1. ОСНОВНЫЕ ПОНЯТИЯ. СПРАВОЧНЫЙ МАТЕРИАЛ

1.1 Основные понятия

Центральным понятием курса будет трудоемкость алгоритма. Рассмотрим это понятие подробнее.
Пусть на входе некоторого алгоритма имеется информация объема n (например, n – длина массива, размер матрицы СЛАУ, длина перемножаемых чисел и т.п.), а на выходе ответ – результат обработки входной информации.

Определение. *Трудоемкость алгоритма T(n)* – максимально возможное количество действия для решения данной задачи среди всех возможных входов длины n.

Замечание. Иногда под сложностью алгоритма подразумевают не максимальное количество операций, а среднюю трудоемкость. Порядок средней и максимальной трудоемкостей, как правило, одинаков.

Характеристики алгоритма:

T(n) – (от англ. Time) – количество операций, необходимых для обработки информации объема n.
S(n) – (от англ. Space) – объём необходимой памяти для обработки информации.

Алгоритмы решения задачи могут быть написаны на разных языках:

- **машинные:**
  - высокого уровня;
  - низкого уровня;
  - машинные коды;
- **математические:**
  - математические реализации (машины Тьюринга);
  - конечные автоматы;
  - рекурсивные функции;

Все эти языки эквивалентны с точки зрения набора решаемых задач (любая задача, реализованная на одном языке, может быть переписана на другой), но они неэквивалентны с точки зрения трудоемкости. Трудоемкость алгоритма, реализованного на одном языке, может существенно отличаться от того же алгоритма, записанного на другом языке.
Существуют некоторые договорённости о том, какие именно функции считаются эквивалентными.

Договоренности.

Определение. Функция *f(n)* и *g(n)* называются *эквивалентными* и обозначаются, как *f(n) ~g(n)*, если выполняется хотя бы одно из двух условий:

1) $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 1$ либо 2) $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = C \neq 0$

Мы будем пользоваться вторым определением выполнения эквивалентности как наиболее общим.

Пример: $(n^3 - n^2) \sim n^3$ эквивалентны по условиям 1) и 2);
$(n^3 - n^2) \sim 2n^3$ по 1) условию не эквивалентны, но эквивалентны по 2) условию.

Определение. Функции *f(n) пренебрежимо мала* по сравнению с функцией и *g(n)* обозначается *f(n)<<g(n)* или *f(n) = o(g(n))*, если выполняется условие $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

$2^n >> n^2$ (показательная функция растет быстрее, чем степенная).
$n! >> a^n >> n^a >> \log_2 n$, *где n > 1, a > 0.*
Теорема: $n! >> a^n >> n^a$, $(a > 0) >> \log_2 n$
Без доказательства.
Договоренность. В дальнейшем под *log n* будем подразумевать $\log_2 n$.

1.2. Справочный материал. Сравнение скорости роста основных функций

Формула Стирлинга:

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}.$$

Пример. Сравним скорости роста функций $2^{n!}$ и $n^{\sqrt{n}}$
Решение. 1) Прологарифмируем обе функции по основанию 2, получим:

n! и $\sqrt{n}\log n$.

2) Используя формулу Стирлинга: $n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$, получим:

$\left(\frac{n}{e}\right)^n \sqrt{2\pi n}$ и $\sqrt{n} \cdot \log n$

Сократим на $\sqrt{n}$:

$\left(\frac{n}{e}\right)^n \sqrt{2\pi}$ и $\log n$

3) Так как значение некоторых частей выражений много меньше значения других, то их можно не рассматривать:

$\frac{1}{2}\log(2\pi) + (n\log n - n\log e)$ и *log n.*

т.к. *nloge << nlogn.*

Получаем: *nlogn >> logn*

n >> 1

Следовательно, $2^{n!} >> n^{\sqrt{n}}$

Функция $2^{n!}$ растет быстрее, чем $n^{\sqrt{n}}$

Определение. Будем говорить, что *задача решаема за полиномиальное время,* если для неё существует некоторый алгоритм с трудоёмкостью *T(n) ≤ p(n)*, где *p* – некоторый многочлен.

Примерами задач, решаемых за полиномиальное время, являются:

1) сортировки;
2) решение СЛАУ (метод Гаусса).

Чем же хороши задачи, решаемые за полиномиальное время?

Для ответа на этот вопрос рассмотрим две таблицы:

Таблица 1.1. – Время, необходимое для вычисления на машине с быстродействием 10^6 опер/сек

n / ***T(n)***	**10**	**20**	**30**	**50**	**100**
n	*10^{-5} сек*	*$2\cdot10^{-5}$ сек*	*$3\cdot10^{-5}$ сек*	*$5\cdot10^{-5}$ сек*	*10^{-4} сек*
n logn	*$3\cdot10^{-5}$ сек*	*$8\cdot10^{-5}$ сек*	*$15\cdot10^{-5}$ сек*	*$3\cdot10^{-4}$ сек*	*$7\cdot10^{-4}$ сек*
n^2	*10^{-4} сек*	*$4\cdot10^{-5}$ сек*	*$9\cdot10^{-5}$ сек*	*$2\cdot5\cdot10^{-3}$ сек*	*10^{-2} сек*
n^5	*10^{-1} сек*	*3·2 сек*	*10^{-5} сек*	*5.2 мин*	*≈ 3 часа*
2^n	*10^{-3} сек*	*1 сек*	*18 мин*	*36 лет*	*$3\cdot10^{16}$ лет*

Таблица 1.2. – Объем доступной решаемой задачи за одно и то же время (таблица прогресса)

машины / ***T(n)***	**Современные**	**В 10 раз быстрее современных**	**30**
n	*k*	*10k*	*100k*
n logn	*k*	$(10 - \varepsilon)k$, где $(\varepsilon \to 0$ при $k \to \infty)$	$(100 - \varepsilon)k$, где $(\varepsilon \to 0$ при $k \to \infty)$
n^2	*k*	$\sqrt{10} \approx 3.16k$	*10k*
n^3	*k*	$\sqrt{10} \approx 2.2k$	$\sqrt[3]{100} \approx 4.6k$
n^5	*k*	$\sqrt{10} \approx 1.6k$	$\sqrt[5]{100} \approx 2.5k$
Неполиномиальные алгоритмы			
2^n	*k*	*3.3 + k*	*6.6 + k*
n!	*k*	$k + \varepsilon$, где $(\varepsilon \to 0$ при $k \to \infty)$	$k + \varepsilon$, где $(\varepsilon \to 0$ при $k \to \infty)$

Комментарии к таблице прогресса. При повышении производительности машины в несколько раз для алгоритма с полиномиальной производительностью объем доступной решаемой задачи повышается в разы, а у неполиномиальных алгоритмов увеличивается «на сколько-то», то есть НЕ в разы.

Домашнее задание №1
Отсортировать по скорости роста, начиная с наименьшей:
$log^3 n, 2^n, 3^{\sqrt{n}}, 3^{\log n}, n \cdot \log n, \frac{n^2}{log^4 n}, 2^{n!}, (2^n)!, n^{n^{2^n}}, 2^{3^{2^{2^n}}}$.

2 НОВЫЕ БЫСТРЫЕ ВЕРСИИ СТАРЫХ АЛГОРИТМОВ

2.1. Сортировки массивов

Справка. Пусть дан массив $A = (a_1, a_2, \ldots, a_n)$, состоящий из n элементов. Для всех элементов определены отношения <, >, =. Тогда *сортировка* – это процесс, в ходе которого элементы массива переставляются таким образом, что выполняется одно из следующих неравенств:
$a_1 \le a_2 \le \ldots \le a_n$ – массив отсортирован по возрастанию (в прямом порядке) или
$a_1 \ge a_2 \ge \ldots \ge a_n$ – массив отсортирован по убыванию (в обратном порядке).

2.1.1 Метод прямого выбора (SelectSort)

Метод прямого выбора – пример простейшего метода сортировки, обладающего квадратичной трудоёмкостью.

Алгоритм. Просматриваем весь массив, находим минимальный элемент, ставим его на первое место. Потребуется *n-1* сравнений и одна перестановка. При втором проходе просматриваем массив, начиная со второго элемента, находим минимум среди просматриваемой (не отсортированной) части и ставим его на второе место, и т.д. Потребуется $n - 2$ сравнений и одна перестановка. Для полного упорядочивания придется совершить *n-1* проходов не отсортированной части.

Трудоемкость алгоритма: $n + (n-1) + (n-2) + \cdots + 2 = \frac{n(n+1)}{2} - 1 \approx \frac{1}{2}n^2$.

2.1.2 Быстрая сортировка методом двоичных вставок (MergeSort)

Рассмотрим случай, когда количество элементов массива равно степени двойки $n = 2^k$

Алгоритм. 1. Разобьем массив на пары и упорядочим каждую. Получаем серии длины 2.

2. При втором проходе сливаем полученные на первом этапе пары в упорядоченные четвертки. Получаем серии длины 4.
3. При третьем проходе получаем упорядоченные восьмерки, и т.д., пока длина серии не станет равно количеству элементов массива (на *k*-том проходе).

Оценим трудоемкость алгоритма.
Сначала оценим трудоемкость слияния двух упорядоченных массивов из k и l элементов (считаем только сравнения):

k:

2	3	8	10	14

l:

3	5	6	10	11

Минимальная трудоемкость – меньшее из k и l – $\min(k,l)$.
Максимальная трудоемкость: $k + l - 1$ (единицу вычитаем потому, что сравниваем все элементы, а самый последний просто дописываем к упорядоченной группе).
В MergeSort при получении упорядоченных пар потребуется $\frac{n}{2} \cdot 1$ действий.
При получении упорядоченных четверок $\frac{n}{4} \cdot 3$ действий. Здесь $\frac{n}{4}$ есть количество четверок в массиве, а 3 – это максимальная трудоемкость получения упорядоченных четверок (считая только сравнения).
Таким образом суммарная трудоемкость сортировки составит:

$$T(n) = \frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 3 + \frac{n}{8} \cdot 7 + \cdots + 1 \cdot (n-1) \le n + n + \cdots + n = n \cdot k = k \cdot 2^k = n \log_2 n.$$

таким образом, трудоемкость MergeSort составляет $T(n) = n \log_2 n$.

Теорема. *Сортировка на порядок быстрее, чем n $\log_2 n$ невозможна.*

Доказательство. Заметим, что сортировка с помощью некоторого алгоритма эквивалентна блужданию по двоичному дереву, где в узлах этого дерева находятся операторы сравнения. После операции сравнения маршрут двоится. Напомним, что *двоичное дерево* – граф, у которого ветвление в каждой вершине не больше, чем на две ветви:
Тогда *развилками* дерева будут *сравнения* двух элементов массива; а *листьями* – все возможные варианты *перестановки* элементов массива (у массива длины n их будет $n!$).
В этом случае *трудоемкость алгоритма* – высота дерева, другими словами, максимальный путь от вершины до листа дерева.
Дерево с k листьями имеется высоту не меньше, чем $\log_2 k$ (т.к. дерево высоты h имеет не более, чем 2^k листьев). Итак, трудоемкость произвольной сортировки будет не меньше, чем $\log_2(n!)$.

$$T(n) \ge \log_2(n!)$$

Воспользовавшись формулой Стирлинга, имеем:

$$T(n) \ge \log_2(n!) \ge \log_2\left[\left(\frac{n}{e}\right)^n \sqrt{2\pi n}\right] = n\log_2 n - n\log_2 e + \log_2\sqrt{2\pi n} \approx n\log_2 n$$

данные величины пренебрежимо малы по сравнению с $n\log_2 n$,
поэтому их во внимание не принимаем

Теорема доказана.

2.2. Преобразование Фурье (БПФ)

2.2.1. Дискретное преобразование Фурье

Как известно из вычислительной математики, большое значение имеет следующее преобразование массивов: $(f_0, f_1, \dots, f_{N-1}) \leftrightarrow (A_0, A_1, \dots, A_{N-1})$, которое находится по формулам *прямого дискретного преобразования Фурье:*

$$A_k = \frac{1}{N}\sum_{j=0}^{N-1} exp\left\{-2\pi i \frac{kj}{N}\right\} f_j, k = \overline{0, N-1}, i = \sqrt{-1} \quad \textbf{(2.1)}$$

Формула обратного преобразования Фурье:

$$f_k = \frac{1}{N}\sum_{j=0}^{N-1} exp\left\{2\pi i \frac{kj}{N}\right\} A_j, k = \overline{0, N-1}, i = \sqrt{-1} \quad \textbf{(2.2)}$$

Трудоемкость вычисления по формулам *(2.1)* и *(2.2)* CN^2, где C – *const*, т.к. каждый из N коэффициентов состоит из N слагаемых и каждое слагаемое вычисляется за C действий ($C = 5$ для прямого и обратного преобразования).

2.2.2. Полубыстрое преобразование Фурье (ППФ)

Попробуем найти алгоритм, который реализует преобразование Фурье несколько быстрее.

Идея, наводящая на алгоритм быстрого преобразования Фурье (БПФ):

Рассмотрим случай, когда $N = p_1 \cdot p_2$
Представим k и j в виде: $k = k_1 + p_1 k_2$ $\quad 0 \le k \le N-1$
$j = j_2 + p_2 j_1$ $\quad 0 \le j \le N-1$
Здесь k_1 – остаток от деления k на p_1 $\quad k_1 = k \bmod p_1$
k_2 – частное от деления k на p_1 $\quad k_1 = k \operatorname{div} p_1$

$0 \le k_1 \le p_1 - 1$
$0 \le k_2 \le p_2 - 1$

аналогично

j_1 – частное от деления j на p_2 $\quad j_1 = j \text{ div } p_2$
j_2 – частное от деления j на p_2 $\quad j_2 = j \bmod p_2$

$$0 \le j_1 \le p_1 - 1$$
$$0 \le j_2 \le p_2 - 1$$

Когда число j меняется от 0 до N, то j_1 и j_2 пробегают независимым образом все возможные значения в своих диапазонах. Поэтому, вместо однократного суммирования можно применить двукратное:

$$A_k = \frac{1}{p_1 \cdot p_2} \sum_{j_1=0}^{p_1-1} \sum_{j_2=0}^{p_2-1} \exp\left\{-2\pi i \frac{(k_1+p_1k_2)(j_2+p_2j_1)}{p_1p_2}\right\} \cdot f_{j_2+p_2j_1} = \vdots\vdots\vdots$$

$A^{(0)}(k_1,k_2)$

Можно игнорировать, т.к. оно добавляет к $\exp(\alpha) = \cos\alpha + i\cdot\sin\alpha$ целое число периодов и сама exp не меняется

Заметим, что:

$$\frac{(k_1+p_1k_2)(j_2+p_2j_1)}{p_1p_2} = \frac{j_1k_1}{p_1} + \frac{j_2}{p_1p_2} \cdot (k_1 + p_1k_2) + k_2j_1$$

$$\vdots\vdots\vdots = \frac{1}{p_1}\sum_{j_1=0}^{p_1-1} \frac{1}{p_2}\sum_{j_2=0}^{p_2-1} f_{j_2+p_2j_1} \cdot exp\left\{-2\pi i\left(\frac{j_1k_1}{p_1} + \frac{j_2}{p_1p_2} \cdot (k_1 + p_1k_2)\right)\right\} =$$

$A^{(1)}(k_1,j_2)$

$$= \frac{1}{p_2}\sum_{j_2=0}^{p_2-1} exp\left\{-2\pi i\left(\frac{j_1k_1}{p_1} + \frac{j_2}{p_1p_2} \cdot (k_1 + p_1k_2)\right)\right\} \cdot \frac{1}{p_1}\sum_{j_1=0}^{p_1-1} f_{j_2+p_2j_1} \cdot exp\left\{-2\pi i\left(\frac{j_1k_1}{p_1}\right)\right\}$$

$A^{(2)}(k_1,k_2)$

Итак, мы будем вычислять преобразование Фурье не по *(2.1)*, а по формулам *(2.3а)* и *(2.3б)*:

$$A^{(1)}(k_1, j_2) = \frac{1}{p_1}\sum_{j_1=0}^{p_1-1} f_{j_2+p_2j_1} \cdot exp\left\{-2\pi i\left(\frac{j_1k_1}{p_1}\right)\right\} \quad \textbf{(2.3а)}$$

$0 \le k_1 \le p_1 - 1$
$0 \le j_2 \le p_2 - 1$

$$A^{(2)}(k_1, k_2) = \frac{1}{p_2}\sum_{j_2=0}^{p_2-1} A^{(1)}(k_1, j_2) \cdot exp\left\{-2\pi i\left(\frac{j_2}{p_1p_2} \cdot (k_1 + p_1k_2)\right)\right\} \quad \textbf{(2.3б)}$$

$0 \le k_1 \le p_1 - 1$
$0 \le j_2 \le p_2 - 1$

Оценим трудоемкость вычисления преобразования Фурье по формулам 2.3:

Массив $A^{(0)}$ переходит в $A^{(1)}$, а оттуда в $A^{(2)}$:

$$A^{(0)} \to A^{(1)} \to A^{(2)}$$

Этот переход осуществляется в 2 этапа.

В формуле *(2.3а)* имеем $p_1 \bullet p_2$ коэффициентов, каждый их которых есть сумма p_1 слагаемых – итого $p_1 \bullet p_2 \bullet p_1$ действий. В *(2.3б)* соответственно $p_1 \bullet p_2 \bullet p_2$ действий. Итого

$$T(n) = \underbrace{p_1 \cdot p_2 \cdot p_1}_{\text{по (2.3а)}} + \underbrace{p_1 \cdot p_2 \cdot p_2}_{\text{по (2.3б)}} = p_1 \cdot p_2 \cdot (p_1 + p_2) = N(p_1+p_2).$$

Если $p_1 \approx p_2 \approx \sqrt{N}$, то трудоемкость будет $T(n) = n^{3/2} \ll n^2$, что существенно меньше трудоемкости при обычном преобразовании Фурье. Т.е. получаем метод, позволяющий реализовать преобразование Фурье значительно быстрее. Этот метод назван *полубыстрым преобразованием Фурье.*

2.2.3. Быстрое преобразование Фурье (БПФ)

Применим выше изложенную идею для случая $N = 2^r$.
Представим числа k и j в двоичной записи:
$k = \sum_{L=1}^{r} k_L 2^{L-1}, k = \overline{k_r, \dots, k_1}$ - двоичная запись числа k, $k_L = \overline{0,1}$
$j = \sum_{m=1}^{r} j_{r+1-m} 2^{m-1}, j = \overline{j_1, \cdots, j_r}$ - двоичная запись числа j, $j_i = \overline{0,1}$.
Аргументов экспоненты всякий раз является число $\frac{k \cdot j}{N}$:

$$\frac{k \cdot j}{N} = k \cdot j \cdot 2^{-r} = \sum_{L=1}^{r} k_L 2^{L-1} \sum_{m=1}^{r} j_{r+1-m} 2^{m-1} 2^{-r} = \sum_{m=1}^{r} \underbrace{\left(\sum_{L=1}^{r} k_L 2^{L+m-r-2} \right)}_{\text{Все целочисленные слагаемые в данной сумме можно отбросить, т.к. они добавят целое число периодов к аргументу и не повиляют на значение } \exp(2\pi i)} j_{r+1-m} =$$

$$= \sum_{m=1}^{r} \left(\sum_{L=1}^{r-m+1} k_L 2^{L+m-r-2} \right) j_{r+1-m} + \underbrace{S}_{\text{Целое число, не влияющие на значение } \exp(-2\pi i). \text{ Его можно отбросить}}$$

$$A_k = A(k_1, k_2, \dots, k_r) = \frac{1}{N} \sum_{j=0}^{N-1} f_j exp\left\{ -2\pi i \frac{kj}{N} \right\} = \vdots\vdots\vdots$$

Учтём, что, когда j пробегает все значения от *0* до *N-1*, индексы $j_1 \dots j_r$ меняются независимым образом, принимая значения либо 0 либо 1.

$$\vdots\vdots\vdots = \frac{1}{2} \sum_{j_r=0}^{1} \frac{1}{2} \sum_{j_{r-1}=0}^{1} \cdots \frac{1}{2} \sum_{j_1=0}^{1} f_{j_r + 2 j_{r-1} + \cdots + 2^{r-1} j_r} \cdot exp\left\{ -2\pi i \left[\sum_{m=1}^{r} \left(\sum_{L=1}^{r-m+1} k_L 2^{L+m-r-2} \right) j_{r+1-m} \right] \right\} =$$

Суммирование по m распишем подробно и каждую соответствующую компоненту заносим в соответствующую сумму.

$$= \frac{1}{2} \sum_{j_r=0}^{1} exp\left\{ -2\pi i \cdot j_r \cdot 2^{-r} \sum_{L=1}^{r} k_L 2^{L-1} \right\} \times \frac{1}{2} \sum_{j_{r-1}=0}^{1} exp\left\{ -2\pi i \cdot j_{r-1} \cdot 2^{1-r} \sum_{L=1}^{r-1} k_L 2^{L-1} \right\} \times$$

$$\times = \frac{1}{2} \sum_{j_{r-2}=0}^{1} exp\left\{ -2\pi i \cdot j_{r-2} \cdot 2^{2-r} \sum_{L=1}^{r-2} k_L 2^{L-1} \right\} \times \dots \times \frac{1}{2} \sum_{j_2=0}^{1} exp\{ -2\pi i \cdot j_2 \cdot 2^{-2} (2k_2 + k_1) \} \times$$

$$\times \frac{1}{2} \sum_{j_1=0}^{1} exp\{ -2\pi i \cdot j_1 \cdot 2^{-1} k_1 \} f_{j_r + 2 j_{r-1} + \cdots + 2^{r-1} j_1}.$$

Преобразование Фурье можно вычислять по следующей формуле *(2.4)*:
$A^{(s+1)}(k_1, k_2, \dots, k_{s+1}, j_{s+2}, .., j_r) =$

$$= \frac{1}{2} \sum_{j_{s+1}=0}^{1} exp\left\{ -2\pi i j_{s+1} 2^{-(S+1)} \sum_{L=1}^{S+1} k_L 2^{L-1} \right\} A^{(s)}(k_1, .., k_s, j_{S+2}, .., j_r)$$

где *s = 0,...,r-1*. ***(2.4)***

Итак, необходимо выполнить r шагов, постепенно переходя по формуле *(2.4)* от массива $A^{(0)}$ к массиву $A^{(S)}$.
Данную формулу можно переписать в виде рекуррентной последовательности:
$A^{(0)}(j_1, j_2, .., j_r) = f_{j_r + 2 j_{r-1} + \cdots + 2^{r-1} j_1}$,

$$A^{(1)}(k_1, j_2, .., j_r) = \frac{1}{2} \sum_{j_1=0}^{1} exp\{ -2\pi i j_1 2^{-1} k_1 \} \cdot A^{(0)}(j_1, j_2, .., j_r),$$

$$A^{(2)}(k_1, k_2, j_3, .., j_r) = \frac{1}{2} \sum_{j_2=0}^{1} exp\{ -2\pi i j_2 2^{-2} (2k_2 + k_1) \} \cdot A^{(1)}(k_1, j_2, .., j_r),$$

$$A^{(3)}(k_1, k_2, k_3, j_4, .., j_r) = \frac{1}{2} \sum_{j_3=0}^{1} exp\{ -2\pi i j_3 2^{-3} (4k_3 + 2k_2 + k_1) \} \cdot A^{(2)}(k_1, k_2, j_3, .., j_r),$$

и т.д.
$A^{(r)}(k_1, k_2, \dots, k_r) = A_{k_1+2k_2+\cdots+2^{r-1}k_r}$.

Трудоемкость вычисления каждого коэффициента по формуле *(2.4)* равна *const* (при условии, что заранее уже заполнен массив частичных сумм разрядов всех числе от *0* до *N-1*). Размер массива $N*r$.

Итого, трудоемкость БПФ по формуле *(2.4)* составляет:

$C \cdot 2^r \cdot r = C\, N \log N$, где 2^r – N элементов в каждом массиве,

r – число шагов.

Трудоемкость преобразования Фурье по формулам *(2.4)* существенно меньше. Чем у ранее изученных методов с трудоемкостями $C\, n^2$ и $C\, n^{3/2}$.

Аналогичные формулы можно вывести не только лишь в случае $N = 2^r$ (как в классическом БПФ), но и в случае $N = k_1 \cdot k_2 \cdot \ldots \cdot k_r$. В этом случае трудоемкость составит:

$$T(n) = N(k_1 + k_2 + \ldots + k_r).$$

Заметим, что и для *полубыстрого* и для *быстрого преобразования Фурье* обратное преобразование вычисляется по тем же формулам, что и прямое только в *exp* нет знака « – » и отсутствуют множители $\frac{1}{p_1}$ и $\frac{1}{p_2}$ в *полубыстром* и ½ в *быстром преобразовании Фурье*.

Домашнее задание №2

Выразить $A^3(1, 0, 1, 1)$ через A^2 и экспоненту по формуле быстрого преобразования Фурье. $N = 2^4$.

2.3. Быстрая свертка

2.3.1. Понятие свертки

Определение. Пусть даны два массива:

$a = (\ldots, a_{-1}, a_0, a_1, a_2, \ldots)$, $b = (\ldots, b_{-1}, b_0, b_1, b_2, \ldots)$ – конечные или бесконечные в одну или в обе стороны. Тогда ***сверткой*** последовательности a и b называется массив

c = (.., c_{-1}, c_0, c_1, c_2, …), где $c_i = \sum_{k+l=i} a_k b_l$ ***(2.5а)***

т.е. суммирование производится по всевозможным сочетаниям, в которых $k + l = i$. Обозначается свёртка как: $c = a*b$.

В случае, когда последовательности (массивы) *a* и *b* бесконечны в обе стороны, выражение для c_i можно представить в виде

$$c_i = \sum_{k=-\infty}^{\infty} a_k b_{i-k} \quad \textbf{(2.5б)}$$

Аналогичным образом определяется свертка для конечных массивов (в формулу (*2.5.б)* подставляются конечные пределы суммирования).

Пример. Даны два массива:

$a = (a_0, a_1)$;

$b = (b_0, b_1, b_2)$.

Определим свёртку этих массивов , используя формулу *(2.5а)*:

$c_0 = a_0 b_0$,

$c_1 = a_0 b_1 + a_1 b_0$,

$c_2 = a_1 b + a_0 b_2$,

$c_3 = a_1 b_2$.

Рассмотрим классический пример свёртки.

1) При умножении многочленов:
   $a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$,
   $bx) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$.
   $a(x)*b(x) = c(x) = c_0 + c_1 x_1 + \ldots + c_{2n-2} x^{2n-2}$,
   $c = a*b$.
2) При перемножении числе столбиком (если не делать переносы в старшие разряды).
3) Пусть *a* и *b* – дискретные независимые случайные величины, такие что $P(a = i) = a_i$, $P(b = i) = b_i$. Причем *a* и *b* – обязательно независимые случайные величины (независимые случайные величины – те, для которых справедливо высказывание: P((x ∈ A) и (y ∈ B)) = P(x ∈ A) · P(y ∈ B),

где x, y – значения случайных величин, A и B – некоторые множества из области значений соответствующих случайных величин).

Тогда, случайная величина $c = a + b$ имеет распределение, которое есть ни что иное, как всёртка массивов a_i и b_i.

$$c_i = \{P(c = a + b = i)\} = \{\sum_k^\infty P(a = k) \cdot P(b = i - k)\} = \sum_k^\infty a_k b_{i-k} = (a * b)_i .$$

2.3.2. Обычный и быстрый алгоритм свертки

Рассмотрим два конечных вектора (массива). Для удобства их длина одинакова и равна n. Если же массивы имеют разное количество элементов n и m, причем $n>m$, то более короткий массив дополняем n-m нулями до n элементов (выравниваем массивы по более длинному).

Имеем:

$a = (a_0, a_1, a_2, \ldots, a_{n-1})$

$b = (b_0, b_1, b_2, \ldots, b_{n-1})$.

Оценим трудоемкость вычисление свертки по обычному алгоритму:

$c_0 = a_0 b_0$ - состоит из 1 слагаемого

$c_1 = a_0 b_1 + a_1 b_0$ - состоит из 2 слагаемых

$c_2 = \ldots$ - состоит из 3 слагаемых

до середины массива c количество слагаемых, из которых состоят его элементы, увеличивается до n

$$c_{n-1} = \underbrace{a_0 b_{n-1} + \ldots + a_{n-1} b_0}_{n \text{ слагаемых}},$$

а от c_n до c_{2n-2} уменьшается. Последний элемент массива – c_{2n-2} – имеет одно слагаемое. Итого:

$$T(n) = 1 + 2 + 3 + \ldots + (n-1) + n + (n-1) + \ldots + 3 + 2 + 1 = n^2.$$

Если реализовать вычисления по формуле *(2.5a)*, т.е. нерационально – путем организации трех вложенных циклов (во внутреннем цикле будет проверка по условию $k + l = i$), то трудоемкость будет даже не n^2, а n^3. На самом деле, используя в формулах свертки преобразование Фурье, можно снизить трудоемкость с n^2 до $n\log(n)$. Для вывода этих формул сначала докажем теорему.

Теорема 2.1. *Пусть $a = (a_0, a_1, a_2, \ldots, a_{n-1}, 0, 0, 0 \ldots 0)$*
и $b = (b_0, b_1, b_2, \ldots, b_{n-1}, 0, 0, 0 \ldots 0)$ – массив длины $2n$ полученные из первоначальных массивов длины n путем дописывания n нулей. Соответственно, $F(a)$ и $F(b)$ – преобразования Фурье от этих массивов, т.е.

$$F(a) = (\widetilde{a_0}, \widetilde{a_1}, \ldots, \widetilde{a_{2n-1}})$$

$$F(b) = (\widetilde{b_0}, \widetilde{b_1}, \ldots, \widetilde{b_{2n-1}}).$$

Тогда преобразование Фурье от их свертки:

$$F(c) = F(a * b) = (\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{2n-1})$$

есть ничто иное, как покоординатное произведение массивов $F(a)$ и $F(b)$, помноженное на $2n$: $c_k = 2n\, \tilde{a}_k \cdot \tilde{b}_k$.

Доказательство. Введем переменную $w = exp\{-2\pi i/2n\}$.

Тогда, по формулам преобразования Фурье:

$$\tilde{a}_k = F_k(a) = \frac{1}{2n} \sum_{j=0}^{2n-1} a_j exp\left\{-2\pi i \frac{kj}{2n}\right\} = \frac{1}{2n} \sum_{j=0}^{2n-1} a_j w^{kj} .$$

Аналогично,

$$\tilde{b}_k = F_k(a) = \frac{1}{2n} \sum_{j=0}^{2n-1} b_j w^{kj} = \frac{1}{2n} \sum_{l=0}^{2n-1} b_l w^{kl} .$$

Теперь мы можем перемножить и . Посмотрим, что получится в результате:

$$\tilde{a}_k \cdot \tilde{b}_k = \frac{1}{(2n)^2} \sum_{j=0}^{2n-1} \sum_{l=0}^{2n-1} a_j b_l w^{k(l+j)} = \vdots\vdots\vdots$$

Ввёдем новую переменную $m = l + j$.

$$\vdots\vdots\vdots = \frac{1}{(2n)^2}\sum_{m=0}^{2n-1}\sum_{j=0}^{2n-1} a_j b_{m-j} w^{km}.$$

$$\sum_{j=0}^{2n-1} a_j b_{m-j} = c_m$$

В результате получаем

$$\tilde{c}_k = \frac{1}{(2n)}\sum_{m=0}^{2n-1} c_m w^{km}.$$

Теорема доказана.

Используя ***Теорему 2.1***, в которой фактически сказано, что преобразование Фурье от свертки есть

$$F(c) = F(a*b) = 2n\, F(a) \circ F(b)$$

Операция покоординатного перемножения массивов

приходим к выводу, что можно выполнять свертку с помощью преобразования Фурье по следующей формуле:

$$a*b = F^{-1}(2nF(a) \circ F(b)), \quad \textbf{(2.6)}$$

где F^{-1} – обратное преобразование Фурье. => Свертка массивов a и b может быть выполнена по формуле *2.6*.

Трудоемкость свертки по формуле *2.6:*

$$T(n) = 3 \cdot 2n \cdot \log(2n) \approx n \log n << n^2$$

т.к. в расчетах применяются 2 прямых и одно обратно преобразование Фурье с одинаковой трудоемкостью *2nlog(2n)*, и длина обрабатываемого массива *2n*

трудоемкость покоординатного перемножения массивов

2.4. Быстрое умножение

2.4.1. Быстрое умножение чисел

При обычном умножении столбиком нам потребуется n^2 операций, где n – количество разрядов чисел. Попробуем произвести умножение чисел быстрее. Для этого поступим следующим образом:

Пусть x и y – два n-разрядных числа ($n = 2k$).

Разобьем их запись пополам, т.е. на два отрезка длины k;

$x = a \cdot 2^k + b$

, где

$y = c \cdot 2^k + d$

$$x = \underbrace{x_1 x_2 \ldots x_k}_{a = \text{старшие разряды}} \underbrace{x_{k+1} \ldots x_{2k}}_{b = \text{младшие разряды}}$$

$$y = \underbrace{y_1 y_2 \ldots y_k}_{c = \text{старшие разряды}} \underbrace{y_{k+1} \ldots y_{2k}}_{d = \text{младшие разряды}}$$

Рассмотрим умножение $x \cdot y$ по обычным формулам:

4 произведения

$$x \cdot y = (a \cdot 2^k + b)(c \cdot 2^k + d) = a \times c \cdot 2^k + (a \times d + b \times c) \cdot 2^k + b \times d \quad \textbf{(2.7)}$$

1 суммирование

При перемножении по *(2.7)* получаем рекуррентную формулу, для вычисления трудоёмкости:

$$T(n) = 4 \cdot T(n/2) + 1 \cdot n$$

4 произведения чисел вдвое меньшей

одно сложение

Таким образом, в *(2.7)* четыре произведения и одно сложение. Попробуем обойтись меньшим количеством произведений для умножения двух числе по формуле *(2.7)*. Рассмотрим модификацию формулы *(2.7)* – формулу *(2.8)*.

$$\begin{cases} u = (a+b)(c+d) \\ v = a \cdot c \\ w = b \cdot d \end{cases}$$

Тогда $xy = v\,2^{2k} + (u - v - w)2^k + w$ **(2.8)**

При перемножении чисел по *(2.8)* нам потребуется 3 умножения и 4 действия сложения и вычитания. Итого получаем трудоемкость:

$$T(n) = 3\,T(n/2) + 4n. \quad \textbf{(2.9)}$$

Временно забудем, что числа $(a+b)$ и $(c+d)$ могут быть не k-разрядные, а $k+1$-разрядные. Организуем процесс умножения двух n-разрядных чисел по формуле *(2.8)* следующим образом: на входе имеем два длинных числа. Каждое из них разобьем пополам, и получается, что нам нужно 3 раза перемножить числа вдвое меньшей разрядности. Каждое из этих чисел снова делим пополам и перемножаем по формуле *(2.8)*. Подобная процедура дробления осуществляется до тех пор, пока в результате очередного разделения не получатся одноразрядные числа, которые перемножаем по обычной таблице умножения.

Оценим трудоемкость подобных вычислений по формуле *(2.8)*. Она оценивается по рекуррентной формуле *(2.9)*.

Теорема 2.2. *Если трудоемкость некоторого алгоритма задается формулой*

$$T(n) = \alpha \cdot T(n/k) + c \cdot n^{\beta},$$

где k *– целое число,* $k > 1$

$\beta < \log_k \alpha$, $\beta > 0$

α *– обычное целое (хотя это необязательно),*

то существует следующая оценка трудоемкости длинного алгоритма:

$$T(n) \approx n \log_k \alpha$$

Без доказательств.

Комментарии. Что делать, если происходит переполнение разрядов при вычислении по формуле ***(2.8)***?

Предположим, что при сложении (при вычислении u) у нас получились не k, а $(k+1)$-разрядные числа, т.е. a и b имеют $k+1$ разряд.

Выделим старший разряд: $a_1, b_1 = 0$ или 1, но один из них обязательно равен 1. Тогда произведение можно осуществить следующим образом:

$$a \cdot b = a_1 b_1 \cdot 2^{2k} + (a_1 b_1 + a_2 b_2)\,2^k + a_2 b_2$$

флаг (0 или 1)

равно либо b_2, либо a_2, либо в худшем случае $a_2 + b_2$

В случае, когда возникает переполнение, перемножение $k+1$ разрядных числе выполняется по формуле *2.10*. В ней по-прежнему одно умножение $(a_2 b_2)$ и одно сложение, которое может и не понадобиться. На трудоемкости (рекуррентная формула *2.9*) это существенно не отражается:

$T(n) = 3\,T(n/2) + 4n$ **(2.9')**

или $T(n) = 3\,T(n/2) + 5n$.

Согласно ***Теореме 2.2,*** на трудоемкости перемножения по формуле *2.8* (формула быстрого умножения) это не сказывается. Получаем:

$$T(n) \approx n^{\log 3} \approx n^{1.58} << n^2$$

2.4.2. Быстрое умножение матриц

Оценим трудоемкость обычного умножения двух матриц $n \times n$. Трудоемкость будет иметь порядок n^3, т.к. в матрице-результате перемножения будет $n \times n = n^2$ элементов и каждый из них вычисляется за n операций попарного умножения и сложения.

Попробуем применить ту же идею, что и быстрого умножения чисел, при перемножении двух матриц $n \times n$. Разделим каждую из них на 4 матрицы вдвое меньшего размера:

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \quad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \quad A_i, B_i = \frac{n}{2} \times \frac{n}{2}$$

$$C = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$$

$$C_1 = A_1B_1 + A_2B_2 \quad C_3 = A_3B_1 + A_4B_3$$
$$C_2 = A_1B_2 + A_2B_4 \quad C_4 = A_3B_2 + A_4B_4$$

Итак, при применении обычных формул блочного произведения матриц получаем рекуррентную формулу:

$$T(n) = \underbrace{8T\left(\frac{n}{2}\right)}_{\text{трудоемкость перемножения матриц вдвое меньшего размера.}} + \underbrace{cn^2}_{\text{трудоемкость сложений.}}$$

Соответственно по ***Теореме 2.2*** получаем $T(n) = cn^{\log_2 8} = cn^3$, то есть трудоемкость такая же, как и при обычном умножении.

Однако существуют формулы Штрассена для блочного умножения матриц. В этих формулах будет не 8, а 7 попарных умножений матриц размера $\frac{n}{2} \times \frac{n}{2}$.

Формула Штрассена:

$$\left.\begin{aligned} M_1 &= (A_2 - A_4)(B_3 + B_4) \\ M_2 &= (A_1 + A_4)(B_1 + B_4) \\ M_3 &= (A_1 - A_3)(B_1 + B_2) \\ M_4 &= (A_1 + A_2)B_4 \\ M_5 &= A_1(B_2 - B_4) \\ M_6 &= A_4(B_3 - B_1) \\ M_7 &= (A_3 + A_4)B_1 \\ C_1 &= M_1 + M_2 - M_4 + M_6 \\ C_2 &= M_4 + M_5 \\ C_3 &= M_6 + M_7 \\ C_4 &= M_2 - M_3 + M_5 - M_7 \end{aligned}\right\} \quad \textbf{(2.11)}$$

7 умножений и 18 сложений и вычитаний в этих формулах.

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

Следовательно, по ***Теореме 2.2*** получаем

$$T(n) = cn^{\log_2 7} = cn^{2,81} \ll cn^3.$$

2.4.3. Очень быстрое умножение числе (алгоритм Шенхаге – Штрассена)

Как уже было замечено, умножение многоразрядных чисел есть сверка двух массивов с переносом в старшие разряды. Если делать свертку с применением БПФ, то **трудоемкость** подобного алгоритма $T(n) = n \log n$. Однако в таком случае на вход свертки подаются целочисленные массивы, а на выходе получается массив вещественных чисел, которые нужно округлить до ближайшего целого.

Замечание. При большой погрешности вычислений существует опасность округлить не в ту сторону. Следовательно, для уменьшения вероятности ошибки нужно увеличивать разрядность вычислений.

Тогда трудоемкость алгоритма составит:

$$T(n) = \underbrace{n\log_2 n}_{\text{трудоемкость быстрого перемножения}} \cdot \underbrace{\log_2(\log_2 n)}_{\text{т.к. мы увеличиваем разрядность чисел, с которыми работаем}}.$$

Домашнее задание №3. Найти произведение 3871 и 9211 по формуле быстрого умножения чисел.
Домашнее задание №4. Найти произведение 8329 и 5631 по формуле быстрого умножения чисел.

3. ЗАДАЧИ НА ГРАФАХ

3.1. Справочный материал

Кратко напомним определения, известные из курса дискретной математики.

Определения. *Граф* – пара $G(V,E)$, где $V = \{V_1, V_2, \ldots, V_n\}$ – множество вершин графа G, E – множество ребёр (дуг) графа: $E=\{ (V_{1i}, V_{2i}) \}$.
Рёбра можно задавать разными способами, например матрицей инцидентности или списком ребёр.

Неориентированный граф – граф, удовлетворяющий условию, что если $(a, b) \in E$, то и $(b, a) \in E$, т.е. порядок расположения концов дуг графа не существенен.
Матрица инцидентности такого графа симметрична.

Взвешенный граф – граф, каждой дуге которого приписан её вес.

Цикл в графе – маршрут, заданный последовательностью вершин, каждая вершина посещается по одному разу, и начальная вершина совпадает с конечной.

Граф без цикла – куст.

Связный граф – граф, из любой вершины которого можно дойти до любой другой.

Связный куст – дерево.

Компонента связности – совокупность вершин, для каждой из которых существует пусть в любую другую вершину этой совокупности.

3.2. Поиск минимального остова в связном неориентированном взвешенном графе

Задача. *Дан граф G(V,E) – связный, неориентированный, взвешенный. Нам нужно выделить в нем минимальный (по суммарному весу ребер) связный граф с теми же вершинами – остов (остовное дерево), т.е. исключить из графа часть ребёр таким образом, чтобы сумма весов оставшихся была минимальна, и получившийся граф по-прежнему был связным.*

Идея решения. Для решения этой задачи обычно применяется *алгоритм Краскалла* (классический пример т.н. «жадного алгоритма»).

Алгоритм Краскалла

1. Сначала упорядочиваем все ребра по возрастанию весов.
2. Заводим таблицу: в левой колонке список ребер, в правой компоненты связности.
3. В первой строчке список ребер пустой и все компоненты связности одновершинные. Берем минимальное по стоимости (по весу) ребро включаем его в список ребер. Соответствующие две вершины объединяем в одну компоненту связности.
4. Берем следующее по стоимости (весу) ребро, добавляем к содержимому предыдущей строчки левого столбца; объединяем компоненты связности. Если концы ребра уже принадлежат одной и той же компоненте связности, то данное ребро в состав минимального остова не включается.
5. Повторяем эти операции до тех пор, пока все вершины не окажутся в одной единственной компоненте связности (для этого потребуется включить в состав остова ровно $n-1$ ребро).

Пример. Пусть имеется граф, изображенный на рисунке 3.1.

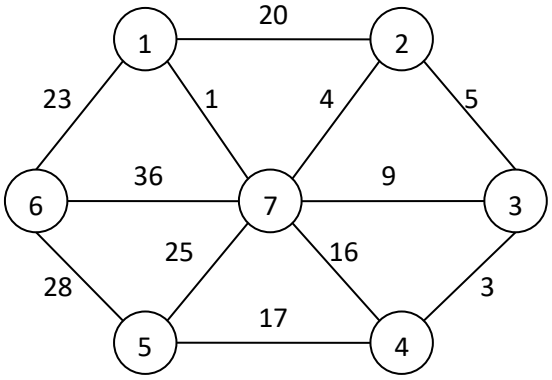


Рисунок 3.1. – Исходный граф

Решение. Составим таблицу, в которой будем отражать порядок выбора ребер для минимального остова.

Подграф (ребра)	**Связи (компоненты связности)**
Пустой	1,2,3,4,5,6,7
(1,7) – минимально по цене	(1,7),2,3,4,5,6
(1,7) + (3,4)	(1,7),2,(3,4),5,6
(1,7)(3,4) + (2,7)	(1,2,7),(3,4),5,6
(1,7)(3,4) (2,7) + (2,3)	(1,2, 3,4,7),5,6
(1,7)(3,4) (2,7) (2,3) ~~+ (3,7)~~*	(1,2, 3,4,7),5,6
(1,7)(3,4) (2,7) (2,3) ~~+ (4,7)~~**	(1,2, 3,4,7),5,6
(1,7)(3,4) (2,7) (2,3) + (4,5)	(1,2, 3,4,5 ,7),6
(1,7)(3,4) (2,7) (2,3)(4,5) ~~+ (1,2)~~***	(1,2, 3,4,5 ,7),6
(1,7)(3,4) (2,7) (2,3) (4,5) + (1,6)	(1,2, 3,4,5 ,6,7)

Пояснения.

* - вершины 3 и 7 уже находятся в одной компоненте связности на момент включение в остов ребра (3,7);

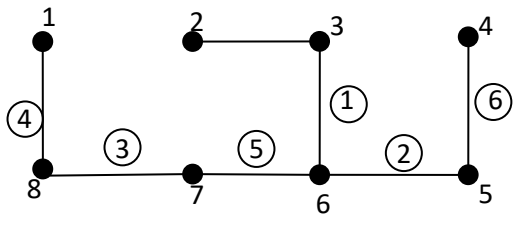
** - следующее по весу ребро – (4,7), однако вершины 4 и 7 уже находятся в одной компоненте связности на момент включения в остов ребра (4,7);

*** - аналогично.

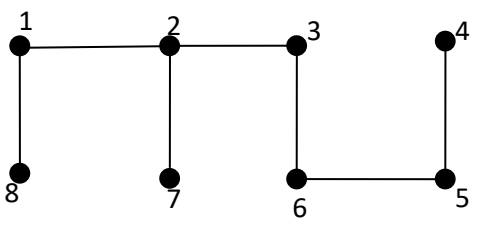
Теорема 3.1. *Алгоритм Краскалла всегда выдает минимальный по стоимости остов.*

Доказательство. Заметив, что в результате работы алгоритма Краскалла мы всегда получаем дерево (так как мы не включаем ребра, вершины которых лежат в одной компоненте связности, следовательно возникновение циклов невозможно).

Предположим, что получившееся в результате работы алгоритма Краскалла дерево не минимально по своей стоимости и существует другое, предположительно «оптимальное» дерево. Например, такое, как на рисунке 3.2. Цифрами в кружочках обозначены ребра в порядке их добавления в дерево.



Дерево, полученное в результате работы алгоритма Краскалла



предполагаемое «оптимальное» дерево

Рисунок 3.2.

Рассмотрим подробно тот этап работы алгоритма Краскалла, на котором мы впервые добавили в остов «неправильное» ребро, т.е. то, которого нет в «оптимальном» дереве. Первые два по породку добавления ребра (3,6) и (6,5) присутствуют в «оптимальном» дереве. Расхождение начинается после добавление ребра (7,8).

Добавил ребро (7,8) к «оптимальному» дереву (см. рис 3.3):

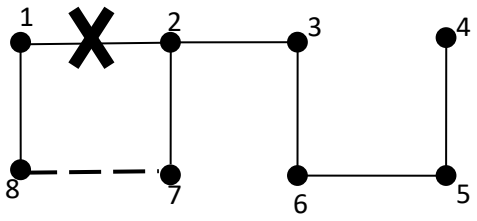


Рисунок 3.3.

При этом в «оптимальном» дереве возникает цикл, и мы можем убрать из него любое ребро, например, (1,2), участвующее в цикле, и при этом граф останется связанным. Заметим, что при этом общий вес полученного дерева будет меньше веса «оптимального» дерева, т.к. ребро (7,8) самое меньшее по весу в этом цикле (т.к. в алгоритме Краскалла мы по очереди добавляем все ребра, начиная с самого минимального по весу, меньше ребра (7,8) по весу только ребра (3,6) и (6,5), а они в этот цикл не входят). В этом примере мы можем построить меньший, чем предполагаемый «оптимальный», по суммарному весу остов.

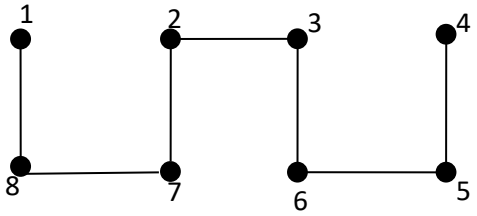


Рисунок 3.4. – Минимальный остов.

Он отличается от «оптимального» заменой ребра (1,2) на меньшее по весу ребро (7,8).

Данные рассуждения можно обобщить для случая произвольного графа.

Теорема доказана.

Оценим трудоемкость работы алгоритма Краскалла

Пусть в графе было *n* вершин и *k* ребер ($k \leq \frac{n\,(n-1)}{2}$). При быстрой сортировке на первый эта работы алгоритма затрачивается *k logk* действий. Далее нам потребуется ровно *(n-1)* результативных этапов (результативным будем считать этап работы алгоритма, в результате которого осуществляется добавление ребра к минимальному остову и слияние компонент связности), и не больше, чем *k – (n-1)* нерезультативных этапов. Нерезультативный этап состоит из одного сравнения – проверки принадлежности вершин добавляемого ребра на нахождение в одной компоненте связности. В случае же результативного этапа мы должны переприсвоить номера компонент связности n вершинам – итого n действий.

Таким образом, трудоемкость алгоритма Краскалла:

$$T(n) = \underbrace{(n\text{-}1)\cdot n}_{\text{Трудоемкость результативных этапов}} + \underbrace{(k-n+1)\cdot 1}_{\text{Трудоемкость нерезультативных этапов}} + \underbrace{k\,\log k}_{\text{Трудоемкость сортировки}} \approx n^2 \log n$$

Следовательно, алгоритм Краскалла – полиномиальный.

3.3. Нахождение кратчайшего расстояния

Дан связный неориентированный взвешенный граф G. Если существует ребро с вершинами v_i и v_j, то стоимость перехода из v_i в v_j составит $C(i,j) = C(v_i, v_j)$. Если же ребра $v_i – v_j$ нет, то полагаем $C(i,j) = \infty$.

На выходе в данном графе выделяется вершина v_0. Надо найти кратчайшее расстояние от v_0 до всех остальных вершин.

Для решения задачи поиска кратчайшего расстояния используется (кроме прямого перебора) два алгоритма: Форда – Беллмана и Дейкстры.

Рассмотрим простейший алгоритм поиска кратчайшего расстояния – алгоритм Форда-Беллмана. Этот алгоритм является классическим примером алгоритма на поиск транзитивного замыкания.

Общая идея работы всех алгоритмов на поиск транзитивного замыкания

Пересчитываем что-либо (в нашем случае, стоимости вершин) до тех пор, пока это что-то не стабилизируется. Как только произойдет стабилизация, необходимо остановиться. Ответ получен.

3.3.1. Алгоритм Форда – Беллмана

Формируем массив $D^{[k]}(i)$ – минимально возможная стоимость переезда (перехода, перевозки) из вершины v_0 в v_1 на каждом этапе работы нашего алгоритма.

Первоначально он задается как $D^{[0]}(i) = (0, \infty, \infty, \ldots, \infty)$.

Затем пересчитываем стоимости всех вершин по формуле

$$D^{[k+1]}(i) = \min_j(D^{[k]}(j) + C(i,j)) \quad \textbf{(3.1)}$$

до тех пор, пока система не стабилизируется (так называемое *транзитивное замыкание*). В результате мы получим стоимости переезда из каждой вершины графа до заданной v_0 эти стоимости будут минимально возможными.

Пример. Дан граф (рис 3.5). Найти расстояние от нулевой вершины до всех остальных.

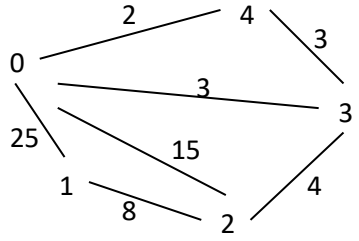


Рисунок 3.5 – Исходный граф

Решение:

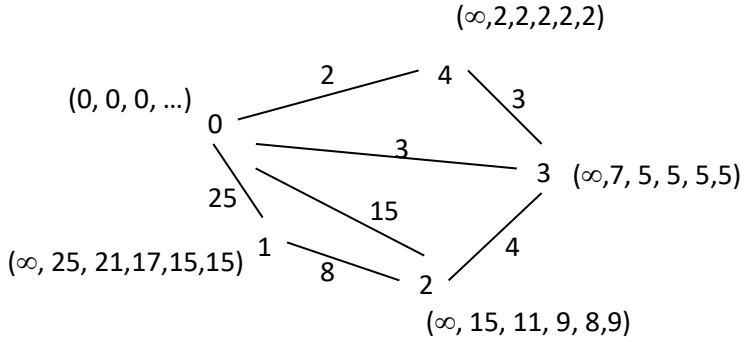


Рисунок 3.6 – Стоимости переездов из вершины v_0

Первоначальный массив стоимостей переходов выглядит так:

$D^{(0)}=(0, \infty, \infty, \infty, \infty)$. Сосчитаем стоимость вершины i на k-том шаге. В вершину j мы могли попасть из нулевой вершины, из первой вершины и т.д.

Тогда:

$$D^{(k+1)}(i) = (D^{(k)}(0) + (0,i), D^{(k)}(1) + (1,i), \ldots)$$

стоимость перехода из вершины 0 в вершину i.

стоимость вершины 0 на предыдущем шаге + стоимость переезда из нулевой вершины в i-ую.

Стоимость вершины *i* на каждом шаге считаем по формуле ***(3.1)***:

$D^{[k+1]}(i) = min\ (D^{[k]}(0) + C(0,i), D^{[k]}(1) + C(1,i), D^{[k]}(2) + C(2,i), D^{[k]}(3) + C(3,i), D^{[k]}(4) + C(4,i)\ ).$

Вычислим стоимости вершин данного графа на первом шаге:

$D^{[1]}(1) = min\ (D^{[0]}(0) + C(0,1), D^{[0]}(1) + C(1,1), D^{[0]}(2) + C(2,1), D^{[0]}(3) + C(3,1), D^{[0]}(4) + C(4,1)\ )$
$= min\ (0 + 25, \infty + 0, \infty + 6, \infty + \infty, \infty + \infty) = min\ (25, \infty, \infty, \infty, \infty) = 25;$

$D^{[1]}(2) = min\ (D^{[0]}(0) + C(0,2), D^{[0]}(1) + C(1,2), D^{[0]}(2) + C(2,2), D^{[0]}(3) + C(3,2), D^{[0]}(4) + C(4,2)\ )$
$= min\ (0 + 15, \infty + 6, \infty + 0, \infty + 4, \infty + \infty) = min\ (15, \infty, \infty, \infty, \infty) = 15;$

$D^{[1]}(3) = min\ (D^{[0]}(0) + C(0,3), D^{[0]}(1) + C(1,3), D^{[0]}(2) + C(2,3), D^{[0]}(3) + C(3,3), D^{[0]}(4) + C(4,3)\ )$
$= min\ (0 + 7, \infty + \infty, \infty + 4, \infty + 0, \infty + 3) = min\ (7, \infty, \infty, \infty, \infty) = 7;$

$D^{[1]}(4) = min\ (D^{[0]}(0) + C(0,4), D^{[0]}(1) + C(1,4), D^{[0]}(2) + C(2,4), D^{[0]}(3) + C(3,4), D^{[0]}(4) + C(4,4)\ )$
$= min\ (0 + 2\ \infty + \infty, \infty + \infty, \infty + 3, \infty + 0) = min\ (2, \infty, \infty, \infty, \infty) = 2;$

Вычислим стоимости вершин данного графа на втором шаге:

$D^{[2]}(1) = min\ (D^{[1]}(0) + C(0,1), D^{[1]}(1) + C(1,1), D^{[1]}(2) + C(2,1), D^{[1]}(3) + C(3,1), D^{[1]}(4) + C(4,1)\ )$
$= min\ (0 + 25, 25 + 0, 15 + 6, 7 + \infty, 2 + \infty) = min\ (25, 25, 21, \infty, \infty) = 21;$

$D^{[2]}(2) = min\ (D^{[1]}(0) + C(0,2), D^{[1]}(1) + C(1,2), D^{[1]}(2) + C(2,2), D^{[1]}(3) + C(3,2), D^{[1]}(4) + C(4,2)\ )$
$= min\ (0 + 15, 25 + 6, 15 + 0, 7 + 4, 2 + \infty) = min\ (15, 31, 15, 11, \infty) = 11;$

$D^{[2]}(3) = min\ (D^{[1]}(0) + C(0,3), D^{[1]}(1) + C(1,3), D^{[1]}(2) + C(2,3), D^{[1]}(3) + C(3,3), D^{[1]}(4) + C(4,3)\ )$
$= min\ (0 + 7, 25 + \infty, 15 + 4, 7 + 0, 2 + 3) = min\ (7, \infty, 21, 7, 5) = 5;$

$D^{[2]}(4) = min\ (D^{[1]}(0) + C(0,4), D^{[1]}(1) + C(1,4), D^{[1]}(2) + C(2,4), D^{[1]}(3) + C(3,4), D^{[1]}(4) + C(4,4)\ )$
$= min\ (0 + 2, 25 + \infty, 15 + \infty, 7 + 3, 2 + 0) = min\ (2, \infty, \infty, 10, 2) = 2$ – наблюдается стабилизация.

Вычислим стоимости вершин данного графа на третьем шаге:

$D^{[3]}(1) = min\ (0 + 25, 21 + 0, 11 + 6, 5 + \infty, 2 + \infty) = min\ (25, 21, 17, \infty, \infty) = 17;$

$D^{[3]}(2) = min\ (0 + 15, 21 + 6, 11 + 0, 5 + 4, 2 + \infty) = min\ (15, 27, 11, 9, \infty) = 9;$

$D^{[3]}(3) = min\ (0 + 7, 21 + \infty, 11 + 4, 5 + 0, 2 + 3) = min\ (7, \infty, 15, 5, 5) = 5$ – стабилизация;

$D^{[3]}(4) = min\ (0 + 2, 21 + \infty, 11 + \infty, 5 + 3, 2 + 0) = min\ (2, \infty, \infty, 8, 2) = 2$ – стабилизация;

Вычислим стоимость вершина данного графа на четвертом шаге:

$D^{[4]}(1) = min\ (0 + 25, 17 + 0, 9 + 6, 5 + \infty, 2 + \infty) = min\ (25, 17, 15, \infty, \infty) = 15;$

$D^{[4]}(2) = min\ (0 + 15, 17 + 6, 9 + 0, 5 + 4, 2 + \infty) = min\ (15, 23, 9, 9, \infty) = 9$ – стабилизация;

$D^{[4]}(3) = min\ (0 + 7, 17 + \infty, 9 + 4, 5 + 0, 2 + 3) = min\ (7, \infty, 13, 5, 5) = 5$ – стабилизация;

$D^{[4]}(4) = min\ (0 + 2, 17 + \infty, 9 + \infty, 5 + 3, 2 + 0) = min\ (2, \infty, \infty, 8, 2) = 2$ – стабилизация;

Вычислим стоимости вершин данного графа на пятом шаге:

$D^{[5]}(1) = min\ (0 + 25, 15 + 0, 9 + 6, 5 + \infty, 2 + \infty) = min\ (25, 15, 15, \infty, \infty) = 15$ – стабилизация;

$D^{[5]}(2) = min\ (0 + 15, 15 + 6, 9 + 0, 5 + 4, 2 + \infty) = min\ (15, 21, 9, 9, \infty) = 9$ – стабилизация;

$D^{[5]}(3) = min\ (0 + 7, 15 + \infty, 9 + 4, 5 + 0, 2 + 3) = min\ (7, \infty, 13, 5, 5) = 5$ – стабилизация;

$D^{[5]}(4) = min\ (0 + 2, 15 + \infty, 9 + \infty, 5 + 3, 2 + 0) = min\ (2, \infty, \infty, 8, 2) = 2$ – стабилизация;

Массив стоимостей вершин перестал изменяться. Таким образом, мы получили кратчайшее расстояние от всех вершин данного графа до нулевой вершины.

Оценим *трудоемкость* алгоритма Форда-Беллмана.

В нем участвуют три цикла: внешний типа While выполняется до тех пор, пока не произошла стабилизация. Следующий по вложенности цикл – по вершинам графа. Самый внутренний цикл (нахождение минимума) – по переменной j. Итого,

$T(n) = l \cdot n \cdot n$, где l – количество проходов внешнего цикла While.

Так как $l \le n$ (потому что самый длинный оптимальный маршрут включает в себя прохождения n-1 вершин, плюс делаем еще один проход, чтобы убедиться в стабилизации), то

$2n^2 \le T(n) \le n^3$.

3.3.2. Алгоритм Дейкстры

Описание алгоритма Дейкстры

Ищем расстояние от нулевой вершины

$S = \{0\}$

$D[i] = C(0,i) \qquad i = 0 \dots n$

$While\ S \ne V\ do$

1) выбираем вершину *w*, которая принадлежит множеству вершин *V\S* (*V* без *S*) с минимальной стоимостью *D(w)*;
2) *S := S + w* (добавляем вершину *w* к множеству *S*);
3) для всех верши $v \in V \setminus S$ н *do* $D(v) := min(D(v), D(w) + C(w,v))$ пересчитываем стоимости всех остальных вершин.

Пример. Рассмотрим работу алгоритма на уже знакомом графе:

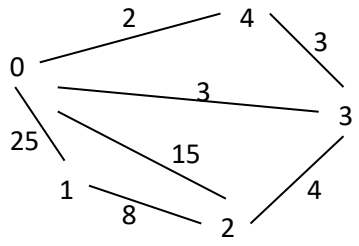


Рисунок 3.7 – Исходный граф

Решение. Составим таблицу:

S	***w***	***D(w)***	***D(1)***	***D(2)***	***D(3)***	***D(4)***
0	Не определенны		25	15	7	2
(0,4)	4	2	25*	15	5	–**
(0,4,3)	3	5	25	9***	–	–
(0,4,3,2)	2	9	15	–	–	–
(0,4,3,2,1)	1	15	–	–	–	–

Пояснения.

* - 25 = min(25, 2 + ∞), где C(4,1) = ∞ - стоимость переходы напрямую от 4 вершины к первой, так как эти вершины не соединены ребром;

** - стоимость маршрута от нулевой до четвертой вершины не пересчитывается, так как четвертая вершина уже вошла во множество S.

Прочерки в остальных ячейках таблицы также означают отсутствие пересчета маршрутов для соответствующих вершин;

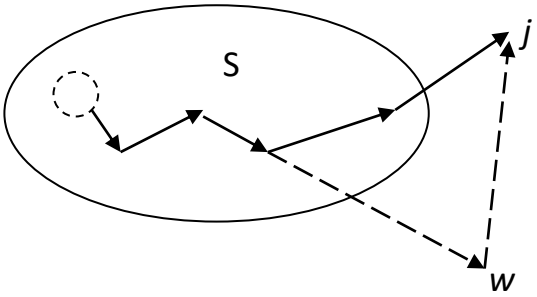
*** - 9 = min(11, 5 + 4), где 4 – вес пути из 3 вершины во вторую.

Комментарии. В алгоритме Дейкстры на каждом шаге

S – растущее множество вершин, для которых уже найдены их подлинные стоимости;

w – добавляемая к множеству S вершина;

D[j] – минимально возможная стоимость маршрута следующего вида:



Все этапы маршрута принадлежат множеству *S*. Только один (последний) переход не лежит в множестве *S*.

Рисунок 3.8.

Замечание. Алгоритм Дейкстры работает корректно, так как на каждом шаге стоимость каждой вершины $j \in (V \setminus S)$ пересчитывается по формуле (*3.2)*, что соответствует выбор из двух маршрутов – старого и нового.

Оценим *трудоемкость* данного алгоритма.

В процессе реализации алгоритма Дейкстры *n* раз выполняется цикл *While*. Внутри цикла *While* имеется цикл *For*, в котором нам приходится пересчитывать *n-1, n-2, n-3, …, 1* значений. Каждое из них – минимум из двух числе, одно из которых – сумма двух слагаемых.

Следовательно, получаем квадратичную трудоемкость:

$$T(n) = ((n-1) + (n-2) + \dots + 1)\cdot 2 \approx n^2$$

То есть алгоритм Дейкстры в целом работает на порядок быстрее, чем алгоритм Форда-Беллмана.

Замечание. Алгоритм Дейкстры может работать только с графами, для которых стоимости переезда $C(i,j) \geq 0$. Алгоритм Форда-Беллмана справляется с графами и с отрицательными стоимостями переезда, но работает дольше.

3.4. Нахождение диаметра, радиуса и центра графа

Определения.

Диаметров графа (взвешенного) называется максимально возможное расстояние между вершинами.

$diamG = \max\limits_{u,v} d(u,v)$ - максимальная стоимость маршрута $u \to v$.

Радиусом графа называется минимально возможный радиус круга (центра круга выбирается среди всех вершин), внутри которого помещается весь граф.

$r(G) = \min\limits_{u} \max\limits_{v} d(u,v)$,

где $r(u) = \max\limits_{v} d(u,v)$ - минимально возможный радиус круга (в который входит весь граф) с центром в вершине u.

Центр графа - та вершина u_0, для которой достигается минимально возможный радиус круга: u_0: $r(u_0) = r(G)$.

При применении алгоритма Дейкстры и диаметр, и радиус графа можно найти за $T = n\cdot n^2$ операция, где n^2 – трудоемкость однократного применения алгоритма Дейкстры.

Алгоритм Дейкстры мы запускаем n раз: с начала от v_0, потом от v_1 и т.д. перебирая все возможные вершины u. Получаем квадратную матрицу $d(u,v) = n \times n$, по которой мы за n^2 операций находим диаметр и радиус графа.

3.5. Задача об изоморфизме графов

Определение. Два взвешенных графа G_1 и G_2 называются *изоморфными*, если существует взаимооднозначное отображение вершин, сохраняющих расстояние, т.е. после перенумерации вершин графа G_1 получаем ровно граф G_2.

Оценим трудоемкость этой задачи методом прямого перебора.

На выходе имеем два графа из n вершин.

Существует всего $n!$ различных перестановок (перенумераций) для n-элементного массива и при каждой перенумерации необходимо проверить n^2 – попарных расстояний. Таким образом, трудоемкость этих операций составит $T(n) = n!\cdot n^2$.

Нетрудно заметить, что этот алгоритм не полиномиальный и его трудоемкость велика. На данный момент не известны алгоритмы решения данной задачи для графов произвольного вида за полиномиальное время, но и не доказано, что такого алгоритма не существует.

В тоже время для графов некоторых специальных видов (для деревьев) данные алгоритмы существуют.

3.6. Задача коммивояжера. Её решение методом ветвей и границ.

Задача. *Имеется взвешенный неориентированный связный граф. Необходимо найти гамильтонов цикл наименьшей длины, т.е. нужно обойти все вершины графа, побывав в каждой из них по одному разу, затратив как можно меньше денег.*

Оценим трудоемкость методом прямого перебора.

Имеем $n!$ всевозможных маршрутов. Стоимость каждого маршрута – сумма n ребер. Следовательно,

$$T(n) = n!\cdot n$$

$n!$ можно заменить $(n-1)!$, т.к. маршрут проходит через все вершины, и поэтому в качестве стартовой мы можем взять вершину v_i, располагая оставшиеся вершины $(n-1)$ произвольным образом.

$T(n) = (n-1)!\cdot n = n!$ – неполиномиальная.

Для более быстрого решения нашей задачи существует метод ветвей и границ. На самом деле это целая группа методов, они используются для решения множества задач. Их объединяет общая идея,

но в каждом случае реализация метода ветвей и границ своя. Впервые этот мтеод был придуман для решения задачи коммивояжера.

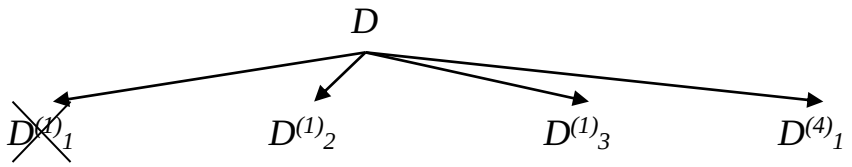
Идея метода ветвей и границ. Пусть нам необходимо найти минимум некоторой функции $f(x)$, $x \in D$. Предположим, что у нас есть:

а) Алгоритм дробления множества D на всё уменьшающиеся части (вплоть до одноэлементных множеств).

б) Для каждой части D_i, полученной в результате дробления, имеется некоторая оценка $H(D_i) \le f(x)$, $\forall x \in D_i$. Эта величина оценивает значение функции f на интервале D_i снизу, причем на одноэлементных множествах эта оценка совпадает со значением функции f.

Тогда для нахождения минимума поступаем следующим образом.

Возьмем точку x из множества D (желательно, что бы $f(x)$ была как можно меньше). Объявим f(x) рекордом r. Делим множество D на части:



$H(D^{(1)}_1) < r$ $\quad H(D^{(1)}_2) > r$ $\quad H(D^{(1)}_3) = r$ $\quad H(D^{(1)}_4) < r,$

где f – функция-оценка минимума.

Для множества $D^{(1)}_2$ справедливо:

$H(D^{(1)}_2) > r$, тогда для всех $y \in D^{(1)}_2$ выполняется $f(y) \ge H(D^{(1)}_2) > r$, следовательно, на этом множестве мы минимума не достигнем, поэтому данное множество отбрасываем. Таким образом, ветвь мы отрубаем тогда и только тогда, когда функция-оценка минимума на этом множестве больше либо равна рекорда. Далее работаем с множествами $D^{(1)}_1$ и $D^{(1)}_4$, и , не смотря на то, что множество $D^{(1)}_1$ кажется более перспективным, может случится так, что реальное значение минимума будет достигнуто на $D^{(1)}_4$ (так как мы не требовали, чтобы выполнялось $H(D^{(i)}_k = min\ f(x)$, а $H(D^{(i)}_k$ является лишь некоторой оценкой этого минимума снизу).

$$H(D^{(i)}_k) \le min\ f(x),\ x \in D^{(i)}_k.$$

Множество $D^{(1)}_3$ мы можем в процессе работы не рассматривать, если нам достаточно найти хотя бы одну точку, в которой достигается минимум, если же нам наобходимо найти все точки, то множество $D^{(1)}_3$ считаем перспективным.

Замечание. Подобное дробление множества D продолжаем до тех пор, пока не доберемся до одноэлементных множеств (нижнего этажа дерева).

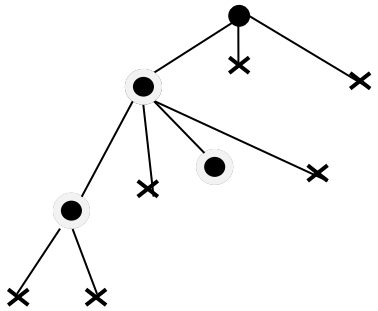


Рисунок 3.9. – Дробление множества, где

● перспективное множество

× неперспективное множество

При этом возможны два вариант дробления.

Схема одновременного ветвления.

На каждом шаге мы работает со всеми перспективными множествами:

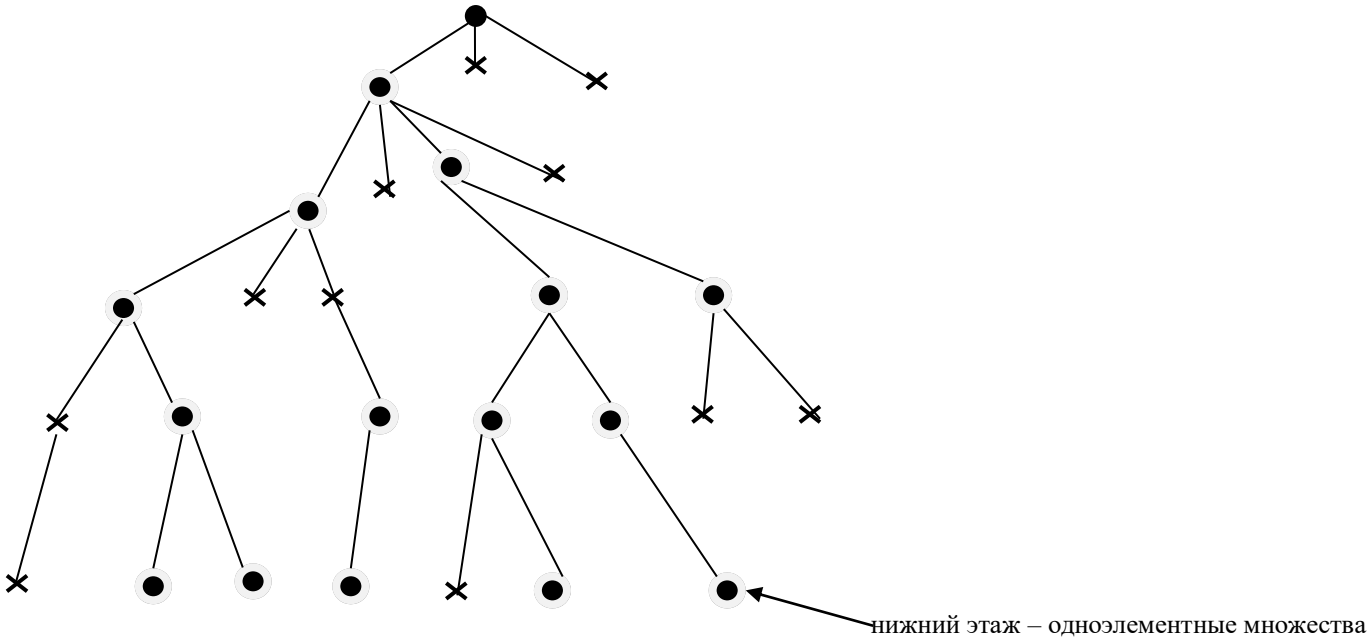


Рисунок 3.10 – Одновременное дробление

Схема одностороннего ветвления.

Отличие этой схемы от предыдущей заключается в том, что на каждом шаге мы работает только с одним перспективным множеством, а не со всеми сразу, как в схеме одновременного ветвления (см.рис.3.11).

Здесь

◯ перспективное множество;

⊗ неперспективное множество;

? отложенное перспективное множество;

ветвь, ставшая неперспективно после обновления рекорда.

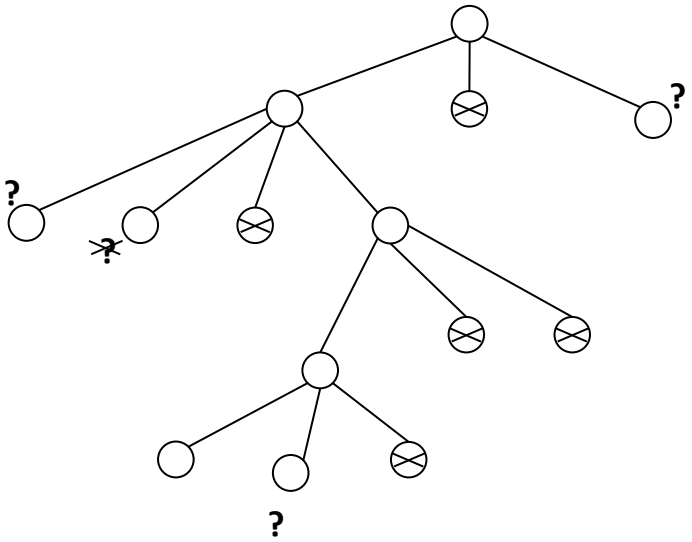


Рисунок 3.11 – Одностороннее дробление

По ходу дробления на каждом шаге из всех множеств выбирается самое перспективное – то, на котором меньшего всего оценка *f*. С ним и будем работать дальше. Рассмотрение остальных перспективных множеств пока отложим. Эта схема предпочтительнее, т.к. при ее применении мы быстрее доберемся до нижнего этажа, где сможем обновить рекорд, а после его обновления многие множества, ранее перспективные, при новом рекорде могут перейти в разряд неперспективных, и их мы рассматривать вообще не будем. Таким образом трудоемкость схемы одностороннего ветвления существенно меньше схемы одновременного ветвлении.

Алгоритм одностороннего ветвления

На каждом шаге оставляем в работе только одно, самое перспективное множество. Таким образом доходим до нижнего этажа дерева, при этом все находящиеся на нем множества станут одноэлементными. Обновляем рекорд. Выкидываем уже неперспективные отложенные ветви. Среди оставшихся перспективных ветвей выбираем самую перспективную (либо оценке *f*, либо по уровню, на котором расположена в дереве – чем ближе к нижнему этажу, тем перспективнее отложенная ветвь).

Подобную процедуру подъема и спуска продолжаем до тех пор, пока не останется ни одной перспективной ветви.

Применение метода ветвей и границ для решения задачи коммивояжера.

В нашем случае исходное множество D – множество всевозможных маршрутов, проходящих по всем вершинам графа (так называемых гамильтоновых циклов), для определенности стартующих из точки 0. Напомним, что Гамильтонов цикл должен содержать все вершины графа, причем. Переходя по его ребрам, можно обойти все вершины, побывав в каждой только по одному разу.

В предложенной схеме дробления буду возникать подмножества D следующего вида:

$$D_{(j)}^{(i)} = [(v_1, v_2, \dots, v_k), \{u_1, \dots, u_l\}].$$

Множество $D_{(j)}^{(i)}$ состоит из всевозможных маршрутов, у которых $v_1, v_2, \dots, v_k$ – первоначальный участок, а следующий переход из вершины v_k мы можем совершить в любую вершину, кроме указанной в фигурных скобках вершин $u_1, \dots, u_l$. Также мы не можем идти в вершины $v_1, v_2, \dots, v_k$, так как тогда маршрут будет уже не гамильтонов.

Составим оценку *f* для множества $D_{(j)}^{(i)}$ следующего вида:

$$f\left(D_{(j)}^{(i)}\right) = C(v_1, v_2) + \dots + C(v_{k-1}, v_k) + \sum_{u \neq v_1, v_2, \dots, v_{k-1}} \alpha(u) + \sum_{w \neq w_2, w_3, \dots, w_k} \beta(w)$$

где

$\alpha(u)$ – стоимость выхода (выезда) из вершины u (т.е. минимально возможная цена, за которую мы можем выехать в какую-либо другую допустимую вершину),

$\beta(w)$ – минимально возможная стоимость въезда в вершину w после уже уплаченных стоимостей выездов.

Пример. Пусть у нас имеется гамильтонов цикл в ориентированном графе (для орграфа матрица стоимости ребер не симметрична относительна своей главной диагонали). По главной диагонали расположены бесконечности, а не нули, потому что в гамильтонов цикле петли запрещены.

	0	1	2	3	4	5
0	∞	3	6	7	∞	4
1	2	∞	7	4	3	2
2	8	7	∞	5	4	3
3	2	8	4	∞	6	∞
4	∞	2	4	8	∞	7
5	3	5	6	4	1	∞

Имеем первоначальный маршрут: *D = [(2, 5, 0), {1}].*

Таким образом, из вершины 0 мы можем идти в любую, кроме 1 (так как она запрещенная), 2 и 5 (так как уже прошли).

Вычисляем оценку *H*. Для этого:

1. Вычеркиваем из имеющейся матрицы 2 и 5 строки, так как стоимости ребер, по которым можно попасть во 2 и 5 вершины, нам не понадобятся, в эти вершины мы уже въезжали и больше туда не собираемся.

2. Вычеркиваем 5 и 0 столбцы, так как уже въезжали в вершины 5 и 0.

	~~0~~	1	2	3	4	~~5~~	
0	∞	3	6	7	∞		4
1	2	∞	7	4	3		2
~~2~~	~~8~~	~~7~~	~~∞~~	~~5~~	~~4~~		~~3~~
3	2	8	4	∞	6		∞
4	∞	2	4	8	∞		7
~~5~~	~~3~~	~~5~~	~~6~~	~~4~~	~~1~~		~~∞~~

3. Получилась матрица:

	1	2	3	4
0	3	6	7	∞
1	∞	7	4	3
3	8	4	∞	6
4	2	4	8	∞

Стоимость переезда из 0 в 1полагаем равно бесконечности, так как он запрещен. В каждой строке находим минимальных элемент (минимальную стоимость выезда из вершин 0, 1, 3 и 4, α), т.е. находим α:

	1	2	3	4	α
0	3	6	7	∞	6
1	∞	7	4	3	3
3	8	4	∞	6	4
4	2	4	8	∞	2

«Уплачиваем» найденные стоимости выездов, т.к. вычитаем из каждого элемента строки минимальный элемент данной строки. Стоимости выездов «уплачены», однако мы еще должна заехать в вершины 1, 2, 3, 4. Находим для них оценку β.

	1	2	3	4
0	3	6	7	∞
1	∞	7	4	3
3	8	4	∞	6
4	2	4	8	∞
β	0	0	1	0

Итак, оценка

$H(D) = \underbrace{C(2,5) + C(5,0)}_{\text{из первоначальной матрицы}} + (\alpha(0) + \alpha(1) + \alpha(3) + \alpha(4)) + (\beta(0) + \beta(2) + \beta(3) + \beta(4)) =$

$= 3 + 3 + (6 + 3 + 4 + 2) + (0 + 0 + 1 + 0) = 22.$

То есть, проехав по любому маршруту из множества D [(2, 5, 0), {1}], мы не можем потратить менее 22 долларов. Осталось записать схему возможных вариантов дробления маршрутов B. Рассмотрим два варианта схемы на примере пятиэлементного множества:

Первый вариант

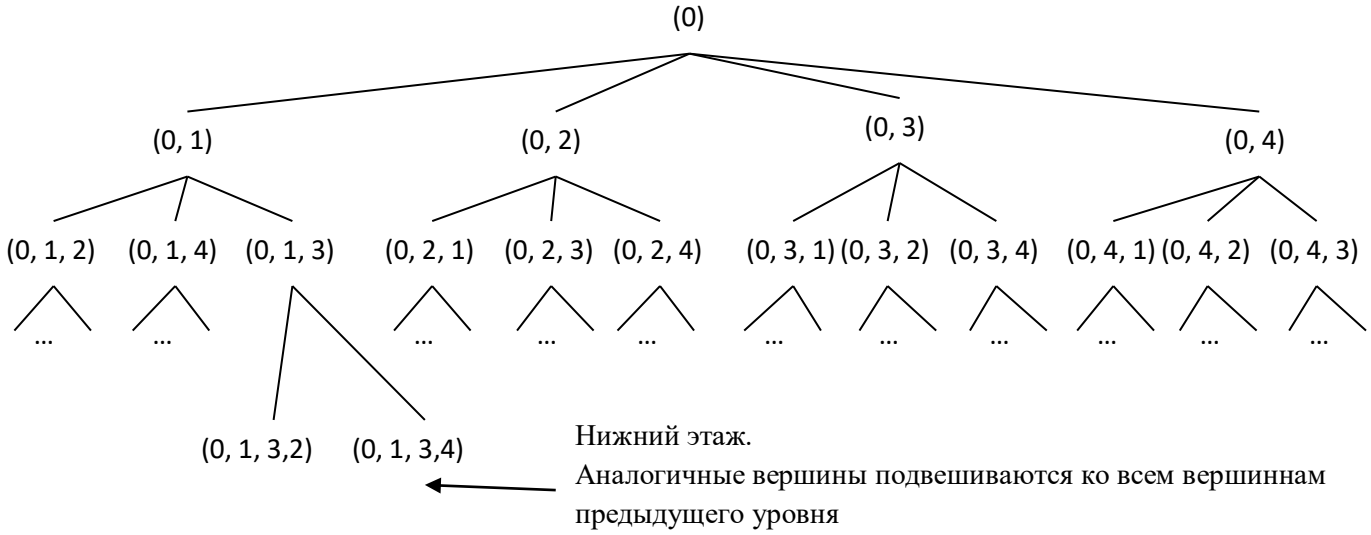


Рисунок 3.12

Тут не возникает запретных вершин.

Второй вариант

При этой схеме дробления на каждом этапе ветка делится на две новые ветви, но также возникают и запретные вершины.

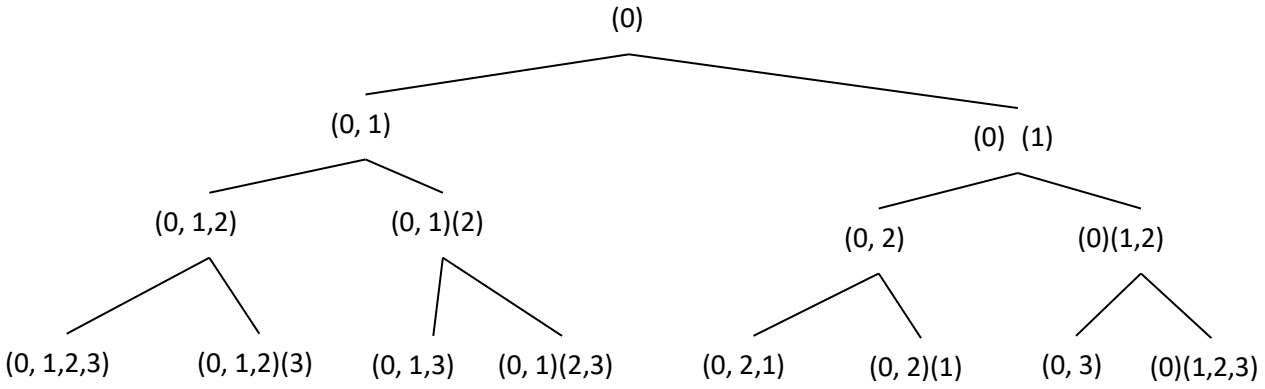


Рисунок 3.13

и т.д.

<u>Замечание.</u> Обычно метод ветвей и границ позволяет существенно уменьшить объем перебора. Однако это справедливо не для каждой задачи, например для задачи коммивояжера до сих пор неизвестен алгоритм, который гарантированно бы решал ее за полиномиальное время. Впрочем. Для большинства графов он дает существенный выигрыш по сравнению с прямым перебором.

Домашнее задание №5 (А, Б, В)

А) Найти остов минимального веса для связанного взвешенного неориентированного графа, заданного матрицей стоимостей переездов из одной вершины в другую:

0	2	7	4	6	3
2	0	4	5	6	8
7	4	0	8	7	∞
4	5	8	0	5	7
6	6	7	5	0	3
3	8	∞	7	3	0

Б) Найти кратчайшее расстояние от третьей до всех остальных вершин графа, заданного матрицей стоимостей переездов из одной вершины в другую

Б1) с помощью алгоритма Форд-Беллмана

Б2) с помощью алгоритма Дейкстры

Матрица стоимостей:

0	2	7	4	6	3
3	0	4	5	6	1
2	4	0	8	7	∞
4	∞	8	0	5	7
∞	7	8	4	0	3
2	4	∞	7	8	0

В) Определить оценку H, если первоначальный маршрут (0,3) {1,5}.

Матрица стоимостей переездов:

0	2	7	4	6	3
3	0	4	5	6	1
2	4	0	8	7	∞
4	∞	8	0	5	7
∞	7	8	4	0	3
2	4	∞	7	8	0

4. ЗАДАЧИ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

4.1 Задачи динамического программирования. Её решение методом динамического программирования.

Определение. *Задача динамического программирования (ДП)* – это задача оптимального управления некоторым многошаговым процессом.

Подобных задач, ровно как и методов их решения, существует великое множество. Изобретаются они в каждом конкретном случае индивидуально, но все объединены общей идеей решения – так называемого метода решения задачи через чайник: что бы вскипятить воду, нужно налить её в чайник, поставить его на плиту, включить плиту и т.д. Если же вода в чайнике уже налита, то нужно её вылить. А что делать дальше, мы уже знаем. Таким образом, в процессе решения мы сводим задачу к той, решать которую уже научились на предыдущем шаге.

Рассуждения при этом приводятся примерно следующие.

Нам необходимо найти оптимальное решение в конце маршрута. Если бы мы знали оптимальное решение для всех предыдущих этапов, то нашли бы решение для последнего. Чтобы найти решение для предпоследнего этапа, мы должны знать решение для второго с конца этапа, и т.д. То есть, разрабатывается метод динамического программирования с конца. Реализуется же он с начала: высчитываем оптимальные значение с самого начала и находим оптимальную стоимость. После этого необходимо найти собственно оптимальный маршрут. Его находим с конца.

Задача. *Рассмотрим задачу оптимального управления многошаговым процессом. Над ребрами данного графа проставлены стоимости переходов из одной вершины в другую. Необходимо найти путь по которому с минимальными затратами можно попасть из* $S^{(0)}$ *в* $S^{(5)}$ *(см. рисунок 4.1).*

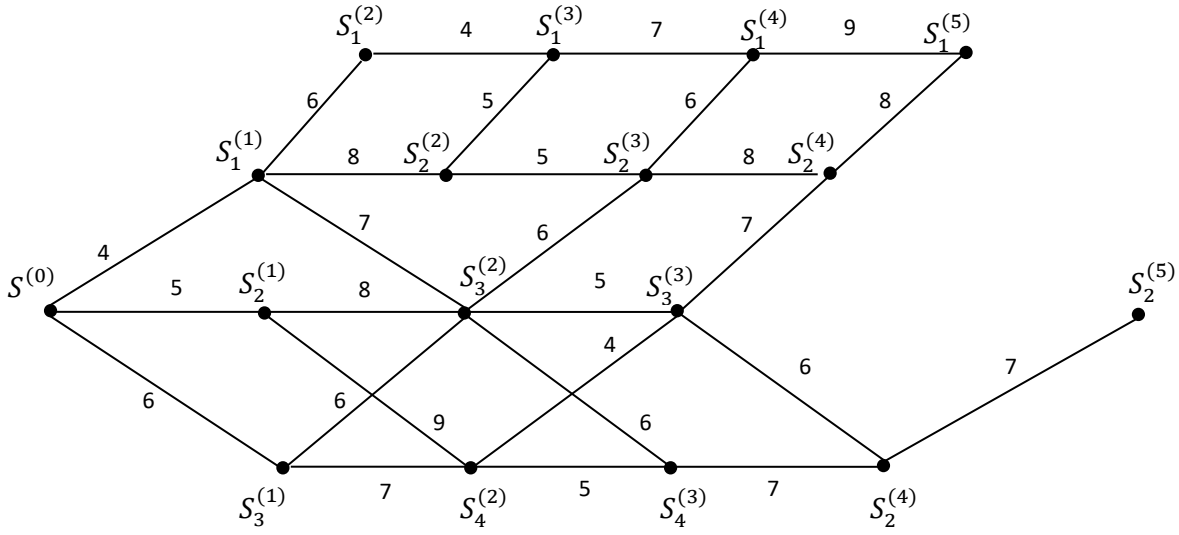


Рисунок 4.1. – исходный граф.

Идея решения. В принципе мы умеем решать подобные задачи по алгоритму Дейкстры и Форда-Беллмана. Попробуем теперь решить эту задачу методом динамического программирования. Он изобретается с конца. Нам необходимо найти минимально возможную сумму, имея которую, мы можем добраться до $S^{(5)}$.

Рассуждаем так: если бы мы знали «стоимости» всех вершин. Из которых мы можем попасть в $S^{(5)}$ (то есть вершин $S_1^{(4)}, S_2^{(4)}, S_3^{(4)}$), то мы бы нашли стоимости всех вершин $S^{(5)}$ (для этого к стоимости вершин из $S^{(4)}$ прибавляем стоимости переездов и выбираем минимум из получившихся сумм). Чтобы найти стоимости $S^{(4)}$ мы должны знать стоимости $S^{(3)}$ и т.д. Так спускаемся до вершины $S^{(0)}$, стоимость которой равна нулю.

Итак, $f(S_j^i) = min_k(f(S_k^{(i-1)} + C(S_k^{(i-1)}, S_j^{(i)})$

Здесь $f(S_j^i)$ – минимально возможная сумма денег, имея которую мы можем добрать от $S^{(0)}$ до $S_j^{(i)}$.

Имеем:

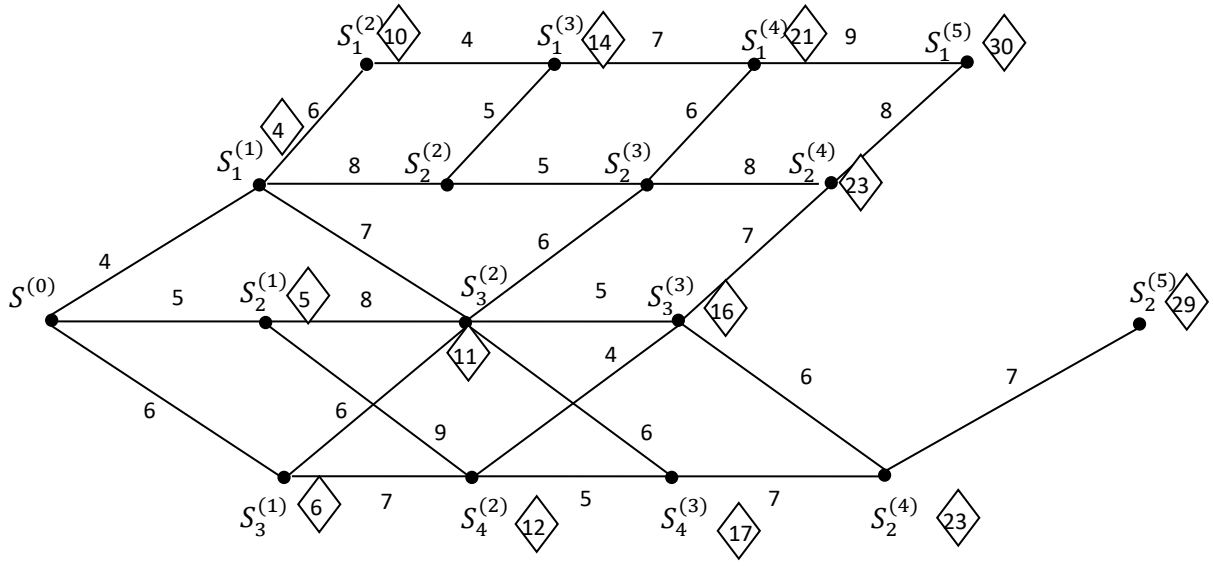


Рисунок 4.2. – Здесь 10 – полученный стоимости вершин.

Реализация метода ДП происходим от начала к концу (то есть слева направо в нашем случае). Самый внешний цикл – по *i*; в нём в прямом порядке перебираем уровни вершин. Следующий по вложенности цикл – по *j*; в нём перебираем вершины одного уровня. Самый внутренний цикл – по *k*. Изменение направления прохода двух вложенных циклов не повлияет на конечный результат.

Последний этап – восстановление оптимального пути – реализуется из конца в начало. Для этого смотрим, из какой вершины предыдущего уровня была достигнута стоимость нашей вершины, постепенно продвигаясь справа налево.

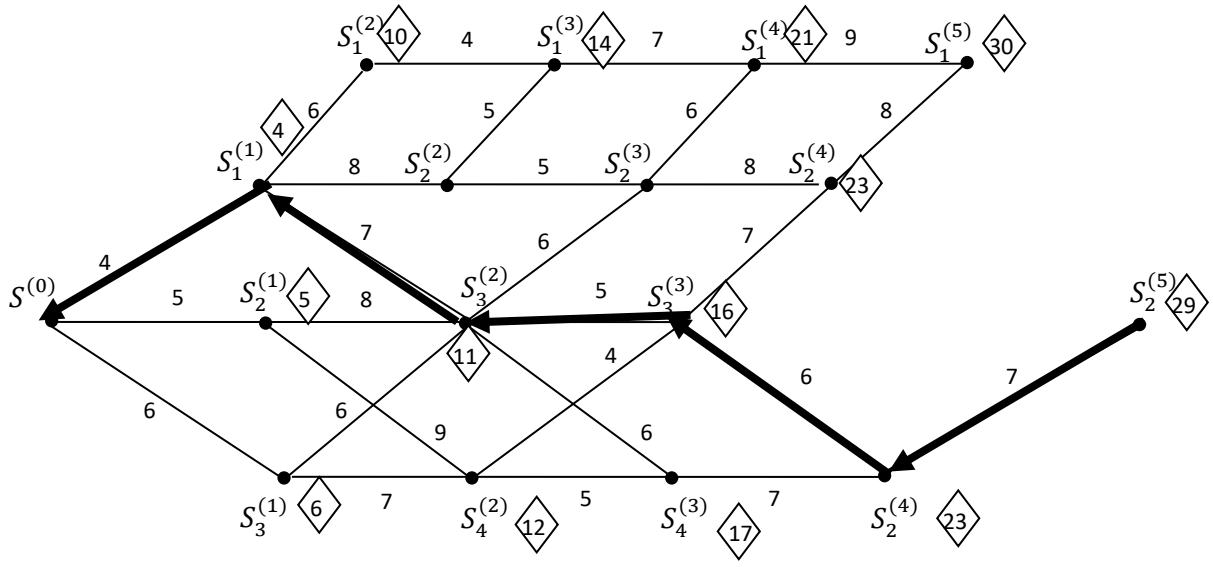


Рисунок 4.3. – Восстановление оптимально пути.

4.2. Задача об оптимальном наборе самолетом скорости и высоты

Задача. *Необходимо, имея стартовую нулевую скорость v и стартовую нулевую высоту h, набрать некоторую скорость V и высоту H, минимизировав при этом суммарные затраты топлива. Если в какой-то момент времени t мы увеличиваем скорость на dv, а высоту на dh, то затраты горючего на это изменение могут быть определены по формуле:*

$$C_v(v(t),h(t))dv + C_h(v(t),h(t))dh,$$

где v(t) и h(t) – скорость и высота в момент времени t;

$C_v(v(t),h(t))$ и $C_h(v(t),h(t))$ – коэффициенты пропорциональности затрат топлива.

Идея решения. Нам необходимо минимизировать интеграл, выбирая оптимальный вариант набора сокрости и высоты (ищем оптимальное управление):

$$\int_{(0,0)}^{(v,h)} Cv\big(v(t),h(t)\big)dv + Ch\big(v(t),h(t)\big)dh.$$

Дискретизируем эту задачу. Для этого разобьем весь участок приращения скорости и высоты на несколько меньших интервалов:

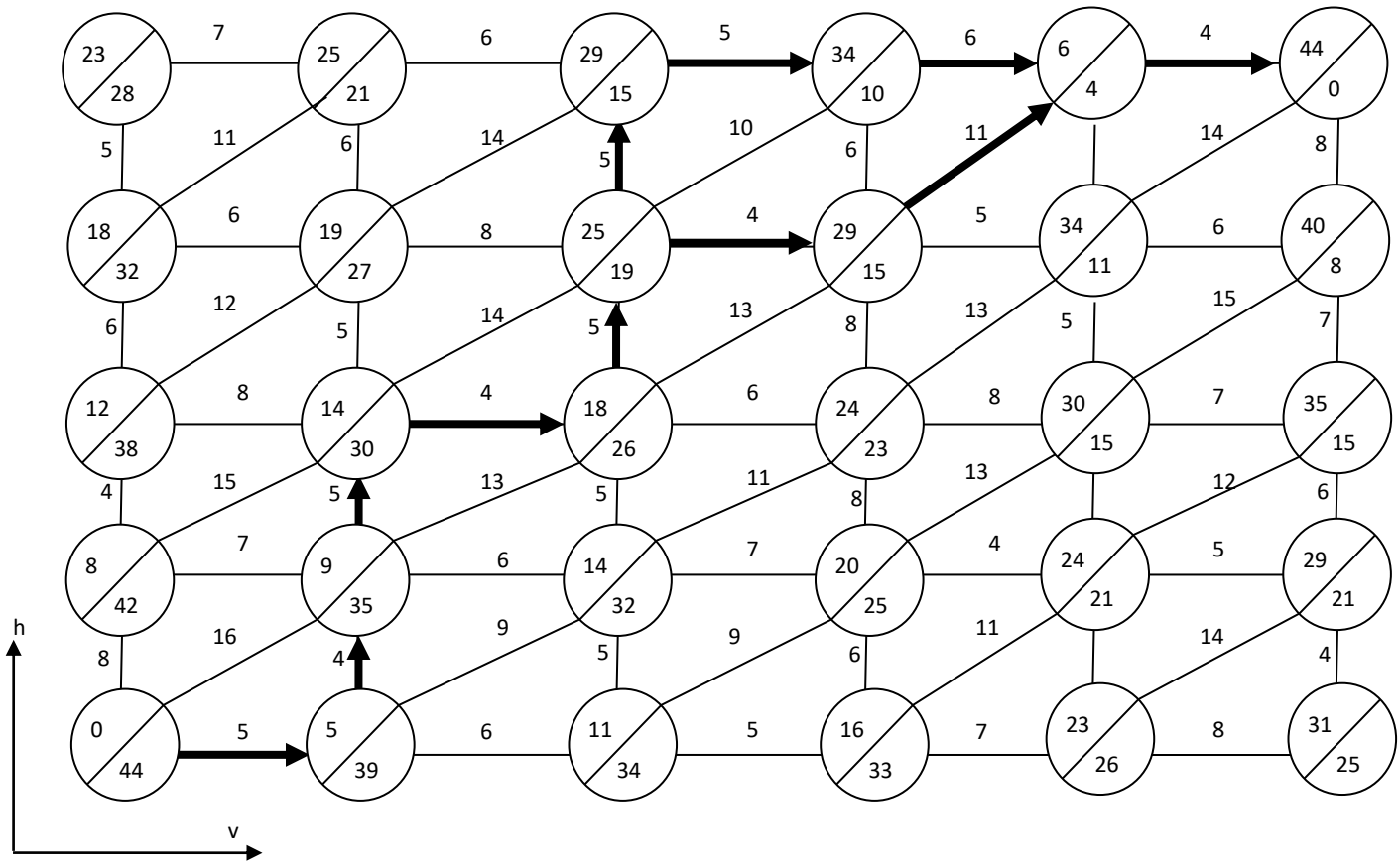


Рисунок 4.4. – Дискретизация исходной задачи.

Заполняем стоимости, начиная с левого нижнего угла, и записываем их в левый сектор круга. Аналогично считаем стоимости вершин с правого верхнего угла и записываем их в правый сектор. Глядя на полученный стоимости, восстанавливаем оптимальный путь: 87 получили из 79 + 8; 79 из 70 + 9 и т.д.

Комментарии

- В принципе, при восстановлении оптимального пути, возможно ветвление маршрута (когда минимум получен на пути не из одной, а из большего количества вершин).
- Все вершины графа разбиваются на группы состояний по диагоналям. Но в группу $S^{(i)}$ мы попадем не только из $S^{(i-1)}$, но и из $S^{(i-2)}$.
- При проходе слева направо и справа налево, как и следовало ожидать, стоимость пути одинакова и равна 87. В каждом кружке сумма чисел больше либо равна 87. Причем сумма равна 87 в кружках, лежащих на оптимальном пути, а для остальных превышает 87. В каждом кружке – левое число – стоимость маршрута из левого нижнего угла до данного кружка. Правое число – стоимость маршрута из правого верхнего угла до данного кружка.

4.3. Задача грабителя (задача о рюкзаке)

Задача. *Имеется склад, на котором есть некоторый ассортимент товаров. Запас каждого товара считается неограниченным. Товары имеют две характеристики:* m_i *– масса,* c_i *– стоимость;* $i = \overline{1,n}$.

Необходимо выбрать набор товаров так, что бы его суммарная масса не превосходила заранее фиксированную массу M(т.е. $\Sigma m_i \leq M$*), и стоимость набора была как можно больше (*$\Sigma c_i \rightarrow max$*).*

Идея решения. Считаем, что все массы m_i целочисленные. Решим эту задачу методом динамического программирования. Изобретаем метод, т.е. формулу.

Нам необходимо найти *f(M)*, т.е. максимально возможную сумму m_i при заданном *M*. Предположим, что к этому моменту мы знаем, как решать эту задачу для всех меньших значений грузоподъемности. Тогда смотрим, какой товар мы положили в рюкзак предпоследним. Если это был первый товар стоимость c_1, то мы должны оптимизировать стоимость рюкзака при грузоподъемности $M - m_1$. Если

это был второй товар стоимость c_2, то оптимизация при грузоподъемности $M - m_2$, и т.д. Среди этих величин выбираем максимальную.

Таким образом, получаем формулу: $f(M) = \max\limits_{1 \le i \le n} (f(M - m_i) + C_i)$.

Пример. Рассмотрим решение этой задача при следующем наборе товаров:

m:	3	5	8	– массы товаров,
c:	8	14	23	– стоимости товаров.

Решение.

Вычислим последовательно:

f(0) = 0; f(1) = 0; f(2) = 0; f(3) = 8; f(4) = 8;

f(5) = 14 = max(f(5-3)+8; f(5-5)+14; f(5-8)+23); - не рассматриваем при поиске максимума, так как *f(-3)* не определена.

f(6) = 16; f(7) = 16; f(8) = 23; f(9) = 24 = max(f(9-3)+8; f(3-5)+14; f(9-8)+23);

f(10) = 28; f(11) = 31; f(12)= 32; f(13) = 37 = max(f(13-3)+8; f(13-5)+14; f(13-8)+23).

Оценим трудоемкость решения задачи о рюкзаке методом ДП.

Для применения метода ДП все массы должны быть целочисленные (а стоимости – необязательно). Тогда если k – количество видов товаров, m – грузоподъемность, то имеем m шагов внешнего цикла (по грузоподъемности) и на каждом шаге находим максимум среди k чисел, каждое из которых является суммой двух слагаемых. В итоге получаем трудоемкость: $T = m \cdot k$.

4.4. Задача о перемножении матриц

Как известно из высшей математики, умножение матриц ассоциативно, то есть результат перемножения зависит только от порядка матриц и не зависит от расстановки скобок:

$$(A*B)*C = A*(B*C).$$

Результат перемножения от расстановки скобок не зависит, зато трудоемкость этого перемножения при разных расстановках скобок может отличаться существенно.

Оценим трудоемкость умножения двух матриц $A(p \times q)$ и $B(q \times r)$:

$$C(p \times r) = A(p \times q) * B(q \times r),$$

каждый элемент матрицу $C(p \times r)$ есть сумма q попарных произведений.

Трудоемкость перемножения двух матриц: $T = p \cdot q \cdot r$.

Пример. Рассмотрим пример перемножения матриц разными способами:

Пусть имеется четыре матрицы разных размерностей:

$M_1[10 \times 20]$, $M_2[20 \times 50]$, $M_3[50 \times 1]$, $M_4[1 \times 100]$.

Решение.

Рассмотрим умножение матриц в следующей порядке:

$M_1 \cdot (M_2 \cdot \underbrace{(M_3 \cdot M_4)})$

$\underbrace{[50 \times 1] \cdot [1 \times 100]}_{} = [50 \times 100]$ трудоём. $50 \cdot 1 \cdot 100 = 5000$

$\underbrace{[20 \times 50] \cdot [50 \times 100]}_{} = [20 \times 100]$ трудоём. $20 \cdot 100 \cdot 50 = 100\,000$

$\underbrace{[10 \times 20] \cdot [20 \times 100]}_{} = [10 \times 100]$ трудоём. $10 \cdot 20 \cdot 100 = 20\,000$

} 125 000 операций

Рассмотрим другой вариант умножения матриц:

$(\underbrace{M_1 \cdot (M_2 \cdot M_3)}) \cdot M)$

$\underbrace{[20 \times 50] \cdot [50 \times 1]}_{} = [20 \times 1]$ трудоём. $20 \cdot 50 \cdot 1 = 1000$

$\underbrace{[10 \times 20] \cdot [20 \times 1]}_{} = [10 \times 1]$ трудоём. $10 \cdot 20 \cdot 1 = 200$

$\underbrace{[10 \times 1] \cdot [1 \times 100]}_{} = [10 \times 100]$ трудоём. $10 \cdot 1 \cdot 100 = 1000$

} 2200 операций

При умножении вторым способом действий потребовалось значительно меньше (2200 против 125 000). Придумаем формулу метода ДП применительно к данной задачу.

Допустим, нам необходимо оптимальным образом перемножить 5 матриц. i-тая матрица имеет размерность $[r_{i-1} \times r_i]$. Смотрим, где стояла последняя пара скобок: существует 4 варианта:

$$(M_1)\cdot(M_2\cdot M_3\cdot M_4\cdot M_5) \quad (1)$$
$$[r_0 \times r_1] \quad [r_1 \times r_5]$$

$$(M_1\cdot M_2)\cdot(M_3\cdot M_4\cdot M_5) \quad (2)$$
$$[r_0 \times r_2] \quad [r_2 \times r_5]$$

$$(M_1\cdot M_2\cdot M_3)\cdot(M_4\cdot M_5) \quad (3)$$
$$[r_0 \times r_3] \quad [r_3 \times r_5]$$

$$(M_1\cdot M_2\cdot M_3\cdot M_4)\cdot(M_5) \quad (4)$$
$$[r_0 \times r_4] \quad [r_4 \times r_5]$$

При первом варианте расстановок скобок нам потребуется
$f(1,1) + f(2,5) + r_0r_1r_5$ операций.
Здесь $f(k,l)$ – минимальное количество действий, за которое можно вычислить произведение матриц с k-той по l-тую.
Для последующих трех вариантов:

$$f(1,2) + f(3,5) + r_0r_2r_5$$
$$f(1,3) + f(4,5) + r_0r_3r_5$$
$$f(1,4) + f(5,5) + r_0r_4r_5$$

операций соответственно.
Здесь $f(k,p)$ – минимальное количество действий, за которое можно вычислить произведение матриц с k-той по p-тую.
Выбираем минимум среди всех этих чисел и получаем общую формулу:

$$f(k,p) = \min_{k\le j\le p-1}\left(f(k,j) + f(j+1,p) + r_{k-1}\cdot r_j \cdot r_p\right).$$

<u>Замечание</u>. В отличии от задачи о рюкзаке, где была динамика по одному параметру (грузоподъемности), здесь динамика по двум параметрам: начало и конец блока перемножаемых матриц.
Итак, в окончательном виде эта задача решается с помощью следующего алгоритма.

1) Заполняем трудоемкости матриц: Трудоемкости по главной диагонали равны 0:
   for i:=0 to n do f(i,i):=0;
2) Внешний цикл по t – длине перемножаемого блока;
   Средний цикл по k – местоположению блока;
   Внутренний – поиски минимума по j.
   for t:=1 to n-1 do
   for k:=1 to n-1 do

$$f(k,k+t) = \min_{k\le j\le k+t-1}\left(f(k,j) + f(j+1,k+t) + r_{k-1}\cdot r_j \cdot r_{k+t}\right).$$

Для матриц M_1, M_2, M_3, M_4 из рассмотренного выше примера расставим скобки оптимальным образом.

f(1,1)	*f(1,2)*	*f(1,3)*	*f(1,4)*
	f(2,2)	*f(2,3)*	*f(2,4)*
		f(3,3)	*f(3,4)*
			f(4,4)

Итак, заполняем такую матрицу в следующем порядке:
f(1,1) = f(2,2) = f(3,3) = f(4,4) = 0
f(1,2) = min (f(1,1) + f(2,2) + 10·20·50) = 10 000
f(2,3) = min (f(2,2) + f(3,3,) + 20·50·1) = 1000
f(3,4) = min (f(3,3) + f(4,4) + 50·1·100) = 5000
f(1,3) = min (f(1,1) + f(2,3) + 10·20·1; f(1,2) + f(3,3) + 10·50·1) = 1200

f(2,4) = min (f(2,2) + f (3,4) +20·50·100; f(2,3) + f (4,4) + 20·1·100) = 3000
f(1,4) = min (f(1,1) + f(2,4) + 10·20·100; f(1,2) + f(3,4) + 10·50·100; f(1,3) + f(4,4) + 10·1·100) = 2200
После вычисления оптимальных трудоемкостей восстанавливаем оптимальную расстановку скобок.
Смотрим, где достигнут минимум в *f(1,4)*. Он достигнут в *f(1,3) + f(4,4) + 10·1·100.* Следовательно, последними скобками будут *(M_1 M_2 M_3)(M_4).*
Далее смотрим, где достигнут минимум в *f(1,3)*. Он достигнут в *f(1,1) + f(2,3) + 10·20·1*. Т.е. следующими скобками будут *(M_1 (M_2 M_3) M_4.*
Т.к. у нас 3 вложенных цикла, длина каждого порядка *n* (*n* – количество перемножаемых матриц), то трудоемкость решаемой задачи методом ДП $T = C \cdot n^3$.
При решении этой задачи методом прямого перебора получаем не полиноминальную трудоемкость, а намного большую. То же самое получаем и для задачи о рюкзаке: её решение методом динамического программирования имеет полиномиальную трудоемкость (2 вложенных цикла), а методом прямого перебора получаем неполиномиальную трудоемкость.

Домашнее задание №6 (А, Б)
А) Задача грабителя (о рюкзаке). Имеет склад, на котором присутствует ассортимент товаров (каждого товара неограниченный запас). У каждого товара своя стоимость C_i и масса m_i. Выбрать набор товаров так, чтобы его суммарных вес не превышал заданную грузоподъемность M притом, что суммарная стоимость этого набора товаров была бы максимальной.

Номер товара, i	m_i	C_i
1	5	9
2	7	13
3	11	21

Максимальная грузоподъемность: M = 23; 24.
Б) Расставить скобки при перемножении матриц оптимальным образом.
M_1=[10×20], M_2=[20×5], M_3=[5×4], M_4=[4×30], M_5=[30×6].

5 КЛАССЫ P И NP

5.1 Машина Тьюринга

Определение. Многоленточной машиной Тьбринга (МТ) называется свертка: **Q, T, I, *δ*, q_0, q_{end}**. В этом наборе:

Q –множество состояний;

T – множество символов на лентах (алфавит);

I – множество входных символов, I⊆T (входной алфавит);

δ – функция перевода.

$\delta(q,a_1,a_2,\ldots,a_k) =(q',(a_1',m_1),(a_2',m_2),\ldots,(a_k',m_k))$

$a_1,a_2,\ldots,a_k$ - обозреваемые символы на лентах

$a_1',a_2',\ldots,a_k'$ – новые символы которые записывают вместо $a_1,a_2,\ldots,a_k$

$m_1,m_2,\ldots,m_k$ – команды перехода к соседней клетке, т. е. либо L (сдвиг в лево) , либо R (сдвиг вправо), либо S (остановка на месте).

Все символы на лентах – буквы некоторого заранее зафиксированного алфавита

b – специальный выделенный пустой символ (служит для разделения слов)

q_0 – начальное состояние

q_{end} – конечное состояние машины, попав в которое она останавливается.

С помощью МТ можно реализовать любые програмы написаные на языке высокого уровня.

Входное слово *w допускается* машиной Тьюринга (ТМ) М, если машина Тьюринга, начав работу из состояния q_0 имея слово *w* ? записанное на нашей ленте, рано или поздно закончит работу, т.е . попадет в состояние $q_{end.}$ При этом обычно в ячейках ленты записывается в закодированном виде ответ (см. пример выше).

Определение. Мгновенное описание МТ М – полная информация о МТ в некоторый момент времени t, оно состоит из :

1) текущего состояния q_1 ;
2) содержимого всех лент;
3) номеров обозреваемых ячеек.

Зная мгновенное описание C_t, т.е. всю информацию, мы можем смоделировать по функции переходов δ последующие мгновенное описание C_{t+1} и т. д.

Можно придумать МТ, которая складывает, вычитает, возводит в степень,находит минимальный остов, т. е. выполняет любые операции.

> **Тезис Черча.** Все что можно числить с помощью некоторого алгоритма (понятие алгоритма не формализируется) может быть описано на языке МТ.

5.2 Недетерменированные машины Тьюринга(НДМТ)

Определение. НДМТ – машина Тьюринга, в которой функция перехода δ – многозначна, т. е., если в обычной МТ по мгновенным показаниям C_i однозначно определяется C_{i+1} (записывается $C_i \vdash\!\!\rightarrow C_{i+1}$) , то в НДМТ этот переход имеет несколько вариантов :

$$C_i \vdash\!\!\rightarrow C_{i+1}$$
$$C_i \vdash\!\!\rightarrow C'_{i+1}$$
$$C_i \vdash\!\!\rightarrow C''_{i+1}$$
и т. д.

Фактически процесс переход а мгновенных описаний ветвится, и на каждом шаге машина как бы угадывает правильное направление перехода(существует другая модель, аналогичная НДМТ – МТ с оракулом, т.е. с некоторым встроенным устройством, которое правильно предугадывает последовательность действий – куда пойти, чтобы получить правильный ответ).

Пример НДМТ. Рассмотрим задачу о разбиении.

Постановка задачи: имеется некоторый конечный набор натуральных чисел. Можно ли это множетсво разбить на непересекающихся подмножестватак, чтобы $\sum_{m\in M_1} m = \sum_{m\in M_2} m$ т.е. сумма чисел в каждом подмножестве была одинаковой.

На обычной МТ и на любом ее аналоге эта задача за полинамиальное время не решается (на данный момент алгоритм неизвестен). На НДМТ эту задачу нетрудно решить за полинамиальное время следующим образом: проходим множество М и для каждого элемента этого множетсва оракул предстказывает, куда этот элемент нужно вписать, в M_1 или в M_2 . При этом еще параллельно происходит суммирование элементов первого и второго подмножеств. Дойдя до конца, сравниваем счетчики сумм. Если они равны, то задача решена, в противном случае так разделить исходное множество нельзя. Итак, НДМТ решает задачу о разбиение за линейное время T(n) = тю

Если же данную задачу мы попытаемся решить на олбычной машине Тьюринга , то потребуется 2^{n-1} пдействий, так как вариантов разбиения множества на два подмножества $2^n/2$. При каждом варианте потребуется n /2 операций сложения. Итого трудоемкость $T(n)= 2^{n-1}\frac{n}{2} = n\cdot 2^{n-2}$ действий.

Определение. *Класс P (P-TIME)* - класс задач, решаемых за полиномиальное время на обычной (детерминированной) МТ.

Примеры подобных задач:

- алгоритм Дейкстры (задача о нахождении кратчайшего расстояния на графе — трудоемкость n^2*);*
- сортировки — трудоемкость n^2 или $n\ log(n)$*;*
- преобразование Фурье;

- вычисление определителя матрицы — решаем ее не разложением по строке или столбцу (тогда трудоемкость *n!),* а с помощью метода Гаусса приводим ее к треугольному виду и перемножаем диагональные элементы (трудоемкость n^3).

Определение. *Класс NP (NP-TIME)* — класс задач, решаемых за полиномиальное время на недетерминированной *МТ.*

Примеры подобных задач:

- *NP* содержит *P,* следовательно все задачи из класса *P* попадают в *NP;*
- задача о разбиении;
- задача коммивояжера в следующей постановке: существует ли гамильтонов цикл длины L≤C, где *C* — заданное число.

Последняя задача решается на *НДМГ* следующим образом: сначала на каждом шаге мы должны правильно угадать вершину, в которую нужно идти, а потом суммировать стоимости переходов для данного маршрута. Если сумма не превосходит С, то ответ положительный.

Класс *NP* содержит класс *P.* Но совпадают ли эти классы - неизвестно (скорее всего ответ будет отрицательный).

Определение. Две функции $f_1,f_2: N\to N$ со свойствами

$$f_i(n)\geq 0, \quad \lim_{n\to\infty} f_i = \infty, \qquad i=1,2$$

называются *полиномиалъно-эквивачентными,* если существуют два полинома p_1 и p_2 такие, что $p_1(f_1(n))\geq f_2(n)$ и $p_2(f_2(n))\geq f_1(n)$.

Пример:

♦***<u>Полиномиально-эквивалентные функции</u>:***

- n^2 и n^3

С одной стороны, $n^2 \leq n^3$, с другой стороны $(n^2)^2 > n^3$, то есть свойство полиномиальной эквивалентности выполняется, хотя эти функции не эквивалентны, так как $n^2 \ll n^3$.

- 2^n и 10^n

$2^n \leq 10^n$

$(2^n)^4=(2^4)^n=16^n>10^n$

♦ ***<u>Полиномиально-неэквивалентные функиыи</u>***:

n и Inn

n≥lnn

в какую бы степень α мы ни возвели In n, всегда будет $(\mathrm{In}\ n)^a \ll n$. Следовательно, не выполняется свойство полиномиальной эквивалентности.

n и 2

$(n)^\beta<2^n$=> не выполняется свойство полиномиальной эквивалентности.

$n!> 2^n$

$n!>(2^2)^\beta$=> не выполняется свойство полиномиальной эквивалентности.

Теорема 5.1 Если некоторая задача может быть решена на НДМТ M_1 за некоторое время T(n), то она же может быть решена на ДМТ M_2 за время $T_1 = \mathbf{2}^{T^2(n)}$.

Доказательство. Введем число *k* , равное максимальной степени ветвления функции переходов δ нашей исходной *НДМГ* M_1**.** Построим обычную МТ M_2**,** которая перебирает все возможные варианты ветвления функции δ и все их просчитывает. Исходная M_1 делает T(n) шагов. На каждом шаге присутствует не более k вариантов ветвления. Итого получаем не больше, чем $k^{T(n)}$ вариантов. Трудоемкость каждого варианта T(n). Итого общая трудоемкость M_1:

$T_1(n) \le n^{T(n)} \cdot T(n) = 2^{\log k \cdot T(n) + \log T(n)} \le 2^{T^2(n)}$

Теорема доказана.

Комментарий. Как следует из ***Теоремы 5.1,*** любую задачу, решаемую на ***НДМТ*** за полиномиальное время, мы можем решить и на обычной ***МТ,*** но за большее время (не полиномиально эквивалентное).

Теорема 5.2Любая задача, решаемая на к—ленточной НДМТ (к-ленточной МТ) за время T(n), может быть решена на одноленточной МТ такого же типа **(**если была k-ленточная НДМТ, то будет одноленточная НДМТ, в случае к-ленточной МТ будет одноленточная МТ) за время $T_i < CT^2(n)$

Доказательство. Научимся моделировать работу к-ленточной на одноленточной машине такого же вида. Поступаем для этого следующим образом: необходимо информацию о мгновенном описании ***к*** - ленточной машины записать на одну ленту (т.е. все символы на лентах + ***N*** обозреваемых ячеек):

Делаем это следующим образом:

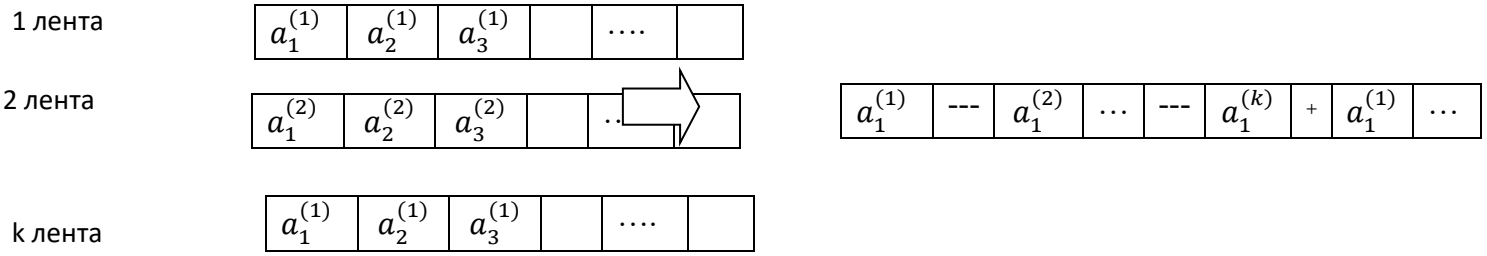


Рисунок5.1 Моделирование работы k-ленточной МТ на одноленточной

Плюс или минус в ячейках результирующей ленты стоит в зависимости от того, находится ли над соответствующей ячейкой одной из лент многоленточной машины считывающая головка, или нет.

Промоделируем 1 шаг работы исходной k-ленточной машины $м_i$ на новой одноленточной машине. Для этого сначала проходим всю ленту и ищем плюсы, **Т**.е. те ячейки, над которыми находятся считывающие головки. Параллельно просматриваем содержимое данных ячеек.

На следующем проходе меняем расположение плюсов в соответствии со старой функцией перехода. При этом в соответствии с инструкциями *L* и *R* перемещаем плюсики. Потом переходим в новое состояние.

Новая функция переходов δ' для построенной таким образом одноленточной машины, гораздо больше по количеству команд, нежели исходная функция δ k-ленточной машины, и при этом она будет однозначной, если исходная ленточная машина была детерминирована, а если исходная ***k***-ленточная машина была недетерминирована, то неоднозначность сохранится.

То есть для моделирования одного шага k-ленточной машины нам потребуется два раза пройти всю ленту, новой одноленточной машины. Заметим, что длина исписанного участка каждой ленты исходной машины не превосходит T(n),так как за *T(n)* шагов мы не можем уйти больше чем на ***T(n)*** клеток от первой ячейки.

Итого на один шаг нам потребуется $k \cdot T(N)$ действий. Для моделирования ***T(n)*** шагов нам потребуется

$$T_1(n) \le T(n) \cdot k(n) \approx c \cdot T(n) \le T(n) \cdot k \cdot T(n) = c \cdot T^2(n)$$

действий.

Теорема доказана.

Комментарий. Таким образом, при замене *k*- ленточной машины на одноленточную, трудоемкость задач, решаемых на одноленточной и *k*-ленточной МТ (НДМТ и ДМТ), полиномиально эквивалентна.

5.3 Сводимость. NP-полнота.

Будем говорить, что задача L_1, полиномиально трансформируема к задаче L_2, если существует трансформатор T, который преобразует входную информацию (входное слово) ω_1 для задачи L_1 во входное слово ω_2 для задачи L_2 причем ответы при решении

этих задач (задачи L_1 с входной информацией ω_1 и задачи L_2 с информацией ω_2) будут одинаковые (в случае, когда ответом задачи может являться «ДА» или «Нет») (смотри рис. 5.2).

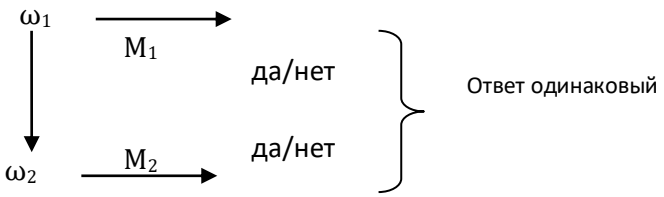


Рисунок 5.2

Пояснение к рисунку 5.2 Пусть МТ M_1 решает задачу L_1, а МТ M_2 решает задачу L_2. Каждая машина дает ответ да/нет. T – трансформатор.

Если же ответ выглядит более сложным образом, то нам потребуется не только входной трансформатор, но и выходной трансформатор T_1(см. рисунок 5.3)

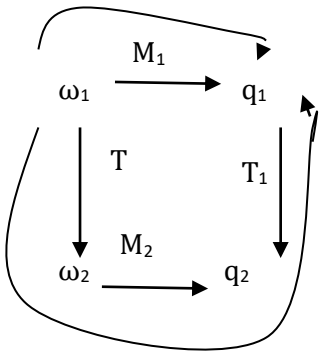


Рисунок 5.3

Пояснения к рисунку 5.3:

q_1q_2 – выходная информация для M_1 и M_2

T_1 – трансформирует q_2 в q_1

Результаты прохождения по обоим путям (стрелки сверху и снизу) одинаковы. Приэтом время работы M_2 полиномиально эквивалентно времени работы машины M_1, и оба трансформатора T и T_1 работают полиномиальное время.

Если все эта условия выполняются, то мы говорим, что задача L_1, полиномиально трансформируема к задаче L_2.

Определение. Задача L_0 из класса NP называется NP-полной задачей, если любая другая задача L из класса NP может быть к этой задаче L_0 полиномиально трансформируема (сводима).

Фактически NP- полные задачи - это самые сложные задачи в классе NP, то есть как только мы научимся решать за полиномиальное время T(n) какую-либо NP-полную задачу, мы, тем самым, научимся решать за время P(T(n)) любую задачу из класса NP.

Комментарий. Трансформаторы T и T_1 есть некоторые детерминированные машины Тьюринга, которые строятся индивидуально для каждых задач L_1 и L_2.

Возникает вопрос: существуют ля вообще NP-полные задачи? Ответ на данный вопрос положительный.

Например, NP полными задачами являются следующие задачи:

1. Выполнимость (выполнима ли данная булева формула), т.е. существует ли некоторый набор переменных, такой, что формула на этом наборе принимает истинное значение.
2. Задача о клике (найдется ли в данном неориентированном графе G клика размера к, т.е. подграф размера к, каждая вершина которого связана с любой другой).
3. Задача о гамильтонове контуре (найдется ли в данном графе G гамильтонов цикл).
4. Задача об узельном покрытии (найдется ли в данном графе G узельное покрытие размера к, т.е. такие к вершин, что любое ребро графа инцидентно какой-либо вершине из подмножества к (т.е. входит или выходит из какой-либо вершины узельного покрытия)).
5. Задача о разбиении (можно ли разбить некоторое множество M натуральных чисел на два подмножества так, чтобы суммы этих подмножеств были одинаковы).

Подобного рода NP-полных задач известно несколько сотен. Научившись быстро решать хоть одну из них (то есть за полиномиальное время), мы и любую из них сможем решить тоже за полиномиальное время. Если же, напротив, сможем доказать, что некоторая NP-полная задача не решаема за полиномиальное время ни на одной ДМТ, мы тем самым докажем, что и другие NP-полные задачи не могут быть решены за полиномиальное время.

Теорема 5.3 Задача выполнимости булевой формулы NP-полна.

Доказательство.

Для начала докажем, что задача выполнимости лежит в классе NP. Для этого достаточно предъявить некоторую НДМТ, которая решит эту задачу за полиномиальное время. Это нетрудно сделать, т.к. для решения данной задачи необходимо один раз правильно угадать набор переменных и проверить, что на этом наборе переменных формула принимает значение, равное 1. Для формулы длины n такую проверку можно сделать за cn^2 действий.

Докажем теперь, что любая задача L, из класса NP может быть полиномиально сведена к задаче выполнимости.

Итак, мы имеем некоторую НДМТ, которая за полиномиальное время T решает нашу задачу L.

Пусть ω - входная информация машины M(входное слово для нашей машины).

$|\omega| = n$ - объем входной информации.

Если машина M допускает слово ω, то найдется некоторая последовательность мгновенных описаний:

$Q_0 \rightarrow Q_1 \rightarrow \ldots \rightarrow Q_{end}$,

причем

Q_0 -мгновенное описание следующего вида:

- в ячейках ленты записано слово ω;
- обозревается первая ячейка;
- мы находимся в состоянии q_0.

Q_{end} - следующего вида:

- в ячейках записано все что угодно (выходная информация);
- обозревается любая ячейка;
- мы находимся в состоянии q_{end}.

Длина этой последовательности мгновенных описаний не превосходит P(n), где P(n) - некоторый многочлен.

Основная идея доказательства ***Теоремы 5.3*** состоит в том, что мы будем строить булеву формулу Z, которая «моделирует» работу машины M на входном слове ω, т.е. моделирует последовательность мгновенных описаний (5.1). А именно, Z принимает значение 1 тогда и только тогда, когда присваивание значений 0 или 1 переменным в этой формуле соответствует допускаемой последовательности мгновенных описаний (5.1).

Переменные в нашей формуле Z будут трех видов:

1)C(i, j, t) =1 тогда и только тогда, когда i-ая клетка ленты содержит символ x_j в момент времени t;

2)S(k t) =1 тогда и только тогда, когда машина M в момент времени t находится в состоянии q_k

3)H(i, t) = l, когда головка M в момент времени t находится над i-той ячейкой ленты.

Таким образом, значения переменной C отвечают за содержимое ленты, S — за текущее состояние, H - за положение головки. Т.е., зная значения всех переменных C, S, H, мы имеем полную информацию о последовательности мгновенных описаний машины M.

Заметим, что $1 \le i \le P(n)$, т.к. за время работы P(n) (то есть за P(n) шагов) мы доберемся не далее, чем до P(n)-той ячейки.

$0 \le t \le P(n)$ $1 \le j \le C$ $1 \le x \le C$, где C некоторая константа.

C=|T|+|Q| -зависит только от машины;

|T| - количество символов алфавита данной машины;

|Q| - количество состояний.

Таким образом нам потребуется $CP^{(2)}(n)$ переменных первого типа;

$CP(n)$ переменных второго типа;

$P^{(2)}$ переменных третьего типа.

Итого нам потребуется

$$CP^{(2)}(n)+ CP(n)+ P^{(2)}=P^2(n)\times Const \text{ переменных.}$$

Для каждой переменной (при двоичной кодировке) потребуется не более чем

$$\log_2 P^2(n) \times Const \le P(n)$$

символов на ленте.

Для дальнейшего доказательства Теоремы 5.3 нам потребуется Лемма 5.4

Лемма 5.4 Для любого набора переменных $x_1,..,x_к$ существует формула $U(x_1,..,x_к)$, которая равна 1 тогда и только тогда, когда равно одна из переменных $x_1,..,x_к$ принимает значение 1, а все остальные - нули.

Доказательство. Предъявим формулу:

$$U(x_1,..,x_к)=(x_1 \vee \ldots \vee x_к)\&(\&_{i<j}(\neg x_i \vee \neg x_j)) \qquad (5.2)$$

Формула (5.2) представляет собой набор КНФ, поэтому она равна 1, когда все ее элементарные сомножители равны 1.

$x_1 \vee \ldots \vee x_к=1$ тогда и только тогда, когда хоть одна из переменных x_i равна 1.

$\neg x_i \vee \neg x_j=1$ тогда и только тогда, когда x_i и x_j одновременно в 1 не обращаются.

$\&_{i<j}(\neg x_i \vee \neg x_j)=1$ т.к. мы перебираем все вариант по i, j, то получаем, что никакие две переменные одновременно в 1 не обращаются.

Замети, что длина формулы (5.2) порядка k^2.

Лемма доказана.

Вернемся к доказательству теоремы.

Искомую формулу Z будем искать в виде конъюнкции:

Z=A&B&C&D&E&F&G, где

A: отвечает за то, что в каждый момент времени головка обозревает ровно одну ячейку.

B: отвечает за то, что в каждый момент времени в каждой ячейке находится ровно один символ.

C: отвечает за то, что в каждый момент времени машина находится ровно в одном состоянии.

D: отвечает за то, что при переходе к следующему шагу машина меняет содержимое не более одной ячейки.

E: отвечает за то, что все изменения на каждом шаге происходят в строгом соответствии с функцией перехода δ.

F: отвечает за то, что в момент времени t = 0 мы находимся в состоянии q0 и на входе имеем слово ω, обозреваем первую ячейку.

G: отвечает за то, что в момент времени P(n) мы находимся в состоянии q_{end}.

Для удобства будем считать, что машина M всякий раз работает ровно P(n) шагов и каждое мгновение описание содержит P(n) ячеек.

- $A = A_1 \& A_2 \& \ldots \& A_{P(n)}$
  $A_i = U(H(l,t), H(2,t), \ldots, H(P(n),t))$

  $A_i = 1$ тогда и только тогда, когда ровно одна из переменных $H(k,t) = 1$, т.е. в момент времени t мы обозреваем ровно одну ячейку (т.к. берем конъюнкцию по всем t). Здесь P(n) — длина ленты.

- $B = \&_{i,t} B_{it} \qquad 1 \le i,t \le P(n)$
  $B_{it} = U(C(i, 1, t), C(i, 2, t), \ldots, C(i, const, t))$
- $C = C_1 \& C_2 \& \ldots \& C_{P(n)}$
  $C_t = U(S(1, t), \ldots, S(const, t))$
  Здесь P(n) - время работы машины.
- $D = \&_{i,j,t} (H(i,t) \vee C\{i,j,t) \equiv C(i,j,t+1))$.

Знак «≡» обозначает совпадение значений двух переменных, то есть
$(x \equiv y) = (x \& y) \vee (\neg x \& \neg y)$.

Если H(i, t) = 1, т.е. мы находимся в i-той ячейке, то с этой ячейкой мы можем делать все что угодно.

Если H(i,t) = 0, т.е. головка находится на другой ячейке, то должно выполняться $C(i,j,t+1) \equiv C(i,j,t+1)$, т.е. содержимое 1-той ячейки при переходе от момента времени t к моменту времени t+1 не меняется.

- $E = \&_{i,j,k,t} E_{i,j,k,t}$

$E_{i,j,k,t} = \neg C(i,j,t) \vee \neg H(i,t) \vee S(k,t) \vee (\vee_l (C(i,j_l,t+1) \& S(k_l,t+1) \& H(i_l,t+1)))$,

где l пробегает все возможные разветвления многозначной функции переходов δ.

q_{kl} новое состояние, в соответствии с многозначной функцией переходов δ;

x_{jl} новый символ, который мы напишем в i-той ячейке;

$i_l = i+1, i, i-1$ – номер новой обозреваемой ячейки;

$1 \le l \le L$, где L – максимально возможное число ветвлений нашего алгоритма.

Комментарий.

$E_{i,j,k,t}$ будет равно й в том случае, если C(i,j,t)=1, H(i,t)=1,S(k,t)=1(т.е. начальный фрагмент E равен 0).

C(i,j,t)=1, H(i,t)=1,S(k,t)=1 означает, что в момент времени t мы видим символ x_j в i-той ячейке ленты и находимся в состоянии q_k, то есть мы должны работать в строгом соответствии с функцией переходов δ.

Если же хоть одно из этих условий не выполняется то мы можем делать что угодно.

- $F = S(0,0) \& H(1,0) \&_i (C(1,j_1,0) \& C(2,j_2,0) \& \ldots \& C(n,j_n,0)) \& C(n+1,b,0) \& \ldots \& C(P(n),b,0)$
  $x_{j1} \ldots x_{jn}$ – символы входного слова ω
  Во всех остальных ячейках находится пустой символ b.
- G=S(end,P(N))

Построенная нами формула Z моделирует работу исходной НДМТ M: она принимает значение 1 тогда и только тогда, когда мы прошли от начального состояния до конечного в соответствии с работой машины M и функцией переходов.

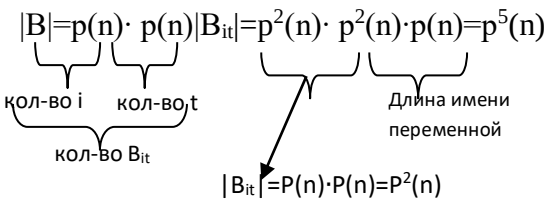
Осталось убедиться, что длина формулы Z тоже полином, и, следовательно, её выполнимость может быть проверена за полиномиальное время (функция длины n может быть проверена за время n^2). Длина формулы Z складывается из длин ее слагаемых:

$$|A| = p(n) \cdot |A_i| = p(n) \cdot p^2(n) \cdot p(n) = p^4(n)$$

кол-во A_i — кол-во переменных в A_i — длина записи имени переменной

Каждый конъюнкт A_i, представляет собой функцию U и зависит от P(n) символов. Длина функции U, зависящей от P(п) символов составляет $P^2(n)$.

Оценим длину формулы B:



Аналогичным образом оцениваются длины всех остальных формул.

Складывая длины всех этих формул, получаем:

$$|Z| \le const \cdot p^5(n) .$$

Как мы знаем, выполнимость формулы длины k на НДМТ может быть проверена за время, не превосходящее k^2.

Таким образом, мы доказали, что наша исходная задача из класса NP полиномиально трансформировалась к задаче выполнимости, решаемой за полиномиальное время.

Теорема доказана.

В ***Теореме 5.3*** мы по НДМТ M и входной информации ω для нее построили формулу Z, которая моделирует работу этой машины, т.е. принимает значение 1 тогда и только тогда, когда исходная НДМТ M закончит свою работу (стартовав со слова ω). При этом выполнимость формулы Z проверяется за время работы, которое полиномиально эквиваентно P(n) - времени работы M.

Осталось заметить, что входной трансформатор, который преобразует матрицу M+ ω в формулу Z, работает полиномиальное время.

Следствие 5.5 Задача выполнимости КНФ NP полна.

Доказательство. Построенная в ***Теореме 5.3*** формула Z уже почти находится в КНФ, а именно:

A – уже в КНФ, т.к. U()- уже в КНФ, B - уже в КНФ, C –уже в КНФ, D - почти в КНФ, E - почти в КНФ, F - уже в КНФ.

«Почти в КНФ» — означает, что формулы D и E есть конъюнкции большой длины от формул маленькой длины (длина ≤const), причем формулы маленькой длины мы можем спокойно привести в КНФ, при этом длина формулы вместо k станет не больше, чем $k \cdot 2^k$ (по таблице истинности исходной формулы находим 0, их не больше 2^k штук; потом составляем СКНФ — в ней будет не более 2^k конъюнктов, длина каждого конъюнкта не более к , где к— количество переменных).

Следовательно, при преобразовании форму D и E в КНФ их длина увеличивается не более, чемв const раз.

Для формулы D эта const = 3, для E const = 3m, где m — максимально возможное количество ветвлений в формуле переходов δ.

Замечание. Таким образом, формулу Z мы можем (увеличив ее длину не более, чем в const раз) заменить на равносильную ей формулу Z_1, которая находится в КНФ.

Следствие доказано.

Теорема 5.6 Задача о клике—NP-полная.

Доказательство. Сначала проверим, что задача о клике принадлежит классу NP. В самом деле, нетрудно построить НДМТ, которая сначала угадывает за один проход нашу клику размера к (подграф из к вершин, в котором каждая вершина связана с каждой), а затем за время к проверяет, что каждая вершина соединена с каждой.

Докажем теперь, что любая задача S из класса NP может быть полиномиально трансформируема к задаче о клике. Для этого сначала трансформируем задачу S к задаче выполнимости некоторой формулы Z, находящейся в конъюнктивной нормальной форме(см. Следствие 3.5), а уже задачу выполнимости формулы Z сведем к задаче о клике.

Итак, пусть задача S сведена к задаче выполнимости формулы Z, причем

$$Z=\&_{i=1}^{k}(\vee_{j=1}^{k_i} x_{ij})$$

представляет собой конъюнкцию дизъюнкций, где x_{ij} — литералы, то есть переменные или отрицания переменных. Наша задача — по формуле Z построить граф G, такой, что в графе G найдется клика размера к тогда и только тогда, когда формула Z выполнима.

Граф G =(V, E), где V - множество вершин графа, E - множество ребер.

Вершины индексируются парами (i, j), где $1 \le i \le k$, $1 \le j \le k_i$

Строим граф G следующим образом: каждой вершине соответствует вхождение переменной в формулу Z. Две вершины соединены ребром $(|i_1,j_1|)(|i_2, j_2|)\in E$ тогда и только тогда, когда соответствующие литералы лежат в разных конъюнкциях и один литерал не совпадает с отрицанием другого.

Пример. $Z = (x_1 \vee \bar{x}_2)\&(x_2 \vee \bar{x}_3)\&(x_3 \vee \bar{x}_1)$.

Решение. Литералов 6 - поэтому в графе будет 6 вершин.

Вершинам соответствуют литералы:

[1,1] - x_1 [2, 1] - x_2 [3,1] - x_3

[1,2] - $\bar{x}_2$ [2,2] - $\bar{x}_3$ [3,2] - $\bar{x}_1$

Соединим вершины, руководствуясь вышеизложенным принципом:

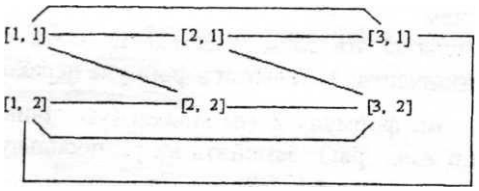


Рисунок 5.4

[1,1] и [1,2] не соединяем потому, что они входят в один конъюнкт.

[1, 1] и [3, 2] не соединяем потому, что хоть они и лежат в разных . конъюнктах, но один является отрицанием другого и т.д.

Заметим, что формула Z выполнима, так как в построенном графе G найдется клика размером 3 (т.к. в Z было 3 конъюнкта). При этом Z = 1 когда $x_1= x_2= x_3= 0$

либо $x_1= x_2= x_3= 1$.

Заметим, что первому набору значений переменных соответствует 3-клика [1, 1]»[2,1], [3,1] , а второму набору 3-клика [1,2], [2,2], [3,2] .

Докажем теперь, что в графе G найдется клика размера k тогда и только тогда, когда формула Z выполнима.

В самом деле, если Z на некотором наборе переменных выполнима, то в каждом конъюнкте хотя бы один литерал обращается в 1. Пусть это будут литералы $x_{1,i1}\ldots x_{k,ik}$

Тогда соответствующие им к вершине графа G образуют клику, так как вершины

$([l, i_l]; \lfloor p, i_p \rfloor)$ соединены друг с другом, так как $l\neq p$ и $x_{i,l}\neq x_{i,p}$ так как они оба обращаются в 1.

Обратно, если в построенном нами графе G нашлась k-клика, то:

1)все эти к вершин лежат в разных конъюнктах (так как вершины из одного конъюнкта ребрами не соединяются);

2) мы можем подобрать такие значения переменных, что хотя бы один литерал в каждом конъюнкте обращается в единицу.

В самом деле, имеем клику из k вершин (все вершины из разных конъюнктов)

$$[1, i_1]; \lfloor k, i_k \rfloor$$

Если литерал $x_{p,ip}$ есть некоторая переменная, то эту переменную полагаем равной единице. Если же он есть отрицание некоторой переменной, то эту переменную полагаем равной нулю. Остальные переменные (не входящие в этот конъюнкт) могут принимать любые значения. Тогда в первом

конъюнкте литерал $x_{1,i2}$ обращается в единицу, во втором конъюнкте $x_{2,i2}$ =1, то есть наша формула Z=l.

Таким образом, мы доказали, что функция Z выполняется тогда и только тогда, и только когда в графе G найдется клика размера k.

Теорема доказана.

Теорема 5.7 Задача об узельном покрытии NP-полна. (Узельное покрытие графа G - такой набор вершин $V_0 \subset V$, что любое ребро графа G инцидентно какой-либо вершине из V_0).

Доказательство. Докажем сначала, что задача об узельном покрытии принадлежит классу NP. Для этого построим НДМТ, которая угадывает узельное покрытие V_0 за время k, а потом за полиномиальное время n·k проверяет, что каждое ребро графа инцидентно какой-либо вершине из V_0 (то есть каждое ребро проверяется на смежность с каждой вершиной).

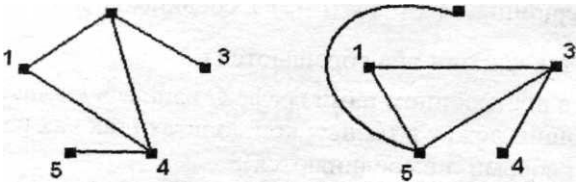
Докажем теперь, что любая задача P из класса NP может быть полиномиально трансформируема к задаче об узельном покрытии.

Вспомним (см. Теорему 5.6), что любая задача T из класса NP может быть полиномиально трансформируема к задаче о клике. Осталось только лишь трансформировать задачу о клике к задаче об узельном покрытии.

Пусть нам дан граф G-(V, E).

Рассмотрим дополнительный граф $\bar{G}$ =(V, $\bar{E}$). То есть ребро (u,v) $\in \bar{E}$ тогда и только тогда, когда (u,v)$\notin$E (то есть как бы инвертируем граф G по его ребрам. Тогда множество S вершин графа G образует клику в графе G тогда и только тогда, когда его дополнение будет узельным покрытием графа $\bar{G}$.

Иллюстрация сказанного на рисунке 5.5:



Клика размера 3 -1,2.4 в G => Узельное покрытие 3.5 в G

Рисунок 5.5

В самом деле, если S - клика в G, то V\S - узельное покрытие в $\bar{G}$. Возьмем произвольное ребро из (u,v)$\in$ E.

Нам необходимо доказать, что либо u, либо v попадают в узельное покрытие V\S. Если это не так, то есть u и v $\in$ S, тогда мы получим противоречие (так как S - клика в графе G, то с одной стороны, (u,v)$\in$E, и с другой стороны (u,v)$\in \bar{E}$).

Докажем теперь противное: если V\S - узельное покрытие в $\bar{G}$, то S- клика в G.

Возьмем произвольные u и v из S. Надо доказать, что (u,v)$\in$ E. Если это не так, то тогда по построению $\bar{E}$ ребро (u,v)$\in \bar{E}$, но u и v попадают в S, и поэтому не лежат в V\S, следовательно, это ребро не инцидентно никакой из вершин V \ S. Получаем противоречие, которое завершает доказательство ***Теоремы 5.7.***

Теорема доказана.

Итак, мы выявили много NP-полных задач и все они друг другу полиноминально эквивалентны, поэтому как только мы научимся любую из них решать за полиномиальное время, это будет означать, что мы можем любую задачу из класса NP решить за полиномиальное время, следовательно, класс NP совпадает с классом P.

Если же, напротив, для какой-то из них нам удастся доказать, что какая-либо из NP-полных задач не решается за полиномиальное время на обычной НДМТ, мы, тем самым, докажем, что любая другая задача не может быть решена за полиномиальное время.
Грубо говоря, все NP- полные задачи имеют приблизительно одинаковую сложность.

Домашнее задание №7.
По заданной в КНФ Z построить граф G и найти в нем возможные «клики».
$Z=(x_1 \vee \bar{x}_2 \vee \bar{x}_4)\&(\bar{x}_2 \vee \bar{x}_4)\&(x_3 \vee \bar{x}_4)\&(\bar{x}_1 \vee \bar{x}_4)$

5.4 Иерархия по сложности. Труднорешаемые задачи.

Рассмотрим следующие классы задач:

- *P (P-time)* – класс задач, решаемых за полиномиальное время на ДМТ.
- *NP (NP-time)* – класс задач, решаемых за полиномиальное время на НДМТ.
- *P-space* – класс задач, которые решаются на обычной машине Тьюринга с использованием не более, чем полиномиальной памяти.
- *NP-space* – класс задач, которые могут быть решены на НДМТ с полиномиальной памятью.

Теорема 5.8 P-time $\subset$ P-space

NP-time $\subset$ NP-space

Доказательство. Ясно, что, работая полиномиальное время, мы используем не более, чем полиномиальную память.

Теорема 5.9 P-space=NP-space

Доказательство. Что P-space $\subset$ NP-space очевидно, так как детерминированная машина всегда может рассматриваться как недетерминированная.

Докажем обратное вхождение $\supset$.

Пусть T – некоторая задача, которую мы умеем решать на НДМТ M с использованием полиномиальной памяти. Научимся ее решать также с использованием не более, чем полиномиальной памяти на обычной машине Тьюринга M_1.

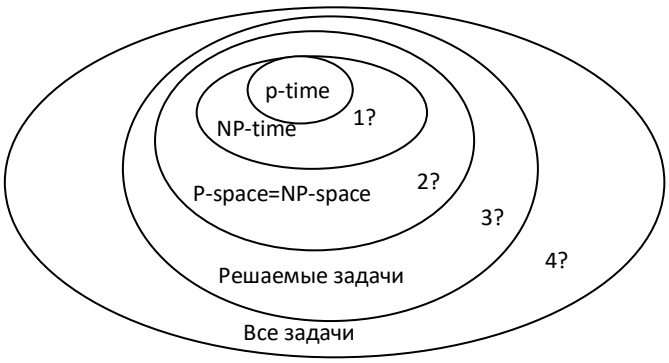
Наша M_1 перебирает все возможные варианты в работе исходной НДМТ M и после окончания разбора очередного варианта освобождает память. Поэтому нам потребуется столько же памяти, сколько требуется для работы исходной машины M, но при этом надо вводить дополнительную

память для кодирования номера рассматриваемого варианта. Для этого потребуется не более, чем P(n) ячеек(при двоичной кодировке).

Таким образом, теорема доказана, так как работу любой НДМТ с неполиномиальной памятью мы смогли промоделировать на ДМТ с полиномиальной памятью.

Теорема доказана.

Классы сложности.



1? – означает: выполняется ли равенство NP-time = P-time или нет. Ответ на этот вопрос неизвестен и стоит $1000000.

2? – означает: выполняется ли равенство NP-space= Np-time или нет (то есть существуют ли задачи, которым хватает полиномиальной памяти, но за полиномиальное время даже на НДМТ их решить нельзя). Ответ на этот вопрос так же неизвестен.

3?- означает: существуют ли задачи, которые нельзя решить, используя только полиномиальную память. Ответ на этот вопрос дается

Теоремой 5.10

Существует задача(задача непустоты дополнения полурасширенного выражения), которая решаема. Но для ее решения требуется больше чем

$$\left.2^{2^{2^{2^{\cdots}}}}\right\}^{n} = k \text{ раз}$$

памяти, где k - константа, сколь угодно большая. Тем более эта задача не решаема за полиномиальное время с использованием полиномиальной памяти.

Без доказательства.

4? – означает: существуют ли неразрешаемые задачи? Ответ на данный вопрос дается в **Разделе 6.**

6 НЕРАЗРЕШИМЫЕ ЗАДАЧИ

6.1 Новая модель алгоритма вычислений.

Новая модель алгоритма вычислений представляет собой машину с неограниченными регистрами (МНР).

Определение. МНР – машина с бесконечной лентой ячеек, которые пронумерованы от 1 до ∞, причем в каждой ячейке содержится некоторое натуральное число.

Машина работает по программе, написанной на языке МНР. Язык МНР состоит из команд 4 типов

Тип	Команда	Действие МНР
Обнуление	Z(n)	X_n=0,обнуление n-ой ячейки
Прибавление 1	S(n)	Inc(x_n)
Переадресация	T(m,n)	x_n:=x_m
Условный переход	J(m,n,q)	If (x_n=x_m) then goto q?где q – номер строки программы

В программе на каждой строке находится по одной программе.

Теорема 6.1 Все что можно вычислить, можно вычислить на МНР.

Доказательство. Научимся моделировать на МНР работу машины Тьюринга (а все что можно вычислить – можно вычислить на машине Тьюринга(тезис Черча)).

Теорема доказана.

Замечание о трудоемкости вычисления на МНР.

С одной стороны на МНР более быстрый доступ к памяти по сравнению с МТ(при работе на МТ мы за один шаг смещаемся только в соседнюю ячейку, а при работе на МНР за один шаг можем переместиться в любую), другой стороны, на МНР значение переменной за одно действие изменяется всего на 1, так что некоторые программы будут быстрее выполнятся на машине Тьюринга, некоторые на МНР.

В данном случае нас интересует не трудоемкость, а только то, разрешима ли данная задача или нет, а классы разрешимых задач у нас совпадают.

Пример. Программа сложения двух чисел:

Вход:

1 2 3 4 5

X	Y	*	*	*	*…

Выход:

1 2 3 4

*	*	X+Y	*	*…

Рисунок 6.1 Иллюстрация работы программы

Программа

1: T(1,3)

2: Z(4)

3: S(3)

4: S(4)

5: J(2,4,7)

6: J(1,1,3) – фактический оператор безусловного перехода

7: -------- - пустой оператор (выход), т. е . любой оператор, ничего не меняющий в третьей ячейке

6.2 Нумерация программ

Теорема 6.2 Все программы на языке МНР можно эффективно перенумеровать, то есть каждой программе соответствует некоторый номер так, что разным программам соответствуют разные номера, и в то же время каждому натуральному числу будет соответствовать некоторая программа.

Доказательство. Сначала научимся нумеровать команды МНР.

Тип команды	Номер, присваиваемый данной команде
Z(n)	4n-3
S(n)	4n-2
T(m,n)	4π(m,n)-1
J(m,n,q)	4π(m,π(n,q))

Где π – некоторая специальная функция, нумерующая пары натуральных чисел. В качестве примера такой функции приведем $\pi(m,n)=2^{m-1}(2n-1)$.

Заметим, что функция π – хорошая. То есть разным парам соответствуют разные числа и каждому числу соответствует некоторая пара. Кроме того, обе функции – прямая (π) и обратная (π^{-1}) эффективно вычислимы. Покажем это:

$\pi(3,4) = 2^2(2\cdot4-1)=4\cdot7=28$

$\pi^{-1}(28)=2^{2^\circ}\cdot\frac{28}{2^2} = 4\cdot7 = 2^{3-1}(2\cdot4-1)$

° - ставим 2, так как это максимальная степень двойки, на которую делится 28

$\pi^{-1}(54)= \pi^{-1}(2\cdot27)= \pi^{-1}(2(2\cdot14-1))=(2,14)$

$\pi^{-1}(100)=(3,13)$

Рассмотрим примеры кодировки команд:

$T(3,2)\rightarrow4\pi(3,2)-1=2^2(2\cdot2-1)=4\cdot48-1=191$

$J(2,1,3)4\pi(2,\pi(1,3))=4\pi(2,5)=4\cdot29=72.$

Построенная нами нумерация хорошая, так как:

во-первых, разным командам соответствуют разные числа,

во-вторых, каждому натуральному числу соответствует некоторая команда,

в-третьих, нумерация и в ту и в другую сторону эффективно вычисляема, то есть по команде нетрудно вычислить число, ей соответствующие, и так же по числу определяемому командой.

Пример.

Определить команду имеющую код 247.

1. Определяем остаток от деления на 4: 247 div 4=3
2. Так как получено 3, значит мы, при вычислении номера команды, вычитали1, следовательно, мы переводили в число команду Т.
3. Восстанавливаем обратно: 247+1=248=4·62
   П(k,l)=62
   k=2

l=16
Результат: T(2,16).
Определить команду. Имеющую код 264.

1. 264 div 4 = 0
2. J(m,n,q)=4π(m,π(n,q))
3. 264=4·66
   66=π(m,π(n,q))
   m=2
   n=2
   q=9

Результат: J(2,2,9).

Теперь осталось научиться нумеровать программы. Программа есть конечная последовательность команд. Каждой команде поставим в соответствие ее номер по описанному выше алгоритму. После этого надо научиться нумеровать все возможные конечные последовательности натуральных чисел.

Делаем это следующим способом.

Рассмотрим функцию φ, которая переводит в набор из k натуральных чисел в натуральное число по следующей формуле:

$$\varphi(n_1,n_2..n_k) = \frac{2^{n_1}+2^{n_1+n_2}+2^{n_1+n_2+n_3}+\cdots+2^{n_1+\cdots+n_k}}{2}$$

Если число $\varphi(n_1,n_2..n_k)$ записать в двоичной записи, то оно может, у примеру выглядеть так:

$$\Phi(1,2,3)=\frac{2^1+2^3+2^6}{2} = 2^0+2^2+2^5=100101$$

n_3 n_2 n_1

$\Phi^{-1}(13)=(1,2,1)$ так как 13 1101

n_3 n_2 n_1

Таким образом, функция φ – хорошая, так как:

1) Разные наборы переходят в разные числа;
2) Каждому натуральному числу соответствует набор чисел, который в него приходит;
3) Функции φ и φ^{-1} эффективно вычислимы.
4) Таким образом мы научились нумеровать все программы на языке МНР и доказали ***Теорему 6.2***

Теорема доказана.

6.3 Неразрешимые проблемы

Определение. Проблема А (x∈A) называется разрешимой, если функция

$$X_A \begin{cases} 1, x \in A \\ 0, x \notin A \end{cases}$$

Вычислима, то есть можно построить, например, МНР, которая по входу x через конечное время работы выдает 1 или 0, в зависимости от того, попадает x в множество А или нет.

Пример. Разрешимые проблемы:

1. Четно ли число x.
2. Простое ли число x.
3. Ориентированный граф или нет(т.е. будет ли его матрица симметричной).

Определение. Проблема A (x∈A) называется частично разрешимой, если функция

$$X_A^{\leftrightarrow}\begin{cases}1, x \in A\\ \leftrightarrow, x \notin A\end{cases}$$

Эффективно вычислима, т. е. можно написать МНР, которая в случае, если x∈A за конечное число шагов выпадает 1, и зацикливается, если x∉A.

Знак $\leftrightarrow$ означает, что алгоритм расходится при подаче на вход числа x.

Лемма 6.3 Если проблема A разрешима, то она частично разрешима.

Доказательство. У нас есть программа, которая вычисляет функцию $X_A(x)$.

Припишем к этой программе блок, который зациклит программу, если она выдала ноль. Т.о. мы получим программу, которая вычисляет функцию $X_A^{\leftrightarrow}(x)$.

Лемма доказана

Теорема 6.4 *Если проблема A (x∈A) и проблема* $\overline{A}$ *(x∈*$\overline{A}$*)* частично разрешимы, то *сама проблема A будет разрешима.*

Доказательство. Пусть у нас есть две программы: p_1 и p_2. Первая вычисляет функцию:

$$X_A^{\leftrightarrow}\begin{cases}1, x \in A\\ \leftrightarrow, x \notin A\end{cases}$$

Вторая – функцию:

$$X_{\overline{A}}^{\leftrightarrow}\begin{cases}1, x \in A\\ \leftrightarrow, x \notin A\end{cases}$$

Наша задача – написать программу, которая вычисляет

$$X_A\begin{cases}1, x \in A\\ 0, x \notin A\end{cases}$$

Для этого поступаем следующим образом: возьмем вход x и сделаем один шаг программой p_1 и один шаг программой p_2. Если какая-то из них закончила работу, то ответом будет 1, если закончила работать программа p_1, и ответом будет 0, если закончила работать программа p_2. Потом опять берем вход x и делаем по два шага каждой программой и пишем в ответе 0 или 1 в зависимости от того, какая программа, p_1 или p_2 закончила работу. Если же ни одна из них не закончила работу, то берем вход x и делаем по три шага. Рано или поздно либо p_1, либо p_2 выдадут 1, следовательно, наша программа закончит работу.

Теорема доказана.

Теорема 6.5 *Существуют неразрешимые проблемы (например, проблема останова).*

Доказательство. Введем универсальную функцию $a(n,k)=P_n(k)$- результат работы программы номером n(нумерацию берем из теоремы 6.2), на выходе программы k. Заметим, что функция a(n,k) частично вычислима. В самом деле, по номеру n можно восстановить саму программу, а потом промоделировать ее работу на входе k.

Замечание. Функция $\varphi(n)$ называется частично вычисляемой, если существует программа, которая либо выдает значение этой функции, если x принадлежит области определения φ, либо зацикливается, если x не принадлежит области определения φ (т.е. функция вычисляема не везде, а только на ее области определения).

Рассмотрим проблему останова в следующей постановке: машина P_n на входе k не зацикливается. Т. е. возможно написать программу, которая по коду исходной программы и входным данным выдает ответ 1, если $P_n(k)$ не зацикливается, и 0 если зацикливается.

Для доказательства рассмотрим проблему останова в упращенном варианте. Докажем, что функция:

$$g(n) = \begin{cases} 1, P_n(n)\downarrow \\ 0, P_n(n)\not\downarrow \end{cases}$$

не вычислима.

Предположим противное, что функция g(n) вычислима. То есть у нас есть некоторая МНР, которая эту функцию вычисляет. Поступим следующим образом, рассмотрим функцию h(n)

$$h(n) = \begin{cases} P_n(n)+1, \text{если } g(n)=1\big(\text{т. е. } P_n(n)\downarrow (\text{сходится})\big) \\ 0 \quad , \text{если } g(n)=0\big(\text{т. е. } P_n(n)\not\downarrow (\text{расходится})\big) \end{cases}$$

То есть применяя «метод испорчивания по диагонали». Наша функция h(n) вычислима (при условии сделанного выше предположения вычислимости g(n)). Построенная нами функция h(n) не совпадает ни с какой другой частично вычислимой функцией, так как:

$h(x)\neq P_1(x)$ (т.к. $h(1)\neq P_1(1)$)

$h(i)\neq P_i(i)$.

Таким образом, построенная функция h не совпадает ни с какой другой частично вычислимой функцией (их все мы смогли перенумеровывать в Теореме 6.2).Таким образом, функция h не совпадает ни с какой другой вычислимой функцией.

Теорема доказана.

Данный метод доказательства («ипорчивание по диагонали») очень похож на доказательство нечетности множества действительных чисел.

Теорема 6.6 *Вещественные числа нельзя перечитать.*

Доказательство. Докажем, что нельзя пересчитать даже вещественные числа из интервала [0;1]. Пусть их можно пересчитать. $x^{(1)}, x^{(2)}$.. – последовательность, которая пересчитывает все числа из интервала [0;1]. Рассмотрим их десятичную запись:

$x^{(1)} = 0, x_1^{(1)}, x_2^{(1)} \ldots.$

$x^{(2)} = 0, x_1^{(2)}, x_2^{(2)} \ldots.$

Рассмотрим число x, которое отличается от $x^{(1)}$ в первом знаке после запятой, от $x^{(2)}$ во втором знаке после запятой и т. д. $x=0, x_1^{(1)} \oplus 1, x_2^{(1)} \oplus 1 \ldots$ ($\oplus$ - означает(a+b) mod 10). Т.к. x – целое число, оно не совпадает ни с одним из x_i.

Полученное противоречие доказывает несчетность множества вещественных чисел.

Теорема доказана.

В ***Тоереме 6.5*** мы очень похожим образом строили функцию h, которая не совпадает ни с одной частично вычислимой функцией P_i.

Заметим, что проблема останова, не будучи разрешимой, тем не менее является частично разрешимой (нетрудно промоделировать работу нашей машины при данном входе(такая функция либо выдает результат, либо зациклится).

Следствие 6.6 *Проблема неостанова не является частично разрешимой.*

Доказательство. Если бы проблема неостанова была бы частично разрешимой, то в силу доказанной выше частичной разрешимости проблемы останова и ***Теоремы 6.4*** мы получили бы, что проблема останова была бы разрешимой, но , как мы уже знаем, это не так по Теореме 6.5.

Следствие доказано.

6.4 Теорема об ускорении

Теорема 6.7 Теорема Блюма об ускорении.

Существуют функции, которые можно вычислять ве быстрее и быстрее почти всюду, а именно: пусть M_1 – некоторая МНР, вычисляющая функцию f(n), пусть $t_1(n)$ – время ее работы.

Тогда существует машина M_2, которая также вычисляет функцию f(n), причем для почти всех n время ее работы $t_2(n) \leq \varphi(t_1(n))$, где φ – произвольная функция ускорения со свойствами $\varphi(n) \to \infty$.

$n \to \infty$

Комментарий. Если $\varphi(n)=\frac{n}{100}$, то это означает, что новая машина $M2$ работает как минимум в 100 раз быстрее машины M_1.

Если $\varphi(n)=\sqrt{n}$, то ускорение уже происходит не в разы, а на порядок.

Без доказательства.

Таким образом, нашу функцию f мы можем вычислять все быстрее и быстрее, но при этом исключительное множество(множество чисел, для которых ускорение не происходит) становится все больше.

Если задать распределение вероятности входов, то существует некоторая программа, которая будет оптимальным образом на этих входах работать, то есть будет иметь минимальное математическое ожидание времени работы.

ЛАБОРАТОРНЫЕ РАБОТЫ

Сортировка массивов

1. Метод «Пузырька»
2. Метод прямого выбора(SelectSort)
3. Метод прямого слияния (MergeSort)

Преобразование Фурье

4. Дискретное преобразование Фурье.
5. «Полу-быстрое» преобразование Фурье.
6. Быстрое преобразование Фурье.

Свертки

7. Свертка с помощью обычного алгоритма.
8. Свертка с помощью быстрого алгоритма (через ДПФ).

Умножение чисел

9. Умножение чисел столбиком.
10. Быстрое умножение чисел.

Задачи на графах

11. Поиск минимального остова в связанном неориентированном графе с помощью алгоритма Краскалла.
12. Нахождение кратчайшего расстояния с помощью алгоритма Форда-Беллмана.
13. Нахождение кратчайшего расстояния с помощью алгоритма Дейкстры.

Задачи динамического программирования

14. Задача грабителя(задача «о рюкзаке»).
15. Задача о перемножении матриц.

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

Вариант расчетно-графического задания выбирается по номеру в журнале. Символ b в матрице расстояний означает ∞.

Вариант1. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант2. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

Вариант3. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант4. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i-j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант5. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

C=
$$\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & 7 \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 5 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант6. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

C=
$$\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 8 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 8 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 6 & b \end{matrix}$$

Вариант7. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^j, B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант8. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

C=
$$\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & b \\ 0 & 0 & b & 3 & 7 & b \end{matrix}$$

Вариант9. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

C=
$$\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 6 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 6 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант10. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

C=
$$\begin{matrix} b & 13 & 7 & 6 & 2 & 9 \\ 8 & b & 4 & 6 & 5 & b \\ 8 & 4 & b & 3 & 8 & 2 \\ 5 & 6 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 6 & 3 & 7 & b \end{matrix}$$

Вариант11. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант12. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант13. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант14. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i-j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант15. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & 7 \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 5 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант16. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 8 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 8 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 6 & b \end{matrix}$$

Вариант17. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^j, B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант18. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант19. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант20. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант21. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i-j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант22. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & 7 \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 5 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 7 & b \end{matrix}$$

Вариант23. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одностороннего ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 8 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 8 & 1 & 4 & b & 9 \\ 10 & 0 & 8 & 3 & 6 & b \end{matrix}$$

Вариант24. Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^j, B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

Вариант25. Написать 2 программы, решающие задачу коммивояжера. Граф G задан матрицей c[i,j]. Использовать алгоритмы прямого перебора и метод ветвей и границ (схема одновременного ветвления). Сравнить трудоемкость двух алгоритмов.

$$C=\begin{matrix} b & 13 & 7 & 5 & 2 & 9 \\ 8 & b & 4 & 7 & 5 & b \\ 8 & 4 & b & 3 & 6 & 2 \\ 5 & 8 & 1 & b & 0 & 1 \\ b & 6 & 1 & 4 & b & b \\ 10 & 0 & b & 3 & 7 & b \end{matrix}$$

Ответы к домашним заданиям

Домашние задание №1. Отсортировать функции по скорости роста, начиная с наименьшей:

$$log^3 n, 2^n, 3^{\sqrt{n}}, 3^{logn}, n \cdot logn, \frac{n^2}{log^4 n}, 2^{n!}, (2^n)!, n^{n^{2^n}}, 2^{3^{2^{2^n}}}$$

Решение.

Для примера сравним скорости роста функций $2^{n!}$ и $n^{n^{2^n}}$.

1)Прологарифмируем оба выражения по основанию 2:

n! и $n^{2^n} \cdot logn$

2)Используя формулу Стирлинга: $n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$, получим

$\left(\frac{n}{e}\right)^n \sqrt{2\pi n}$ и $n^{2^n} \cdot logn.$ Далее:

3)Так как значение некоторых частей выражений много меньше значения других, то их можно рассматривать:

$\frac{1}{2}\cancel{\log(2\pi n)} + (nlogn - \cancel{nloge})$ и $\cancel{logn} + 2^n \cdot logn.$

Т.к. nloge≪nlogn

Получаем: $nlogn \ll 2^n \cdot logn$

$n \ll 2^n$.

Аналогичное неравенство выполняется и для исходных функций.

$n! \ll n^{2^n} \cdot logn$

Следовательно, $2^{n!} \ll n^{n^{2^n}}$.

Функция $n^{n^{2^n}}$ растет быстрее, чем $2^{n!}$.

Сравнивая остальные функции аналогичным образом, получаем сведущую последовательность:

Ответ: $log^3 n \ll n \cdot logn \ll 3^{logn} \ll \frac{n^2}{log^4 n} \ll 3^{\sqrt{n}} \ll 2^n \ll (2^n)! \ll 2^{n!} \ll n^{n^{2^n}} \ll 2^{3^{2^{2^n}}}$

Домашние задание №2.

Выразить $A^3(1,0,1,1)$ через A^2 и эспоненту по формуле быстрого преобразования Фурье. $N=2^4$.

Решение.

$A^{(s+1)}(k_1,\dots,k_{s+1}1,j_{s+2},\dots,j_r)=\frac{1}{2}\cdot\sum_{j_{s+1}=0}^{1}A^{(s)}(k_1,\dots,k_{s+1}1,j_{s+2},\dots,j_r)\cdot$
$exp\{-2\pi i(\sum_{l=0}^{s+1}k_l\,2^{l-1})2^{-(s+1)}j_{s+1}\}.$

r=4;

S=2;

k_1=1;

k_2=0;

k_3=1;

k_4=1;

$$\sum_{l=0}^{3}k_t2^{t-1}=k_12^0+k_22^1+k_32^2=1\cdot1+0\cdot2+1\cdot4=5.$$

У нас имеются все данные, которые нужно подставить в формулу:

$A^3(1,0,1,1)=\frac{1}{2}\left(A^2(1,0,1,1)\cdot exp\left\{-2\pi i\cdot\frac{1}{8}\cdot5\cdot0\right\}+A^2(1,0,1,1)\cdot exp\left\{-2\pi i\cdot\frac{1}{8}\cdot5\cdot1\right\}\right)=\frac{1}{2}\cdot(A^2(1,0,1,1)\cdot$
$exp\{0\}+A^2(1,0,1,1)\cdot exp\left\{-\frac{5\pi i}{4}\cdot1\right\})=\frac{A^2(1,0,1,1)+A^2(1,0,1,1)\cdot exp\left\{-\frac{5\pi i}{4}1\right\}}{2}$

Ответ: $A^3(1,0,1,1)=\frac{A^2(1,0,1,1)+A^2(1,0,1,1)\cdot exp\left\{-\frac{5\pi i}{4}1\right\}}{2}$.

Домашние задание №3. Найти произведение 3871 и 9211 по формуле быстрого умножения чисел.

Решение.

Разобьем наши числа на разряды по формуле:

$x=a\cdot2^k+b$

$y=c\cdot2^k+d$.

Формула быстрого умножения чисел имеет вид:

$$\begin{cases}U=(a+b)(c+d)\\V=a\cdot c\\W=b\cdot d\end{cases}$$

$x\cdot y=V\cdot2^{2k}+(U-V-W)\cdot10^k+W$.

Если в процессе вычисления мы получаем(k+1) разрядные числа, то их произведение вычисляется по формуле (*):

$a\cdot b=a_1\cdot b_1+(a_1\cdot b_2\ a_2\cdot b_1)2k+a_2\cdot b_2$

В данном случае k=2.

Получаем $x=38\cdot10^2+71$, $y=92\cdot10^2+11$.

$$\begin{cases}U=(38+71)(92+11)\\V=38\cdot92\\W=71\cdot11\end{cases}$$

1)Рассчитываем U:

U=(38+72)(92+11)=109·103(двухразрядное умножение с двумя переполнениями).

Получаем трехразрядные числа. Следует воспользоваться формулой (*), где

$a_1=1$, $a_2=9$;

$b_1=1$, $b_2=3$.

$109\cdot103=1\cdot1\cdot10^4+(09+03)\cdot10^2+09\cdot03=11227$.

Получили 09·03 – двухразрядное умножение без переполнения:

$$\begin{cases} U_1=(0+9)(0+3)=9\cdot3\ (\text{одноразрядное без переполнений}) \\ V_1=0\cdot0(\text{одноразрядное без переполнений}) \\ W_1=9\cdot3\ (\text{одноразрядное без переполнений}) \end{cases}$$

$09\cdot03=0\cdot10^2+(27-0-27)\cdot10+27=27$

Таким образом, $U=1\cdot1\cdot10^4+(09+03)10^2+09\cdot03=11227$.

2)Рассчитаем V:

V=38·92 (двух разрядное умножение без переполнений)

$$\begin{cases} U_2=(3+8)(9+2)=11\cdot11\ (\text{одноразрядное с двумя переполнениями}) \\ V_2=3\cdot9=27(\text{одноразрядное без переполнений}) \\ W_2=8\cdot2\ (\text{одноразрядное без переполнений}) \end{cases}$$

В U_2 мы получили одноразрядное с двумя переполнениями.

Вычисляем его по формуле(*):

$a_1=1$, $a_2=1$;

$b_1=1$, $b_2=1$.

$U_2=11\cdot11=1\cdot1\cdot10^2+(1\cdot1+1\cdot1)\cdot10^1+1\cdot1=121$.

Таким образом, $V=38\cdot92=27\cdot10^2+(121-27-16)\cdot10^1+16=3496$.

3)Рассчитаем W:

W=71·11(двух разрядное умножение без переполнений)

$$\begin{cases} U_3=(7+1)(1+1)=8\cdot2\ (\text{одноразрядное без переполнений}) \\ V_3=7\cdot1=7(\text{одноразрядное без переполнений}) \\ W_3=1\cdot1=1\ (\text{одноразрядное без переполнений}) \end{cases}$$

Таким образом, $W=71\cdot11=7\cdot10^2+(16-7-1)\cdot10+1=781$.

Теперь мы можем вычислить $3871\cdot9211=3496\cdot10^4+(11227-3496-781)\cdot10^2+781=35655781$.

Ответ: 3871·9211=35655781.

Домашние задание №4.

Найти произведение 8329 и 5631 по формуле быстрого умножения чисел.

Решение.

Разобьем наши числа на разряды по формуле:

$x=a\cdot 2^k+b$

$y=c\cdot 2^k+d$.

Формула быстрого умножения чисел имеет вид:

$$\begin{cases} U=(a+b)(c+d) \\ V=a\cdot c \\ W=b\cdot d \end{cases}$$

$x\cdot y=V\cdot 2^{2k}+(U-V-W)\cdot 10^k+W$.

Если в процессе вычисления мы получаем(k+1) разрядные числа, то их произведение вычисляется по формуле (*):

$a\cdot b=a_1\cdot b_1+(a_1\cdot b_2\ a_2\cdot b_1)2k+a_2\cdot b_2$

В данном случае k=2. Получаем $x=83\cdot 10^2+29$, $y=56\cdot 10^2+31$.

$$\begin{cases} U=(83+29)(56+31) \\ V=83\cdot 56 \\ W=29\cdot 31. \end{cases}$$

1)<u>Рассчитаем U:</u>

$U=(83+29)(56+31)=112\cdot 87$(двухразрядное умножение с одним переполнением)

Следует воспользоваться формулой (*):

$a_1=1$, $a_2=12$;

$b_1=0$, $b_2=87$.

$a\cdot b=112\cdot 87=1\cdot 0\cdot 10^4+(1\cdot 87+12\cdot 0)\cdot 10^2+12\cdot 87$.

Получили $12\cdot 87$ – двухразрядное умножение без переполнений:

$$\begin{cases} U_1=(1+2)(8+7)=3\cdot 15\ \text{(одноразрядное с одним переполнением)} \\ V_1=1\cdot 8\text{(одноразрядное без переполнений)} \\ W_1=2\cdot 7\ \text{(одноразрядное без переполнений)} \end{cases}$$

Вычислим U_1 по формуле (*):

$a'_1=0$, $a'_2=3$;

$b'_1=1$, $b'_2=5$.

$a'\cdot b'=3\cdot 15=0\cdot 1\cdot 10^2+(0\cdot 5+3\cdot 1)\cdot 10^1+3\cdot 5=45$.

Теперь мы можем вычислить$12\cdot 87=8\cdot 10^2+(45-8-14)\cdot 10^1+14=1044$ и досчитать

$a\cdot b=112\cdot 87=87\cdot 10^2+(1\cdot 87+12\cdot 0)\cdot 10^1+12\cdot 87=9744$

Таким образом, $U=87\cdot 10^2+(1\cdot 87+12\cdot 0)\cdot 10^1+12\cdot 87=9744$

2) <u>Рассчитаем V:</u>

$V=83\cdot 56$(двухразрядное умножение без переполнений)

$$U_2 = (8+3)(5+6) = 11 \cdot 11 \text{ (одноразрядное с двумя переполнениями)}$$
$$V_2 = 8 \cdot 5 = 40 \text{(одноразрядное без переполнений)}$$
$$W_2 = 3 \cdot 6 = 16 \text{ (одноразрядное без переполнений)}$$

U_2 мы получили одноразрядное умножение с двумя переполнениями. Вычисляем его по формуле(*):

$a_1=1$, $a_2=1$;

$b_1=1$, $b_2=1$.

$U_2=11\cdot11=1\cdot1\cdot10^2+(1\cdot1+1\cdot1)\cdot10^1+1\cdot1=121$.

Таким образом, $U_2=83\cdot56=40\cdot10^2+(121-40-18)\cdot10^1+18=4648$.

3)Рассчитаем W:

W=29·31 (двухразрядное умножение без переполнений)

$$\begin{cases} U_3 = (2+9)(3+1) = 11 \cdot 4 \text{ (одноразрядное без переполнений)} \\ V_3 = 2 \cdot 3 = 6 \text{(одноразрядное без переполнений)} \\ W_3 = 9 \cdot 1 = 9 \text{ (одноразрядное без переполнений)} \end{cases}$$

Вычисляем U_3 по формуле (*):

$a_1=1$, $a_2=1$;

$b_1=0$, $b_2=4$.

$11\cdot4=1\cdot0\cdot10^2+(1\cdot4+1\cdot0)\cdot10^1+1\cdot4=44$.

Таким образом, $W=2931=6\cdot10^2+(44-6-9)\cdot10^1+9=899$

Теперь мы можем вычислить $8329\cdot5631=4648\cdot10^4+(9744-4648-899)\cdot10^2+899=4690059$.

Ответ: 8329·5631=4690059.

Домашние задание №5(А,Б,В).

А) Найти сотов минимального веса для связанного взвешенного неориентированного графа, заданного матрицей стоимостей переездов из одной вершины в другую:

0	2	7	4	6	3
2	0	4	5	6	1
7	4	0	8	7	∞
4	5	8	0	5	7
6	6	7	5	0	3
3	1	∞	7	3	0

Решение.

1) Упорядочить все ребра графа по возрастанию их стоимости:

(1,5)=1;(0,1)=2;(0,5)=3;(4,5)=3;(0,3)=4; (1,2)=4; (1,3)=5; (3,4)=5; (0,4)=6; (1,4)=6; (0,2)=7; (2,4)=7; (3,5)=7; (2,3)=8; (2,5)=∞;

2) Создаем таблицу: слева ребра, справа компоненты связанности.
Изначально список пуст и все компоненты связанности одновершинные. Берем минимальное по стоимости ребро и включаем его в список ребер. Две соответствующие вершины объединяем в одну компоненту связанности. Если обе вершины ребра уже принадлежат какой либо компоненте связанности, то данное ребро мы бракуем.
Данную процедуру повторяем, пока все вершины не окажутся в одной компоненте связанности (для этого потребуется включить в наш остов ровно n-1=4 ребра).

Подграф(ребра)	Связи(компоненты связанности)
0	0,1,2,3,4,5
(1,5)- минимально по цене	0,{1,5},2,3,4
(1,5)+(0,1)	{0,1,5},2,3,4
(1,5)+(0,1)+(0,5)	{0,1,5},2,3,4
(1,5)+(0,1)+(4,5)	{0,1,4,5},2,3
(1,5)+(0,1)+(4,5)+(0,3)	{0,1,3,4,5},2
(1,5)+(0,1)+(4,5)+(0,3)+(1,2)	{0,1,2,3,4,5}

Таким образом, мы нашли остов минимального веса(его вес равен 14) для заданной матрицы стоимостей:

Б) Найти кратчайшие расстояния от второй до все остальных вершин графа,заданного матрицей стоимостей переездов из одной вершины в другую

Б1) Решение:

Формируем первоначальный массив стоимостей: $D^{[0]}=(\infty, \infty,0, \infty,\infty,\infty)$.

Стоимость вершины i на каждом шаге считаем по формуле:

$D^{[k+1]}(i)=\min(D^{[k]}(0)+C(0,i), D^{[k]}(1)+ C(1,i), D^{[k]}(2)+ C(2,i), D^{[k]}(3)+ C(3,i), D^{[k]}(4)+ C(4,i), D^{[k]}(5)+ C(5,i))$

Вычислим стоимости вершин данного графа на первом шаге:

$D^{[1]}(0)=\min(D^{[0]}(0)+C(0,0), D^{[0]}(1)+ C(1,0), D^{[0]}(2)+ C(2,0), D^{[0]}(3)+ C(3,0), D^{[0]}(4)+ C(4,0), D^{[0]}(5)+ C(5,0))=\min(\infty, \infty,2, \infty,\infty,\infty)=2.$

$D^{[1]}(1)=\min(D^{[0]}(0)+C(0,1), D^{[0]}(1)+ C(1,1), D^{[0]}(2)+ C(2,1), D^{[0]}(3)+ C(3,1), D^{[0]}(4)+ C(4,1), D^{[0]}(5)+ C(5,1))=\min(\infty+2, \infty+0,0+2, \infty+\infty,\infty+\infty,\infty+2)=2.$

$D^{[1]}(2)=\min(D^{[0]}(0)+C(0,2), D^{[0]}(1)+ C(1,2), D^{[0]}(2)+ C(2,2), D^{[0]}(3)+ C(3,2), D^{[0]}(4)+ C(4,2), D^{[0]}(5)+ C(5,2))=\min(\infty+7, \infty+0,0+0, \infty+8,\infty+7)=0.$

$D^{[1]}(3)=\min(D^{[0]}(0)+C(0,3), D^{[0]}(1)+ C(1,3), D^{[0]}(2)+ C(2,3), D^{[0]}(3)+ C(3,3), D^{[0]}(4)+ C(4,3), D^{[0]}(5)+ C(5,3))=\min(\infty, \infty,8, \infty,\infty,\infty)=8.$

$D^{[1]}(4)=\min(D^{[0]}(0)+C(0,4), D^{[0]}(1)+ C(1,4), D^{[0]}(2)+ C(2,4), D^{[0]}(3)+ C(3,4), D^{[0]}(4)+ C(4,4), D^{[0]}(5)+ C(5,4))=\min(\infty, \infty,7, \infty,\infty,\infty)=7.$

$D^{[1]}(5)=\min(D^{[0]}(0)+C(0,5), D^{[0]}(1)+ C(1,5), D^{[0]}(2)+ C(2,5), D^{[0]}(3)+ C(3,5), D^{[0]}(4)+ C(4,5), D^{[0]}(5)+ C(5,5))=\min(\infty, \infty,\infty, \infty,\infty,\infty)=\infty.$

Вычислим стоимости вершин данного графа на втором шаге:

$D^{[2]}(0)=\min(D^{[1]}(0)+C(0,0), D^{[1]}(1)+ C(1,0), D^{[1]}(2)+ C(2,0), D^{[1]}(3)+ C(3,0), D^{[1]}(4)+ C(4,0), D^{[1]}(5)+ C(5,0))=\min(2+0,4+3,0+2,8+4,7+\infty,\infty+2)=2.$

$D^{[2]}(1)=\min(D^{[1]}(0)+C(0,1), D^{[1]}(1)+C(1,1), D^{[1]}(2)+C(2,1), D^{[1]}(3)+C(3,1), D^{[1]}(4)+C(4,1), D^{[1]}(5)+C(5,1))=\min(2+2,4+0,0+4, 8+\infty,7+7,\infty+4)=4.$

$D^{[2]}(3)=\min(D^{[1]}(0)+C(0,3), D^{[1]}(1)+C(1,3), D^{[1]}(2)+C(2,3), D^{[1]}(3)+C(3,3), D^{[1]}(4)+C(4,3), D^{[1]}(5)+C(5,3))=\min(2+4,4+5,0+8,8+0,7+4,\infty+7)=6.$

$D^{[2]}(4)=\min(D^{[2]}(0)+C(0,4), D^{[2]}(1)+C(1,4), D^{[2]}(2)+C(2,4), D^{[2]}(3)+C(3,4), D^{[2]}(4)+C(4,4), D^{[2]}(5)+C(5,4))=\min(2+6,4+6,0+7,8+5,7+0,\infty+8)=7.$

$D^{[2]}(5)=\min(D^{[2]}(0)+C(0,5), D^{[2]}(1)+C(1,5), D^{[2]}(2)+C(2,5), D^{[2]}(3)+C(3,5), D^{[2]}(4)+C(4,5), D^{[2]}(5)+C(5,5))=\min(2+3,4+1,0+\infty, 8+7,7+3,\infty+0)=5.$

Вычислим стоимости вершин данного графа на третьем шаге:

$D^{[3]}(0)=\min(D^{[2]}(0)+C(0,0), D^{[2]}(1)+C(1,0), D^{[2]}(2)+C(2,0), D^{[2]}(3)+C(3,0), D^{[2]}(4)+C(4,0), D^{[2]}(5)+C(5,0))=\min(2+0,4+3,0+2,6+4,7+\infty,5+2)=2.$

$D^{[3]}(1)=\min(D^{[2]}(0)+C(0,1), D^{[2]}(1)+C(1,1), D^{[2]}(2)+C(2,1), D^{[2]}(3)+C(3,1), D^{[2]}(4)+C(4,1), D^{[2]}(5)+C(5,1))=\min(2+2,4+0,0+4, 6+\infty,7+7,5+4)=4.$

$D^{[3]}(3)=\min(D^{[2]}(0)+C(0,3), D^{[2]}(1)+C(1,3), D^{[2]}(2)+C(2,3), D^{[2]}(3)+C(3,3), D^{[2]}(4)+C(4,3), D^{[2]}(5)+C(5,3))=\min(2+4,4+5,0+8,6+0,7+4,5+7)=6.$

$D^{[3]}(4)=\min(D^{[2]}(0)+C(0,4), D^{[2]}(1)+C(1,4), D^{[2]}(2)+C(2,4), D^{[2]}(3)+C(3,4), D^{[2]}(4)+C(4,4), D^{[2]}(5)+C(5,4))=\min(2+6,4+6,0+7,6+5,7+0,5+8)=7.$

$D^{[3]}(5)=\min(D^{[2]}(0)+C(0,5), D^{[2]}(1)+C(1,5), D^{[2]}(2)+C(2,5), D^{[2]}(3)+C(3,5), D^{[2]}(4)+C(4,5), D^{[2]}(5)+C(5,5))=\min(2+3,4+1,0+\infty,6+7,7+3,5+0)=5.$

Массив стоимостей вершин перестал меняться. Таким образом, мы получили кратчайшее расстояние от всех вершин данного графа до второй вершины.

Б2) Решение:

Первоначально в множество S включаем только ту вершину, от которой требуется найти расстояние до всех других вершин графа. В нашем случае это вершина $V_2=2$.

Считаем стоимости проезда от вершины 2 до всех остальных вершин:

$D[V_0]=D[0]=C(V_2,V_0)=C(2,0)=2;$

$D[V_1]=D[1]=C(V_2,V_1)=C(2,1)=4;$

$D[V_3]=D[3]=C(V_2,V_3)=C(2,3)=8;$

$D[V_4]=D[4]=C(V_2,V_4)=C(2,4)=7;$

$D[V_5]=D[5]=C(V_2,V_5)=C(2,5)=\infty;$

S	W	D[W]	D[0]	D[1]	D[3]	D[4]	D[5]
2	-	-	2	4	8	7	∞

Далее выбираем среди всех вершин графа, за исключением вершины $V_2=2$, минимальную по тоимости:

$D[W]=\min(D[V_0], D[V_1], D[V_3], D[V_4], D[V_5])=\min(2,4,8,7,\infty)=2,$

$W=V_0=0;$

Пересчитываем стоимости проезда от вершины 0 до всех остальных вершин, кроме вершины $V_0=0$:

$D[V_1]=D[1]=\min(D[V_1],D[W]+C(V_0,V_1))=\min(4,2+C(0,1))=\min(4,2+2)=4$

$D[V_3]=D[3]=\min(D[V_3],D[W]+C(V_0,V_3))=\min(8,2+C(0,3))=\min(8,2+4)=6$

$D[V_4]=D[4]=\min(D[V_4],D[W]+C(V_0,V_4))=\min(7,2+C(0,4))=\min(7,2+6)=7$

$D[V_5]=D[5]=\min(D[V_5],D[W]+C(V_0,V_5))=\min(\infty,2+C(0,5))=\min(\infty,2+3)=5$

S	W	D[W]	D[0]	D[1]	D[3]	D[4]	D[5]
2	-	-	2	4	8	7	∞
2,0	0	2	-	4	6	7	5

Выбираем среди всех вершин графа, за исключением 0, 2, минимальную по стоимости:

$D[W]=\min(D[V_1], D[V_3], D[V_4], D[V_5])=\min(4,6,7,5)=4$,

$W=V_1=1$

Пересчитываем стоимости проезда от вершины 2 до всех остальных вершин, кроме вершины $V_0=0$, $V_1=1$.

$D[V_3]=D[3]=\min(D[V_3],D[W]+C(V_1,V_3))=\min(6,4+C(1,2))=\min(6,4+5)=6$

$D[V_4]=D[4]=\min(D[V_4],D[W]+C(V_1,V_4))=\min(7,4+C(1,4))=\min(7,4+6)=7$

$D[V_5]=D[5]=\min(D[V_5],D[W]+C(V_1,V_5))=\min(5,4+C(1,5))=\min(5,4+1)=5$

S	W	D[W]	D[0]	D[1]	D[3]	D[4]	D[5]
2	-	-	2	4	8	7	∞
2,0	0	2	-	4	6	7	5
2,0,1	1	4	-	-	6	7	5

Выбираем среди всех вершин графа, за исключением 0, 2, 1 минимальную по стоимости:

$D[W]=\min(D[V_3], D[V_4], D[V_5])=\min(6,7,5)=5$,

$W=V_5=5$

Пересчитываем стоимости проезда от вершины 5 до всех остальных вершин, кроме вершины $V_0=0$, $V_1=1$, $V_2=2$ $V_5=5$

$D[V_3]=D[3]=\min(D[V_3],D[W]+C(V_5,V_3))=\min(6,5+C(5,3))=\min(6,5+7)=6$

$D[V_4]=D[4]=\min(D[V_4],D[W]+C(V_5,V_4))=\min(7,5+C(5,4))=\min(7,5+8)=7$

И, наконец $D[W]=\min(D[V_3], D[V_4])=\min(6,7)=6$,

$W=V_3=3$

S	W	D[W]	D[0]	D[1]	D[3]	D[4]	D[5]
2	-	-	2	4	8	7	∞
2,0	0	2	-	4	6	7	5
2,0,1	1	4	-	-	6	7	5
2,0,1,5	5	5	-	-	6	7	-
2,0,1,5,3	3	6	-	-	-	7	-
2,0,1,5,3,4	4	7	-	-	-	-	-

В результате мы получили таблицу, содержащую кратчайшее расстояние от вершины V_2 до всех остальных вершин данного графа.

В) Определить оценку H, если первоначальный маршрут (4,0,3){1,5}.

Матрица стоимостей переездов:

0	2	7	4	6	3
3	0	4	5	6	1
2	4	0	8	7	∞
4	∞	8	0	5	7
∞	7	8	4	0	3
2	4	∞	7	8	0

Решение.

1) Вычеркиваем нулевую и четвертую строки, так как мы уже выехали из нулевоц и четвертой вершин.
2) Вычеркиваем нулевой и третий столбцы, так как мы уже выезжали в нулевую и третью вершины.
   Получаем следующую матрицу:

	1	2	4	5
1|	∞	4	6	1
2|	4	∞	7	∞
3|	∞	8	5	∞
5|	4	∞	8	∞

Стоимость переезда (3,5) полагаем равным ∞, т. к. он запрещен.

1) В каждой строке находи минимальный элемент (стоимость выезда):

   α_i=1,4,5,4. Выплачиваем найденные стоимости выездов, т. е. все элементы соответствующие строки уменьшаем на α_i:

	1	2	4	5
1|	∞	3	5	0
2|	0	∞	3	∞
3|	∞	3	0	∞
5|	0	∞	4	∞

2) В каждом столбце находим минимальный элемент (стоимость въезда):

   β_i=0,3,0,0.

3) Вычисляем наименьшую стоимость маршрута:

   H(D)=C(4,0)+C(0,3)+α(1)+α(2)+α(3)+α(5)+β(1)+β(2)+β(4)+β(5)=∞+4+(1+4+5+4)+(0+3+0+0)=∞.

   Ответ: H(D)=∞.

Домашнее задание №6(А,Б)

А) Задача грабителя (о рюкзаке). Имеется склад, на котором присутствует ассортимент товаров (каждого товара неограниченный запас). У каждого товара своя стоимость C_i и масса m_i. Выбрать набор товаров так, чтобы его суммарный вес не превышал заданную грузоподъемность M притом, что суммарная стоимость этого набора товаров была бы максимальной.

Номер товара, i	m_i	C_i
1	5	9

2	7	13
3	11	21

Максимальная грузоподъемность: M=23;24.

Решение.

Вычислим ***f(M)*** - максимально возможную **СТОИМОСТЬ** товаров при грузоподъемности ***M:***

f(0)=0;
f(1)=0;
f(2)=0;
f(3)=0;
f(4)=0;
f(5)=max(f(5-5)+9,f(5~7)+l3,f(5-ll)+21)=max(0+9)=9;
f(6)=max(f(6-5)+9, f(6~7)+J3,f(6-ll)+21)= max(0+9)=9;
f(7)=max(f(7-5)+9, f(7-7)+l3,f(7-U)+21)= max(0+9, 0+13)=13;
f(8)= (f(8-5)+9, f(8~7)+l3,f(8-11)+21)= max(0+9, 0+13)=13;
f(9)= max(f(9-5)+9, f(9~7)+l3,f(9-11)+21)=max(0+9, 0+13)=13;
f(10)=max(f(10-5)+9, f(10~7)+13,f(10-ll)+21)=max(9+9, 0+13)=18;
f(U)=max(f(ll-5)+9, f(ll~7)+13,f(ll-ll)+21)=max(9+9, 0+13, 0+21)=21;
f(12)=max(f(12-5)+9, f(12-7)+13,f(12-ll)+21)= max(13+9, 9+13, 0+21)=22;
f(13)=max(f(13-5)+9, f(13~7)+13,f(13-ll)+21)=max(13+9, 9+13, 0+21)=22;
f(14)=ma(f(14-5)+9, f(14-7)+13,f(14-U)+21)=max(13+9, 13+13, 0+21)=26;
f(15)=max(f(15~5)+9, f(15-7)+13,f(15-ll)+21)=max(18+9, 13+13, 0+21)=27;
f(16)=max(f(16-5)+9, f(16~-7)+13,f(16-ll)+21)=max(21+9, 13+13, 9+21)=30;
f(17)=max(f(17-5)+9, f(17-7)+13,f(17-U)+21)=max(22+9, 18+13, 9+21)=31;
f(18)=max(f(18^5)+9, f(18-7)+13,f(18-ll)+21)= max(22+9, 21+13, 13+21)=34;
f(19)=max(f(19-5)+9, f(19~7)+13,f(19-U)+21)=max(26+9, 22+13, 13+21)=35;
f(20)=max(f(20-5)+9, f(20-7)+13,f(20-ll)+21)=max(27+9, 22+13, 13+21)=36;
f(21)=max(f(21-5)+9, f(21-7)+13,f(21-U)+21)=max(30+9, 26+13, 18+21)=9;
f(22)=max(f(22)+9, f(22)+13,f(22-ll)+21)=max(31+9, 27+13, 21+21)=42;
f(23)=max<f(23-5)+9, f(23-7)+13,f(23-U)+21)=max(34+9,30+13, 22+21)=43;
f(24)=max(f(24-5)+9, f(24-7)+13,f(24-ll)+21)= max(35+9, 31+15, 22+21)=44.

Определим оптимальный набор товаров при M=23:

f(23)=43;

f(23-5)+9=f(18)+9=34+9=43; => берём товар (m_1=5 , C_1=9).

Новая грузоподъемность M=23-5=18

f(18)=34;

f(18-5)+9= f(13)+9=22+9=31; $\Rightarrow$ берём товар (m_1=5 , C_1=9), так как 34≠31

f(18)=34;

f(28-7)+13= f(11)+13=21+13=34; берём товар (m_2=7 , C_2=13)

Новая грузоподъемность M=18-7=11

f(11)=21;

f(11-5)+9= f(6)+9=9+9=18; $\Rightarrow$ *не* берём товар (m_1=5 , C_1=9), так как 18≠21

f(11)=21;

f(11-7)+13= f(4)+13=0+13=13;⇒не берём товар (m_2=7 , C_2=13), так как 13≠21

f(11)=21;

f(11-11)+21= f(0)+21=0+21=21;⇒ берём товар (m_3=11 , C_3=21)

Новая грузоподъемность M=11-11=0⇒ грузоподъемность исчерпана.

Получили оптимальный набор товаров при M=23;

1 товар (m_1=5 , C_1=9),

1товар (m_2=7 , C_2=13)

1товар (m_3=11 , C_3=21).

Определим оптимальный набор товар при M=24:

f(24)=44;

f(24-5)+9= f(19)+9=35+9=44; ⇒ берём товар (m_1=5 , C_1=9)

Новая грузоподъемность M=24-9=19

f(19)=35;

f(19-5)+9= f(14)+9=26+9=35; ⇒ берём товар (m_1=5 , C_1=9)

Новая грузоподъемность M=19-5=14

f(14)=26;

f(14-5)+9= f(9)+9=13+9=22; ⇒ берём товар (m_1=5 , C_1=9) т к 22≠26

f(14)=26;

f(14-7)+13= f(7)+13=13+13=26; ⇒ берём товар (m_2=7 , C_2=13)

Новая грузоподъемность M=14-7=7

f(7)=13;

f(7-5)+9= f(2)+9=0+9=9; ⇒ не берём товар (m_1=5 , C_1=9) т к 9≠13

f(7)=13;

f(7-7)+13= f(0)+13=0+13=13; ⇒ берём товар (m_2=7 , C_2=13)

Новая грузоподъемность M=7-7=0⇒ грузоподъемность исчерпана. Получили оптимальный набор товаров при M=24:

2 товара (m_1=5 , C_1=9)

2 товара(m_2=7 , C_2=13)

Б) Расставить скобки при перемножении матриц оптимальным образом.

M_1=[10×20], M_2=[20×5], M_3=[5×4], M_4=[4×30], M_5=[30×6].

Решение.

r_0=10, r_1=20, r_2=5, r_3=4, r_5=6.

Вычислим оптимальные трудоемкости перемножения матриц:

f(1,1)= f(2,2)= f(3,3)= f(4,4)= f(5,5)=0.

f(1,2)= f(1,1)+ f(2,2)+ $r_0 \cdot r_1 \cdot r_2$=0+0+10·20·5=1000;

f(2,3)= f(2,2)+ f(3,3)+ $r_0 \cdot r_1 \cdot r_2$=0+0+20·4·5=400;

f(3,4)= f(3,3)+ f(4,4)+ $r_0 \cdot r_1 \cdot r_2$=0+0+5·4·30=600;

f(4,5)= f(4,4)+ f(5,5)+ $r_0 \cdot r_1 \cdot r_2$=0+0+4·30·6=720;

f(1,3)=min(f(1,1)+ f(2,3)+ $r_0 \cdot r_1 \cdot r_2$=0+400+10·20·4=1200;*f(1,2)+ f(3,3)+* $r_0 \cdot r_1 \cdot r_2$=1000+0+10·4·5=1200)=1200

f(2,4)=min(f(2,2)+ f(3,4)+ $r_1 \cdot r_2 \cdot r_4$=0+600+20·5·30=3600;*f(2,3)+ f(4,4)+* $r_1 \cdot r_3 \cdot r_4$=400+0+20·4·30=2800)=2800

f(3,5)=min(f(3,3)+ f(4,5)+ $r_2 \cdot r_3 \cdot r_5$=0+720+5·4·6=840;*f(3,4)+ f(5,5)+* $r_2 \cdot r_4 \cdot r_5$=600+0+5·30·6=900)=840

f(1,4)=min(f(1,1)+ f(2,4)+ $r_0 \cdot r_1 \cdot r_4$=0+2800+10·20·30=8800;*f(1,2)+ f(3,4)+* $r_0 \cdot r_2 \cdot r_4$=1000+600+10·5·30=3100; *f(1,3)+ f(4,4)+* $r_0 \cdot r_3 \cdot r_4$=1200+0+10·4·30=2400)=2400;

f(2,5)=min(f(2,2)+ f(3,5)+ $r_1 \cdot r_2 \cdot r_5$=0+840+20·5·6=1440;*f(2,3)+ f(4,5)+* $r_1 \cdot r_3 \cdot r_5$=400+720+20·4·6=1600; *f(2,4)+ f(5,5)+* $r_1 \cdot r_4 \cdot r_5$=2800+0+20·30·6=6400)=1400;

f(1,5)=min(f(1,1)+ f(2,5)+ $r_0 \cdot r_1 \cdot r_5$=0+1440+10·20·6=2640;*f(1,2)+ f(3,5)+* $r_0 \cdot r_2 \cdot r_5$=1000+840+10·5·6=2140; *f(1,3)+ f(4,5)+* $r_0 \cdot r_3 \cdot r_5$=1200+720+10·4·6=2160;

f(1,4)+ f(5,5)+ $r_0 \cdot r_4 \cdot r_5$=2400+0+10·30·6=4200)=2140;

Восстановим оптимальную расстановку скобок:

min f(1,5) достигнут на f(1,2)+ f(3,5):

$(M_1 M_2) \cdot (M_3 M_4 M_5)$

min f(3,5) достигнут на f(3,3)+ f(4,5):

$(M_1 M_2) \cdot (M_3 (M_4 M_5))$

Ответ: $(M_1 M_2) \cdot (M_3 (M_4 M_5))$

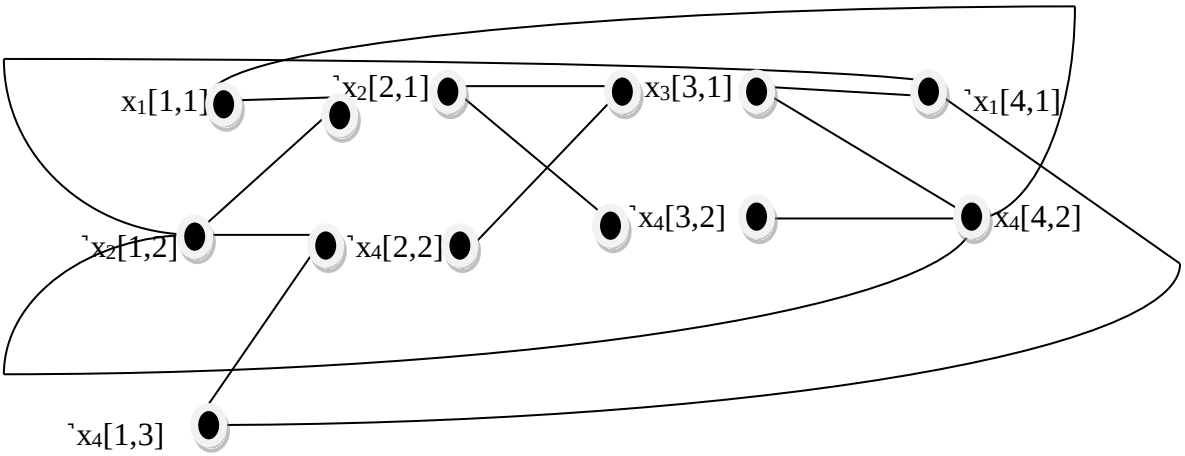
Домашние задание№7

По заданной в КНФ Z построить граф G и найти в нем возможные «клики».

$Z=(x_1 \vee \bar{x}_2 \vee \bar{x}_4)\&(\bar{x}_2 \vee \bar{x}_4)\&(x_3 \vee \bar{x}_4)\&(\bar{x}_1 \vee \bar{x}_4)$

Решение.

В графе будет 9 вершин



Построим для Z таблицу истинности.

X_1	X_2	X_3	X_4	Z
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0

1	1	1	1	0

По таблице нетрудно заметить, что Z=1 на шести наборах переменных. Значит, данный граф имеет ровно 6 «клик»:

1. [1,1],[2,1],[3,1],[4,2]
2. [1,2],[2,1],[3,1],[4,1]
3. [1,2],[2,1],[3,1],[4,2]
4. [1,2],[2,1],[3,2],[4,2]
5. [1,2],[2,2],[3,1],[4,1]
6. [1,3],[2,2],[3,1],[4,1]

Ответ: Получили 6 вышесказанных «клик».

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Ахо А., Ульман Дж., Хопкрофт Дж. Построение и анализ алгоритмов. –М.Мир, 1987, стр.520
2. Томас Кормен. Алгоритмы. Построение и анализ. Второе издание. Москва-Санкт-Петербург-Киев,2005, стр 1296.
3. H.S. Wilf Algorithms and complexity. Internet Edition,1994,стр 136
4. М. Гэри, Д.Джонсон. Вычислительные машины и трудно решаемые задачи. Москва «Мир» 1982 стр 416.
5. Носов НН Теория алгоритмов и анализа их сложности.

   http://intsys.msu.ru/stuff/vnosov/theoralg.htm

6. Кузюрин НН Курс лекций «Сложность комбинаторных алгоритмов»

   http://discopal.ispras.ru/ru.lectures.htm