in W$ сопоставляется некоторое подмножество (возможно, пустое) ее предшественников P_w . Это реальные данные, в том виде, как они появляются при разработке проекта, и они могут быть неполны и противоречивы.

Противоречивость связана с возможностью контуров: работы, составляющие контур, не могут начаться и ждут друг друга. Первая задача, встающая при разработке проекта, — это поиск контуров и их размыкание. Отметим, что формализовать процесс размыкания контуров (а одна подходящая задача нам встретится в дальнейшем) в данной ситуации невозможно; априорно допустимо любое упорядочение работ, и единственная приемлемая возможность разрешения коллизии — это экспертный ее анализ, вне рамок математической модели.

После того как отношение предшествования P очищено от контуров, его следует транзитивно замкнуть, так как разработчики, представляющие исходную информацию, видят, как правило, только ближайших предшественников каждой работы. Транзитивное замыкание \bar{P} найдет для каждой работы всех ее предшественников.

Теперь появляется возможность некоторой унификации начал работ определения тех из них, которые имеют общий набор предшественников. Рассмотрим для этого совокупность множеств \bar{P}_w , $w \in W$, и удалим в этой совокупности повторы; мы получим некоторый набор множеств π_α , $\alpha \in A$, и каждому $w \in W$ будет отвечать индекс $\alpha(w)$, такой что $\bar{P}_w = \pi_{\alpha(w)}$.

Следующий шаг может вызвать у читателя некоторое недоумение, поэтому лучше немного объяснить. Нам сейчас существенно классифицировать работы и разбить их на группы, одинаково входящие или не входящие в множества π_α . У нас есть математический аппарат для такой классификации — это аппарат разбиений множеств с операцией произведения разбиений.

Каждому π_α сопоставим разбиение $\Pi_\alpha : W = \pi_\alpha \cup (W \setminus \pi_\alpha)$. Рассмотрим произведение $\bar{\Pi} = \{\rho_\beta\}_{\beta \in B}$ разбиений Π_α . Разбиение $\bar{\Pi}$ мельче любого из разбиений-сомножителей, а следовательно, каждое из множеств π_α является объединением некоторого набора множеств ρ_β . Нам этот набор понадобится. Обозначим через $B(\alpha) \subset B$ тот набор индексов, для которого $\pi_\alpha = \bigcup_{\beta \in B(\alpha)} \rho_\beta$.

Так как объединение всех множеств ρ_β совпадает с W и эти множества не пересекаются, то каждый элемент W принадлежит ровно одному из них. Иными словами, каждому $w \in W$ однозначно сопоставляется такой индекс $\beta(w) \in B$, что $w \in \rho_{\beta(w)}$.

Теперь мы имеем все, что нужно для построения первоначального графа. Построим граф (M, N, T) , где $M = A \cup B$ (считая, что выбранные нами множества индексов A и B дизъюнкты), $N = W \cup F$, $T(w) = (\alpha(w), \beta(w))$, дополнительное множество дуг F будет определено позднее.

Сейчас отметим, что с дугами-работами все достаточно просто: каждая работа — это дуга, начала таких дуг — индексы из A , концы — индексы из B . Но получился граф, в котором начала и концы принадлежат разным множествам (такой граф называется двудольным). Нужны дополнительные дуги, чтобы собрать граф из фрагментов. Ими будут дуги из множества F . Они как бы собирают каждое множество π_α из элементов разбиения $\bar{\Pi}$, и множество F состоит из дуг, соответствующих всем парам (β, α) , где начало дуги $\beta \in B(\alpha)$, конец дуги $\alpha \in A$. Дуги из множества F принято называть *фиктивными работами*.

Пример. Пусть множество работ состоит из 11 элементов $W = \{a, b, c, d, e, f, g, h, i, j, k\}$. Пусть $P_a = P_b = P_c = \emptyset$, $P_d = \{b\}$, $P_e = P_f = \{a\}$, $P_g = P_k = \{c\}$, $P_h = \{b, d, e, k\}$, $P_j = \{f, g, h\}$, $P_i = \{c, g\}$. Построим транзитивное замыкание отношения P :

	a	b	c	d	e	f	g	h	i	j	k	
a												π_0
b												π_0
c												π_0
d			b									π_1
e				a								π_2
f					a							π_2
g						c						π_3
h		a	b	c	d	e					k	π_4
i							c					π_5
j		a	b	c	d	e	f	g	h		k	π_6
k							c					π_3

Последним столбцом в этой таблице записаны номера множеств, среди которых оказалось семь различных. Произведение соответствующих разбиений дает нам множества $\rho_1 = \{a\}$, $\rho_2 = \{b\}$, $\rho_3 = \{c\}$, $\rho_4 = \{g\}$, $\rho_5 = \{d, e, k\}$, $\rho_6 = \{f, h\}$, $\rho_7 = \{i, j\}$.

Теперь осталось построить граф. Множество F состоит из дуг, ведущих из ρ_1 в π_2 , ρ_2 в π_1 , из ρ_3 в π_3 , из $\rho_1, \rho_2, \rho_3, \rho_5$ в π_4 , из ρ_3, ρ_4 в π_5 , из $\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6$ в π_6 . Получаем граф, изображенный на рис. 8.17 (пунктирные пути — фиктивные). В этом графе есть начальная вершина, из которой исходит дуги (начинаются работы), не имеющие предшественников, и конечная вершина — заключительное событие, в нем заканчиваются дуги, за которыми не должны следовать другие дуги. Обозначим начальную вершину графа через i_0 , а заключительную через i_1 .

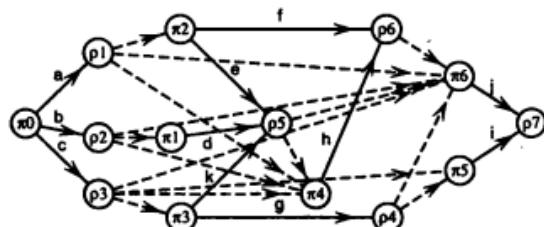


Рис. 8.17. Первоначальный сетевой график

Получившийся граф может быть существенно упрощен, и эти упрощения нужно сделать, так как предложенная здесь универсальная и логически правильная конструкция неудобна для использования.

Имеется несколько типов преобразований, которые упрощают граф. Прежде всего, если начало и конец какой-либо фиктивной дуги соединены “обходным” путем, не содержащим этой дуги, то такая дуга может быть удалена из графа. Во-вторых, если какая-либо фиктивная дуга — единственная дуга, выходящая из вершины или входящая в вершину, то дуга может быть удалена из графа склеиванием ее начала и конца в одну вершину. Эти операции можно применять в любом порядке до полного насыщения.

Пример (продолжение). Дуга (ρ_1, ρ_6) может быть удалена, так как существует обходный путь $\rho_1 \rightarrow \rho_2 \rightarrow \rho_6 \rightarrow \rho_6$. Аналогично удаляются дуги (ρ_2, ρ_6) , (ρ_3, ρ_6) , (ρ_4, ρ_6) и (ρ_5, ρ_6) , а также (ρ_3, ρ_5) , (ρ_1, ρ_4) , (ρ_2, ρ_4) и (ρ_3, ρ_4) . А теперь можно склеить вершины ρ_1 и ρ_2 , ρ_2 и ρ_1 , ρ_3 и ρ_4 , ρ_4 и ρ_5 , ρ_5 и ρ_6 , ρ_6 и ρ_6 . Таким образом, окончательно получаем граф, представленный на рис. 8.18. Отметим, что этот график изображает ту же зависимость работ, что и график на рис. 8.16, а.

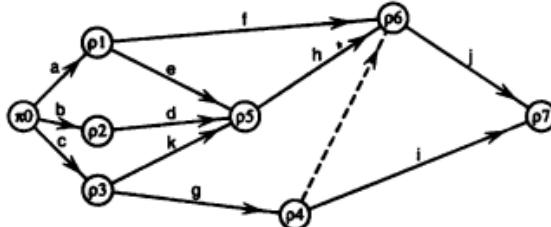


Рис. 8.18. Окончательный вид сетевого графика

Сам по себе этот график, называемый *сетевым графиком* проекта, очень важен для общего наглядного представления о предстоящей работе. Но он может быть использован и для получения важных количественных характеристик, прежде всего, связанных со временем выполнения.

Представим себе, что каждой работе $w \in W$ сопоставляется время ее исполнения $t_w > 0$. Для единства фиктивным работам также сопоставляется время их выполнения, равное 0. Работа не может начаться прежде, чем будут выполнены все предшествующие ей работы, поэтому продолжительность выполнения всего проекта не может быть меньше, чем сумма продолжительностей работ, входящих в любой путь от начального до заключительного события. Сумму продолжительностей работ, входящих в путь, будем называть дальше *длиной пути*. Путь наибольшей длины называется *критическим путем*.

Критические пути и пути, близкие к ним по длине, наиболее существенны для соблюдения сроков выполнения проекта, эти пути находятся и при первоначальном анализе сетевого графика, и в процессе выполнения проекта, когда реальность уже внесла свои изменения в структуру графика и в оценку времен выполнения работ.

Критический путь вычисляется почти по методу Дейкстры, нужны совсем небольшие поправки, связанные с тем, что ищется путь максимальной, а не минимальной длины. Как и в методе Дейкстры, каждой вершине графа $i \in M$ (каждому событию) сопоставляется число $v[i]$, называемое в этой задаче *ранним наступлением* события i и равное наибольшей длине пути от начального события до i .

Ранние начала событий могут быть определены из соотношений

$$v[i] = \max\{t_j + v[\text{beg } j] \mid \text{end } j = i\}$$

и условия $v[i_0] = 0$ для начального события i_0 . Время раннего наступления заключительного события называется *критическим временем*, проект не может быть выполнен быстрее, конечно, без изменения исходных данных.

Алгоритм нахождения ранних наступлений событий

Состояние вычислительного процесса. Разбиение множества вершин $M = M_0 \cup M_1 \cup M_2$. Аналогично алгоритму Дейкстры, это множества вершин с вычисленными, вычисляемыми и еще не затронутыми вычислением ранними временами наступления. Каждой вершине $i \in M$ сопоставлено целое неотрицательное число $r[i]$ — количество еще не просмотренных дуг с концом i и число $v[i]$ — раннее наступление события i .

Начальное состояние.

$$\begin{aligned}M_0 &= \emptyset, & M_1 &= \{i_0\}, & M_2 &= M \setminus M_1, \\v[i] &= 0, & r[i] &= |\{j \mid \text{end } j = i\}|, & i &\in M.\end{aligned}$$

Стандартный шаг.

- Если множество M_1 пусто, завершить вычисления.
- Если же оно не пусто, то выбрать в нем любую вершину, назовем ее i_1 . Перевести эту вершину в M_0 .
- Для каждой дуги j , выходящей из i_1 , и ее конца $i_2 = \text{end } j$ снизить $v[i_2]$ и $v' = r[j] + v[i_1]$ и увеличить $v[i_2]$ до v' , если v' больше. Уменьшить $r[i_2]$ на 1 и при $r[i_2] = 0$ перевести i_2 из M_2 в M_1 .

Аналогично ранним наступлениям событий могут вычисляться поздние наступления. Взяв в качестве исходных моментов для каждого события критическое время проекта и считая это время уже вычисленным моментом для заключительного события, мы вычисляем заново поздние наступления для всех остальных событий по формуле

$$V[i] = \min\{-t_j + V[\text{end } j] \mid \text{beg } j = i\}.$$

Нахождение поздних моментов наступления событий вполне аналогично нахождению ранних началь.

Вычислим теперь для каждой работы j величину

$$d[j] = V[\text{end } j] - t[j] - v[\text{beg } j].$$

Эта величина называется *резервом времени работы* j . Резерв времени каждой работы неотрицателен, так как представляет собой разность длины критического пути и длины какого-то конкретного пути от начальной вершины до конечной. Работы (дуги), имеющие нулевой резерв времени, называются *критическими*.

Лемма. Каждая критическая дуга лежит на каком-нибудь критическом пути.

Доказательство очевидно.

Модель сетевого графика имеет большое значение как хороший организующий инструмент, выделение критических путей позволяет организаторам выделить область наибольшего внимания, хотя, конечно, она не исчерпывается критическими путями по многим причинам.

Прежде всего, сами продолжительности работ удается указать лишь примерно; по мере выполнения работ, когда выясняются их истинные продолжительности, сетевой график для оставшейся части проекта должен пересчитываться. Полезно несколько расширять область контроля, вводя в нее так называемые *субкритические пути*, такие, у которых резерв времени мал и небольшое изменение продолжительности может вывести их на критический путь.

Редко какие практические проекты анализируются полностью в технике критических путей *).

Дело в том, что отдельные работы, входящие в проект, обычно связаны не только технологическими предшествованиями, но и совместно используемыми и обычно дефицитными ресурсами, в частности рабочей силой и оборудованием. Правильная модель должна составлять "расписание" выполнения проекта, в этом расписании требуется согласование выполнения работ по каждому из ресурсов. Могут накладываться дополнительные ограничения на выполнение работ, в частности выполнение работ в одних случаях может прерываться, а в других нет. Используемые ресурсы могут быть *складируемыми*, как деньги и материалы, и *нескладируемыми*, как рабочая сила, время используемого оборудования или электроэнергия.

Так как задача составления расписаний хорошо всем известна по школьному быту, то на каждом курсе находится один или несколько студентов, обуреваемых идеей алгоритмизации составления расписания занятий. Поэтому я должен предупредить, что это очень сложная задача. Она сложна еще и потому, что если ресурсы не дефицитны, то ручное составление расписания не составляет большого труда, а когда становится трудно, то у диспетчера-человека есть в запасе волшебное средство, которого пока нет у машины, — попросить кого-нибудь из участников системы ослабить свои ограничения.

Существует особая наука *теория расписаний*, изучающая и систематизирующая задачи такого рода, а также различные приближенные методы их решения (на точные методы надежды почти нет). Особое место среди них занимают *эвристические* методы, в которых делаются попытки описать логику и технику действий диспетчера *).

Алгоритм поиска наименее загруженных связей			Конфигурация
a	—	—	$\Pi_0 := W, \emptyset$
b	a	a	$\Pi_1 := \{a\} \cup \{b, c, d, e, f\}$
c	b, e	a, b, d, e	$\Pi_3 := \{a, b, d, e\} \cup \{c, f\}$
d	—	—	Π_0
e	d	d	$\Pi_2 := \{d\} \cup \{a, b, c, e, f\}$
f	d	d	Π_2

Приоритетное разбиение:

$$\Pi_1 \times \Pi_2 = \{a\} \cup \{d\} \cup \{b, c, e, f\}$$

$$\Pi_1 \times \Pi_2 \times \Pi_3 = \{a\} \cup \{d\} \cup \{b, e\} \cup \{c, f\}$$

$$p_1 = \{a\}$$

$$\Pi_1 = \{a\} = p_1$$

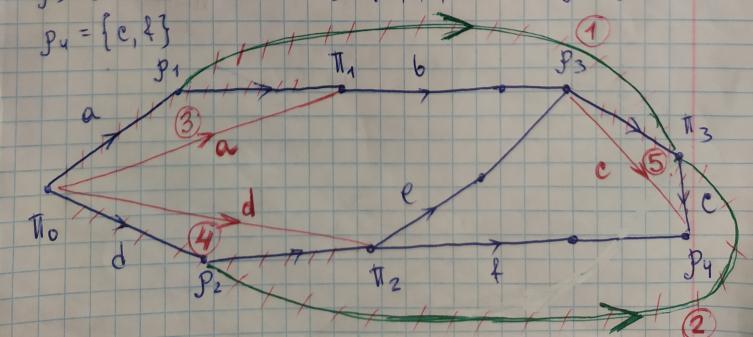
$$p_2 = \{d\}$$

$$\Pi_2 = \{d\} = p_2$$

$$p_3 = \{b, e\}$$

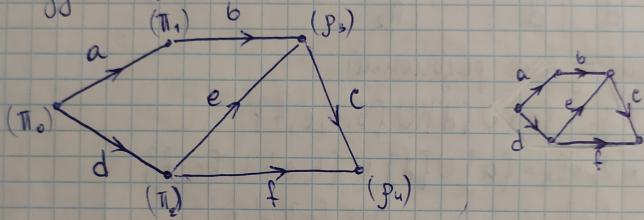
$$\Pi_3 = \{a, b, d, e\} = p_1 \cup p_2 \cup p_3$$

$$p_4 = \{c, f\}$$



- ① удаляем $p_1 \rightarrow \Pi_3$ (т.к. есть путь $p_1 \xrightarrow{b} p_3 \rightarrow \Pi_3$)
- ② удаляем $p_2 \rightarrow \Pi_3$ (т.к. есть путь $p_2 \xrightarrow{f} \Pi_2 \rightarrow p_4 \rightarrow \Pi_3$)
- ③ склеиваем Π_0 и Π_1 (т.к. в Π_1 только $p_1 \rightarrow \Pi_1$)
- ④ склеиваем Π_0 и Π_2 (т.к. в Π_2 только $p_2 \rightarrow \Pi_2$)
- ⑤ склеиваем p_3 и p_4 (т.к. в p_4 только $\Pi_3 \rightarrow p_4$)

Результат:



~~+
нр~~

49 Задача о максимальном паросочетании в графе. Алгоритм построения

Опр

Граф $< M, N >$ называется простым или двудольным, если множество его вершин разбито на два множества M_b и M_c и все начала дуг принадлежат M_b , а концы M_c

Опр

Набор ребер $J \subset N$ называется паросочетанием, если $\forall j_1, j_2 \in J, j_1 \neq j_2$ начала и концы этих дуг различны

Опр

Максимальное паросочетание - максимальное по числу ребер паросочетание

Опр

Цепью длины k называется некоторый простой путь, содержащий k ребер

Опр

Чередующей цепью (относительно некоторого паросочетания) называется цепь, в которой ребра поочередно принадлежат/не принадлежат паросочетанию.

Опр

Увеличивающей цепью называется чередующаяся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию

Теорема (Бержа)

Паросочетание является максимальным $\Leftrightarrow \nexists$ увеличивающих относительно него цепей

Опр

Насыщенная вершина - принадлежащая паросочетанию

Опр (Алгоритм Куна)

Из каждой ненасыщенной вершины графа (из одной доли) будем искать увеличивающую цепь. Для этого применим DFS (поиск в глубину). Мы выбрали ненасыщенную вершину v , рассмотрим все вершины, инцидентные с ней. Назовем текущую вершину to , если она ненасыщенная, то мы нашли увеличивающую цепь, иначе запустим поиск от вершины

$p((v, to), (to, p))$ пробуем найти увеличивающую цепь из нее.

Если из вершины p мы нашли увеличивающую цепь, то "прочередуем" ребра. Уберем ребро (p, to) и добавим (v, to)

После того, как все вершины будут просмотрены, текущее паросочетание будет максимальным.

В этом параграфе мы увидим, как терминология теории графов может работать при изучении комбинаторных свойств связей между элементами двух множеств, скажем A и B . Связи, которые имеются в виду, формально можно описать отношением на $A \times B$. Интересуют нас взаимнооднозначные отображения между подмножествами A и B .

Пример. В качестве удачной иллюстрации назовем всем известную задачу о расстановке на шахматной доске ладей, не находящихся под ударом друг друга. Множества вертикалей и горизонталей выступают в роли A и B , их прямое произведение — это множество полей доски; группа не бьющих друг друга ладей задает взаимнооднозначное отображение между использованными вертикалями и горизонталями. Например, ладьи, расположенные в a5, b7 и e4, задают отображение вертикалей {a, b, e} в горизонтали {4, 5, 7}.

Если рассматривать доски произвольного размера и ограничивать множество полей доски, на которые разрешается ставить ладьи, то это будет уже полное описание интересующей нас сейчас задачи.

Граф $\langle M, N \rangle$ называется *простым*, или *двудольным*, если множество его вершин разбито на два множества M_b и M_e и все начала дуг принадлежат M_b , а все концы — M_e .

Такие графы удобно рассматривать при описании соответствий между элементами двух множеств. Одна из наиболее характерно возникающих в этой связи является задача построения полных или частичных взаимнооднозначных соответствий между элементами этих множеств, т.е. составление допустимых пар элементов (i, k) , $i \in M_b$, $k \in M_e$, с неповторяющимися i и k .

Набор дуг $J \subset N$ называется *паросочетанием*^{*)}, если для любых $j_1, j_2 \in J$, $j_1 \neq j_2$, начала и концы этих дуг различны: $\text{beg } j_1 \neq \text{beg } j_2$, $\text{end } j_1 \neq \text{end } j_2$.

Посмотрим, как можно построить паросочетание, максимальное по числу входящих в него дуг.

Сначала будет предложен алгоритм, который строит (с постепенным увеличением размера) некоторое паросочетание, а потом окажется, что оно действительно максимально. Мы заранее отметим это свойство алгоритма в его названии.

Вместе с паросочетанием будет строиться некое специальное множество вершин. Скажем, что множество вершин M' контролирует дугу j , если начало или конец этой дуги принадлежит M' . Нас интересует *контрольное множество*, в котором для любой дуги j найдется контролирующая ее вершина^{**)}. Если размер паросочетания мы стремимся максимизировать, то контрольное множество естественно искать наименьшего размера.

Оказывается, размер наибольшего паросочетания и наименьшего контрольного множества совпадают. Это дает нам пример замечательного явления — *двойственности экстремальных задач*, о которой будет немного подробнее сказано в следующей главе.

В алгоритме вместе с паросочетанием будет строиться некоторое множество, составленное из начал и концов дуг, входящих в текущее паросочетание (по одной вершине от каждой дуги), так что размеры паросочетания и множества совпадают, и все дуги паросочетания контролируются. Назовем это множество *тестовым*. Работа алгоритма закончится, когда тестовое множество станет контрольным, т.е. будет контролировать все дуги из N .

Нам будет удобно обозначать через $\text{beg } K$ и $\text{end } K$ множества, соответственно, начал и концов любого множества дуг K :

$$\text{beg } K = \{\text{beg } j \mid j \in K\}, \quad \text{end } K = \{\text{end } j \mid j \in K\}.$$

Алгоритм построения максимального паросочетания

Состояние вычислительного процесса. На каждом шаге алгоритма имеется текущее паросочетание N' , разбитое (так определено состояние) на два подмножества $N' = N'_b \cup N'_e$. Объединение M' множеств $M'_b = \text{beg } N'_b$ и $M'_e = \text{end } N'_e$ составляет текущее тестовое множество. Каждой дуге $j \in N'_e$ будет сопоставлена другая дуга $\delta(j)$ — ее *дублер*, — не входящая в паросочетание и имеющая тот же конец, что и j ($\text{end } \delta(j) = \text{end } j$). Будет удобно приписать каждой дуге из N'_e положительный целочисленный *ранг* $\rho(j)$. Откуда же эта информация берется? Немного терпения! Каждая дуга оснащается ею при включении ее в N'_e , вначале же это множество пусто.

Начальное состояние. Первоначально множество N' пусто. Соответственно, пусты и множества N'_b , N'_e и M' .

Стандартный шаг. Если все дуги из N контролируются множеством M' , работа алгоритма завершается.

Пусть нашлась дуга j_0 , не контролируемая множеством M' , так что $\text{beg } j_0 \notin \text{beg } N'_b$, $\text{end } j_0 \notin \text{end } N'_e$. Возможны четыре случая:

- 1) $\text{beg } j_0 \notin \text{beg } N'$, $\text{end } j_0 \notin \text{end } N'$.

Мы нашли дугу с началом и концом, не встречающимися в текущем паросочетании. Эту дугу можно добавить к паросочетанию. Положим $N' := N' \cup \{j_0\}$, $N'_b := N' \setminus N'_e$, $N'_e := \emptyset$, $M' := \text{beg } N'$.

- 2) $\text{beg } j_0 \notin \text{beg } N'$, $\text{end } j_0 \in \text{end } N'$.

Так как $\text{end } j_0 \notin \text{end } N'_e$, но $\text{end } j_0 \in \text{end } N'$, то существует такая дуга $j_1 \in N'_b$, что $\text{end } j_0 = \text{end } j_1$. Переведем дугу j_1 из множества N'_b в N'_e , назначив ей дублером дугу j_0 (вот и появился дублер) и приписав им всем ранг 1.

- 3) $\text{beg } j_0 \in \text{beg } N'$, $\text{end } j_0 \in \text{end } N'$.

Существует такая дуга $j_1 \in N'_b$, что $\text{end } j_0 = \text{end } j_1$, но, кроме того, существует такая дуга $j_2 \in N'_e$, что $\text{beg } j_0 = \text{beg } j_2$. Переведем дугу j_1 из N'_b в N'_e , назначив ей дублером дугу j_0 и приписав им всем ранг $\rho(j_2) + 1$. Отметим, что так наращиваются постепенно цепочки из дуг и дублеров, и каждая кончается парой дуг первого ранга.

- 4) $\text{beg } j_0 \in \text{beg } N'$, $\text{end } j_0 \notin \text{end } N'$.

Существует такая дуга $j_2 \in N'_e$, что $\text{beg } j_0 = \text{beg } j_2$. Присоединение j_0 к цепочке, начинающейся с j_2 , дает нам цепочку с нечетным числом дуг. В ней дуг из паросочетания на одну меньше, чем дублеров (считая j_0). При этом начала и концы дублеров — это те же вершины, что у дуг паросочетания, а начало дублера первого ранга и конец дуги j_0 в паросочетании не встречались. Заменим дуги паросочетания их дублерами. Размер множества N' вырастет на 1, и получится новое паросочетание. Переведем все дуги паросочетания из N'_e в N'_b .

Лемма. Описанный алгоритм сходится за конечное число шагов, которое не превосходит $(s+1)(s+2)/2$, где s — размер максимального паросочетания.

Доказательство. Рассмотрим в качестве характеристики состояния вычислительного процесса пару (r, k) , где $r = |N'|$, $k = |N'_e|$. Возможные значения r ограничены сверху размером максимального паросочетания s , а k в любой момент не превосходит r . Вычисления начинаются с пары $(0, 0)$, а дальше на каждом шаге либо r увеличивается на 1 и k обращается в нуль (случаи 1 и 4), либо k увеличивается на 1. Таким образом, всего имеется $(s+1)(s+2)/2$ возможных значений этой характеристики процесса, и вычислительный процесс не может вернуться к уже встречавшемуся значению. Следовательно, он окончится за конечное число шагов.

Но окончиться этот процесс может только в случае, если все дуги графа контролируются. Значит, получающееся в результате работы алгоритма множество контролирует все дуги графа. \square

Теорема. Размер максимального паросочетания в двудольном графе равен минимальному размеру контрольного множества.

Доказательство. Прежде всего, размер любого паросочетания не превосходит размера любого контрольного множества, так как каждая дуга из паросочетания должна контролироваться отдельной вершиной. Далее, согласно лемме, описанный алгоритм строит паросочетание N' и контрольное множество M' одного размера. Любое паросочетание не превосходит по размеру N' , и следовательно, размер N' максимальен. Любое контрольное множество имеет размер не меньше $|M'|$, и следовательно, размер M' минимален. \square

Обозначим через $\Gamma(A)$ множество вершин, в которые по дугам графа можно перейти из множества начал дуг $A \subset M_b$:

$$\Gamma(A) = \{\text{end } j \mid \text{beg } j \in A\}.$$

Назовем *дефицитом* графа максимальную разность размеров множества начал и множества концов:

Следствие 1. Размер максимального паросочетания в графе равен $|M_b| - \delta$.

Доказательство. Можно искать контрольное множество в виде пары $(M_b \setminus A, \Gamma(A))$ (каждая дуга либо кончается в $\Gamma(A)$, либо начинается в дополнении M_b до A). Если так, то размер минимального множества равен

$$\begin{aligned} \min\{|M_b \setminus A| + |\Gamma(A)| \mid A \subset M_b\} &= |M_b| - \max\{|A| - |\Gamma(A)|\} = \\ &= |M_b| - \delta. \end{aligned} \quad \square$$

Следствие 2. Для того чтобы размер максимального паросочетания был равен $|M_b|$, необходимо и достаточно, чтобы для любого $A \subset M_b$ размер $\Gamma(A)$ был не меньше размера A .

Доказательство. Утверждение прямо вытекает из предыдущего следствия. \square

Следствие 3. Пусть $a[K, L]$ — матрица из нулей и единиц, при чем $|K| = |L|$. Для того чтобы существовало такое взаимнооднозначное отображение $\psi : K \rightarrow L$, что $a[k, \psi(k)] = 1$ для всех $k \in K$, необходимо и достаточно, чтобы любая подматрица $a[K', L']$, где $|K'| + |L'| > |K|$, имела ненулевые элементы *).

Доказательство. Рассмотрим двудольный граф $\langle M, N \rangle$, у которого $M_b = K$, $M_e = L$, а дуги соответствуют единицам в матрице $a[K, L]$. Взаимнооднозначное отображение, о котором говорится в теореме, — это просто паросочетание максимального размера; для того чтобы оно существовало, необходимо и достаточно, чтобы минимальный размер контрольного множества был равен $|K|$. Между тем каждой нулевой подматрице $a[K', L']$ соответствует контролирующее множество $(K \setminus K') \cup (L \setminus L')$, размер которого равен $2|K| - (|K'| + |L'|)$. Значит, для существования *полного паросочетания* необходимо и достаточно, чтобы такой подматрицы не нашлось. \square

Теорема (Дилворт^{*}). Минимальное число путей, которыми можно покрыть все вершины графа без контуров $\langle M, N \rangle$, равно размеру максимального множества попарно несравнимых вершин.

Доказательство. В каждом множестве попарно несравнимых вершин каждая такая вершина должна лежать на отдельном пути при покрытии множества вершин путями. Поэтому вершин в любом таком множестве не больше, чем путей в любом покрытии.

Итак, максимум не превосходит минимума, и доказательство будет завершено, если мы предъявим множество и покрытие одного размера. Мы воспользуемся теоремой о максимальном паросочетании.

Граф $\langle M, N \rangle$ задает отношение P на множестве вершин M . Рассмотрим его транзитивное замыкание \bar{P} и по нему построим двудольный граф $\langle \bar{M}, \bar{N} \rangle$, где $\bar{M} = \bar{M}_b \cup \bar{M}_e$. Каждое из этих двух множеств — как бы отдельный экземпляр множества M .

Построим в графе $\langle \bar{M}, \bar{N} \rangle$ максимальное паросочетание \bar{N}' и минимальное контрольное множество $\bar{M}' = \bar{M}'_b \cup \bar{M}'_e$. Пусть размер паросочетания равен s , так что $|\bar{M}'| = |\bar{N}'| = s$. Теперь уже множество \bar{M} нам не будет нужно, мы вспоминаем, что каждая дуга паросочетания начинается и кончается в M . Контрольное множество M' состоит из двух подмножеств, M'_b и M'_e , суммарный размер которых равен s , и для каждой дуги из паросочетания ровно одна из инцидентных ей вершин принадлежит контрольному множеству.

Покажем, что дуги, включаемые в паросочетание \bar{N}' , порождают покрытие исходного графа $(|M| - s)$ путями. Для этого покажем, что граф $\langle M, \bar{N} \rangle$ состоит из $(|M| - s)$ компонент связности. Возьмем граф без дуг $\langle M, \emptyset \rangle$, состоящий из $|M|$ компонент связности, в котором все вершины покрыты $|M|$ путями длины 0. Будем добавлять к нему по одной дуге паросочетания. Каждое такое добавление соединяет два простых пути в один, так как каждая вершина графа не больше одного раза — концом, и уменьшает число компонент связности на 1. После s итераций у нас останется $(|M| - s)$ компонент связности, и каждая из них будет путем. Покрытие графа путями нашлось!

Теперь нам нужно раздобыть множество попарно несравнимых вершин размера $(|M| - s)$. На его роль естественно претендует множество $D = M \setminus (M'_b \cup M'_e)$. Чтобы претензии были законными, нужно, прежде всего, чтобы размер этого множества был таким, как надо. Для этого необходимо, чтобы множества M'_b и M'_e не пересекались. Докажем это.

В самом деле, пусть они имеют общую вершину i_0 . Так как $i_0 \in M'_e$, то в паросочетании имеется дуга j_1 , кончающаяся в i_0 , — ее начало не может принадлежать контрольному множеству. Равным образом, так как $i_0 \in M'_b$, то в паросочетании имеется дуга j_2 , начинающаяся в i_0 , — ее конец не может принадлежать контрольному множеству. По транзитивной замкнутости отношения \bar{P} пара (*beg* j_2 , *end* j_1) содержится в \bar{P} , а в силу сказанного она не может контролироваться. Значит, предположение о непустоте $M'_b \cap M'_e$ неверно.

Покажем теперь, что вершины множества D попарно несравнимы. Это уже просто. Существование пути из, скажем, i_1 в i_2 равносильно тому, что пара (i_1, i_2) содержится в \bar{P} и при этом не контролируется, что невозможно. \square

Теорема Дилвортса имеет важные комбинаторные приложения. Вот одно из них, в котором графы уже даже и не видны.

Следствие. Пусть a_1, \dots, a_n — произвольная последовательность чисел. Максимальная длина возрастающей подпоследовательности, содержащейся в ней, равна минимальному количеству невозрастающих подпоследовательностей, на которые можно разложить последовательность a_1, \dots, a_n .

Доказательство следствия основано на построении подходящего графа (какого? придумайте сами) и использовании теоремы Дилвортса. \square

Алгоритм построения максимального паросочетания

Клик №10 занял
141 чр.



$$= \{a, b, c, d\}$$

$$= \{1, 2, 3, 4\}$$

$$\bar{N} = \emptyset$$

$$\textcircled{1}. \quad p = (a, 1)$$

$$\bar{N} = \{(a, 1)\} \quad a \leftarrow 1$$

$$X(\bar{N}) = \{b, c, d\} \quad Y(\bar{N}) = \{2, 3, 4\}$$

$$\textcircled{2}. \quad p = (c, 1), (\cancel{1}, a), (a, 3)$$

$$\bar{N} = \{(c, 1), (a, 3)\} \quad c \leftarrow 1 \quad a \cancel{\leftarrow} 1$$

$$X(\bar{N}) = \{b, d\} \quad Y(\bar{N}) = \{2, 4\} \quad a \leftarrow 3$$

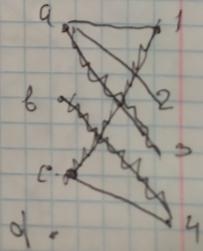
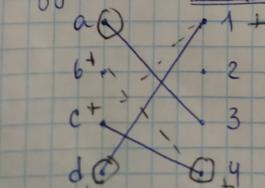
$$\textcircled{3}. \quad p = (d, 1), (\cancel{1}, c), (c, 4)$$

$$\bar{N} = \{(a, 3), (d, 1), (c, 4)\} \quad d \leftarrow 1 \quad c \cancel{\leftarrow} 1$$

$$X(\bar{N}) = \{b\} \quad Y(\bar{N}) = \{2\}$$

Результат:

$$\bar{N} = \{(a, 3), (d, 1), (c, 4)\}$$



50 Теорема Кенига

Опр

Вершинное покрытие графа - множество вершин, такое, что любое ребро графа имеет хотя бы одну конечную вершину из этого множества.

Опр

Вершинное покрытие называется наименьшим, если никакое другое вершинное покрытие не имеет меньшего числа вершин.

Теорема (Кёнига)

В любом двудольном графе число ребер в макс. паросочетании равно числу вершин в наименьшем вершинном покрытии

Теорема алгоритм постр. макс. N - корректен

1. $|N_{\max}| = |\bar{V}_A|$ $|\bar{N}| < |\bar{V}|$

2. $|\bar{V}_A|$ $\exists u, \text{begin}(u) \notin |\bar{V}_A|$
 $\text{end}(u) \notin |\bar{V}_A|$



+	+
+	-
-	+
-	-

невозможен по алгоритму

эту дугу можно добавить
в паросочетание

Теорема Кенига

$|N_{\max}| = |V_{mm}|$

число вершин в максимальном паросочетании равно числу вершин в листе максимального контролирующим множестве

Dok-bo,

$$|\bar{N}| \leq |\bar{V}|$$

17. Теорема Кемпа.

Теорема (Кемп) Рядок максимального пересечения в пятиугольной решетке равен максимальному рядочку континуального шифрования.

Доказательство.

Значит, что рядок в пересечении не больше рядочка в континуальном шифре, так как континуальный рядок в пересечении дополнение контролируется общей единицей.

Лемма.

Арифметика $\mathbb{Z}/N\mathbb{Z}$ строится за конечное число шагов, не больше $(8+1)(8+2)/2$, где 8 - рядок максимального рядочка.

Доказательство леммы.

(r, k) -континуальный рядок, $r = |N|$, $k = |Y(N)|$.
 $r \leq s$, $k \in N$.

Рассмотрим начальное с $(0, 0)$, на каждом шаге надо $r=r+1$ и $k=0$, т.к. $k=k+1$. Там при $r=s$, имеем $(s+1)(s+2)/2$ возможных значений (r, k) и (r, k) не может вернуться к уже перебранным значениям \Rightarrow окончить за конечное число шагов, а т.к. окончание - состояние, при котором все ячейки континуума ≥ 1 , то получившееся шифрованное континуум будет весь ячейка решетки.

Доказательство теоремы.

Рассмотрим $|M| = |N|$.

пересечение континуал. шифрованно.

\forall пересечение $\leq |N| \Rightarrow |N|$ - макс. \forall континуал. шифр. $\geq |M| \Rightarrow$
по рядочку $|M|$ - мин.

$$|N_{opt}| = |V|$$

\leq

51 Алгоритм построения контролирующего множества

15. Алгоритм построения контролирующего множества.

Вершина Дуги - минимальное множество вершин, которое индуцирует все дуги.

Алгоритм построения минимального контролирующего множества вершинной пары графа.

От каждого дуги N берут по вершине как начальную.

* Начальное состояние, соответственно, никакое \emptyset .

1. Некоторое начальное состояние \bar{N}
2. Пусть есть $P: X(\bar{N}) \rightarrow Y(\bar{N})$; он не найдется, так как N - max.
- Сейчас ищут + на все вершины, которые будут индуцированы в течение пути.
3. От каждого дуги из N выбирают $\text{end}(u) \in M_2$ - начальную $\text{beg}(u) \in M_1$ - не начальную

Пример.

1. $\bar{N} = \{(b, 1), (g, 3)\}$
 $X(\bar{N}) = \{a, c, d\}$, $Y(\bar{N}) = \{2, 3, 4\}$.

2. $b \rightarrow 1$
 $c \rightarrow 1$
 $d \rightarrow 1$

3. $(b, 1)$ - выбираем 1
 нач. верн.
 $(g, 3)$ - выбираем 2
 нач. начн.

$\bar{V} = \{g, 1\}$.

16. Задача о максимальном пересечении в графе.
Алгоритм Кохсайда.

Пересечение - набор дуг, не имеющих общих начальных концов.

$G = \langle M, N \rangle$ называется дублетным, если его максимальное вершин

$M = M_1 \cup M_2$, $u \in N : \text{begin}(u) \in M_1$, $\text{end}(u) \in M_2$. (все начала дуг
принадлежат M_1 , а все концы M_2)

Решение задачи.

Найти max пересечение в дублетном графе.

Конгруэнтные множества - множества вершин, с которыми для
каждой дуги, имеющей концы из этого множества, ее вершина (начало или конец
дуги - и это эта вершина)

Пример max пересечения в них конгруэнт. множества совпадают.
(тезис Кенига).

Алгоритм поиска max пересечения

$G = \langle M, N \rangle$, N - текущее пересечение

$X(N) \subset M_1$, $Y(N) \subset M_2$

Нарастание

$N = \emptyset$

Шаг:

1. Дуги из \bar{N} направ. в обратную сторону

2. Найти вершины из $X(\bar{N}) \rightarrow Y(\bar{N})$. Если вершины нет, то пересечение
заканчивается.

3. $P = \{v_1, v_2, v_3, v_4, \dots, v_k\} \subset \bar{N}, v_i \notin \bar{N}, v_i \in N$

$\bar{N} \setminus (V_{\text{нр}} \cup V_{\text{нр}})$ - множество из \bar{N}

4. Направление все дуги из M_1 в M_2 , переходят к 1 пункту.

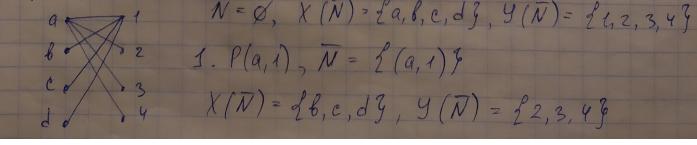
Пример

$\bar{N} = \emptyset, X(\bar{N}) = \{a, b, c, d\}, Y(\bar{N}) = \{1, 2, 3, 4\}$

$a \xrightarrow{\text{нр}} 1, b \xrightarrow{\text{нр}} 2, c \xrightarrow{\text{нр}} 3, d \xrightarrow{\text{нр}} 4$

1. $P(a, 1), \bar{N} = \{(a, 1)\}$

$X(\bar{N}) = \{b, c, d\}, Y(\bar{N}) = \{2, 3, 4\}$



$$2. P = \{(b, 1), (1, a), (a, 3)\} \quad \bar{N} = \{(b, 1), (a, 3)\}$$

Уз $X(\bar{N})$ нет языка $\mathcal{L}(\bar{N})$ -ан. завершит.

Д-бо языков алгоритмов:

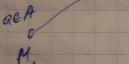
1. Старт. Г-к. компонт вор уб. с \bar{N} на 1.
2. Проверка конца языка - бесконечное языки.



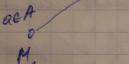
$$|P_{out}'| > |P_A| \quad \text{Проверка языка}$$



Если $|P_{out}'| \leq |P_A|$, то язык конечен.



$$V(P_{out}') \setminus V(P_A)$$



Первый вершину, которая не направляется
направо, и будет именем языка.

Первый $A = V(P_{opt}') \setminus V(P_A)$ - из них будем
выбирать первый язык, (из них первое слово)

Первый $a \in A$ первый язык из них, но определи P_{opt}' и P_A)
Первое слово удаляется в любом месте, а дальше длина не
меньше.

Но язык не может быт пустым, запечатлен в M_1 .
(если может, то $|P_{opt}'| > |P_A|$ и все пусто)

Начало, если концов композиции
и ненулевому началу - то запечатлено опять.

$$B = V(P_A) \setminus V(P_{opt}) \subset M_1$$

52 Задача о назначениях. Венгерский метод

первое что нашел

Дана неотрицательная матрица размера $n \times n$, где элемент в i -й строке и j -м столбце соответствует стоимости выполнения j -го вида работ i -м работником. Нужно найти такое соответствие работ работникам, чтобы расходы на оплату труда были наименьшими. Если цель состоит в нахождении назначения с наибольшей стоимостью, то решение сводится к решению только что сформулированной задачи путём замены каждой стоимости C на разность между максимальной стоимостью и C .

Алгоритм основан на двух идеях:

если из всех элементов некой строки или столбца вычесть одно и то же число $u, y, uy, ;, ., (), , , ,$.

Алгоритм проще описать, если сформулировать задачу, используя [[двудольный граф]]. Дан [[полный двудольный граф]] " $G = (S, T; E)$ " с " n " вершинами, соответствующими работникам ("S"), и " n " вершинами, соответствующими видам работ ("T"); стоимость каждого ребра " $c(i, j)$ " неотрицательна. Требуется найти [[совершенное паросочетание|совершенное]], или [[совершенное паросочетание|полное]] [[паросочетание]] с наименьшей стоимостью.

Будем называть функцию $y: (S \cup T) \rightarrow \mathbb{R}$, $y(i) + y(j) \leq c(i, j)$, $y(v) = \sum_{v \in S \cup T} y(v)$, $y(i) + y(j) = c(i, j)$, \mathbf{G}_y , $y()$.

== Алгоритм в терминах двудольных графов ==

Алгоритм хранит в памяти потенциал y и ориентацию (задание направления) каждого жёсткого ребра, обладающую тем свойством, что рёбра, направленные от T к S образуют паросочетание, которое мы обозначим M . Ориентированный граф, состоящий из жёстких рёбер с заданной ориентацией, мы обозначаем \vec{G}_y , $*(*M)$.

Изначально y везде равно 0, и все рёбра направлены от S к T (таким образом, M пусто). На каждом шаге или модифицируется y так, что увеличивается множество вершин Z , определённое в следующем абзаце, или изменяется ориентация, чтобы получить паросочетание с большим числом рёбер; при этом всегда остаётся верным, что все рёбра из M являются жёсткими.

Процесс заканчивается, если M — совершенное паросочетание.

Пусть на каждом шаге $R_S \subseteq S < /math > < math > R_T \subseteq T < /math >$, $M < /math > (< math > R_S < /math > ^- < math > S < /math >, , < math > R_T < /math > ^- < math > T < /math >). < math > Z < /math >, < math > R_S < /math > < math > \vec{G}_y < /math > ([[]]).$

Если $R_T \cap Z < /math >, , < math > \vec{G}_y < /math > < math > R_S < /math > < math > R_T < /math > ., .1. < blockquote > : * < math > S < /math > < math > T < /math > < math > T < /math > < math > S < /math > .. * < math > S < /math >, ^- < math > T < /math > ., < math > S < /math > < math > T < /math > . < math > \vec{G}_y < /math >, , , < math > M < /math >, < math > M < /math >, , . < /blockquote >$

Если $R_T \cap Z < /math >, < math > \Delta := \min\{c(i, j) - y(i) - y(j) : i \in Z \cap S, j \in T \setminus Z\} < /math >. < math > \Delta < /math >, < math > Z \cap S < /math > < math > T \setminus Z < /math > . < blockquote >, (i, j). < math > i \in Z < /math >, < math > j \notin Z < /math >, < math > i < /math > < math > R_S < /math > < math > \vec{G}_y < /math >, < math > j < /math > ., < math > \vec{G}_y < /math > (i, j)., < math > (i, j) \in M < /math >, < math > i \notin R_S < /math > . < math > i < /math > < math > R_S < /math > < math > \vec{G}_y < /math >, < math > i < /math > - , < math > R_S < /math > ..(k, i). < math > k < /math > < math > R_S < /math > < math > \vec{G}_y < /math >, < math > j < /math > , < math > k \neq j < /math > . < math > (k, i) \in \vec{G}_y < /math >, < math > (i, k) \in M < /math > ., < math > i < /math > < math > M < /math > : < math > (i, j) < /math > < math > (i, k) < /math >, , < math > M < /math > ^- .. < /blockquote >$

Увеличим y на $\Delta < /math > < math > Z \cap S < /math > < math > y < /math > < math > \Delta < /math >, < math > Z \cap T < /math > . < math > y < /math > . < blockquote >, (i, j), < math > i \in S < /math >, < math > j \in T < /math > : * < math > i \notin Z < /math >, < math > j \notin Z < /math >, c(i, j) - y(i) - y(j), y(i), y(j) * < math > i \in Z < /math >, < math > j \in Z < /math > , c(i, j) - y(i) - y(j), y(i) < math > \Delta < /math >, y(j) * < math > i \notin Z < /math >, < math > j \in Z < /math >, c(i, j) - y(i) - y(j) < math > \Delta < /math >, , * < math > i \in Z < /math >, < math > j \notin Z < /math >, c(i, j) - y(i) - y(j) < math > \Delta < /math >, , < math > \Delta < /math > ^- . < /blockquote >$

Граф \vec{G}_y , M .
 $\vec{G}_y(i, j), i \in S, j \in T, c(i, j) - y(i) - y(j)$,
 $i \notin Z, j \in Z, (j, i) \in \vec{G}_y$,
 $R_S, (j, i) \in \vec{G}_y, M$.

Ориентируем новые рёбра от S к T .
По определению Δ , Z , R_S , $(Z, V \in Z, R_S, V - R_S, Z, c(i, j) - y(i) - y(j), V - \min\{c(i, j) - y(i) - y(j) : i \in Z \cap S, j \in T \setminus Z\}, c(i, j) - y(i) - y(j), ST, ij)$.

Повторяем эти шаги до тех пор, пока M не станет совершенным паросочетанием; в этом случае оно даёт назначение с наименьшей стоимостью. Время выполнения этой версии алгоритма равно $O(n^4)$:
 M , n (Z), $O(n^2)$.

53 Задача коммивояжера. Метод ветвей и границ

Задача (Задача коммивояжера)

Дали n городов (вершины графа), между городами есть какие-то дороги, по каждой можно проехать за какое-то фикс. время (вес ребра). Необходимо найти самый дешевый маршрут, проходящий через все города только один раз (тут возможны варианты) с последующим возвращением в исходный город.

Очевидно наивное решение за $O(n!)$, что неприменимо на практике

Опр (метод ветвей и границ) Метод для решения различных задач оптимизации. В отличии от полного перебора отсеивает подмножества допустимых решений.

$f(x)$ - ф-я, которую мы хотим минимизировать

D - мн-во допустимых решений

Ветвление - разбиение мн-ва допустимых решений на подмн-ва

Ветвление можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево поиска или дерево ветвей и границ.

Процедура нахождения оценки (границы) заключается в поиске верхних и нижних границ для решения задачи на подобласти.

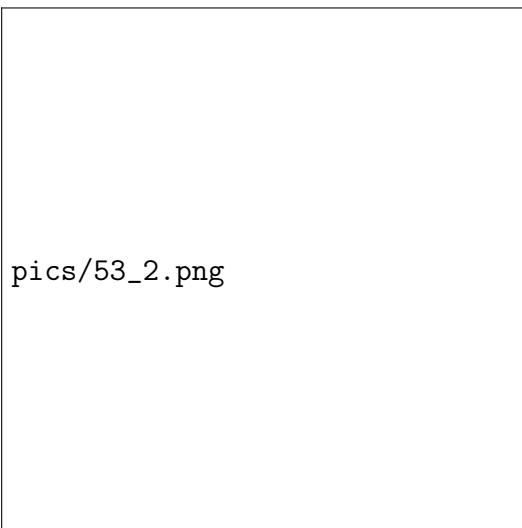
В основе метода лежит след. идея, если нижняя граница на подобласти A больше, чем верхняя граница на ранее просмотренной B , то A может быть исключена из дальнейшего рассмотрения. Обычно минимальную из полученных верхних оценок записывают в глобальную переменную M

Задача (решение)

Зафиксируем какую-нибудь вершину, очевидно, что выбор не важен
Мы можем начать строить дерево путей. Будем поддерживать абсолютный минимум в переменной M . Если мы дошли до листа, то обновляем минимум. Дальше, делая обход, мы можем сделать оценку, обозначим ее S . Допустим, мы прошли какую-то часть пути (часть ветви), обозначим сумму весов на ней за W , оставшийся график назовем G' К оценке добавим самое короткое ребро (r) из вершины, на которой мы остановились до G' и вес минимального остовного дерева на G' . Таким образом, мы оценили снизу решение задачи на этой ветви.

$$S = W + \omega(r) + \omega(MST(G'))$$

Если $S > M$, то можно дальше не спускаться вниз и не рассматривать этот путь дальше, он гарантированно не будет оптимальнее.



В качестве нижней оценки можно использовать и более хитрые функции. А вообще достоинством MST является быстрое получение ответа, хотя оценка и немножко грубовата.

19. Задача коммивояжера. Маршрут вейвей и гамильтон.

Постановка задачи.

Торговец должен обойти все города, находясь в каждом из них не более раз и вернувшись в тот город, с которого начал. В какой последовательности он должен обходить города, чтобы длина суммарного пути была наименьшей?

М-циклическое дерево

с $[M, M]$ - начальной расстановкой (не обязательно единичной)

Городы $i = i_0, i_1, \dots, i_m = i_0$ - последовательность города деревьев,

из $M_i, k \in 1..m, m = 1/41$, который изображает не единичную матр.

$$C(i) = \sum_{k \in 1..m} [i_{k-1}, i_k] \rightarrow \min. \quad \text{Алгоритм}$$

Всего таких деревьев $(m-1)!$

Алгоритм.

1. Сумма до нуля

2. Определение начального

3. Рассматриваем второе минимальное и ~~все остальные~~ следующее значение d_{ij} , у которого максимум суммы второго минимальных.

4. если все d_{ij} на (i, j) значение не ∞ .

5. Следующий d_{ij} все отрицательные d_{ij} и ~~все остальные~~ оставить.

Пример:

①	$\begin{array}{cccccc} \infty & 95 & 85 & 45 & 105 & 45 \\ 85 & \infty & 45 & 55 & 75 & 45 \\ 85 & 85 & \infty & 65 & 25 & 25 \\ 15 & 45 & 25 & \infty & 55 & 15 \\ 25 & 45 & 55 & 25 & \infty & 25 \end{array}$
	$\begin{array}{cccccc} \infty & 0 & 0 & 0 & 0 & 0 \\ 0 & \infty & 0 & 10 & 20 & 0 \\ 0 & 0 & \infty & 0 & 0 & 0 \\ 0 & 10 & 0 & \infty & 0 & 0 \\ 0 & 0 & 0 & 0 & \infty & 0 \\ 0 & 20 & 30 & 0 & 0 & 0 \end{array}$
	$\xrightarrow{d_j}$
	$\begin{array}{cccccc} 0 & 10 & 0 & 0 & 0 & 0 \end{array}$

$$\textcircled{2} \quad M_0 = d_i + d_j = 165$$

③	$\begin{array}{ c c c c c } \hline & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \infty & 40 & 90 & 0 & 60 \\ 2 & 20 & \infty & 0 & 10 & 20 \\ 3 & 30 & 0 & \infty & 40 & 0 \\ 4 & 0 & 50 & 10 & \infty & 40 \\ 5 & 0 & 10 & 30 & 0 & \infty \\ \hline \end{array}$	d_i $\begin{array}{c} (\text{старт}) \\ \text{и конец} \end{array}$	$\begin{array}{ c c c c c } \hline & 0 & 0 & 0 & 0 & 30 \\ \hline 0 & \infty & 0 & 0 & 0 & 0 \\ 0 & 0 & \infty & 0 & 0 & 0 \\ 0 & 0 & 0 & \infty & 0 & 0 \\ 0 & 0 & 0 & 0 & \infty & 0 \\ \hline \end{array}$	d_j	$M_1 = 10 + 10 = 20$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_0 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_2 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_3 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_4 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_5 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_6 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_7 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_8 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_9 = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{10} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{11} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{12} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{13} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{14} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{15} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{16} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{17} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{18} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{19} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{20} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{21} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{22} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{23} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{24} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{25} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{26} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{27} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{28} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{29} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{30} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{31} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{32} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{33} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{34} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{35} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{36} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{37} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{38} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{39} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{40} = 0$
	$\xrightarrow{d_j}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$\xrightarrow{d_i}$	$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$	$M_{41} = 0$

54 Метод динамического программирования. Задача линейного раскроя

Формула: $v[i] = \max_{j \in N_i} \{c_j + v[i - l_j]\}$ $v[0] = 0$, $v[l_0] = \text{старт}$

Параметры: $n=3$ $\begin{array}{|c|ccc|} \hline l & 4 & 7 & 3 \\ \hline c & 5 & 10 & 4 \\ \hline \end{array}$ $l_0=16$

Решение	Длинна	Номер	
0	0	0	$v[6] = \{5 + v[2], 4 + v[3]\}$
1	0	0	$v[8] = \{5 + v[5], 10 + v[2], 4 + v[6]\}$
2	0	0	
3	4	3	$v[9] =$
4	5	1	$v[10] =$
5	5	1	$v[11] =$
6	8	3	$v[12] =$
7	10	2	$v[13] =$
8	10	2	$v[14] =$
9	12	3	$v[15] =$
10	14	3	$v[16] =$
11	15	1	
12	16	3	
13	18	2	
14	20	2	
15	20	1	
16	21	2	

Образ: $0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 16$ 3 груб. группой 3,
1 груб. группой 7

20. Метод генетического программирования. Графы линейного
54
последовательности.

Графа - каскад, который передает из состояния в состояние
и $i \in M$: $f(N_i) : u \in N_i \rightarrow f(u)$

N_i - начальное.

т.е. N_i - каскад состояний имеет свое состояние и
последующий каскад - каскад состояний и

следующий.

$\langle M, V, N_i \rangle$ - означает график (граф переходов)

где, для всех $t \in T$, из t идет заселение N_i .

$f_i \in M$: $N_i \rightarrow f_i$. т.е. из состояния i каскад имеет

уровня i - последующее состояние

$f_i(t)$ - заселение на конц. T каскадом $t \in T$ то - момент это

бесконечный.

Н.у. о. f запад $\max_{t \in T} f(t)$

$$f(t) = C[u] + \beta[u] \cdot f(tu)$$

$C[u]$ - заселение от которого не u -й засел.

$\beta[u]$ - коэффициент заселения, $\beta[u] \in [0, 1]$

$f(tu)$ - заселение t по tu состоянию без заселения u

$f(v)$ - max заселение по tu состоянию, начиная с i

$$v[i] = \max_{t \in T} f(t) = \max_{u \in N_i} \max_{t \in T} f(t) = \max_{u \in N_i} (C[u] + \beta[u] \cdot \max_{t \in T} f(t))$$

$v[i] = \max(C[u], \beta[u])$

$v[i_0] = 0$

\Rightarrow рекурсивная формула для заселения max по приведенным
переменным от V состояния V графа (при заселении Bernoulli)

Граф линейного каскада

c_i - один засел i - много заселений

t_i - один засел $x_i \in Z$ - мало заселений выше i

Каждое заселение содержит заселение i и заселение t_i .

Решение: $\sum c_i x_i \leq l$ где $c_i \in \{0, 1\}$

ограничение $\sum c_i x_i \leq l$ $\forall i \in N_i$, где $i \in N_i$

$$\sum_{i=1}^n c_i x_i \rightarrow \max \text{ all } - [l : l \in \emptyset] \text{ - ограничение free moment first last заселений}$$

заселение первого заселения, определяемое первым заселением

$$N_i = \{j \in S : h_j \leq i\}$$

заселений

55 Приближенные методы решения дискретных задач.
Жадные алгоритмы

56 Алгоритмы с гарантированной оценкой точности.
Алгоритм Эйлера

57 Жадные алгоритмы. Задача о системе различных представителей

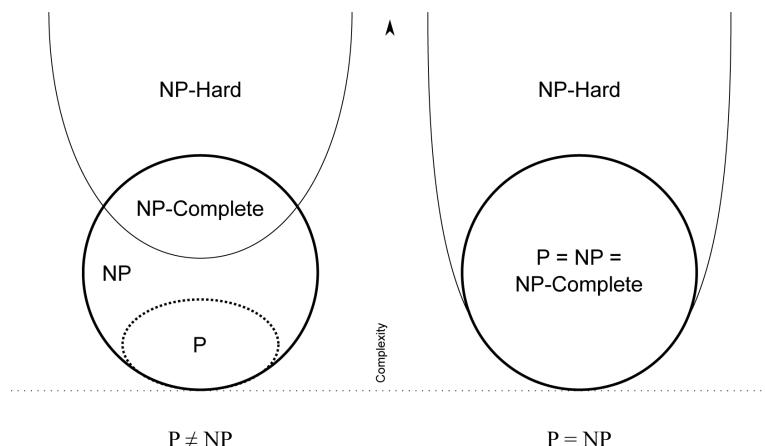
58 Приближенные методы решения дискретных задач

NP - класс всех задач поиска

P - класс всех задач поиска, решение которых может быть найдено за полиноминальное время

$P \neq NP$ - нерешенная задача

Задача поиска называется NP-полной, если к ней сводятся все задачи поиска. В предположении $P \neq NP$ не существует полиноминальных точных алгоритмов для NP-полных задач. Для них можно найти в течении полиноминального времени решение, близкое к оптимальному. Алгоритм, возвращающий такие решения называется приближенным



Пример (приближенные алгоритмы)

1. Жадные алгоритмы
2. Алгоритмы с гарантированной оценкой точности
3. \mathcal{E} -приближенные алгоритмы
4. Локальный поиск
5. Генетические алгоритмы
6. Муравьиные алгоритмы
7. Нейронные сети
8. Гибридные алгоритмы

- Жадный алгоритм на каждом шаге делает локально наилучший выбор в надежде, что итоговое решение будет оптимальным (Алгоритм Дейкстры, задача о рюкзаке не всегда подходит, Алгоритм Краскала)

Жадный алгоритм не всегда дает наилучший результат

- Точность приближенного алгоритма измеряется в том, в какое максимальное число раз может отличаться полученное решение от оптимального :

$$\exists k : \frac{f_a}{f_{opt}} \leq k - \text{гарантированная оценка точности}$$

Алгоритм Кристофида $k = 2$, алг. К. с оценкой $\frac{3}{2}$

- \mathcal{E} -прибл. алг., если $\forall \mathcal{E} > 0$ можно построить алг.:

$$\frac{f_a - f_{opt}}{f_{opt}} < E$$

- Локальный поиск - поиск, который учитывает текущее состояние, а ранее пройденные не учитываются. Основная задача - не нахождение оптимального пути к целевой точки, а оптимизация некоторой целевой функции

- Генетический алгоритм - эвристический алгоритм поиска. Описание:

- (a) Задает целевую функцию для особой популяции
- (b) Создать начальную популяцию *начало цикла*:
 - i. Размножение
 - ii. Мутирование
 - iii. Целевая функция для всех особой
 - iv. Формирование нового поколения
 - v. Если выполняются условия остановки цикла, конец

Когда остановились - нашли решение

- Муравьиный алгоритм (один из алгоритмов решения задачи коммивояжера). В вершинах графа (городах) размещаем муравьев. Начинается их движение, направление определяется вероятностными методами (*маркировка наиболее удачных путей*)

7. Нейронная сеть - математическая модель, система соединенных и взаимод. между собой процессов. Обучается (нахождение коэф. связей между нейронами)
8. Гибридные алгоритмы - сочетают разные подходы

59 Конечные автоматы

Что такое автомат, будем определять постепенно. Пусть заданы:

M — конечное непустое множество, элементы которого называются *состояниями автомата*;

A — конечное непустое множество *внешних воздействий* на автомат;

B — множество *ответов автомата* на внешние воздействия.

Обычно множество A называется *входным алфавитом* автомата, а множество B — *выходным алфавитом*.

Автомат — это процесс, который рассматривается в дискретные моменты времени (такты работы) и в каждый момент времени получает внешние воздействия. В зависимости от воздействия и своего текущего состояния процесс переходит в новое состояние и вырабатывает свой ответ ¹⁾). Нас интересует поведение автомата во времени: его переходы из одного состояния в другое и реакции на воздействия.

Первоначально, в нулевом такте, автомат находится в некотором заданном начальном состоянии $i_0 \in M$. Каждый следующий такт t начинается с того, что в автомат поступает некоторый элемент a , множества A , и в зависимости от этого элемента и текущего состояния процесса i_{t-1} процесс переходит в новое состояние i_t . Попутно вырабатывается действие автомата b_t . Таким образом, в канонических терминах автомат переводит входную строку алфавита A в выходную строку алфавита B .

Правила перехода автомата в новое состояние и выработки им действий определяются в простейшем случае таблицами. Есть более удобные и компактные способы задания правил, но сейчас нам важна не форма зависимости, а набор параметров, определяющий выбор. Итак, пусть правила перехода в новое состояние и выбора действия определяются отображениями $T : M \times A \rightarrow M$ и $D : M \times A \rightarrow B$.

Рассмотрим несколько простых примеров.

Пример 1. Сканирующий автомат. Задается последовательность символов, в которой требуется выделить запись целого числа (в десятичной системе счисления). Эта запись состоит из последовательности цифр, которая окаймлена нецифровыми символами (кроме знаков – и +, которые могут войти в запись числа, и знака . (точка), который использовать запрещено). Для упрощения я выбрал представление числа, в котором запрещено многократное появление знаков – и + и появление точки.

По завершении выделения числа автомат выдает результат и переходит в состояние готовности к следующему сканированию. В случае появления запрещенной последовательности цифр автомат выдает сообщение об ошибке и также переходит в состояние готовности.

Итак, можно выделить следующие состояния автомата:

- S_0 – состояние начальной готовности — нет еще никакой информации о сканируемом числе, если какие-либо символы и поступили, то они не были использованы;
- S_1 – состояние чтения числа — прочтен символ – или +, который определил знак и запретил появление любого символа, кроме цифры;
- S_2 – состояние чтения числа — прочтена хотя бы одна цифра.

Различие состояний S_1 и S_2 — в их реакции на появление нецифрового символа; когда известен только знак числа, рано вырабатывать результат.

Хотя на входе могут появляться любые символы и с точки зрения вычисления числа существенно, например, какая именно цифра появилась для состояния автомата и вырабатываемых действий важны только категории символов — Цифра (Ц), Знак (З), Прочий символ (П).

Теперь можно составить таблицу переходов (левая таблица):

Ц	З	П	
S_0	S_2	S_1	S_0
S_1	S_2	S_0	S_0
S_2	S_2	S_0	S_0

Ц	З	П	
S_0	D_2	D_1	D_0
S_1	D_3	D_4	D_5
S_2	D_3	D_4	D_6

Правая таблица указывает действия, которые должны быть выполнены в той или иной ситуации. Перечислим их в порядке появления:

- D_0 – начальное сканирование — ничего не делать;
- D_1 – прочтен знак числа — установить нулевое значение числа и знаковый множитель, зависящий от прочтенного символа;
- D_2 – прочтена первая цифра — установить знаковый множитель, равный +1, и начальное значение числа по прочтенному цифре;
- D_3 – прочтена очередная цифра — пересчитать значение числа по прочтённой цифре, т. е. умножить на 10 и прибавить цифру;

*D*4 – прочтен знаковый символ, который уже не нужен, — это ошибка;

*D*5 – прочтен посторонний символ, до того как начато чтение числа;

*D*6 – прочтен посторонний символ, завершивший чтение числа, — выдать в качестве результата произведение числа на знаковый множитель.

В качестве упражнения измените автомат таким образом, чтобы в записи числа допускались одиночные пробелы между цифрами и после знака числа.

Пример 2. Печатающий автомат^{*)}. Имеется возрастающая последовательность натуральных чисел, которая должна быть напечатана. В случае, когда больше двух чисел идет подряд, их нужно печатать как диапазон значений, т.е. печатать начальное и конечное значение группы чисел, идущих подряд, и знак тире между ними. Например, ряд чисел

7,11,12,13,14,23,27,28,32,33,34,35,36,43

должен выглядеть так:

7,11–14,23,27,28,32–36,43.

На вход автомата поступает последовательность символов из такого трехбуквенного алфавита:

Начало группы (Н), Продолжение группы (П), Конец данных (К).

В нашем примере последовательность чисел превращается в последовательность букв входного алфавита:

Н,Н,П,П,П,Н,Н,П,Н,П,П,П,Н,К.

Автомат имеет следующие состояния:

*S*0 – начальное состояние;

*S*1 – прочтено начало группы;

*S*2 – группа содержит два числа;

*S*3 – группа содержит больше чем два числа.

Таблицы переходов и действий:

	Н	П	К
<i>S</i> 0	<i>S</i> 1	<i>S</i> 0	<i>S</i> 0
<i>S</i> 1	<i>S</i> 1	<i>S</i> 2	<i>S</i> 0
<i>S</i> 2	<i>S</i> 1	<i>S</i> 3	<i>S</i> 0
<i>S</i> 3	<i>S</i> 1	<i>S</i> 3	<i>S</i> 0

	Н	П	К
<i>S</i> 0	<i>D</i> 1	<i>D</i> 0	<i>D</i> 0
<i>S</i> 1	<i>D</i> 2	<i>D</i> 3	<i>D</i> 4
<i>S</i> 2	<i>D</i> 5	<i>D</i> 3	<i>D</i> 6
<i>S</i> 3	<i>D</i> 7	<i>D</i> 3	<i>D</i> 8

Переходы достаточно ясны. Опишем действия:

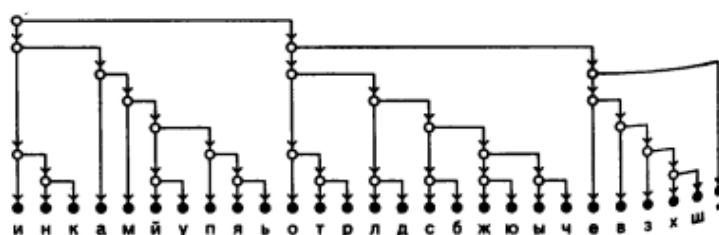
- D0 – ошибка, в нулевом состоянии может появиться лишь начало группы;
- D1 – запомнить поступившее начало группы;
- D2 – напечатать имеющееся начало группы и запятую, затем выполнить действие D1;
- D3 – запомнить поступившее число как конец группы;
- D4 – напечатать имеющееся начало группы и точку;
- D5 – напечатать имеющуюся группу из двух чисел и запятую, затем выполнить действие D1;
- D6 – напечатать имеющуюся группу из двух чисел и точку;
- D7 – напечатать имеющуюся группу и запятую, затем выполнить действие D1;
- D8 – напечатать имеющуюся группу и точку.

Легко видеть, что эти довольно многочисленные действия на самом деле компонуются из простых стандартных действий. Можно сопоставить каждому элементарному действию двоичный разряд — флаг — и набирать необходимые действия в виде битовой последовательности требуемых флагов.

Пример 3. Декодирующий автомат. Хороший пример автомата дает код Хаффмена (см. с. 122): зададимся целью построить алгоритм, декодирующий текст, который записан этим кодом. Рассмотрим пример, использованный при описании алгоритма Хаффмена. Были построены кодовые последовательности

```
- 111    в 010    к 000    е 1100    о 1000    м 0110    и 0010    к 0011  
в 11010    л 10100    А 10101    т 10010    р 10011    з 110110    с 101100  
6 101101    и 011110    в 011100    у 011101    х 1101110    м 1101111    и 1011110  
ч 1011111    к 1011100    н 1011101    я 0111110    ъ 0111111
```

и кодовое дерево, изображенное на рис. 10.1. Знак пробела на рисунке изображен звездочкой. Чтобы получить из этого дерева граф переходов автомата,



нужно каждую дугу, ведущую в тупиковую вершину (в черный кружок), заменить дугой, ведущей в корень дерева (не делайте этого, а только представьте себе). Вся рабочая информация для декодирующего автомата легко представляется следующей таблицей:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	1	2	и	н	а	м	7	я	п	я	11	12	о	т	15	л	17	с	19	ж	ы	22	е	в	з	х
1	10	4	3	к	5	6	8	у	9	ъ	21	14	13	р	16	д	18	б	20	ю	ч	_	23	24	25	ш

где для простоты число обозначает новое состояние автомата, а другой знак — вырабатываемый символ и 0 в качестве нового состояния.

В случае поочередного кодирования элементов из двух алфавитов (как это может произойти, например, в графических файлах, когда изображение раскладывается на монохромные полоски, кодируемые парами “цвет–длина”) автомат составляется из двух декодирующих деревьев и по достижении конечного состояния в одном из деревьев переходит в корень другого дерева.

Пример 4. Автомат для поиска образца в строке. Пусть задан образец — строка $p[1 : m]$ и текст $t[1 : n]$. Построим автомат, состояниями которого будут числа из диапазона $0 : m - 1$. Пребывание автомата в состоянии k означает, что при сканировании текста обнаружено совпадение префикса $p[1 : k]$ образца с концом просмотренной части текста (т. е. с $t[s - k + 1 : s]$, где s — индекс последнего просмотренного символа). При продолжении сканирования текста символ $t[s + 1]$ сравнивается с $p[k + 1]$. Если они совпадают, то при $k + 1 = m$ процесс завершается — образец нашелся, а при меньшем k автомат переходит в состояние $k + 1$. Если же символы не совпали, то автомат переходит в некоторое состояние k' , определяемое наибольшим совпадением конца префикса $p[1 : k + 1]$ с меньшим префиксом. Эти значения $k'(k + 1)$ зависят только от образца p , могут быть вычислены заранее и сведены в таблицу переходов.

Например, при поиске образца *aabbaabaab* в строке над алфавитом $\{a, b\}$ таблица переходов выглядит следующим образом:

Строка	<i>a a b b a a b a a b</i>
Состояние	0 1 2 3 4 5 6 7 8 9
Символ <i>a</i>	1 2 2 1 5 6 2 8 9 2
Символ <i>b</i>	0 0 3 4 0 0 7 4 0 0*

В начале сканирования текста автомат находится в состоянии 0. Символы последовательно проверяются, и автомат, как описано, от каждого символа переходит в новое состояние. Пока автомат не попадет в состояние 0^* , никаких внешних действий не выполняется (такой автомат называется *распознавающим*).

На рис. 10.2 тот же автомат представлен его *графом переходов*: состояниям автомата соответствуют вершины, а переходы из состояния в состояние представлены дугами.

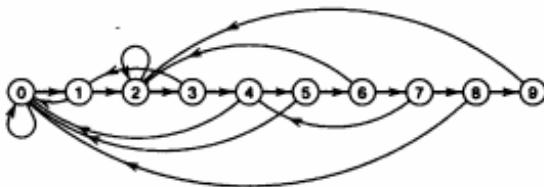


Рис. 10.2. Граф переходов распознавающего автомата.
Жирные нарисованы дуги, повышающие совпадение строк

Упражнение 10.1. Проследите, как происходит поиск *aabbaabaab* при проверке строки $t = abaabbabaabaabaaa$.

Упражнение 10.2. Как нужно изменить определение автомата, чтобы он искал все вхождения строки в текст?

Упражнение 10.3. Как изменится автомат, если текст t может содержать символы, не входящие в образец?

Упражнение 10.4. Попробуйте построить автомат для поиска в тексте строки *abaabaabaaaab*.

Упражнение 10.5. Предложите метод для построения автомата по образцу (отметим, что состояние k' , в которое автомат переходит при несовпадении очередных символов текста и образца, определяется предварительной обработкой для упоминавшегося на с. 103 метода Кнута–Морриса–Пратта).

Пример 5. Автомат для регулярного выражения. Проверка соответствия строки регулярному выражению может также осуществляться автоматом. Назовем строку, удовлетворяющую данному регулярному выражению, регулярной. Состояниями нашего автомата будут всевозможные *регулярные префиксы* — строки, которые могут быть продолжены до регулярных строк. Поступающий в автомат очередной символ приписывается к текущему регулярному преффику, после чего получившаяся строка урезается (слева) до максимального регулярного префикса.

Посмотрите на таблицу автомата для регулярного выражения $\cdot \cdot \cdot (\star a \rho \gamma h y l e | \rho g r o s a l) \cdot \cdot \cdot$. Попробуйте нарисовать граф переходов этого автомата. Автоматы широко используются при разработке трансляторов для алгоритмических языков (по этому поводу см., например, [6]), при описании которых часто применяются рекурсивные описания типа регулярных выражений.

Таблица автомата для регулярного выражения

Состояние	#	<i>s</i>	<i>a</i>	<i>p</i>	<i>r</i>	<i>o</i>	<i>h</i>	<i>y</i>	<i>l</i>	<i>e</i>	*
*	0	1	0	11	0	0	0	0	0	0	0
*s	1	1	2	11	0	0	0	0	0	0	0
*sa	2	1	0	3	0	0	0	0	0	0	0
*sap	3	1	0	11	4	0	0	0	0	0	0
*sapr	4	1	0	11	0	5	0	0	0	0	0
*apro	5	1	0	6	0	0	0	0	0	0	0
*aprop	6	1	0	11	0	0	7	0	0	0	0
*saproph	7	1	0	11	0	0	0	8	0	0	0
*saprophy	8	1	0	11	0	0	0	0	9	0	0
*saprophyl	9	1	0	11	0	0	0	0	0	10	0
*saprophyle	10	19	19	19	19	19	19	19	19	19	19
*p	11	1	0	11	12	0	0	0	0	0	0
*pr	12	1	0	11	0	13	0	0	0	0	0
*pro	13	1	0	14	0	0	0	0	0	0	0
*prop	14	1	0	11	0	15	0	0	0	0	0
*propo	15	16	0	11	0	0	0	0	0	0	0
*propos	16	1	17	11	0	0	0	0	0	0	0
*proposal	17	1	0	11	0	0	0	0	18	0	0
*proposal	18	19	19	19	19	19	19	19	19	19	19
Ok	19	19	19	19	19	19	19	19	19	19	19

60 Числа Фибоначчи. Производящие функции

Опр

Производящая функция — это формальный степенной ряд вида $G(z) = \sum_{n=0}^{\infty} a_n z^n$, порождающий (производящий) последовательность (a_0, a_1, a_2, \dots) .

Свойства*

- Производящая функция суммы (или разности) двух последовательностей равна сумме (или разности) соответствующих производящих функций.
- Произведение производящих функций $A(x) = \sum_{n=0}^{\infty} a_n x^n$ и $B(x) = \sum_{n=0}^{\infty} b_n x^n$ последовательностей $\{a_n\}$ и $\{b_n\}$ является производящей функцией свёртки $c_n = \sum_{k=0}^n a_k b_{n-k}$ этих последовательностей: $A(x)B(x) = \sum_{n=0}^{\infty} c_n x^n$.
- Если $A(x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$ и $B(x) = \sum_{n=0}^{\infty} b_n \frac{x^n}{n!}$ — экспоненциальные производящие функции последовательностей $\{a_n\}$ и $\{b_n\}$, то их произведение $A(x) \cdot B(x) = \sum_{n=0}^{\infty} c_n \frac{x^n}{n!}$ является экспоненциальной производящей функцией последовательности $c_n = \sum_{k=0}^n \binom{n}{k} a_k b_{n-k}$.

Найдем точную формулу для чисел Фибоначчи

$$F(x) = \sum_{k=0}^{\infty} f_k x^k \text{ - производящая функция}$$

$$\begin{aligned} F(x) &= f_0 + f_1 x + \sum_{n=2}^{\infty} f_n x^n = f_0 + f_1 x + \sum_{n=2}^{\infty} f_{n-1} x^n + \sum_{n=2}^{\infty} f_{n-2} x^n = \\ &= f_0 + f_1 x + \underbrace{\sum_{n=1}^{\infty} f_n x^{n+1}}_{x(F(x)-f_0)} + \underbrace{\sum_{n=0}^{\infty} f_n x^{n+2}}_{x^2 F(x)} \end{aligned}$$

$$\Rightarrow F(x) = 1 + x + x^2 F(x) + x(F(x) - 1) \Rightarrow F(x) = \frac{1}{1 - x - x^2}$$

$$(1 - x - x^2) = (1 - \alpha x)(1 - \beta x), \quad \alpha = \frac{1 + \sqrt{5}}{2} \quad \beta = \frac{1 - \sqrt{5}}{2}$$

$$\text{Пусть } F(x) = \frac{1}{1 - x - x^2} = \frac{A}{1 - \alpha x} + \frac{B}{1 - \beta x} \Rightarrow A = \frac{\alpha}{\alpha - \beta} \quad B = \frac{-\beta}{\alpha - \beta}$$

Т.к. для маленьких x :

$$\frac{1}{1 - \gamma x} = \sum_{k=0}^{\infty} \gamma^k x^k$$

$$\Rightarrow F(x) = A \sum_{k=0}^{\infty} \alpha^k x^k + B \sum_{k=0}^{\infty} \beta^k x^k = \sum_{k=0}^{\infty} (A\alpha^k + B\beta^k)x^k$$

$$\Rightarrow f_k = \frac{\alpha^{k+1}}{\alpha - \beta} - \frac{\beta^{k+1}}{\alpha - \beta} = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^{k+1} - \left(\frac{1 - \sqrt{5}}{2} \right)^{k+1} \right)$$

61 Числа Каталана

Задача

Предположим, нужно вычислить сумму $S_0 + S_1 + \dots + S_n$ и можно складывать любые два рядом стоящих числа и ставить результат на их место. Требуется найти число различных последовательностей действий (числа Каталана)

Решение

Обозначим искомое число способов через c_n . Число, что последнее вычисление в любом случае будет сложением сумм для двух последовательных подмножеств вида $0 : k$ и $(k+1) : n$, и схемы с различными k принципиально различны. Но для данного k мы должны иметь c_k вариантов левой части и c_{n-k-1} вариантов правой части, так что

$$c_n = \sum_{k=0}^{n-1} c_k c_{n-k-1}, \quad \text{где } c_0 = 1$$

Введем производящую функцию:

$$C(x) = \sum_{n=0}^{\infty} c_n x^n$$

и рассмотрим:

$$\begin{aligned} C^2(x) &= \sum_{m=0}^{\infty} c_m x^m \cdot \sum_{n=0}^{\infty} c_n x^n = \sum_{m,n=0}^{\infty} c_m c_n x^{m+n} = \sum_{r=0}^{\infty} \sum_{m=0}^r c_m c_{r-m} x^r = \sum_{r=0}^{\infty} c_{r+1} x^r \\ \Rightarrow C(x) &= xC^2(x) + 1 \\ \Rightarrow C(x) &= \frac{1 \pm \sqrt{1 - 4x}}{2x} \end{aligned}$$

По формуле тейлора для функции $f(x) = \sqrt{1 - 4x}$ получим решение

$$\begin{aligned} \frac{d^k}{dx^k} (1 - 4x)^{\frac{1}{2}} &= \frac{1}{2} \left(\frac{1}{2} - 1 \right) \dots \left(\frac{1}{2 - k + 1} \right) (1 - 4x)^{\frac{1}{2} - k} (-4)^k = \\ &= -2^k \cdot 1 \cdot 3 \dots (2k - 3) (1 - 4x)^{\frac{1}{2} - k} = -2(k-1)! C_{2k-2}^{k-1} x^k \end{aligned}$$

Подставляя этот ряд в формулу для $C(x)$, мы должны сначала выбрать знак перед квадратным корнем (знак минус нас устраивает больше) и провести необходимые выкладки. Получаем:

$$C(x) = \sum_{k=1}^{\infty} \frac{1}{k} C_{2k-2}^{k-1} x^{k-1} = \sum_{k=0}^{\infty} \frac{1}{k+1} C_{2k}^k x^k$$