

1. Скорость роста длины записи коэффициентов при реализации метода Гаусса

Думаю билет и так большой, можно это не писать:

Метод Гаусса является точным, но неустойчивым. При его применении к матрицам с элементами типа “real” разработаны различные методы “борьбы” с неустойчивостью метода (операция деления в компьютере выполняется приближенно, младшие разряды числа обрезаются). Все элементы каждой строки массива типа `real` можно умножить на общий знаменатель элементов строки и получить целочисленную матрицу коэффициентов равносильной системы. Так не поступают из-за большой скорости роста элементов матрицы в процессе применения метода Гаусса.

Билет:

Посмотрим, насколько быстро растут коэффициенты при реализации метода Гаусса.

Пусть задана целочисленная матрица размера $n \times m$, причем длина записи каждого из её элементов не превосходит M ($\|a_{ij}\| \leq M$, $i = 1, \dots, m; j = 1, \dots, n$). После первой итерации имеем матрицу вида

$$\begin{pmatrix} a_{11} & a_{12} \dots a_{1n} \\ 0 & \\ \vdots & B^1 \\ 0 & \end{pmatrix}$$

где элементы матрицы B^1 , $i = 2, \dots, m, j = 2, \dots, n$ вычислены по формуле

$$b_{ij}^1 = a_{ij}a_{11} - a_{1j}a_{j1}.$$

При умножении целых чисел их длины складываются, а при сложении длина не превосходит максимума их длин плюс 1. Получаем: (длина записи всех коэффициентов матрицы B^1 не превосходит $2M + 1$)

$$\| b_{ij}^1 \| \leq 2M + 1$$

После k -ой операции матрица имеет вид

$$\begin{array}{ccccccc}
 a_{11} & a_{12} & \dots & a_{1k} & \dots & a_{1n} \\
 0 & b_{22}^1 & \dots & b_{2k}^1 & \dots & b_{2n}^1 \\
 0 & 0 & \dots & b_{kk}^{k-1} & \dots & b_{kn}^{k-1} \\
 0 & 0 & \vdots & 0 & B^k \\
 0 & 0 & \dots & 0
 \end{array}$$

элементы матрицы B^k , $i = k + 1, \dots, m$, $j = k + 1, \dots, n$ вычислены по формуле

$$b_{ij}^k = b_{ij}^{k-1} b_{kk}^{k-1} - b_{kj}^{k-1} b_{jk}^{k-1}$$

В "худшем" случае имеем

$$\|b_{ij}^2\| \leq 2(2M + 1) + 1 = 4M + 3,$$

$$\|b_{ij}^3\| \leq 2(4M + 3) + 1 = 8M + 7,$$

$$\|b_{ij}^4\| \leq 2(8M + 7) + 1 = 16M + 15.$$

По индукции несложно доказать, что после прямого прохода методом Гаусса для матрицы ранга r .

$$\|b_{ij}^r\| \leq 2^r(M + 1) - 1$$

У нас получилось, что длина записи этих коэффициентов растет экспоненциально, а раз длина коэффициентов растет экспоненциально, то и сложение и умножение с числами экспоненциальной длины будет производиться за экспоненциальное время. Получается, метод Гаусса не полиномиален?

Теорема Сильвестра: для всех $k \geq 2$ каждый элемент b_{ij}^k ($i = k + 1, \dots, m$, $j = k + 1, \dots, n$) делится нацело на $b_{(k-1)(k-1)}^{k-2}$.

Длина записи b_{ij}^k уменьшится по крайней мере на длину записи $b_{(k-1)(k-1)}^{k-2}$ минус единица.

Учитывая это рассмотрим как изменится наша оценка длины коэффициентов. Для $k \geq 2$, $i = k + 1, \dots, m$, $j = k + 1, \dots, n$ имеем

$$\begin{aligned}
 \|b_{ij}^k\| &\leq 2\|b_{ij}^{k-1}\| + 1 - (\|b_{(k-1)(k-1)}^{k-2}\| - 1) = \\
 &= 2\|b_{ij}^{k-1}\| - \|b_{(k-1)(k-1)}^{k-2}\| + 2.
 \end{aligned}$$

Учитывая то, что $\|a_{ij}\| \leq M$ получаем

$$\|b_{ij}^2\| \leq 2(2M + 1) - M + 2 = 3M + 4,$$

$$\|b_{ij}^3\| \leq 2(3M + 4) - (2M + 1) + 2 = 4M + 9,$$

$$\|b_{ij}^4\| \leq 2(4M + 9) - (3M + 4) + 2 = 5M + 16,$$

$$\|b_{ij}^5\| \leq 2(5M + 16) - (4M + 9) + 2 = 6M + 25.$$

По индукции несложно доказать, что после прямого прохода методом Гаусса для матрицы ранга r .

$$\| b_{ij}^r \| \leq (r+1)M + r^2$$

Если сравнивать $(r+1)$ и 2^r ситуация стала лучше. Однако, это все равно длиннее чем `integer(16)` и `longinteger(32)`. Большинство современных компьютеров, если результат арифметической операции с числами типа `integer` не помещается в ячейку, просто "обрезают" старшие разряды.

2. Представление «длинного» числа в файле (массиве, списке) как числа в системе счисления по модулю p . Запись из файла. Оценка числа шагов. Вывод в файл. Оценка числа шагов.

Представление

(у Косовской в методичке m , в названии вопроса p , наверное можно просто заменить m на p)

При $m \geq 2$ любое целое неотрицательное число можно представить в m -ичной системе счисления в виде:

$$x = m^{k-1}x_0 + m^{k-2}x_1 + \dots + mx_{k-2} + x_{k-1}.$$

Здесь k - длина записи m -ичного представления числа x , $0 \leq x_i \leq m-1$ при $i = 0, \dots, k-1$.

Числа x_0, \dots, x_{k-1} будем называть **макроцифрами**.

Замечание (важное)

Если m - максимальное целое, которое может быть записано в одну ячейку памяти, то представление в виде списка

k	x_{k-1}	\dots	x_0
-----	-----------	---------	-------

удобно для хранения и осуществления операций с числами произвольной разрядности.

Запись из файла

В файле записано десятичное число, заданное словом $a_1 \dots a_n$, $0 \leq a_i \leq 9$. Хотим представить его динамическим массивом (или списком).

Пример для числа 49583 в случае $m=100$:

думаю его можно вообще не писать в билете, просто чтобы разобраться

A[0]	A[1]	A[2]	A[3]	ch	Примечание
0				4	Считывание одной цифры. 1 шаг
1	4			9	Запись цифры в массив и считывание цифры. 2 шага
1	49			5	Запись числа из 2-х цифр и считывание цифры. 2 шага
2	95	4		8	Перенос 1-й цифры в A[2], запись числа из 2-х цифр и считывание цифры. 3 шага
2	58	49		3	Перенос 1-й цифры в A[2], запись 2-х чисел из 2-х цифр и считывание цифры. 4 шага
3	83	95	4		Перенос 1-й цифры из A[i] в A[i+1] ($i = 1, 2$), запись 2-х чисел из 2-х цифр и считывание цифры. 5 шагов

Небольшой комментарий о том, что здесь происходит:

Сначала длина 0

Считали первое число 4

Затем записали первое число и прибавили 1 к длине

Считываем следующее число 9

Записали 49 не прибавили к длине

Видим 5

Но 495 уже не помещается поэтому надо 4 перенести в следующий разряд а 5 сделать последним элементом

Видим 8

Но 958 уже не помещается поэтому надо 9 перенести в следующий разряд а 8 сделать последним элементом

Видим 3

Но 583 не поместится поэтому надо 5 перенести в следующий разряд а 3 сделать последним элементом

Но 495 не поместится поэтому надо 4 перенести в следующий разряд а 5 сделать последним элементом

Из этого примера получается следующий алгоритм:

писала сама, так что мб где-то могла затупить

1. Считываем новую цифру
2. Если можно дописать ее в 1 ячейку, то дописываем
3. Если нет, то переносим первую цифру уже имеющейся в 1 ячейке числа в следующую ячейку, а новую цифру записываем последней в 1 ячейке
4. Если заняли ранее свободную ячейку, то увеличиваем длину на 1 (длина в 0 ячейке)

Под шагом понимается одна из следующих операций:

- Считывание цифры из файла
- Запись цифры в целочисленный массив
- Выделение первой цифры многозначного числа и ее удаление из него

- Приписывание цифры в конец числа

Шаги не равнозначны: последние два требуют нахождение остатка от деления на 10, а также умножения на 10 и сложения.

При считывании i -той цифры количество "шагов" увеличивается по мере того, как много цифр нужно перенести в следующие элементы массива.

Связь между i и j задается следующим образом: $j = A[0] = \lfloor \frac{i+1}{\|m-1\|} \rfloor$

При $j = 1, \dots, \lfloor \frac{n+1}{\|m-1\|} \rfloor$ выполняются следующие операции:

```
while A[j] < m - 1 do
  { READ(ch);
    for l = 1..j do
      { if l = 1 then ch1 := ch else ch1 := [A[l] : 10\|m\|-1];
        A[l] := A[l] · 10 + ch1 }
    }
  }
```

Первый цикл выполняется $\|m\| - 1$ раз, в его заголовке производится одна операция сравнения.

Второй цикл выполняется j раз, в его заголовке производится 1 операция сравнения и одна операция увеличения j на единицу.

В теле второго цикла 4 операции.

В условном операторе 2 операции: 1 сравнение и либо присваивание, либо выделение последней цифры из числа.

Второе присваивание требует еще 2 операции: умножения на 10 (сдвиг) и сложения.

Всего количество операций для каждого j составит $\|m - 1\|(1 + 4j)$.

Пусть $M = \lfloor \frac{n+1}{\|m-1\|} \rfloor$. Учитывая, что $j = 1, \dots, M$, получаем оценку числа шагов алгоритма:

$$\begin{aligned}
& \sum_{j=1}^M ||m - 1||(1 + 4j) = \\
& ||m - 1||2(M(M - 1) + M) = \\
& ||m - 1||(2M^2 + M) \leq \\
& (n + 1) \left(2 \frac{n + 1}{||m - 1||} + 1 \right) = \\
& O\left(\frac{n^2}{||m - 1||}\right).
\end{aligned}$$

Таким образом, алгоритм выполняется за **квадратичное от длины десятичной записи** исходного числа количество шагов.

$$O(n^2)$$

Замечание

Также стоит отметить, что для записи неотрицательного десятичного числа длины m_1 как `int` или длины m_2 как `long` необходимо, чтобы выполнялись неравенства $10^{m_1} \leq 2^{16} = 65\ 536$ и $10^{m_2} \leq 2^{32} = 4\ 294\ 967\ 296$.

Вывод в файл

При выводе числа необходимо помнить, что в каждом элементе массива записана не последовательность цифр, а число, записанное этими цифрами. Поэтому число, десятичная запись которого меньше, чем длина записи m , необходимо дополнить ведущими нулями.

Пример для числа 10 00 03 45 06:

Оно представлено в массиве в виде

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	6	45	3	0	10

- Проходим по массиву, где записано число (начиная с конца)
- Если длина записи макроцифры меньше, чем длина записи m , то необходимо дополнить ее запись ведущими нулями (не более, чем $||m-1||$ штук).
- Записываем дополненную нулями макроцифру в файл

Таким образом, последовательно выводятся 5 строк цифр 10 00 03 45 06.

Под шагом понимается одна из следующих операций:

- Запись макроцифры в символьную переменную

- Сравнение длины записи макроцифры с $\|m-1\|$, дополнение строки ведущим нулем

Таким образом, для вывода каждой макроцифры требуется $O(\|m - 1\|)$ шагов. Общее число шагов $O(\|m - 1\|k) = O(n)$

Таким образом, алгоритм выполняется за **линейное от длины десятичной записи** исходного числа количество шагов.

$$O(n)$$

3. Сложение двух «длинных» положительных чисел.

Оценка числа шагов.

У нас есть два неотрицательных многоразрядных числа, записанных в массивы A и B.

k_1	x_{k_1-1}	\dots	x_0
-------	-------------	---------	-------

k_2	y_{k_2-1}	\dots	y_0
-------	-------------	---------	-------

Хотим вычислить $x + y$.

Неформально:

Будем складывать их “в столбик” слева направо.

При переполнении в сумме двух макроцифр нужно добавить “начало” к сумме следующих (для этого вспомогательный массив d).

В 0 ячейке счетчик, сколько макроцифр уже сложили ($k \leq \max\{k_1, k_2\} + 1$). --- есть в старых конспектах такое, но в методичке ничего похожего нет

Формально:

Будем последовательно складывать по модулю m числа, записанные в A[i], B[i] и d[i] для $i = 1, \dots, \max\{A[0], B[0]\}$, где $d[1] = 0$, а для $i > 1$ имеем

$d[i] = 1$, если $A[i - 1] + B[i - 1] + d[i - 1] > m$, иначе $d[i] = 0$

Под шагом понимаются следующие операции:

- Вычисление $A[i] + B[i] + d[i] \bmod m$
- Проверка условия $A[i - 1] + B[i - 1] + d[i - 1] > m$
- Вычисление $d[i]$

Общее число шагов не превосходит $3\max\{A[0], B[0]\} + 1$.

Таким образом, алгоритм выполняется за **линейное от максимума из длин записей** исходных чисел количества шагов.

$$O(\max\{A[0], B[0]\}) = O(\max\{k_1, k_2\})$$

Замечание

Для чисел типа int $A[i - 1] < 10^5$, $B[i - 1] < 10^5$, $d[i - 1] < 2$, поэтому $A[i - 1] + B[i - 1] + d[i - 1] < 2 \cdot 10^5 + 2 < 65\,536$, т.е. результат вычисления может быть записан в одну ячейку.

Аналогично для типа long: $A[i - 1] + B[i - 1] + d[i - 1] < 2 \cdot 10^9 + 2 < 4\,294\,967\,296$.

4. Предикаты равенства и неравенств «длинных» положительных чисел. Оценка числа шагов.

У нас есть два многоразрядных числа x и y , записанных в массивы A и B.

k_1	x_{k_1-1}	\dots	x_0
-------	-------------	---------	-------

k_2	y_{k_2-1}	\dots	y_0
-------	-------------	---------	-------

Хотим вычислять предикаты $x = y$, $x \neq y$, $x < y$, $x \leq y$.

Так как $x \neq y \Leftrightarrow \neg(x = y)$ и $x \leq y \Leftrightarrow \neg(y < x)$ или $x \leq y \Leftrightarrow (x < y) \vee x = y$, то достаточно уметь вычислять значения предикатов $x = y$ и $x < y$.

Первые элементы массивов (длины записи чисел) не равны

Сразу можем сказать:

- Если $A[0] \neq B[0]$, то $x = y \Leftrightarrow \text{false}$.
- Если дополнительно $A[0] < B[0]$, то $x < y \Leftrightarrow \text{true}$.
- Если дополнительно $A[0] > B[0]$, то $x > y \Leftrightarrow \text{true}$.

Первые элементы массивов (длины записи чисел) равны

Начиная со старшего разряда (т.е. $A[A[0]]$ и $B[B[0]]$) сравниваем значения чисел в $A[i]$ и $B[i]$, до тех пор, пока они совпадают.

Т.е идем с конца массива до начала.

Если для некоторого i_0 имеем $A[i] \neq B[i_0]$, то $x \neq y$.

Если при этом $A[i_0] < B[i_0]$, то $x < y$.

Если при этом $A[i_0] > B[i_0]$, то $x > y$.

Под шагом понимаем сравнение макроцифр.

В общем случае число шагов вычисления каждого из четырех предикатов
 $\leq \min\{A[0], B[0]\}$.

Таким образом, алгоритм выполняется за **линейное от минимума из длин записей** исходных чисел количества шагов.

$$O(\min\{A[0], B[0]\}) = O(\min\{k_1, k_2\})$$

5. Вычитание двух «длинных» положительных чисел. Оценка числа шагов.

У нас есть два многоразрядных числа x и y , записанных в массивы А и В.

k_1	x_{k_1-1}	\dots	x_0
-------	-------------	---------	-------

k_2	y_{k_2-1}	\dots	y_0
-------	-------------	---------	-------

Хотим вычислить $x - y$.

Случай $x \geq y$

Процедура аналогична процедуре вычисления суммы (вычитаем “в столбик”, подробнее см. билет 2), но если $A[i] < B[i]$, то следует занять единицу из $A[i+1]$.

Для этого введем вспомогательный массив $d[*]$.

Последовательно для $i = 1, \dots, \max\{A[0], B[0]\}$ будем вычислять $A[i] - B[i] - d[i] \pmod m$, где $d[1] = 0$, а при $i > 1$ имеем $d[i] = 1$, если $A[i-1] - B[i-1] - d[i-1] < 0$, иначе 0.

Случай $x < y$

Можно вычислить $y - x$, затем результату присвоить признак отрицательного числа.

Запись отрицательного числа

Это можно сделать, например:

- Отведя дополнительный элемент массива $A[-1]$ или $A[A[0]+1]$, в котором будет храниться 0 для положительных чисел и 1 для отрицательных (или любой другой признак положительности/отрицательности)
- Храня знак числа в $A[0]$, но при этом по всех случаях, описанных выше, следует писать $|A[0]|$

В любом случае, будем считать, что $A[0] > 0$ и имеется признак знака числа.

Под шагом понимается одна из следующих операций:

- Вычисление $A[i - 1] - B[i - 1] - d[i - 1] \pmod{m}$
- Проверка условия $A[i - 1] - B[i - 1] - d[i - 1] > 0$ и вычисление $d[i]$
- Проверка условия $x \geq y$

Общее число шагов не превосходит $4\max\{A[0], B[0]\} + 1$.

Таким образом, алгоритм выполняется за **линейное от максимума из длин записей** исходных чисел количества шагов.

$$O(\max\{A[0], B[0]\}) = O(\max\{k_1, k_2\})$$

6. Умножение «длинного» числа на короткое. Оценка числа шагов.

У нас есть многоразрядное число, записанное в массив A, и макроцифра C.

k	x_{k-1}	\dots	x_0
-----	-----------	---------	-------

C

Хотим вычислить их произведение, т.е. xC .

Первая форма записи алгоритма

Если $C = 0$, то результат равен 0.

Иначе считаем $d = 0$ и последовательно для $i = 1, \dots, A[0]$ вычисляем

$b = A[i]C + d$, $A[i] = b \pmod{m}$ и $d' = \left\lfloor \frac{b}{m} \right\rfloor$. (это целая часть, просто сложна формулы в ворде писать)

Если при $i = A[0]$ имеем $d' \neq 0$, то $A[0] = A[0] + 1$, $A[A[0]] = d'$

Вторая форма записи алгоритма

тут кажется опечатка и нужно $d=b/m$ а не $b=b/m$

```
if C=0 then return {0}
else { d:=0;
       for i:=1..A[0] do
           { b := A[i] · C + d;
             A[i] := b (mod m);
             b := b/m;
           }
       };
if d ≠ 0 then
    {A[0]:=A[0]+1;
     A[A[0]]:=d;
    }
```

Под шагом понимается одна из следующих операций:

- умножение макроцифр
- сложение макроцифр
- вычисление неполного частного и остатка от деления результата предыдущих операций на m

В условном операторе после `else` выполняется одно присваивание и цикл, в котором производятся две операции, необходимые для организации цикла, а также умножение, сложение, вычисление остатка от деления на m , вычисление неполного частного. Всего в цикле 6 шагов.

Общее число операций не превосходит $\max\{1, 2 + 6A[0] + \max\{1, 3\}\} = 5 + 6A[0]$.

Таким образом, алгоритм выполняется за **линейное от длины записи** исходного числа количества шагов.

$$O(A[0]) = O(k)$$

Замечание

Умножение двух макроцифр (для вычисления числа b) - это не одна операция, т.к. по правилам обычного вычисления умножения и сложения целых чисел в компьютере результаты промежуточных вычислений записываются по модулю m .

Есть еще вот такой вброс

В качестве упражнения читателю предлагается разработать алгоритм вычисления произведения двух макроцифр $b = A[i] \cdot C$ и $d := [b/m]$ и проверить, что число шагов этого алгоритма – константа. При этом оценка имеет вид $C'A[0] + 1$ и общая оценка $O(A[0])$ не изменится.

7. Умножение «длинных» чисел. Оценка числа шагов.

У нас есть два многоразрядных числа x и y , записанных в массивы А и В.

k_1	x_{k_1-1}	\dots	x_0
-------	-------------	---------	-------

k_2	y_{k_2-1}	\dots	y_0
-------	-------------	---------	-------

Хотим вычислить их произведение, т.е. xy .

Результат будет записан в массив С, в котором изначально записано число 0.

Будем умножать “в столбик”.

Последовательно при $i = 1, \dots, B[0]$ производим следующие действия:

1. Умножаем число, записанное в А, на $B[i]$ (результат в массиве D1)
 - Умножение числа (A) на макроцифру ($B[i]$) потребует не более $C'A[0] + 1$ шагов, где C' - некоторая константа (см. билет 5).
2. Записываем D1 со сдвигом во вспомогательный массив D, начиная с $D[i-3]$ (в первые $i-2$ элемента записываем 0)
 - Учитывая еще запись нулей, потребуется не более $C'A[0] + i$ шагов.
 - Заметим также, что значение $D[0]$ не превосходит $A[0] + i$.
3. Складываем числа, записанные в массивах С и D ($C[0]$ и $D[0]$ не превосходят $A[0] + i$)
 - Число шагов при сложении двух положительных чисел (см. билет 2) не превосходит $3 \max\{C[0], D[0]\} + 1 \leq 3(A[0] + i) + 1 = 3A[0] + 3i + 1$

```

 $C[0]:=1; C[1]:=0;$ 
 $for i=1..B[0] do$ 
    {  $D1:=A \cdot B[i];$  /* Здесь применяется процедура умножения многоразрядного числа на макроцифру. */
       $for j=0..i-2 do D[j]:=0;$ 
       $for j=i-3..D1[0]+i-3 do D[j]:=D1[j-(i+2)];$ 
       $D:=C+D;$  /* Здесь применяется процедура сложения многоразрядных чисел. */
    }
  
```

Суммируя оценки числа шагов из 1 и 2 шага, получим:

$$\begin{aligned}
& \sum_{i=1}^{B[0]} ((C'A[0] + i) + (3A[0] + 3i + 1)) = \\
& \sum_{i=1}^{B[0]} ((C' + 3)A[0] + 4i + 1) = \\
& (C' + 3)A[0]B[0] + 2B[0](B[0] - 1) + B[0] = \\
& O(A[0]B[0] + B[0]^2).
\end{aligned}$$

В предположении, что $B[0] \leq A[0]$ (это условие проверяется за 1 шаг и в противном случае можно умножить В на А), получаем оценку $O(A[0]B[0])$.

Таким образом, алгоритм выполняется за **произведение длин записей** исходных чисел количества шагов. (**полином от длины записи чисел**).

$$O(A[0]B[0]) = O(k_1 k_2)$$

8. Деление «длинных» чисел. Оценка числа шагов.

У нас есть два многоразрядных числа x и y , записанных в массивы А и В. Предполагаем, что число, записанное в А, больше числа, записанного в В (иначе неполное частное равно 0, а остаток совпадает с делимым).

k_1	x_{k_1-1}	\dots	x_0
-------	-------------	---------	-------

k_2	y_{k_2-1}	\dots	y_0
-------	-------------	---------	-------

Хотим посчитать результат деления x на y (неполное частное и остаток).

Будем подбирать неполное частное делением промежутка, в котором оно может находиться, пополам.

Пусть L и U - нижняя и верхняя границы промежутка,

$M = \lfloor \frac{L+U}{2} \rfloor$ - целая часть середины промежутка,

$z = y \cdot M$ - число, которое будем сравнивать с делимым

если больше делимого, то M надо уменьшить ($U = M$)

если меньше делимого, то M надо увеличить ($L = M$)

Пример деления $x = 45\ 973$ на $y = 261$ при записи в 100-ичной системе счисления.

При делении 3-значного числа на 2-значное получится 1-значное или 2-значное значение (имеются в виду макроцифры). Поэтому можно выбрать $L = 99$ и $U = 9999$

L	U	M	$z = y \cdot M$	$(z < x) \vee (z > x)$	Примечание
99	9999	5049	1317789	$1317789 > 45973$	$[L, M]$
99	5049	2574	671814	$671814 > 45973$	$[L, M]$
99	2574	1336	348696	$348696 > 45973$	$[L, M]$
99	1336	717	187137	$187137 > 45973$	$[L, M]$
99	717	408	106488	$106488 > 45973$	$[L, M]$
99	408	253	66033	$66033 > 45973$	$[L, M]$
99	253	176	45936	$45936 < 45973$	$[M, U]$
176	253	214	55874	$55874 > 45973$	$[L, M]$
176	214	195	50895	$50895 > 45973$	$[L, M]$
176	195	185	48285	$48285 > 45973$	$[L, M]$
176	185	180	46980	$46980 > 45973$	$[L, M]$
176	180	178	46458	$46458 > 45973$	$[L, M]$
176	178	177	46197	$46197 > 45973$	$[L, M]$
176	177	176	45936	$45936 < 45973$	$[M, U]$

Посчитаем число шагов для нахождения неполного частного и остатка.

Под шагом понимается любая из операций, выполнение которой считалось шагом в предыдущих билетах.

Если $C = \lfloor \frac{x}{y} \rfloor$ (неполное частное), то $x = Cy + r$, $\|x\| = \|C\| + \|y\| + d$ и $\|C\| = \|x\| - \|y\| - d$, где $d \in \{0, 1\}$ (здесь $\|a\|$ - длина m -ичной записи числа a)

- Следовательно, $C \in [m^{\lceil \|x\| - \|y\| \rceil - 1}, m^{\lceil \|x\| - \|y\| \rceil}]$. Длина этого отрезка $\leq m^{\lceil \|x\| - \|y\| \rceil} - m^{\lceil \|x\| - \|y\| \rceil - 1} = m^{\lceil \|x\| - \|y\| \rceil - 1}(m - 1) = m^{A[0] - B[0] - 1}(m - 1)$
- Количество делений отрезка пополам $\leq \log_2(U_0 - L_0)$, где U_0 и L_0 - первоначальные значения границ отрезка
- При этом $U_0 - L_0 \leq m^{A[0] - B[0] - 1}(m - 1)$, откуда количество делений отрезка пополам $\leq \log_2(m^{A[0] - B[0] - 1}(m - 1)) = (A[0] - B[0] - 1)\log_2 m + \log_2(m - 1) \leq (A[0] - B[0])\log_2 m$

Для каждого отрезка производятся следующие операции:

1. Вычисление середины отрезка
 - Длина числа-значения середины отрезка $\leq A[0] - B[0]$
 - По билету 2, сложение двух чисел длины $\leq A[0] - B[0]$ требует $\leq 3(A[0] - B[0]) + 1$ шагов
 - На вычисление половины потребуется $(A[0] - B[0]) + 1$ шагов
 - Всего данная операция занимает $\leq 4(A[0] - B[0]) + 2$ шагов
2. Умножение значения середины отрезка на число y
 - Перемножаем числа длины $A[0] - B[0]$ и $B[0]$
 - По билету 6, потребуется $\leq 7(A[0] - B[0])B[0] + 2B[0](B[0] + 1) + B[0] = 7A[0]B[0] + 9B[0]^2 + B[0]$ шагов
3. Сравнение полученного числа с x
 - По билету 3, потребуется $\leq A[0]$ шагов

Складываем эти значения и умножаем на максимальное количество повторов:

$$((4(A[0] - B[0]) + 2) + (7A[0]B[0] + 9B[0]^2 + B[0]) + A[0]) \cdot \log_2(U_0 - L_0) \\ = O(A[0]B[0]\log_2(U_0 - L_0)) = O(A[0]B[0] \cdot (A[0] - B[0])).$$

В конце используется тот факт, что

$$\log_2(U_0 - L_0) \leq (A[0] - B[0]) \cdot \log_2 m$$

Таким образом, алгоритм выполняется за **произведение длин записей исходных чисел и их разности** количества шагов. (**полином от длины записи чисел**).

$$O(A[0]B[0](A[0] - B[0])) = O(k_1 k_2 (k_1 - k_2))$$

Замечание

В окончательной оценке отсутствуют параметры L_0 и U_0 , но очевидно, что в зависимости от начальных приближений будет меняться и число шагов работы алгоритма.

9. Оценки числа шагов метода Гаусса при действиях с «длинными» числами.

Комментарий: очень странный билет наверное попробую либо разобраться и расписать, либо найти что-то получше.

Метод Гаусса — наименеещий алгоритм

теорема: $A_{ij}, \ell \quad (a_{i+2+\ell, j} : a_{ii})$

$$\left(\begin{array}{c|cc} a_{ii} & & \\ \hline & a_{i+2, i+2} & a_{i+2, j..} \end{array} \right) \quad \text{т.е.}$$

$$\|a_{ij}^{(2)}\| \leq 4M + 3$$

$$B_{ij}^{(2)} = a_{ij}^{(2)} : a_{ii} \quad i=3.. ; j=3..$$

$$\|B_{ij}^{(2)}\| \leq 4M + 3 - M = 3(M + 1)$$

$$\begin{array}{c} a_{ii} \\ \hline a_{22}^{(1)} \\ \hline B_{ij}^{(2)} \end{array}$$

$$\|B_{ij}^{(3)}\| \leq 2(3(M + 1)) + \text{const}$$

$$\|B_{ij}^{(3)} : a_{22}^{(1)}\| \leq B M + 2 - 2M + .. = 4M + \text{const}$$

...

$$\|B_{ij}^{(k)} : a_{kk-1, k-1}^{(k-2)}\| \leq (k+1)M + \text{const}$$

а это уже наименеещий

Оценим асимптотику полученного алгоритма. Алгоритм состоит из m фаз, на каждой из которых происходит:

- поиск и перестановка опорного элемента — за время $O(n + m)$ при использовании эвристики "partial pivoting" (поиск максимума в столбце)
 - будем считать, что задана целочисленная матрица размера $n \times m$, причем длина записи каждого из её элементов не превосходит M ($\|a_{ij}\| \leq M, i = 1, \dots, m; j = 1, \dots, n$).
 - можно еще сослаться на теорему Сильвестра и данный факт:

По индукции несложно доказать, что после прямого прохода методом Гаусса для матрицы ранга r

$$\| b_{ij}^r \| \leq (r + 1)M + r^2$$

- сравниваем между собой длинные числа: $O(M * (n + m))$ (?)
- если опорный элемент в текущем столбце был найден — то прибавление текущего уравнения ко всем остальным уравнениям — за время $O(nm)$
 - сложение длинных чисел: $O(M^*n^*m)$ (?)

Очевидно, первый пункт имеет меньшую асимптотику, чем второй. Заметим также, что второй пункт выполняется не более $\min(n, m)$ раз — столько, сколько может быть зависимых переменных в СЛАУ.

Таким образом, **итоговая асимптотика** алгоритма принимает вид $O(\min(n, m) \cdot nm)$.

При $n = m$ эта оценка превращается в $O(n^3)$.

Для длинных числе получается $O(M * n^3)$. (?)

10. Сортировки и оценки числа их шагов: пузырьк, сортировка вставками, сортировка слияниями фон Неймана.

сортировка вставками или сортировка подсчетом.....

Сортировка пузырьком

В сортировке пузырьком имеется два вложенных цикла по $i = 2, \dots, n$ и $j = 1, \dots, i$. Тем самым тело циклов выполняется $\sum_{i=2}^n i = \frac{(n+2)(n-1)}{2}$ раз.

В теле циклов сравниваются значения $a[i]$ и $a[j]$. В случае необходимости (например: сортируем в порядке возрастания, $i < j$ и $a[i] > a[j]$), содержание элементов массива меняется местами. Обмен осуществляется с помощью трех операторов присваивания с использованием вспомогательной переменной z : $z:=x$; $x:=y$; $y:=z$. В теле цикла выполняется не более 4 операций.

Если “шаг” это выполнение операции сравнения или операции присваивания, то “пузырёк” завершает работу за число “шагов”, не превосходящее $4 \frac{(n+2)(n-1)}{2} = 2(n+2)(n-1)$, что составляет $O(n^2)$.

Не учли количество операторов увеличения переменной на 1 и оператора сравнения значения переменной цикла с границей цикла, так как оценка всё равно останется $O(n^2)$.

Массив с многоразрядными числами.

Пусть k - максимальное количество макроцифр в сортируемых многоразрядных числах (т.е. длина записи).

Под “шагом” понимаем соответствующие операции с макроцифрами.

Операции сравнения и операция присваивания многоразрядных чисел осуществляется не более чем за k “шагов”.

Суммарное число шагов не превосходит $4 \frac{k(n+2)(n-1)}{2} = 2k(n+2)(n-1)$, что составляет $O(kn^2)$.

Сортировка вставками (нет в методичке, есть в названии билета)

Общая суть сортировок вставками такова:

1. Перебираются элементы в неотсортированной части массива.
2. Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находиться.

Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Так как в процессе работы алгоритма могут меняться местами только соседние элементы, каждый обмен уменьшает число [инверсий](#) на единицу. Следовательно, количество обменов равно количеству инверсий в исходном массиве вне зависимости от реализации сортировки.

Алгоритм работает за $O(n+k)$, где k — число обменов элементов входного массива, равное числу инверсий. Максимальное количество инверсий содержится в массиве, элементы которого отсортированы по невозрастанию. Число инверсий в таком массиве $n^*(n-1)/2$.

Таким образом, в худшем случае алгоритм работает за $O(n^2)$.

Минимальная оценка встречается в случае уже упорядоченной исходной последовательности элементов - $O(n)$.

Массив с многоразрядными числами.

Пусть k - максимальное количество макроцифр в сортируемых многоразрядных числах (т.е. длина записи).

Под "шагом" понимаем соответствующие операции с макроцифрами.

Операции сравнения и операция присваивания многоразрядных чисел осуществляется не более чем за k "шагов".

Получаем оценку $O(kn^2)$.

Сортировка слияниями (Фон Неймана)

Предположим, длина массива $n = 2^t$. Если это не так, то дополняем его очень большими числами.

Описание алгоритма:

1. Разобьем массив на пары $(a[2i-1], a[2i])$ ($i = 1, \dots, 2^{t-1}$) и упорядочим элементы в каждой паре.

Это $n/2 = 2^{t-1}$ сравнений чисел и не более чем такое же количество обменов значений элементов массива. Всего не более, чем $4 \cdot 2^{t-1} = 2^{t+1}$ "шагов".

2. В цикле по $j = 1, \dots, t-1$ "сливаем" уже упорядоченные наборы из 2^j элементов.
Получаем 2^{t-j-1} упорядоченных набора из 2^{j+1} элементов.

Подобное "слияние" требует не более 2^{j+1} операций сравнения и столько же операций присваивания. Умножаем это на количество "сливаемых" пар 2^{t-j-1} . Следовательно, количество "шагов" в теле цикла не превосходит $2 \cdot 2^{j+1} \cdot 2^{t-j-1} = 2 \cdot 2^t = 2^{t+1}$.

Всего в цикле будет произведено не более $\sum_{j=1}^{t-1} 2 \cdot 2^t = 2 \cdot 2^t(t-1) = 2^{t+1}(t-1)$ «шагов».

Сложив полученные оценки и учитывая, что $t = \log_2 n$ получаем $4 \cdot 2^{t-1} + 2^{t+1}(t-1) = 2^{t+1}t = 2n \log_2 n = O(n \log_2 n)$ «шагов».

Учитывая замечание к алгоритму сортировки «пузырьёк» при сортировке многоразрядных чисел или слов в заданном алфавите в этой оценке требуется дописать мультипликативный сомножитель k , где k – максимальное количество макроцифр в многоразрядных числах или максимальная длина сортируемых чисел, т.е. оценка имеет вид $O(kn \log_2 n)$.

Сортировка подсчетом (есть в методичке, нет в названии билета)

Пусть требуется отсортировать массив целых положительных чисел, наибольшее из которых равно N.

Заведём вспомогательный массив с[1..N], значения элементов которого первоначально равно 0.

В цикле по $i = 1, \dots, n$ значение $c[a[i]]$ увеличиваем на единицу (n шагов).

В цикле по $j = 1, \dots, N$ если $c[j] \neq 0$, то очередным $c[j]$ элементам массива a присваиваем j.

```
for (i=0; i<n; i++)
    c[a[i]]++;
k=0;
for (i=0; i<N; i++)
    for (j=0; j<c[i]; j++)
        a[k++]=i;
```

Посмотрели два массива длиной n и N , в которых либо увеличиваем значение на единицу, либо производим присваивание. $O(n+N)$ шагов.

При сортировке многоразрядных чисел умножаем на k , получаем. $O(k(n+N))$. Где k - максимальное количество макроцифр в сортируемых многоразрядных числах.

Если N намного меньше n или разность $\max a[i] - \min a[i]$ имеет тот же порядок что и n , а границами массива с являются числа $\max a[i]$ и $\min a[i]$, то количество шагов работы алгоритма действительно линейно относительно n .

11. Алгоритмы на графах, различные способы представления графа в компьютере

Матрица смежности

2.1.1. Матрица смежности

Матрица смежности графа — это квадратная матрица $A_{n \times n}$, элементы которой определены так

$$a_{ij} = \begin{cases} 1 & \text{если } \{v_i, v_j\} \in E, \\ 0 & \text{иначе} \end{cases} .$$

$O(n^2)$ памяти

Списки смежности

2.1.2. Списки смежности

Списки смежности — это одномерный массив, i -ым элементом которого является список вершин, смежных с v_i , т.е. окружение вершины v_i .

Очевидно, что для нахождения всех вершин, смежных с v , требуется $\deg(v)$ проверок. Просуммировав эту величину по всем v получаем $\sum_{v \in V} \deg(v) = 2m$.⁴

$O(n+m)$ памяти

Матрица инцидентности

2.1.3. Матрица инцидентности

Матрица инцидентности графа — это матрица $B_{n \times m}$, элементы которой определены так

$$b_{ij} = \begin{cases} 1 & \text{если } v_i \in e_j, \\ 0 & \text{иначе} \end{cases}.$$

Основными свойствами такой матрицы являются следующие два: $\sum_{i=1}^n b_{ij} = 2$, $\sum_{j=1}^m b_{ij} = \deg(v_i)$.

Если для вершины необходимо проверить все вершины, смежные с данной, то придётся в строке, соответствующей этой вершине, найти все столбцы, на пересечении с которыми стоит 1 (m проверок), и в каждом из этих столбцов найти строку, на пересечении с которой стоит 1 (n проверок). Всего не более, чем $\deg(v_i)n + (m - \deg(v_i))$.

Поскольку эти операции следует проделать для каждой вершины, то всего имеем $\sum_{i=1}^n (\deg(v_i)n + (m - \deg(v_i))) = n \sum_{i=1}^n \deg(v_i) + nm + \sum_{i=1}^n \deg(v_i) = 2nm + nm - 2m = 3nm - 2m$.

$O(nm)$ памяти

Массивы V и E

В одномерном массиве V хранятся имена вершин. В двумерном массиве E хранятся пары номеров смежных вершин.

12. Алгоритм поиска в глубину. Оценки числа шагов в зависимости от способа представления графа

Идея поиска в глубину

Пусть есть некоторая начальная вершина графа v . Она помечается единичкой (пройдена). Выбирается вершина u , смежная с v , помечается единичкой. Процесс повторяется с одной из вершин смежных с u .

Если на очередном шаге нет вершин смежных с той, с которой мы работаем, и не помеченных единичкой, то возвращаемся на предыдущий шаг. Попали в начальную вершину - обход графа завершен.

При оценке числа шагов решения задач, связанных с обходом графа, большое значение имеет способ представления графа.

Матрица смежности

2.1.1. Матрица смежности

Матрица смежности графа — это квадратная матрица $A_{n \times n}$, элементы которой определены так

$$a_{ij} = \begin{cases} 1 & \text{если } \{v_i, v_j\} \in E, \\ 0 & \text{иначе} \end{cases} .$$

При использовании матрицы смежности проверку всех вершин на смежность с данной, можно сделать за n проверок. Эта процедура выполняется для каждой из n вершин. Получаем оценку числа шагов $O(n^2)$.

Списки смежности

2.1.2. Списки смежности

Списки смежности — это одномерный массив, i -ым элементом которого является список вершин, смежных с v_i , т.е. окружение вершины v_i .

Очевидно, что для нахождения всех вершин, смежных с v , требуется $\deg(v)$ проверок. Просуммировав эту величину по всем v получаем $\sum_{v \in V} \deg(v) = 2m$.⁴

Получаем оценку числа шагов $O(n+m)$.

Если граф является деревом или лесом ($m < n$) получаем оценку $O(n)$.

Если граф полный ($m = n*(n-1)/2$) получаем оценку $O(n^2)$.

Если степени всех вершин графа не превосходят некоторой константы C (существенно меньшей, чем n) то $m \leq Cn$. А оценка $O(n)$.

Если граф связан и степени вершин произвольны ($m \geq n-1$) получаем оценку $O(n)$.

Матрица инцидентности

2.1.3. Матрица инцидентности

Матрица инцидентности графа — это матрица $B_{n \times m}$, элементы которой определены так

$$b_{ij} = \begin{cases} 1 & \text{если } v_i \in e_j, \\ 0 & \text{иначе} \end{cases}.$$

Основными свойствами такой матрицы являются следующие два:
 $\sum_{i=1}^n b_{ij} = 2$, $\sum_{j=1}^m b_{ij} = \deg(v_i)$.

Если для вершины необходимо проверить все вершины, смежные с данной, то придётся в строке, соответствующей этой вершине, найти все столбцы, на пересечении с которыми стоит 1 (m проверок), и в каждом из этих столбцов найти строку, на пересечении с которой стоит 1 (n проверок). Всего не более, чем $\deg(v_i)n + (m - \deg(v_i))$.

Поскольку эти операции следует проделать для каждой вершины, то всего имеем $\sum_{i=1}^n (\deg(v_i)n + (m - \deg(v_i))) = n \sum_{i=1}^n \deg(v_i) + nm + \sum_{i=1}^n \deg(v_i) = 2nm + nm - 2m = 3nm - 2m$.

Получаем оценку числа шагов $O(nm)$.

Массивы V и E

В одномерном массиве V хранятся имена вершин. В двумерном массиве E хранятся пары номеров смежных вершин.

Построим массив E', в котором вместе с каждой парой $\{vi, vj\}$, присутствует пара $\{vj, vi\}$. Отсортируем его по номеру первого элемента пары. Вместе с элементом vi массива V будем хранить номер строки массива E', в которой впервые встречается пара $\{vi, u\}$ для некоторой вершины u .

Для записи E' потребуется $2m$ присвоений и $O(m * \log m)$ сравнений и присвоений при сортировке.

Получаем оценку $O(n + m * \log m)$.

13. Алгоритм поиска в ширину. Оценки числа шагов в зависимости от способа представления графа.

Идея поиска в ширину:

Пусть есть некоторая начальная вершина графа v . Она помечается единичкой (пройдена). Все вершины, смежные с v , помечаются единичкой, а также помещаются в список. После этого начальная вершина v полностью обработана и помечается двоичкой.

Следующей текущей вершиной становится первая вершина списка. Все ранее не помеченные вершины, смежные с текущей, заносятся в конец списка, а также

помечаются единичкой. Текущая вершина удаляется и помечается двоичкой. Процесс продолжается пока список вершин не станет пустым. Данный список называется очередью.

При оценке числа шагов решения задач, связанных с обходом графа, большое значение имеет способ представления графа.

Матрица смежности

2.1.1. Матрица смежности

Матрица смежности графа — это квадратная матрица $A_{n \times n}$, элементы которой определены так

$$a_{ij} = \begin{cases} 1 & \text{если } \{v_i, v_j\} \in E, \\ 0 & \text{иначе} \end{cases} .$$

При использовании матрицы смежности проверку всех вершин на смежность с данной, можно сделать за n проверок. Эта процедура выполняется для каждой из n вершин. Получаем оценку числа шагов $O(n^2)$.

Списки смежности

2.1.2. Списки смежности

Списки смежности — это одномерный массив, i -ым элементом которого является список вершин, смежных с v_i , т.е. окружение вершины v_i .

Очевидно, что для нахождения всех вершин, смежных с v , требуется $\deg(v)$ проверок. Просуммировав эту величину по всем v получаем $\sum_{v \in V} \deg(v) = 2m$.⁴

Получаем оценку числа шагов $O(n+m)$.

Если граф является деревом или лесом ($m < n$) получаем оценку $O(n)$.

Если граф полный ($m = n*(n-1)/2$) получаем оценку $O(n^2)$.

Если степени всех вершин графа не превосходят некоторой константы C (существенно меньшей, чем n) то $m \leq Cn$. А оценка $O(n)$.

Если граф связен и степени вершин произвольны ($m \geq n-1$) получаем оценку $O(n)$.

Матрица инцидентности

2.1.3. Матрица инцидентности

Матрица инцидентности графа — это матрица $B_{n \times m}$, элементы которой определены так

$$b_{ij} = \begin{cases} 1 & \text{если } v_i \in e_j, \\ 0 & \text{иначе} \end{cases}.$$

Основными свойствами такой матрицы являются следующие два:
 $\sum_{i=1}^n b_{ij} = 2$, $\sum_{j=1}^m b_{ij} = \deg(v_i)$.

Если для вершины необходимо проверить все вершины, смежные с данной, то придётся в строке, соответствующей этой вершине, найти все столбцы, на пересечении с которыми стоит 1 (m проверок), и в каждом из этих столбцов найти строку, на пересечении с которой стоит 1 (n проверок). Всего не более, чем $\deg(v_i)n + (m - \deg(v_i))$.

Поскольку эти операции следует проделать для каждой вершины, то всего имеем $\sum_{i=1}^n (\deg(v_i)n + (m - \deg(v_i))) = n \sum_{i=1}^n \deg(v_i) + nm + \sum_{i=1}^n \deg(v_i) = 2nm + nm - 2m = 3nm - 2m$.

Получаем оценку числа шагов $O(nm)$.

Массивы V и E

В одномерном массиве V хранятся имена вершин. В двумерном массиве E хранятся пары номеров смежных вершин.

Построим массив E', в котором вместе с каждой парой $\{vi, vj\}$, присутствует пара $\{vj, vi\}$. Отсортируем его по номеру первого элемента пары. Вместе с элементом vi массива V будем хранить номер строки массива E', в которой впервые встречается пара $\{vi, u\}$ для некоторой вершины u .

Для записи E' потребуется $2m$ присвоений и $O(m * \log m)$ сравнений и присвоений при сортировке.

Получаем оценку $O(n + m * \log m)$.

14. Задачи, решаемые с помощью этих алгоритмов: выделение компонент связности, проверка на двудольность и выделение долей, выделение остова графа.

Задача выделения компонент связности.

Дано:

Неориентированный граф G с n вершинами и m ребрами. Требуется найти в нем все компоненты связности.

Алгоритм:

Для решения можно использовать как обход в глубину, так и в ширину.

По сути, мы будем проводить серию обходов: сначала запустим обход из первой вершины, и все вершины, которые обойдем при этом, пометим как входящие в 1-ую компоненту связности. Затем, находим 1-ую из оставшихся вершин, которые еще не были помечены, и запускаем обход из неё и т.д. пока все вершины не станут помеченными.

Итоговая асимптотика:

$O(n+m)$

действительно, так как такой алгоритм не запускается дважды от одной и той же вершины => любое ребро будет просмотрено ровно 2 раза (с одного и другого конца).

Проверка на двудольность и выделение долей

Дано:

Неориентированный граф G с n вершинами и m ребрами. Проверить, является ли двудольным и если да, то вывести сами доли.

Алгоритм:

Произведем серию поисков в ширину, то есть будем запускать поиск в ширину из каждой непосещенной вершины. Ту вершину, из которой начинаем идти, помещаем в первую долю. В процессе поиска в ширину, если мы идём в какую-то новую вершину, то мы помещаем ее в долю, отличную от доли текущей вершины. Если же мы пытаемся пройти по ребру в вершину, которая уже посещалась, то мы проверяем, чтобы эта вершина и текущая вершина были из разных долей, иначе граф не является двудольным.

По окончании работы алгоритма мы либо обнаружим, что граф не двудолен, либо найдем разбиение вершин графа на две доли.

Итоговая асимптотика:

$O(m)$.

Выделение остова графа

При поиске в глубину мы получаем дерево, которое и будет являться остовом графа.

15. Нахождение остова минимального веса. Метод Роберта Прима. Оценки числа шагов.

Комментарий к 15 и 16:

Мне кажется, что эти билеты неплохо написаны в методичке, они не такие большие и достаточно понятны. Думаю, можно оставить так.

$\text{OUT}(v)$ - множество вершин орграфа, непосредственно достижимых из v .

$\deg(v)$ - степень вершины v , т.е. количество вершин в окружении.

Задан взвешенный граф $G = (V, E)$, все веса w_{ij} положительны.

Требуется найти остов минимального веса.

Не умаляя общности будем считать, что граф связан и ищется остовное дерево минимального веса.

Вводятся два вспомогательных массива m и d , в которых $m[i]$ и $d[i]$ – это номер вершины, смежной с v_i в остове и вес ребра $\{v_i, m[i]\}$ соответственно.

1. $d := \infty$. Выбираем произвольно вершину v_0 , помечаем её как вошедшую в остов.
2. Для всех вершин $u \in N(v_0)$ делаем присваивания $m[u] := v_0$; $d[u] := w_{v_0 u}$.
3. Среди непомеченных вершин u , у которых $d[u] \neq \infty$ находим вершину $v = \arg(\min_u d[u])$. Помечаем вершину v как вошедшую в остов.
4. Пересчитываем значения массивов m и d для непомеченных вершин u из $N(v)$.

Если $d[u] > w_{vu}$, то $\{m[u] := v; d[u] := w_{vu}\}$.

5. Если имеются непомеченные вершины, то возвращаемся к выполнению п. 3. Иначе алгоритм заканчивает работу и в остов входят все рёбра вида $\{u, m[u]\}$ ($u \neq v_0$), причём вес остова равен $\sum_{u \neq v_0} d[u]$.

В п. 1 совершается n присваиваний.

В п. 2 совершается $2\deg(v_0)$ присваиваний.

Выполнение пунктов 3, 4, 5 производится не более чем $n-1$ раз.

В п. 3 совершается не более $n-i-1$ сравнений.

В п. 4 выполняется не более $\deg(v_i)$ сравнений и $2\deg(v_i)$ присваиваний.

В п. 5 выполняется одно сравнение.

СКОБОЧКА ПОСЛЕ ЕДИНИЧКИ ПРОЕБАНА

$$\begin{aligned} n + 2\deg(v_0) + \sum_{i=1}^{n-1} (n - i - 1 + 3\deg(v_i)) + 1 = \\ n + 2\deg(v_0) + n(n - 1) - \frac{(n - 1)(n - 2)}{2} + 3m \leq \frac{n(n - 1)}{2} + 3m - 1 = \\ O(n^2 + m) = O(n^2). \end{aligned}$$

16. Алгоритм Дейкстры поиска кратчайшего пути. Оценки числа шагов.

2.2.1. Алгоритм Дейкстры поиска кратчайшего пути во взвешенном ориентированном графе. Оценки числа шагов.

Задан взвешенный орграф $G = (V, A)$ (все веса w_{ij} положительны) и две выделенные вершины s – старт и f – финиш. Требуется найти кратчайший путь из s в f .

Вводятся два вспомогательных массива pre и d , в которых хранятся предыдущая вершина в кратчайшем (на данный момент) пути из s в v_i и (известная на данный момент) величина этого кратчайшего расстояния.

1. $pre := \infty$, вершину s помечаем.
2. Для всех вершин $u \in OUT(s)$ делаем присваивания

$$pre(u) := s; \quad d(u) := w_{su}.$$

3. Среди непомеченных вершин u , у которых $pre(u) \neq \infty$ находим вершину $v = \arg(\min_u d(u))$. Помечаем вершину v .
4. Пересчитываем значения массивов pre и d для непомеченных вершин u из $OUT(v)$.
Если $d(u) > d(v) + w_{vu}$, то $\{pre(u) := v; d(u) := d(v) + w_{vu}\}$.
5. Если $v \neq f$, то возвращаемся к выполнению п. 3. Иначе алгоритм заканчивает работу и кратчайшим путём из s в f является $s, \dots, pre(pre(f)), pre(f), f$.

В п. 1 совершается n присваиваний.

В п. 2 совершается $2\|OUT(s)\|$ присваиваний.

Выполнение п.п. 3, 4 и 5 производится не более чем $n - 1$ раз.

При i -ом выполнении п. 3 количество непомеченных вершин не превосходит $n - i$ и, следовательно, нахождение вершины, доставляющей $\max_u d(u)$, требует не более $n - i - 1$ сравнений.

В п. 4 выполняется не более $\|OUT(v)\|$ сложений, $\|OUT(v)\|$ сравнений и $2\|OUT(v)\|$ присваиваний.

В п. 5 выполняется одно сравнение.

Просуммировав полученные оценки имеем

$$\begin{aligned} n + 2\|OUT(s)\| + \sum_{i=1}^{n-1} (n - i - 1 + 4\|OUT(v_i)\|) + 1 = \\ n + 2\|OUT(s)\| + n(n - 1) - \frac{(n - 1)(n - 2)}{2} + 4m \leq \frac{n(n - 1)}{2} + 4m - 1 = \\ O(n^2 + m) = O(n^2). \end{aligned}$$

СКОБОЧКА ПОСЛЕ ЕДИНИЧКИ ПРОЕБАНА

17. Нахождение циклов и мостов в графе. Оценки числа шагов.

Задача нахождения циклов в графах:

Дано:

Ориентированный или неориентированный граф без петель и кратных ребер

Необходимо проверить, является ли граф ациклическим. Если нет, то найти любой цикл.

Алгоритм:

Задача решается с помощью поиска в глубину..

Производится серия поисков в глубину, т.е. из любой вершины, в которой еще не побывали, запустим обход в глубину и при входе в вершину будем красить ее в серый, а при выходе в черный. Если поиск в глубину пытается войти в серую вершину, значит мы нашли цикл. Для неориентированного графа случай, когда поиск в глубину пытается войти в предка не считается. Сам цикл можно восстановить проходом по массиву предков.

Итоговая асимптотика:

$O(n+m)$, где n - количество вершин, m - количество ребер

Задача поиска мостов в неориентированном графе

Мост - ребро, которое не содержится ни в одном цикле (его удаление увеличивает количество компонент связности)

Алгоритм:

Воспользуемся следующей теоремой:

Теорема

Пусть T — дерево обхода в глубину графа G . Ребро (u,v) является мостом тогда и только тогда, когда $(u,v) \in T$ и из вершины v и любого ее потомка нет обратного ребра в вершину u или любого ее предка

Доказательство

►

⇐

Удалим (u,v) из G . Докажем от противного, что мы не сможем достичь ни одного из предков v (в частности, u). Пусть это не так, и w - предпоследняя вершина на пути от v до ее предка x . В силу единственности пути в дереве (w,x) не является ребром дерева. Если (w,x) - обратное ребро, то это противоречит условию теоремы, так как x - предок u . Следовательно, мы не достигнем предков v , а значит количество компонент связности увеличилось, поэтому ребро (u,v) является мостом

⇒

От противного. Пусть существует удовлетворяющее условию обратное ребро (x,w) . Тогда (u,v) лежит на цикле $x \rightsquigarrow v \rightarrow u \rightsquigarrow w \rightarrow x$ и не может быть мостом.

◀

Теперь осталось научиться проверять этот факт для каждой вершины эффективно. Для этого модифицируем поиск в глубину, запоминая дополнительно “время входа” и “время выхода” в каждую вершину.

Введем вспомогательную функцию $\text{ret}(v) = \min\{\text{enter}(v), \text{ret}(x), \text{enter}(w)\}$,

где $\text{enter}(v)$ - время входа в вершину x

$\text{ret}(x)$ - (v, x) - ребро дерева (т.е. x - потомок v) (здесь рассматриваем минимум из всех таких x)

$\text{enter}(p)$ - (v,p) - обратное ребро (здесь рассматриваем минимум из всех таких w)

Тогда:

Ребро (u,v) является мостом тогда и только тогда, когда (u,v) принадлежит дереву обхода в глубину и $\text{ret}(v) > \text{enter}(u)$.

Доказательство

▶

Из вершины v или её потомка есть обратное ребро в её предка тогда и только тогда, когда найдётся такой сын to , что $\text{ret}(to) \leq \text{enter}(v)$.

(Если $\text{ret}(to) = \text{enter}(v)$, то это означает, что найдётся обратное ребро, приходящее точно в v ; если же $\text{ret}(to) < \text{enter}(v)$, то это означает наличие обратного ребра в какого-либо предка вершины v .)

Таким образом, если для текущего ребра (v,t) (принадлежащего дереву поиска) выполняется $\text{ret}[t] > \text{enter}[v]$, то это ребро является мостом; в противном случае оно мостом не является.

◀

Конкретная реализация

```
function dfs(v):
    time = time + 1
    enter[v] = time
    ret[v] = time
    for всех u смежных с v
        if (v, u) – обратное ребро
            ret[v] = min(ret[v], enter[u])
        if вершина u – белая
            dfs(u)
            ret[v] = min(ret[v], ret[u])
            if ret[u] > enter[v]
                ребро (v, u) – мост
```

Итоговая асимптотика:

$O(n+m)$

18. Эйлеров цикл. Оценки числа шагов.

Эйлеров путь - это путь в графе, проходящий через все его рёбра.

Эйлеров цикл - это эйлеров путь, являющийся циклом.

Граф называется **эйлеровым**, если он содержит эйлеров цикл.

Дано:

Задача заключается в том, чтобы найти эйлеров путь в неориентированном мультиграфе с петлями.

Будем пользоваться следующей теоремой:

Теорема (критерий эйлеровости)

Для того, чтобы граф $G=(V,E)$ был эйлеровым необходимо чтобы:

1. Все вершины имели четную степень.
2. Все компоненты связности кроме, может быть одной, не содержали ребер.

Алгоритм:

Алгоритм напоминает поиск в глубину. Главное отличие состоит в том, что пройденными помечаются не вершины, а ребра графа.

Сначала проверим, существует ли эйлеров путь. Затем найдём все простые циклы и объединим их в один - это и будет эйлеровым циклом. Искать все циклы и объединять их будем одной рекурсивной процедурой:

procedure `FindEulerPath (V):`

1. перебрать все рёбра, выходящие из вершины V ;

каждое такое ребро удаляем из графа, и

вызываем `FindEulerPath` из второго конца этого ребра;

2. добавляем вершину V в ответ.

Итоговая асимптотика:

Если реализовать поиск ребер инцидентных вершине и удаление ребер за $O(1)$, то алгоритм будет работать за $O(m)$.

Чтобы реализовать поиск за $O(1)$, для хранения графа следует использовать списки смежных вершин; для удаления достаточно добавить всем ребрам свойство `deleted` бинарного типа.

19. Гамильтонов цикл. Оценки числа шагов.

2.2.3. Нахождение Гамильтонова цикла. Оценки числа шагов.

Алгоритм нахождения Гамильтонова цикла на первый взгляд очень напоминает обход графа в глубину. Однако при удалении вершины из стека алгоритм «забывает», что эту вершину уже посещали. Алгоритм заканчивает работу, если в стеке находятся все n вершин и крайние вершины в стеке смежны.

В результате, если выписать в виде дерева все возможные сокращения стека, то каждая вершина v исходного графа может оказаться в этом дереве не более, чем $\deg(v)$ раз. И каждое появление вершины

v в дереве имеет такую же степень вершины, что и v . Кроме того, высота такого дерева равна n .

Суммарное количество посещений вершин не превосходит $d^n - 1$, где $d = \max_i(\deg(v_i) - 1)$ и не меньше (если граф не имеет Гамильтонова цикла или он был найден в конце работы алгоритма), чем $2^{n'} - 1$, где n' – количество вершин исходного графа, степень которых больше двух.

20. Алгоритм генерации всех независимых множеств. Оценки числа шагов

Определение. Множество V' называется **независимым множеством графа** $G = (V, E)$, если никакие две вершины из V' не смежны.

$$\forall u v ((u \in V' \& v \in V') \rightarrow \{u, v\} \notin E)$$

Определение. Множество V' называется **кликой** в графике $G = (V, E)$, если любые две вершины из V' смежны.

$$\forall u v ((u \in V' \& v \in V') \rightarrow \{u, v\} \in E)$$

Одним из самых эффективных алгоритмов поиска клик является **алгоритм Брана-Кербоша** - метод ветвей и границ для поиска всех клик.

Так как

Теорема 2.2.1. Следующие утверждения равносильны:

1. V' является вершинным покрытием в $G = (V, E)$;
2. $V \setminus V'$ является независимым множеством в $G = (V, E)$;
3. $V \setminus V'$ является кликой в $\bar{G} = (V \setminus V', \bar{E})$;

алгоритм подойдет и для генерации всех независимых множеств
(ищем клики в \bar{G} , в G это будут независимые множества)

Алгоритм оперирует тремя множествами вершин графа и является рекурсивной процедурой **extend (candidates, not)**.

Множество **compsub** - множество, содержащее на каждом шаге рекурсии полный подграф для данного шага. Строится рекурсивно.

Множество **candidates** - множество вершин, которые могут увеличить comps sub.

Множество **not** - множество вершин, которые уже использовались для расширения comps sub на предыдущих шагах алгоритма.

ПОКА candidates НЕ пусто И not НЕ содержит вершины, СМЕЖНОЙ СО ВСЕМИ вершинами из candidates,

```
{ Выбираем вершину v из candidates и добавляем её в comps sub;
Формируем new-candidates и new-not, удаляя из candidates и
not вершины, не СМЕЖНЫЕ с v;
ЕСЛИ new-candidates и new-not пусты;
ТО comps sub клика;
ИНАЧЕ extend (new-candidates, new-not);
Удаляем v из comps sub и candidates, и помещаем в not
}
```

Вычислительная сложность алгоритма линейна относительно количества клик в графе.
В худшем случае алгоритм работает за $O(3^{n/3})$ шагов.

21. Теорема о НМ, ВП, КЛИКА.

2.2.4. Связь между задачами «генерация всех независимых множеств», «нахождение вершинного покрытия», «КЛИКА».

Определение. Множество V' называется вершинным покрытием графа $G = (V, E)$, если каждое ребро графа инцидентно вершине из V' .

$$\forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V'))$$

Определение. Множество V' называется независимым множеством графа $G = (V, E)$, если никакие две вершины из V' не смежны.

$$\forall uv((u \in V' \& v \in V') \rightarrow \{u, v\} \notin E)$$

Определение. Множество V' называется кликой в графе $G = (V, E)$, если любые две вершины из V' смежны.

$$\forall uv((u \in V' \& v \in V') \rightarrow \{u, v\} \in E)$$

Теорема 2.2.1. Следующие утверждения равносильны:

1. V' является вершинным покрытием в $G = (V, E)$;
2. $V \setminus V'$ является независимым множеством в $G = (V, E)$;
3. $V \setminus V'$ является кликой в $\overline{G} = (V \setminus V', \overline{E})$;

Для доказательства запишем формулы, определяющие соответствующие понятия, для каждого из утверждений теоремы.

1. $\forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')).$

2. $\forall uv((u \in V \setminus V' \& v \in V \setminus V') \rightarrow \{u, v\} \notin E) \Leftrightarrow$
 $\forall uv(\{u, v\} \in E \rightarrow \neg(u \in V \setminus V' \& v \in V \setminus V')) \Leftrightarrow$
 $\forall uv(\{u, v\} \in E \rightarrow (u \notin V \setminus V' \vee v \notin V \setminus V')) \Leftrightarrow$
 $\forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')).$

3. $\forall uv((u \in V \setminus V' \& v \in V \setminus V') \rightarrow \{u, v\} \in \overline{E}) \Leftrightarrow$
 $\forall uv(\{u, v\} \notin \overline{E} \rightarrow \neg(u \in V \setminus V' \& v \in V \setminus V')) \Leftrightarrow$
 $\forall uv(\{u, v\} \in E \rightarrow (u \notin V \setminus V' \vee v \notin V \setminus V')) \Leftrightarrow$
 $\forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')).$ ■

Таким образом, наличие алгоритма нахождения одного из этих множеств вершин в графе влечёт наличие алгоритма нахождения оставшихся двух множеств.

22. Отличия между интуитивным и математическим понятиями алгоритма. Представление о рекурсивных функциях. Тезис Чёрча.

Слово «АЛГОРИТМ» происходит от имени узбекского математика Хорезми (по арабски ал-Хорезми), который разработал правила четырёх арифметических действий над числами в десятичной системе счисления. Совокупность этих правил в Европе стали называть «алгорифм». В последствии это слово переродилось в «алгоритм» и сделалось собирательным названием отдельных правил определенного вида (и не только правил арифметических действий). В течение длительного времени его употребляли только математики, обозначая правила решения задач. В начале XX века понятие алгоритма стало объектом математического изучения (прежде им только пользовались), а с появлением электронных вычислительных машин получило широкую известность.

Интуитивное понятие алгоритма:

Первоначально под алгоритмом понимали произвольную строго определенную последовательность действий, приводящую к решению той или иной конкретной задачи. Первые попытки дать более четкое определение алгоритма привели к следующим требованиям:

- **Определенность данных.** Вид исходных данных строго определен.
- **Дискретность** (от лат. *Discretus* разделенный, прерывистый). Процесс разбивается на отдельные шаги. Это свойство указывает, что любой алгоритм должен состоять из конкретных действий, следующих в определенном порядке. Для всех алгоритмов общим является необходимость строго соблюдения последовательности выполнения действий.
- **Детерминированность** (от лат. *Determinate* – определенность, точность). Результат каждого шага строго определен в зависимости от данных, к которым он применен. Это свойство указывает, что любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае.

- **Элементарность шага.** Переход на один шаг прост.
- **Массовость.** Множество возможных исходных данных потенциально бесконечно. Это свойство показывает, что один и тот же алгоритм можно использовать с разными исходными данными.
- **Направленность.** Это свойство говорит, что считать результатом работы алгоритма, если следующий шаг невозможен.

Математическое понятие алгоритма:

В современной математике и информатике активно используются алгоритмы, не удовлетворяющие такому интуитивному определению: недетерминированные вычисления, алгоритмы с оракулом, зацикливающиеся алгоритмы и т.п.

Для математического уточнения определения понятия алгоритма начиная с 30-х годов XX века были введены различные математические понятия алгоритма. Первыми математическими понятиями алгоритма были рекурсивные функции и машины Тьюринга.

Представление о рекурсивных функциях

Определение. Простейшими называются функции натурального аргумента S , O , I_n^m , определяемые равенствами: $S(x) = x + 1$, $O(x) = 0$, $I_n^m(x_1, \dots, x_n) = x_m$ при $1 \leq m \leq n$.

Определение. Функция f от $n + 1$ переменных получена из функции g от n переменных и функции h от $n + 2$ переменных с помощью оператора примитивной рекурсии, если

$$\begin{cases} f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n) \\ f(x_1, \dots, x_n, y + 1) = h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \end{cases} .$$

Определение. Функция называется **примитивно рекурсивной**, если она может быть получена из простейших с помощью применения операторов подстановки и/или примитивной рекурсии.

Всякая примитивно рекурсивная функция определена для любого набора значений своих аргументов.

Нахождение корней функции вызывает некоторые проблемы. Так, например, μ -оператор, действие которого определяется как $g(x_1, \dots, x_n) = \mu y \{f(x_1, \dots, x_n, y) = 0\}$ (наименьшее y , для которого $f(x_1, \dots, x_n, y) = 0$), применённый к примитивно рекурсивной функции, не всегда даёт в результате примитивно рекурсивную функцию.

Но применение ограниченного μ -оператора, действие которого определяется как $g(x_1, \dots, x_n, z) = \mu y_{\leq z} \{f(x_1, \dots, x_n, y) = 0\}$ (наименьшее y , не превосходящее z и для которого $f(x_1, \dots, x_n, y) = 0$, или 0, если такой y не существует), даёт в результате примитивно рекурсивную функцию.

Определение. *Функция называется частично рекурсивной, если она может быть получена из простейших с помощью применения операторов подстановки, примитивной рекурсии и/или μ -оператора.*

Не все частично рекурсивные функции определены для любого набора значений своих аргументов.

Если ввести обобщённый μ -оператор, определяемый как

$$\mu^* y \{f(\bar{x}, y) = 0\} = \begin{cases} \mu y \{f(\bar{x}, y) = 0\} & \text{если такой } y \text{ существует,} \\ 0 & \text{иначе} \end{cases},$$

то можно определить общерекурсивные (или рекурсивные) функции.

Определение. *Функция называется общерекурсивной, если она может быть получена из простейших с помощью применения операторов подстановки, примитивной рекурсии и/или обобщённого μ -оператора.*

Общерекурсивные функции являются всюду определёнными и всякая примитивно рекурсивная функция является общерекурсивной.

Тезис Чёрча.

Всякая интуитивно вычислимая функция является общерекурсивной.

Это утверждение нельзя ни доказать, ни опровергнуть, так как в его формулировке имеется понятие “интуитивно вычислимая функция”, которое не имеет точного математического определения.

23. Машины Тьюринга и их модификации. Тезис Тьюринга-Чёрча.

Машины Тьюринга

Команда машины Тьюринга имеет вид

$$q_r a_i \rightarrow q_t S a_j,$$

где $i, j = 1, \dots, n$, $r = 1, \dots, k$, $t = 0, 1, \dots, k$, S – сдвиг головки влево, вправо или отсутствие сдвига $S \in \{L, R, _\}$. Эта команда читается следующим образом: «Если машина Тьюринга находится в состоянии q_r и обозревает символ a_i , то этот символ заменяется на a_j , головка производит сдвиг S и машина переходит в состояние q_t .»

Команды называются **согласованными**, если они имеют различные левые части, или полностью совпадают.

Программой машины Тьюринга называется конечное непустое множество согласованных команд.

Машина Тьюринга - это структура $\langle A, Q, Q_I, Q_E, P \rangle$, где

- $A = \{a_1, \dots, a_n\}$ - внешний алфавит или алфавит символов, которые могут быть записаны на ленте, содержащий, в частности, пустой символ;
- $Q = \{q_0, q_1, \dots, q_k\}$ - внутренний алфавит или алфавит состояний, в которых может находиться машина, содержащий два выделенных подмножества:
 - Q_I - множество начальных состояний (обычно q_1)
 - Q_E - множество конечных состояний (обычно q_0)
- P - программа

Конфигурацией машины Тьюринга называется слово вида

$$b_1 \dots b_{p-1} q_r b_p \dots b_l,$$

где

- $b_1 \dots b_{p-1} b_p \dots b_l$ — слово в алфавите A , записанное на ленте;
- слева и справа от этого слова на ленте находятся только пустые символы;
- машина находится в состоянии q_r и обозревает p -ый символ этого слова;
- на концах конфигурации находится не более чем по одному пустому символу.

Машина Тьюринга всегда начинает работу в конфигурации вида $q_1 X$, где X — исходные данные.

Протоколом работы машины Тьюринга называется последовательность конфигураций, первая из которых является начальной, а каждая следующая получена из предыдущей в соответствии с одной из команд.

Машина Тьюринга **заканчивает** работу над данными X , если она пришла в состояние q_0 или ни одна из команд не может быть применена к полученной конфигурации.

Тезис Тьюринга-Чёрча

Всякая интуитивно вычислимая функция может быть вычислена на машине Тьюринга.

Это утверждение нельзя ни доказать, ни опровергнуть, так как в его формулировке имеется понятие интуитивно вычислимая, которое не имеет точного математического определения. Однако, доказана равносильность тезиса Чёрча и тезиса Тьюринга-Чёрча.

Модификации

1. Многоленточные (к-ленточные) МТ

В этой модели предполагается, что имеется k лент, на каждой из которых может быть записано свое слово.

Команда k -ленточной машины Тьюринга имеет вид

$$q_r \begin{pmatrix} a_{i_1} \\ \vdots \\ a_{i_k} \end{pmatrix} \rightarrow q_t \begin{pmatrix} S_1 \\ \vdots \\ S_k \end{pmatrix} \begin{pmatrix} a_{j_1} \\ \vdots \\ a_{j_k} \end{pmatrix},$$

где $S_1, \dots, S_k \in \{L, R, _\}$ и обозначают соответственно сдвиги влево, вправо или отсутствие сдвига головки. Эта команда читается следующим образом: «Если машина Тьюринга находится в состоянии q_r и в обозреваемых ячейках лент записаны соответственно символы a_{i_1}, \dots, a_{i_k} , то эти символы заменяются соответственно на a_{j_1}, \dots, a_{j_k} , головка производит сдвиги S_1, \dots, S_k и машина Тьюринга переходит в состояние q_t ».

Обычно предполагается, что 1-ая лента — это входная лента для записи исходных данных, k -ая лента — это выходная лента для записи результата, остальные $k - 2$ ленты — это рабочие ленты.

2. Многоголовчатые (m -головчатые) МТ

В этой математической предполагается, что имеется m головок, каждая из которых может обозревать одну ячейку ленты.

Многоголовчатые машины Тьюринга являются достаточно адекватной моделью для параллельных вычислений, где m процессоров обрабатывают общую память.

В этой модели возможны два варианта модификации:

- запрет на то, чтобы разные головки обозревали одну и ту же ячейку, точнее, требование, чтобы головка с большим номером всегда обозревала ячейку, которая находится правее, чем ячейка, обозреваемая головкой с меньшим номером;
- головкам приписывается приоритет и в случае, если несколько головок обозревают одну и ту же ячейку, команда распространяется только на головку с наивысшим приоритетом, а остальные из этих головок пропускают свой шаг

Команда m -головчатой машины Тьюринга имеет вид

$$q_r(a_{i_1}, \dots, a_{i_m}) \rightarrow q_t(S_1, \dots, S_m)(a_{j_1}, \dots, a_{j_m}),$$

где $S_1, \dots, S_m \in \{L, R, _\}$ и обозначают соответственно сдвиги влево, вправо или отсутствие сдвига головки.

3. Недетерминированные МТ

Недетерминированные машины Тьюринга получаются, если в программе классической машины Тьюринга разрешить использование несогласованных команд.

Недетерминированные машины Тьюринга часто используются для проверки истинности утверждений типа $\exists Y P(X, Y)$ (существует такой объект Y , для которого справедливо утверждение $P(X, Y)$). Для проверки истинности таких утверждений работу недетерминированной машины Тьюринга можно разбить на два этапа:

- этап угадывания, при реализации которого лента в недетерминированном режиме размножается, и на каждой из них выписывается «претендент» на решение;
- этап проверки, при реализации которого машина работает в детерминированном режиме и проверяет конкретного «претендента» на то, является ли он решением.

Вместо одного заключительного состояния q_0 обычно при этом используют два q_Y и q_N (происходящих от слов Yes и No). Недетерминированная машина Тьюринга заканчивает работу в состоянии q_Y , если хоть на одной из лент она пришла в состояние q_Y . Если же на всех лентах недетерминированная машина Тьюринга пришла в состояние q_N , то и вся машина заканчивает работу в этом состоянии q_N .

24. Теорема о числе шагов МТ, моделирующей работу k-ленточной МТ.

3.3.3. Теорема о числе шагов машины Тьюринга, моделирующей работу многоленточной машины Тьюринга.

Теорема 3.3. *По всякой k -ленточной машине Тьюринга MT_k , заканчивающей работу с исходными данными X за t шагов, можно построить одноленточную машину Тьюринга MT_1 , результат работы которой с исходными данными X совпадает с результатом работы исходной и число шагов которой составляет $O(t^2)$.*

Доказательство. Пусть длина записи исходных данных равна n , причём $n \leq t$. Будем считать, что 1-ая лента входная, а последняя — выходная.

На i -ом шаге ($i = 1, \dots, t$) длина записи на j -ой ленте MT_k ($j = 2, \dots, k$) может увеличиться не более чем на единицу (т.е. стать равной i), причём запись нового символа может проходить как в середине слова, так и на одном из его концов. При этом содержимое лент имеет вид

$$\begin{pmatrix} X \\ X_i^2 \\ \vdots \\ X_i^k \end{pmatrix}.$$

Конфигурация моделирующей машины МТ1 в этот момент имеет вид

$$X * q_l X_i^2 * \dots * X_i^k$$

при некотором l .

Для моделирования i -го шага МТк машина МТ1 должна для каждого ($j = 2, \dots, k$)

- сдвинуть головку на символ, обозреваемый МТк на j -ой ленте (не более, чем $\|X_i^{j-1}\| + \|X_i^j\|$ шагов);

- произвести действие, которое МТк производит со словом X_i^j (1 шаг);

- в случае необходимости переместить всё содержимое ленты правее положения головки на одну ячейку вправо (не более, чем $4 \sum_{j'=j}^k \|X_i^{j'}\|$ шагов);

- вернуться в исходное положение (не более, чем $\sum_{j'=j}^k \|X_i^{j'}\|$ шагов).

Всего при моделировании действия МТк с одним символом на i -ом шаге МТ1 совершает не более

$$\begin{aligned} & \sum_{j=2}^k \left(\|X_i^{j-1}\| + \|X_i^j\| + 1 + 4 \sum_{j'=j}^k \|X_i^{j'}\| + \sum_{j'=j}^k \|X_i^{j'}\| \right) \leq \\ & \sum_{j=2}^k \left(i + i + 1 + 5 \sum_{j'=j}^k i \right) = \sum_{j=2}^k (5(k-j)i + 2i + 1) = \\ & \frac{1}{2} 5i(k-1)(k-2) + 2i(k-1) + 2(k-1) = \\ & \frac{3}{2} i(k-1)(3k-4) + 2(k-1). \end{aligned}$$

Просуммировав полученное выражение по $i = 1, \dots, t$ имеем

$$\begin{aligned} & \frac{3}{2} \sum_{i=1}^t i ((k-1)(3k-4) + 2(k-1)) = \\ & \frac{3}{2} (k-1)(3k-4) \frac{t(t-1)}{2} + 2(k-1) = O(k^2 t^2). \end{aligned}$$

Так как k является константой, то получили $O(t^2)$. ■

25. Недетерминированные МТ. Теорема о числе шагов МТ, моделирующей работу недетерминированной МТ.

3.3.5. Недетерминированные машины Тьюринга. Теорема о числе шагов машины Тьюринга, моделирующей работу недетерминированной машины Тьюринга.

Недетерминированные машины Тьюринга получаются, если в программе классической машины Тьюринга разрешить использование несогласованных команд. Как же следует выполнить, например, такую пару команд

$$\begin{aligned} q_1 0 &\rightarrow q_1 L1 \\ q_1 0 &\rightarrow q_2 R0, \end{aligned}$$

Вместо одного заключительного состояния q_0 обычно при этом используют два q_Y и q_N (происходящих от слов Yes и No). Недетерминированная машина Тьюринга заканчивает работу в состоянии q_Y , если хоть на одной из лент она пришла в состояние q_Y . Если же на всех лентах недетерминированная машина Тьюринга пришла в состояние q_N , то и вся машина заканчивает работу в этом состоянии q_N .

3.3.6. Теорема о числе шагов машины Тьюринга, моделирующей работу недетерминированной машины Тьюринга.

Теорема 3.4. *По всякой недетерминированной машине Тьюринга, проверяющей предикат $\exists Y P(X, Y)$ и заканчивающей работу с исходными данными X за t шагов, можно построить одноленточную машину Тьюринга, результат работы которой с исходными данными X совпадает с результатом работы исходной и число шагов которой составляет $2^{O(t)}$.*

Для доказательства этой теоремы докажем две леммы.

Лемма 3.3. *Если недетерминированная машина Тьюринга, проверяющая предикат $\exists Y P(X, Y)$ заканчивает работу с исходными данными X за t шагов, то длина записи «претендентов» Y не превосходит t .*

Утверждение леммы следует из того, что длина записи слова не может быть больше, чем число шагов, затраченных на его выписывание.

Лемма 3.4. *Если недетерминированная машина Тьюринга, проверяющая предикат $\exists Y P(X, Y)$ заканчивает работу с исходными данными X за t шагов, то количество «претендентов» Y не превосходит $2^{O(t)}$.*

Доказательство. Пусть $A = \{a_1, \dots, a_k\}$ — внешний алфавит недетерминированной машины Тьюринга. Количество «претендентов» m не превосходит количества слов в этом алфавите, длина которых не превосходит t , т.е. $m \leq \sum_{i=0}^t k^i = \frac{k^{t+1}-1}{k-1} \leq k^{t+1} = 2^{(t+1)\log k} = 2^{O(t)}$, так как k является константой. ■

Доказательство теоремы. Работу детерминированной машины Тьюринга организуем следующим образом:

— порождаем Y_1 , проверяем $P(X, Y_1) \leq t$ шагов,

⋮

— порождаем Y_m , проверяем $P(X, Y_m) \leq t$ шагов.

Общее число шагов не превосходит $m \cdot t \leq t \cdot 2^{O(t)} = 2^{O(t+\log t)} = 2^{O(t)}$. ■

26. Понятия сложности алгоритма от данных, сложность алгоритма, сложность задачи. Верхняя и нижняя оценки сложности.

Вычислительная сложность

Под вычислительной сложностью алгоритма понимают функцию, зависящую от ДЛИНЫ записи исходных данных и характеризующую

- *число шагов работы алгоритма над исходными данными (временная сложность);*
- *объём памяти, необходимой для работы алгоритма над исходными данными (ёмкостная или зональная сложность).*

Сложность, верхняя и нижняя оценка

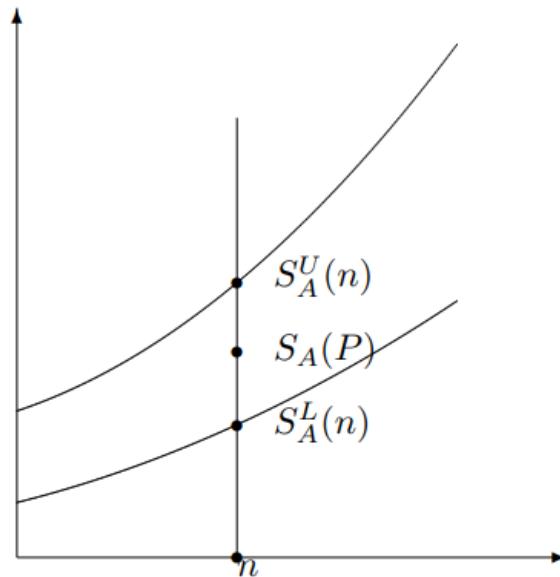
Определение. Сложностью $S_A(P)$ алгоритма A при работе над данными P называется число шагов или объём памяти, затраченные в процессе работы алгоритма A над данными P .

Определение. Верхней (нижней) оценкой сложности алгоритма A при работе над данными длины n называется

$$S_A^U(n) = \max_{P: \|P\|=n} \{S_A(P)\}$$

(соответственно

$$S_A^L(n) = \min_{P: \|P\|=n} \{S_A(P)\}.$$

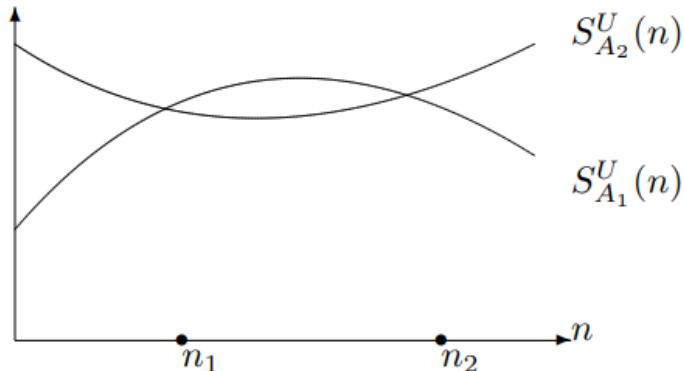


Точная верхняя оценка

Определение. Точной верхней оценкой сложности задачи Z с исходными данными длины n называется

$$S_Z^U(n) = \min_{A: A \text{ решает } Z} \{S_A^U(n)\}.$$

Так, например, если для задачи Z имеется только 2 алгоритма A_1 и A_2 , то точная верхняя оценка сложности задачи Z совпадает с $S_{A_1}^U(n)$ при $0 \leq n \leq n_1$, с $S_{A_2}^U(n)$ при $n_1 \leq n \leq n_2$ и с $S_{A_1}^U(n)$ при $n \geq n_2$.



Нахождение точной верхней оценки сложности задачи довольно трудное (и к тому же неблагодарное) дело. Обычно устанавливают порядок таких оценок или их асимптотику. В дальнейшем изложении будет использована O -символика для указания на порядок роста функций сложности.

$$f(n) = O(g(n)) \iff \exists C \forall n (f(n) \leq C \cdot g(n)).$$

27. Соотношение между временем работы алгоритма и требуемой памятью

В методичке - различие между числом шагов алгоритма и вычислительной сложностью.

Вычислительная сложность - функция от !! ДЛИНЫ !! записи исходных данных.
Требуемая память - функция от !! ДЛИНЫ !! записи исходных данных.

Под вычислительной сложностью алгоритма понимают функцию, зависящую от ДЛИНЫ записи исходных данных и характеристирующую

- *число шагов работы алгоритма над исходными данными (временная сложность);*
- *объём памяти, необходимой для работы алгоритма над исходными данными (ёмкостная или зональная сложность).*

Различие между числом шагов и вычислительной сложностью следует из того, что понимается под словом “шаг”.

Для нахождения вычислительной сложности алгоритма “шаг” должен быть реализован на машине Тьюринга по крайней мере за полиномиальное от длины записи аргументов число ее переходов.

Соотношение между числом и длиной его записи

$$n = 2^k \cdot n_0 + \dots + 2 \cdot n_k + n_{k+1}, \text{ где } n_i \in \{0, 1\}, k_1 \neq 0, ||n|| = k + 1$$

$$\text{Имеем } 2^k \leq n < 2^{k+1} \Rightarrow k \leq \log n < k + 1 \Rightarrow ||n|| - 1 \leq \log n < ||n|| \Rightarrow ||n|| = \lceil \log n \rceil$$

Вывод: длина это примерно логарифм

Пример 1

Число “шагов” вычисления функции $f(n) = 2^n$

Известен алгоритм быстрого возведения в степень, выполняющий $\log n$ “шагов”

Каждый “шаг” - это умножение каких-то двух чисел (они сами по себе могут быть достаточно длинными, и шаги не обязательно выполняются за константу)

Как мы уже знаем, $||n|| \approx \log n$. Значит, длина записи $||2^n|| \approx \log 2^n = n$.

Вывод: время работы алгоритма (число шагов) может быть существенно меньше требуемой памяти (длины записи результата) (и, соответственно, его вычислительной сложности)

Пример 2

Вычисление n-того числа Фибоначчи

$$F_0 = 0, F_1 = 1, \dots, F_{n+2} = F_n + F_{n+1}$$

Пусть матрица P имеет вид

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Тогда $(F_{n-2} F_{n-1}) \cdot P = (F_{n-1} F_n)$, т.е. $(0 1) \cdot P^n = (F_n F_{n+1})$

Следовательно, F_{n+1} вычисляется за n умножений матрицы P на себя и одного умножения на $(0 1)$, линейное число “шагов”.

Но это не значит, что вычислительная сложность линейна.

Известно, что скорость роста чисел Фибоначчи экспоненциальна. Точнее:

$$F_n = [\frac{1}{\sqrt{5}} (\frac{1+\sqrt{5}}{2})^n] \text{ и } \|F_n\| = O(n).$$

По исходному n , длина записи которого $\|n\| \approx \log n$, записывается результат, длина записи которого $\|F_n\| = O(n) = O(2^{\|n\|})$

Вывод: время работы алгоритма (число шагов) может быть существенно меньше требуемой памяти (длины записи результата) (и, соответственно, его вычислительной сложности)

почему-то во всех старых конспектах в этом билете есть такая таблица

Время решения задачи с исх. различие времени n
и её решением сложности f (10^6 оп/с)

$\frac{n}{5}$	10	20	30	40	50	60
n	10^{-5} с	$2 \cdot 10^{-5}$ с	$3 \cdot 10^{-5}$ с	$4 \cdot 10^{-5}$ с	$5 \cdot 10^{-5}$ с	$6 \cdot 10^{-5}$ с
n^2	10^{-4} с	$4 \cdot 10^{-4}$ с	$9 \cdot 10^{-4}$ с	$1,6 \cdot 10^{-3}$ с	$2,5 \cdot 10^{-3}$ с	$3,6 \cdot 10^{-3}$ с
n^5	10^{-1} с	3,2 с	24,3 с	102,4 с	312,5 с	777,6 с
2^n	10^{-3} с	≥ 1 с	≥ 1000 с	16600 с	≥ 1000 дн	≈ 30 тысячелетий

$$f(n) \cdot 10^{-6}$$

Изменение размера исх. данных, обрабатывают решения
за 1 час, при увелечении быстродействия

$\frac{n}{5}$	10^6	10^8	10^9	это как бы оп/с
n	N_1	$100 N_1$	$1000 N_1$	
n^2	N_2	$10 N_2$	$\approx 30 N_2$	
n^5	N_3	$2,5 N_3$	$3,98 N_3$	
2^n	N_4	$N_4 + 6,64$	$N_4 + 9,96$	

$$1 \text{ час} = f(N_i) \cdot 10^{-6} = f(N_i') \cdot 10^{-8} = f(N_i'') \cdot 10^{-9}$$

Вывод: в то время, как для полиномиальных задач увеличение быстродействия помогает что-то сделать, для экспоненциальных задач оно практически ничего не дает.

28. Классы алгоритмов и задач. Схема обозначений.

Задача принадлежит классу сложности C, если существует алгоритм из класса C, решающий эту задачу.

Буква F в начале класса используется для обозначения класса функций, если её нет, то это класс предикатов.

Буквы **D** и **N** обозначают, что в определении класса сложности использована детерминированная или соответственно недетерминированная машина Тьюринга. **D** обычно не пишется(типа дефолт).

Функция сложности - это функция от длины записи исходных данных, ограничивающая число шагов или число шагов соответствующей машины Тьюринга.

Класс функций или предикатов	Детерминированная или недетерминированная	Функция сложности	Временная или ёмкостная
F	[D] N	LOG LIN QLIN P (Poly) EXP-LIN EXP :	[TIME] SPACE

Табл. 3. Обозначения для некоторых классов сложности.

29. Классы P, NP, и P-SPACE. Соотношения между этими классами.

Класс P - это класс предикатов, для которых существует алгоритм, который может быть реализован на детерминированной машине Тьюринга, число шагов которого не превосходит полинома от длины записи исходных данных.

Класс NP - это класс предикатов, для которых существует алгоритм, который может быть реализован на недетерминированной машине Тьюринга, число шагов которого не превосходит полинома от длины записи исходных данных.

Класс P-SPACE - это класс предикатов, для которых существует алгоритм, который может быть реализован на детерминированной машине Тьюринга, число использованных ячеек которой не превосходит полинома от длины записи исходных данных.

В настоящее время известны следующие соотношения между основными классами сложности: $P \subseteq NP \subseteq P\text{-SPACE} \subseteq EXP$. Вопрос о том, строгими или пестроточечными являются первые два включения, объявлен одним из труднейших вопросов математики на XXI век.

Теорема 4.1. Если предикат вида $\exists Y P(X, Y)$ принадлежит классу **NP**, то существует проверяющая его одноленточная машина Тьюринга, число шагов которой составляет $2^{p(n)}$, где $p(n)$ – полином от длины записи исходных данных $n = \|X\|$.

30. Полиномиальная сводимость и полиномиальная эквивалентность.

Определение. Задача Z_1 вида $\exists Y P_1(X, Y)$ при $X \in D_1$ полиномиально сводится к задаче Z_2 вида $\exists Y P_2(X, Y)$ при $X \in D_2$

$$Z_1 \propto Z_2,$$

если существует функция f , отображающая D_1 в D_2 и такая, что

- существует машина Тьюринга, вычисляющая функцию f не более чем за полиномиальное от длины записи исходных данных число шагов ($f \in \mathbf{FP}$);
- задача Z_1 имеет решение с исходными данными X тогда и только тогда, когда задача Z_2 имеет решение с исходными данными $f(X)$

$$\forall X \in D_1 (\exists Y P_1(X, Y) \leftrightarrow \exists Y P_2(f(X), Y)).$$

Далее, если это не будет вызывать неоднозначного прочтения, под задачей Z_i всегда будем понимать задачу вида $\exists Y P_i(X_i, Y)$ при $X_i \in D_i$.

Лемма 4.1. Отношение полиномиальной сводимости рефлексивно $\forall Z (Z \propto Z)$ и транзитивно $\forall Z_1 Z_2 Z_3 (Z_1 \propto Z_2 \ \& \ Z_2 \propto Z_3 \rightarrow Z_1 \propto Z_3)$.

Доказательство непосредственно следует из того, что тождественное отображение принадлежит классу **FP**, а также сумма полиномов является полиномом.

Лемма 4.2. Если $Z_1 \propto Z_2$ и $Z_2 \in \mathbf{P}$, то $Z_1 \in \mathbf{P}$.

Доказательство леммы следует из того, что в качестве алгоритма решения задачи Z_1 можно взять следующий: к исходным данным X задачи Z_1 применяем функцию f , осуществляющую полиномиальную сводимость ($p_1(\|X\|)$ шагов), а затем решаем задачу Z_2 с данными $f(X)$ (длина записи которых не превосходит $p_1(\|X\|)$ за $p_2(p_1(\|X\|))$ шагов. Всего потребуется не более $p_1(\|X\|) + p_2(p_1(\|X\|))$ шагов. Здесь p_1 и p_2 – полиномы.

Определение. Задача Z_1 полиномиально эквивалентна задаче Z_2 ($Z_1 \sim_p Z_2$), если $Z_1 \propto Z_2$ и $Z_2 \propto Z_1$.

Теорема 4.2. Отношение полиномиальной эквивалентности является отношением эквивалентности, то есть оно

- рефлексивно $\forall Z (Z \sim_p Z)$,
- симметрично $\forall Z_1 Z_2 (Z_1 \sim_p Z_2 \rightarrow Z_2 \sim_p Z_1)$ и
- транзитивно $\forall Z_1 Z_2 Z_3 (Z_1 \sim_p Z_2 \& Z_2 \sim_p Z_3 \rightarrow Z_1 \sim_p Z_3)$.

Доказательство очевидно.

Отношение \sim_p разбивает класс **NP** на классы эквивалентности. Один из них – класс **P** – самые «быстро решаемые» задачи из класса **NP**.

31. Полиномиальная сводимость задачи ГЦ к задаче КОМИВОЯЖЁР.

ГАМИЛЬТОНОВ ЦИКЛ (ГЦ)	КОМИВОЯЖЁР
<p>Дано: граф $G = (V, E)$. $(V = n)$</p> <p>Вопрос: существует ли в G гамильтонов цикл?</p> <p>$\exists(v_{i_1}, \dots, v_{i_n})(\{v_{i_1}, \dots, v_{i_n}\} = V \& \{v_{i_1}, v_{i_2}\} \in E \& \dots, \& \{v_{i_n}, v_{i_1}\} \in E)$</p>	<p>Дано: $C = \{c_1, \dots, c_n\}$ – множество городов, $d_{ij} \in \mathbf{Z}_+$ – расстояния между c_i и c_j, $B \in \mathbf{Z}_+$.</p> <p>Вопрос: существует ли маршрут, проходящий через все города, длина которого не больше B?</p> <p>$\exists(c_{i_1}, \dots, c_{i_m})(\{c_{i_1}, \dots, c_{i_m}\} = C \& \sum_{j=1}^{m-1} d_{i_j i_{j+1}} + d_{i_m i_1} \leq B)$</p>

Покажем, что **ГЦ** \propto **КОМИВОЯЖЁР**. Для этого предъявим полиномиальный по времени алгоритм, который по графу $G = (V, E)$ строит исходные данные C , d_{ij} и B с требуемыми свойствами.

$$C = V, \quad B = n,$$

$$d_{ij} = \begin{cases} 1, & \text{если } \{v_i, v_j\} \in E \\ 2, & \text{если } \{v_i, v_j\} \notin E \end{cases}.$$

В графе есть гамильтонов цикл, тогда и только тогда, когда маршрут проходит только между теми городами, расстояния между которыми равно 1.

32. Классы эквивалентности по отношению полиномиальной эквивалентности. Класс P - пример такого класса.

Определение. Задача Z_1 полиномиально эквивалентна задаче Z_2 ($Z_1 \sim_p Z_2$), если $Z_1 \propto Z_2$ и $Z_2 \propto Z_1$.

Теорема 4.2. Отношение полиномиальной эквивалентности является отношением эквивалентности, то есть оно

- рефлексивно $\forall Z(Z \sim_p Z)$,
- симметрично $\forall Z_1 Z_2 (Z_1 \sim_p Z_2 \rightarrow Z_2 \sim_p Z_1)$ и
- транзитивно $\forall Z_1 Z_2 Z_3 (Z_1 \sim_p Z_2 \ \& \ Z_2 \sim_p Z_3 \rightarrow Z_1 \sim_p Z_3)$.

Доказательство очевидно.

Отношение \sim_p разбивает класс **NP** на классы эквивалентности. Один из них – класс **P** – самые «быстро решаемые» задачи из класса **NP**.

Из википедии:

Задачи, принадлежащие классу P

Примерами задач из класса P являются целочисленное сложение, умножение, деление, взятие остатка от деления, умножения матриц, выяснение связности графов, сортировка множества из n чисел, нахождение эйлерова цикла на графе из m рёбер, обнаружение в тексте длиной n некоторого слова, построение покрывающего дерева минимальной стоимости, линейное программирование и некоторые другие.

33. NP-полные задачи. Класс NP-полных задач - класс эквивалентности по отношению полиномиальной эквивалентности.

Задача Z называется NP-полной, если она принадлежит классу NP и любая задача из класса NP полиномиально сводится к ней.

Теорема: Класс NP-полных задач образует класс эквивалентности по отношению полиномиальной эквивалентности в классе NP.

Замечание. Если в определении NP-полной задачи убрать требование её принадлежности классу **NP**, то получим определение NP-трудной задачи. В частности, если задача распознавания $\exists Y P(X, Y)$ является NP-полной, то соответствующая ей задача поиска $(?Y) P(X, Y)$ является NP-трудной.

34. Задача ВЫПОЛНИМОСТЬ (ВЫП). Теорема Кука.

Определение:

Выполнимость(ВЫП)

Дано:

$U = \{u_1, \dots, u_n\}$ - множество произвольных переменных

$C = \{c_1, \dots, c_m\}$ - множество предложений (простых дизъюнкций) над U

Вопрос: выполнимо ли множество C , т.е. существует ли набор значений для переменных из U , для которого истинны все предложения из C ?

$$\exists u_1, \dots, u_n (c_1 \ \& \ \dots \ \& \ c_m).$$

1-ВЫП - каждое C_i содержит 1 переменную

2-ВЫП - каждое C_i содержит 2 переменных

... (и т.д.)

Теорема Кука: Задача ВЫП NP-полна.

Доказательство этой теоремы имеется в [4]. Здесь будет приведена только схема её доказательства.

Во-первых, необходимо доказать, что эта задача принадлежит классу **NP**.

Действительно, порождение всех возможных наборов значений переменных u_1, \dots, u_n может быть осуществлено на недетерминированной машине Тьюринга за n шагов (такая недетерминированная машина Тьюринга, которая в качестве исходных данных имеет строку $\underbrace{\dots}_n$, была приведена в качестве примера в главе 3).

На каждой из полученных в недетерминированном режиме 2^n лент вычисляется $c_1 \ \& \ \dots \ \& \ c_m$ для найденных значений переменных. На двухленточной машине Тьюринга это можно сделать за $O(n\|C\|)$ шагов. Следовательно, на одноленточной — за $O(n^2\|C\|^2)$, т.е. за полином от $\|U\|$ и $\|C\|$.

Для доказательства того, что всякая задача из класса **NP** полиномиально сводится к задаче ВЫП, воспользуемся тем, что каждая такая задача может быть задана программой M для недетерминированной машины Тьюринга, заканчивающей работу не более чем за полином шагов от длины записи исходных данных.

Затем по каждой индивидуальной задаче из класса **NP** (т.е. программе M над алфавитом $S = \{s_0, \dots, s_v\}$ с состояниями $\{q_1, \dots, q_{r-2}, q_Y, q_N\}$ и исходным данным X длины n), проверяемой не более чем за $p(n)$ шагов, построим индивидуальную задачу из ВЫП таким образом, что они имеют решение одновременно.

Для этого введём пропозициональные переменные

$Q_{i,k}$ — «На i -ом шаге программа M находится в состоянии q_k . ($0 \leq i \leq p(n)$, $1 \leq k \leq r$)

$H_{i,j}$ — «На i -ом шаге читающая/пишущая головка обозревает j -ую ячейку». ($0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$.)

$S_{i,j,k}$ — «На i -ом шаге в j -ой ячейке записан символ s_k ». ($0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)$, $0 \leq k \leq v$.)

Очевидно, что количество этих переменных полиномиально от длины записи индивидуальной задачи. Следовательно, длина их записи не превосходит значения этого полинома, умноженного на его логарифм, и тоже полиномиальна.

После этого записывается $6p(n)(p(n) + 1)(r + 1)(v + 1)$ предложений (вид предложений см. в [4]), истинных в том и только том случае, когда недетерминированная машина Тьюринга работает в соответствии с заданной программой над данными X и заканчивает работу в состоянии q_Y не более чем за $p(n)$ шагов.

- В каждый момент времени i ($0 \leq i \leq p(n)$) машина с программой M находится ровно в одном состоянии.
- В каждый момент времени i ($0 \leq i \leq p(n)$) читающая/пишущая головка обозревает ровно одну ячейку.
- В каждый момент времени i ($0 \leq i \leq p(n)$) каждая ячейка содержит ровно один символ из S .
- В момент времени 0 вычисление находится в исходной конфигурации стадии проверки при входе X .
- Не позднее, чем через $p(n)$ шагов M переходит в состояние q_Y и, следовательно, принимает X .
- В каждый момент времени i ($0 \leq i < p(n)$) конфигурация программы M в момент времени $i+1$ получается из конфигурации в момент времени i одноразовым применением команды программы M .

Длина каждого предложения не превосходит $2p(n)$. Таким образом, на выписывание этих предложений требуется $O(p(n)^3rv)$ шагов, т.е. сводимость полиномиальна.

35. Задача 3-ВЫПОЛНИМОСТЬ (3-ВЫП). Её NP-полнота.

3-ВЫПОЛНИМОСТЬ (3-ВЫП)

ДАНО:

У - множество пропозициональных переменных,

С - множество предложений над У с тремя переменными каждое.

ВОПРОС:

Существует ли набор значений для переменных, выполняющий множество С?

$\exists u_1, \dots, u_n (c_1 \& \dots \& c_m)?$

Теорема 4.4. Если для задачи Z и NP -полной задачи Z_1 выполнены условия

$Z \in NP$,
 $Z_1 \propto Z$,
то Z NP -полнна.

Доказательство основано на транзитивности отношения полиномиальной сводимости.

С помощью этой теоремы докажем NP -полноту задач 3-ВЫП и ВП.

Теорема 4.5. Задача 3-ВЫП NP -полнна.

Доказательство. То, что эта задача принадлежит классу NP , доказывается аналогично принадлежности классу NP задачи ВЫП.

1. 3-ВЫП принадлежит NP (так как является подзадачей ВЫП)
2. Покажем, что задача ВЫП полиномиально сводится к задаче 3-ВЫП
Пусть $U = \{u_1, \dots, u_n\}$, $C = \{c_1, \dots, c_n\}$ - исходные данные для ВЫП.
Построим U' , C' для 3-ВЫП. $U \subset U'$

Пусть $c_j = z_1 \vee \dots \vee z_k$, где литерал $z_i = u_j$ или $\neg u_j$

Если $k = 3$, то $c_j \in C'$

Если $k = 2$, то $c_j = z_1 \vee z_2 \Leftrightarrow (z_1 \vee z_2 \vee y^j) \& (z_1 \vee z_2 \vee \neg y^j)$

т.е. введем два новых предложения $c_{j_1}^1$ и $c_{j_2}^2$, $c_{j_1}^{1,2} \in C'$
в них новая переменная $y^j \in U'$

Если $k = 1$, то $c_j = z_1 \Leftrightarrow (z_1 \vee y^j_1 \vee y^j_2) \& (z_1 \vee \neg y^j_1 \vee \neg y^j_2) = c_{j_1}^1 \& c_{j_2}^2 \& c_{j_3}^3 \& c_{j_4}^4$
(раскрыли скобки)

т.е. $y^j_1, y^j_2 \in U'$, $c_{j_1}^1, c_{j_2}^2, c_{j_3}^3, c_{j_4}^4 \in C'$

Если $k \geq 4$, то построим предложения, выполнимость конъюнкции которых равносильна выполнимости исходного предложения.

$$c_j = z_j^1 \vee z_j^2 \vee \dots \vee z_j^{k-1} \vee z_j^k$$

\Leftrightarrow

$$\begin{aligned}
c_j^1 &= z_j^1 \vee z_j^2 \vee y_j^1, \\
c_j^1 &= \neg y_j^1 \vee z_j^3 \vee y_j^2, \\
&\vdots \\
c_j^{l-1} &= \neg y_j^{l-2} \vee z_j^l \vee y_j^{l-1}, \\
&\vdots \\
c_j^{k-3} &= \neg y_j^{k-4} \vee z_j^{k-2} \vee y_j^{k-3}, \\
c_j^{k-2} &= \neg y_j^{k-3} \vee z_j^{k-1} \vee z_j^k.
\end{aligned}$$

Действительно, если исходное предложение выполнимо, например, при $z_j^l = \text{true}$, то $y_j^1, y_j^2, \dots, y_j^{l-2}$ присвоим значения *true*, а $y_j^{l-1}, \dots, y_j^{k-3}$ присвоим значения *false*. Эти значения обеспечивают истинность построенных предложений.

Если выполнена конъюнкция построенных предложений, то хоть при одном l значение z_j^l равно *true*. В противном случае, просматривая предложения сверху в них до предпоследнего включительно получаем, что значения $y_j^1, y_j^2, \dots, y_j^{k-3}$ равны *true*. При этом в последнем предложении стоит дизъюнкция ложных литералов.

Покажем, что построение U' и C' по заданным U и C полиномиально от $\|U\|$ и $\|C\|$.

Пусть K - максимальное число литералов в предложениях, $K \leq \|C\|$.

Количество новых переменных в U' не превосходит $\max\{2, K - 3\} = O(K)$, общее количество переменных в U' составляет $O(\|U\| + \|C\|)$. Переменные можно записать так, что длина записи имени переменной не превосходит $\log(\|U'\|) = O(\log(\|U\| + \|C\|)) = O(\|U\| + \|C\|)$. Таким образом, запись множества U' может быть реализована за $O((\|U\| + \|C\|)^2)$ шагов машины Тьюринга.

Количество предложений в C' не превосходит $\|C\| \cdot \max\{4, K - 2\} = O(\|C\|^2)$. Каждое предложение в C' содержит 3 литерала, поэтому длина его записи составляет $O(3(\|U\| + \|C\|))$. Таким образом, запись множества C' может быть реализована за $O(\|C\|^2(\|U\| + \|C\|))$ шагов машины Тьюринга.

Сложив полученные оценки и учитывая, что $\|U\| \leq \|C\|$, получаем полиномиальность от $\|U\|$ и $\|C\|$ построения U' и C' по заданным U и C : $O((\|U\| + \|C\|)^2) + \|C\|^2(\|U\| + \|C\|) = O(\|C\|^3)$. ■

**36. Задачи ВЕРШИННОЕ ПОКРЫТИЕ (ВП),
НЕЗАВИСИМОЕ МНОЖЕСТВО (НМ), КЛИКА. NP-полнота
задачи ВП. Полиномиальная эквивалентность этих трёх
задач.**

Определение задач

ВЕРШИННОЕ ПОКРЫТИЕ (ВП)

ДАНО: Граф $G = (V, E)$,
 $K \in Z_+, K \leq |V|$.

ВОПРОС: Имеется ли в G вершинное покрытие не более чем из K элементов?

$$\exists V' (V' \subseteq V \ \& \ |V'| \leq K \ \& \ \forall uv (\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')))$$

НЕЗАВИСИМОЕ МНОЖЕСТВО (НМ)

ДАНО: Граф $G = (V, E)$,
 $J \in Z_+, J \geq |V|$.

ВОПРОС: Имеется ли в G независимое множество не менее чем из J элементов?

$$\exists V' (V' \subseteq V \ \& \ |V'| \geq J \ \& \ \forall uv ((u \in V' \ \& \ v \in V') \rightarrow \{u, v\} \notin E))$$

КЛИКА

ДАНО: Граф $G = (V, E)$,
 $J \in Z_+, J \leq |V|$.

ВОПРОС: Содержит ли G клику не менее чем из J вершин?

$$\exists V' (V' \subseteq V \ \& \ |V'| \geq J \ \& \ \forall uv (u \in V' \ \& \ v \in V' \rightarrow \{u, v\} \in E))$$

NP-полнота задачи ВП

Теорема. Задача ВП NP-полнна.

Доказательство.

1. ВП $\in NP$, т.к.
 - a. В недетерминированном режиме порождаем все подмножества V :
 V' , $|V| \leq K$ за $K \log K$ шагов на 2^K лентах ($K \leq |V|$)
 - b. В детерминированном режиме для каждого подмножества V' проверка
 $\forall uv (\{u, v\} \in E \rightarrow (u \in V' \vee v \in V'))$ на машине с прямым доступом
 возможна за $O(|E| \cdot |V|)$
 - c. Детерминированная машина Тьюринга моделирует работу РАМ за число
 ее шагов в кубе \Rightarrow получаем полиномиальное от длины записи графа

число шагов недетерминированной машины Тьюринга, решающей задачу ВП.

2. 3-ВЫП \propto ВП

Пусть $U = \{u_1, \dots, u_n\}$, $C = \{c_1, \dots, c_n\}$ - исходные данные для 3-ВЫП.

Каждой переменной u_i ($i \in [1, n]$) поставим в соответствие две вершины графа - u_i и \bar{u}_i ($2||U||$ шагов) и ребро $\{u_i, \bar{u}_i\}$ ($2||U||$ шагов)

Каждому предложению c_j поставим в соответствие 3 вершины графа:

$a_{j_1}^1, a_{j_1}^2, a_{j_1}^3$ ($3||C||$ шагов) и три ребра $\{a_{j_1}^1, a_{j_1}^2\}, \{a_{j_1}^2, a_{j_1}^3\}, \{a_{j_1}^3, a_{j_1}^1\}$ ($6||C||$ шагов)

Если k -тым ($k = 1, 2, 3$) членом предложения является u_i , добавим ребро $\{u_i, a_{j_k}^k\}$

Если k -тым ($k = 1, 2, 3$) членом предложения является $\neg u_i$, добавим ребро $\{\neg u_i, a_{j_k}^k\}$ (это и предыдущее - $6||C||$ шагов)

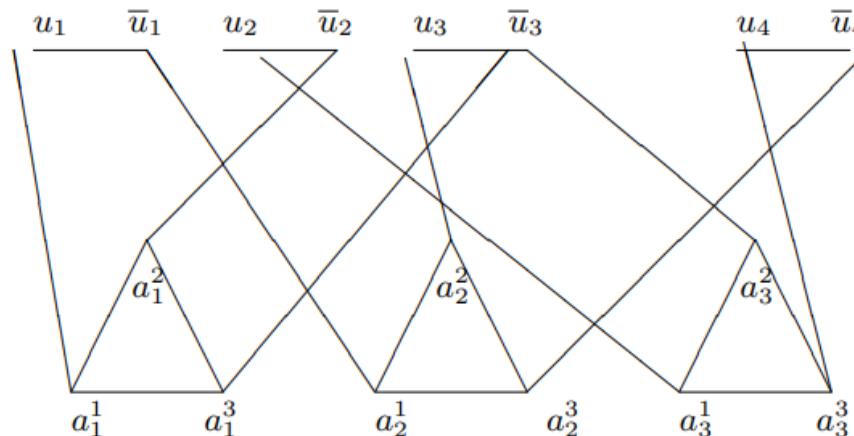
Покрытие каждого из ребер $\{u_i, \bar{u}_i\}$ не может содержать менее n вершин, а покрытие каждого из ребер треугольника $\{a_{j_1}^1, a_{j_1}^2\}, \{a_{j_1}^2, a_{j_1}^3\}, \{a_{j_1}^3, a_{j_1}^1\}$ не может содержать менее $2m$ вершин \Rightarrow положим $K = n + 2m$ (K - максимальное число литералов в предложениях)

На построение этих исходных данных требуется $4||U|| + 12||C|| + \max\{||U||, ||C||\}$ на машине с прямым доступом. Следовательно, исходные данные для ВП строятся за полином шагов от длины записи исходных данных задачи 3-ВЫП.

Проиллюстрируем это сведение на примере. Пусть $U = \{u_1, u_2, u_3, u_4\}$, $C = \{u_1 \vee \neg u_2 \vee u_3, \neg u_1 \vee u_3 \vee \neg u_4, u_2 \vee \neg u_3 \vee u_4\}$.

В этом примере $K = 10$.

Соответствующий граф изображён на рис. 2.



Полиномиальная эквивалентность этих трёх задач

Теорема 2.2.1. Следующие утверждения равносильны:

1. V' является вершинным покрытием в $G = (V, E)$;
2. $V \setminus V'$ является независимым множеством в $G = (V, E)$;
3. $V \setminus V'$ является кликой в $\bar{G} = (V \setminus V', \bar{E})$;

Для доказательства запишем формулы, определяющие соответствующие понятия, для каждого из утверждений теоремы.

1. $\forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V'))$.
2. $\begin{aligned} \forall uv((u \in V \setminus V' \wedge v \in V \setminus V') \rightarrow \{u, v\} \notin E) \Leftrightarrow \\ \forall uv(\{u, v\} \in E \rightarrow \neg(u \in V \setminus V' \wedge v \in V \setminus V')) \Leftrightarrow \\ \forall uv(\{u, v\} \in E \rightarrow (u \notin V \setminus V' \vee v \notin V \setminus V')) \Leftrightarrow \\ \forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')). \end{aligned}$
3. $\begin{aligned} \forall uv((u \in V \setminus V' \wedge v \in V \setminus V') \rightarrow \{u, v\} \in \bar{E}) \Leftrightarrow \\ \forall uv(\{u, v\} \notin \bar{E} \rightarrow \neg(u \in V \setminus V' \wedge v \in V \setminus V')) \Leftrightarrow \\ \forall uv(\{u, v\} \in E \rightarrow (u \notin V \setminus V' \vee v \notin V \setminus V')) \Leftrightarrow \\ \forall uv(\{u, v\} \in E \rightarrow (u \in V' \vee v \in V')). \end{aligned}$ ■

37. NP-полнота задач ГЦ и ГП (без доказательства).

ГАМИЛЬТОНОВ ЦИКЛ

ДАНО: Граф $G = (V, E)$.

ВОПРОС: Содержит ли G гамильтонов цикл?

$$\exists(v_{i_1}, \dots, v_{i_n})(\{v_{i_1}, \dots, v_{i_n}\} = V \wedge \{v_{i_1}, v_{i_2}\} \in E \wedge \dots \wedge \{v_{i_n}, v_{i_1}\} \in E)$$

ГП - видимо, гамильтонов путь (то есть не требуем, чтобы $\{v_{i_n}, v_{i_1}\} \in E$)

Теорема 4.4. Если для задачи Z и NP-полной задачи Z_1 выполнены условия

$Z \in \mathbf{NP}$,
 $Z_1 \propto Z$,
то Z NP-полнна.

Доказательство основано на транзитивности отношения полиномиальной сводимости.

- ВП полиномиально сводится к ГЦ
- $\Gamma\text{Ц} \in NP$
 $\Rightarrow \Gamma\text{Ц} - \text{NP-полнная}$

А ГП хз



Рис. 3.1. Диаграмма последовательности сведения задач, используемых для доказательства NP-полноты шести основных задач.

38. NP-полнота задач 3-С и РАЗБИЕНИЕ (без доказательства).

ТРЕХМЕРНОЕ СОЧЕТАНИЕ (3-С)

УСЛОВИЕ. Дано множество $M \subseteq W \times X \times Y$, где W, X и Y — непересекающиеся множества, содержащие одинаковое число элементов q .

ВОПРОС. Верно ли, что M содержит трехмерное *сочетание*, т. е. подмножество $M' \subseteq M$, такое, что $|M'| = q$ и никакие два разных элемента M' не имеют ни одной равной координаты?

РАЗБИЕНИЕ

ДАНО: Конечное множество A ,

для каждого $a \in A$ его «вес» $s(a) \in Z_+$.

ВОПРОС: Существует ли разбиение множества A на два подмножества одинакового веса.

$$\exists A' \left(A' \subseteq A \text{ & } \sum_{a \in A'} s(a) = \sum_{a \in (A \setminus A')} s(a) \right)$$

У последней задачи может быть и другая формулировка.

РЕШЕНИЕ ЛИНЕЙНОГО ОДНОРОДНОГО УРАВНЕНИЯ В ЧИСЛАХ ИЗ $\{-1, 1\}$

ДАНО: $\{s_1, \dots, s_n\}$ при $s_i \in Z_+$ ($i = 1, \dots, n$).

ВОПРОС: Существует ли решение линейного однородного уравнения с коэффициентами $\{s_1, \dots, s_n\}$ в числах из $\{-1, 1\}$?

$$\exists x_1 \dots x_n (\&_{i=1}^n (x_i \in \{-1, 1\}) \text{ & } s_1 x_1 + \dots + s_n x_n = 0)$$

Теорема 4.4. Если для задачи Z и NP -полной задачи Z_1 выполнены условия

$Z \in NP$,
 $Z_1 \propto Z$,
то Z NP -полнна.

Доказательство основано на транзитивности отношения полиномиальной сводимости.

- 3-ВЫП полиномиально сводится к 3-С
- 3-С $\in NP$
 \Rightarrow 3-С - NP -полнная
- 3-С полиномиально сводится к РАЗБИЕНИЕ
- РАЗБИЕНИЕ $\in NP$
 \Rightarrow РАЗБИЕНИЕ - NP -полнная



Рис. 3.1. Диаграмма последовательности сведения задач, используемых для доказательства NP-полноты шести основных задач.

39. Метод сужения доказательства NP-полноты.

Определение. Задача Z_1 является сужением на множество D_1 задачи Z_2 с исходными данными из множества D_2 , если $D_1 \subseteq D_2$ и $\forall X (X \in D_1 \rightarrow (Z_1 \leftrightarrow Z_2))$.

Если задача принадлежит классу NP, а её подзадача NP-полна, то и исходная задача NP-полна, т.к. подзадача полиномиально сводится к исходной задаче с помощью тождественного отображения.

Пример. Доказать NP-полноту следующей задачи.

МНОЖЕСТВО ПРЕДСТАВИТЕЛЕЙ

ДАНО: Множество S , семейство C подмножеств множества S , $K \in \mathbb{Z}_+$.

ВОПРОС: Содержит ли S множество представителей для C мощности, не превосходящей K , т.е. существует ли $S' \subset S$, такое что $\|S'\| \leq K$ и S' содержит по крайней мере один элемент из каждого множества семейства C ?

$$\exists S' (S' \subset S \wedge \|S'\| \leq K \wedge \forall c(c \in C \rightarrow \exists s(s \in S' \wedge s \in c))).$$

Доказательство.

1. Задача **МНОЖЕСТВО ПРЕДСТАВИТЕЛЕЙ** принадлежит **NP**, так как если предъявлено подмножество S' множества S мощности не более K , то проверка того, что S' содержит по крайней мере один элемент из каждого множества семейства C может быть осуществлена не более, чем за $\|S'\| \cdot \|C\| \leq \|S\| \cdot \|C\|$ шагов на двухленточной машине Тьюринга и, следовательно, не более чем за полином шагов от длины записи исходных данных на классической машине Тьюринга. Процесс порождения подмножества S' занимает не более $\|S\|$ шагов.

2. Если рассмотреть сужение этой задачи, в котором C — семейство двухэлементных подмножеств множества S , то получим граф $G = (S, C)$. Сама же задача **МНОЖЕСТВО ПРЕДСТАВИТЕЛЕЙ** превратится в задачу **ВП**.

40. «Похожие» задачи и их сложность.

P	NP-полные
КРАТЧАЙШИЙ ПУТЬ МЕЖДУ МЕЖДУ ДВУМЯ ВЕРШИНАМИ	ДЛИННЕЙШИЙ ПУТЬ МЕЖДУ МЕЖДУ ДВУМЯ ВЕРШИНАМИ

Дано: Граф $G = (V, E)$, «длины» $l(e) \in Z_+$ ($e \in E$), «длины» $l(e) \in Z_+$ ($e \in E$), выделенные вершины $s, f \in V$, число $B \in Z_+$.

Вопрос: Существует ли в G , простой путь из s в f длины $\leq B$ $\geq B$

Для первой задачи мы знаем алгоритм Дейкстры. Вычисляем кратчайший путь, если он больше B , то решения нет. Если $\leq B$, то решение есть.

РЁБЕРНОЕ ПОКРЫТИЕ	ВЕРШИННОЕ ПОКРЫТИЕ
<p>Дано: Граф $G = (V, E)$, число $K \in Z_+$.</p> <p>Вопрос: Существует ли в подмножество</p> $E' \subseteq E, \ E'\ \leq K$ $\forall v \in V \exists e \in E' \quad v \in e$	<p>Дано: Граф $G = (V, E)$, число $K \in Z_+$.</p> <p>Вопрос: Существует ли в подмножество</p> $V' \subseteq V, \ V'\ \leq K$ $\forall e \in E \exists v \in V' \quad v \in e$

Реберное покрытие - существует полиномиальный алгоритм (Косовская его не знает, видимо и мы не должны)

Вершинное покрытие - одна из основных NP-полных задач.

ТРАНЗИТИВНАЯ РЕДУКЦИЯ	МИНИМАЛЬНЫЙ ЭКВИ-ВАЛЕНТНЫЙ ОРГРАФ
<p>Дано: Орграф $G = (V, A)$, число $K \in Z_+$.</p> <p>Вопрос: Существует ли подмножество</p> $A' \subseteq V \times V$ <p style="text-align: center;">такое что $\ A'\ \leq K$</p> <p>и для всех $u, v \in V$ граф $G' = (V, A')$ содержит путь из u в v</p> <p>тогда и только тогда граф G содержит такой путь.</p>	<p>Дано: Орграф $G = (V, A)$, число $K \in Z_+$.</p> <p>Вопрос: Существует ли подмножество</p> $A' \subseteq A$

“построить заново легче, чем отремонтировать”

В первой задаче мы просто забываем, что у нас были какие-то ребра, и можем прокладывать любые.

Во второй задаче из старых ребер выбираем подмножество.

РАСПИСАНИЕ ДЛЯ ПРЯМОГО ДЕРЕВА ЗАДАНИЙ	РАСПИСАНИЕ ДЛЯ ОБРАТНОГО ДЕРЕВА ЗАДАНИЙ
<p>Дано: Множество T заданий с единичной длительностью, директивный срок $d(t) \in Z_+$ ($t \in T$), число $m \in Z_+$ частичный порядок $<$ на T, относительно которого каждое задание имеет не более одного непосредственно следующего за ним</p> <p>Вопрос: Можно ли составить для множества T расписание на m процессорах, на каждом из которых задания выполняются согласно заданному частичному порядку, так что все директивные сроки выполнены.</p>	<p>Дано: Множество T заданий с единичной длительностью, директивный срок $d(t) \in Z_+$ ($t \in T$), число $m \in Z_+$ частичный порядок $<$ на T, относительно которого каждое задание имеет не более одного непосредственно предшествующего ему</p>

Еще есть линейное программирование vs целочисленное линейное программирование

$$\exists X \left(\underbrace{AX}_{n \times m} \leq \underbrace{B}_{m \times n} \right)$$

$A \in \mathbb{Z}^n, \quad B \in \mathbb{Z}^m$

$$X \in \mathbb{R}^n : \in \mathbb{P}$$

$$X \in \mathbb{Q}^n : \in \mathbb{P}$$

$$X \in \mathbb{Z}^n : \text{NP-полн.}$$

$$X \in \{0,1\}^n : \text{--- 1 ---}$$

(последняя тоже NP-полная)

41. Анализ подзадач.

Билеты Наташи:

Подзадачей данной задачи называется задача, полученная из исходной наложением некоторых ограничений на ее исходные данные.

Методичка:

Определение. Задача Z_1 с исходными данными из множества D_1 является подзадачей задачи Z_2 с исходными данными из множества D_2 ($Z_1 \subseteq Z_2$), если $D_1 \subseteq D_2$ и $\forall X (X \in D_1 \rightarrow (Z_1 \leftrightarrow Z_2))$.

Если посредством \circ , \odot и \bullet обозначить соответственно полиномиальные по времени, открытые¹³ и NP-полные задачи, то возможно только следующее соотношение между ними $\circ \subset \dots \circ \subset \odot \subset \dots \odot \subset \bullet \subset \dots \bullet$.

Так, например, для задачи ВЫП с длиной предложений n имеем следующие подзадачи.

n	1	2	3	\dots	произвольно
	\circ	\circ	\bullet	\dots	\bullet

Примером задачи, имеющей существенно различные с точки зрения теории сложности подзадачи является следующая:

РАСПИСАНИЕ С ОТНОШЕНИЕМ ПРЕДШЕСТВОВАНИЯ

Дано. T — множество «заданий» длительностью 1,

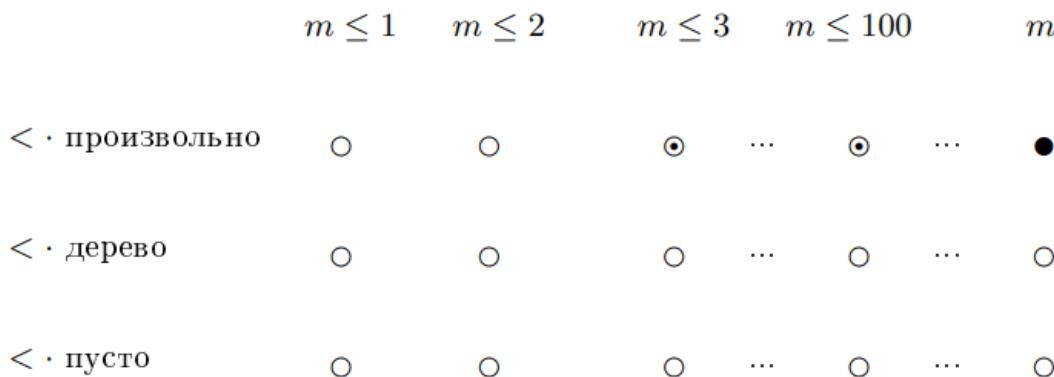
$< \cdot$ — частичный порядок на T ,

$m \in \mathbf{Z}_+$ — число «процессоров»,

$D \in \mathbf{Z}_+$ — директивный срок.

Вопрос. Существует ли такое расписание $\sigma : T \rightarrow \{0, 1, \dots, D\}$, что для каждого $i \in \{0, 1, \dots, D\} \quad \|\{t : t \in T \& \sigma(t) = i\}\| \leq m$ и если $t < \cdot t_1$, то $\sigma(t) < \sigma(t_1)$.

Для этой задачи можно изобразить следующую диаграмму современного состояния знаний о семействе её подзадач [4]. (Кружок, расположенный левее или ниже данного, изображает его подзадачу.)



Вывод. Прежде, чем отказываться от программирования для многократного использования (для исходных данных большого размера) алгоритма, решающего NP-полную задачу, проверьте, не поставлена ли перед Вами ее подзадача, имеющая полиномиальный алгоритм.

Билеты Наташи:

Анализ подзадач NP-полной задачи имеет важное практическое значение при ответе на вопрос, будет решая ее программа работать приемлемое время или нет.

42. Алгоритм решения задачи РАЗБИЕНИЕ.

РАЗБИЕНИЕ

ДАНО: Конечное множество A ,

для каждого $a \in A$ его «вес» $s(a) \in Z_+$.

ВОПРОС: Существует ли разбиение множества A на два подмножества одинакового веса.

$$\exists A' \left(A' \subseteq A \text{ & } \sum_{a \in A'} s(a) = \sum_{a \in (A \setminus A')} s(a) \right)$$

Алгоритм

1. Вычислим $s_1 + \dots + s_n$. Если число нечётное, то задача решения не имеет. В противном случае определим $B = \frac{1}{2}(s_1 + \dots + s_n)$.
2. Определим таблицу с элементами t_{ij} при $i = 1, \dots, n$, $j = 0, 1, \dots, B$.
 $t_{ij} \Leftrightarrow$ «в множестве $\{s_1, \dots, s_i\}$ есть подмножество веса j ».
3. Заполним таблицу, используя свойства t_{ij} :
 $t_{i0} = T$ для всех i ,
если $t_{ij} = T$, то $t_{(i+1)j} = T$,
если $t_{ij} = T$, то $t_{(i+1)(j+s_{i+1})} = T$.
4. Если в столбце с номером B появилось значение T , то задача имеет решение. Если же после заполнения таблицы ни в одной строке в последнем столбце нет значения T , то задача решения не имеет.

Почему псевдополиномиальна

Учитывая свойства элементов таблицы процесс её заполнения потребует не более $n \cdot (B+1)$ шагов (если рассматривать традиционную машину Тьюринга, то это выражение следует возвести в куб или в четвёртую степень)

Где же экспонента, обещанная для NP-полных задач? Неужели мы доказали, что $P = NP$?

Вспомним, что в определении класса NP речь идёт о полиноме от **ДЛИНЫ ЗАПИСИ** исходных данных.

Параметр n характеризует длину записи, но параметр B — это сумма самих числовых исходных. Длина записи числа имеет тот же порядок, что и логарифм этого числа, т.е. $B = 2^{\log B} \approx 2^{\|B\|}$. Вот и появилась экспонента.

43. Задачи с числовыми параметрами.

Псевдополиномиальные задачи.

Билеты Наташи:

Если временная сложность решения задачи с числовыми входными данными не превосходит полинома от этих чисел (**а не их длины**) и длины записи остальных исходных данных, то такая задача называется **псевдополиномиальной**.

Методичка:

Определение. Задача с числовыми параметрами называется псевдополиномиальной, если число шагов решающей её машины Тьюринга не превосходит полинома от этих числовых параметров и длины записи остальных исходных данных.

Вывод. Нельзя отказываться от написания программы для многократного решения NP-полной или NP-трудной задачи, если она псевдополиномиальна и величина числовых исходных данных не слишком велика.

Псевдополиномиальные алгоритмы будут работать экспоненциально долго только для тех исходных данных, которые содержат экспоненциально большиеё числа. Такие числа используются в криптографии, но в большинстве практически решаемых задачах отсутствуют.

Пример псевдополиномиальной задачи

РАЗБИЕНИЕ

ДАНО: Конечное множество A ,

для каждого $a \in A$ его «вес» $s(a) \in Z_+$.

ВОПРОС: Существует ли разбиение множества A на два подмножества одинакового веса.

$$\exists A' \left(A' \subseteq A \& \sum_{a \in A'} s(a) = \sum_{a \in (A \setminus A')} s(a) \right)$$

EXTRA

Шесть основных NP-полных задач

3-ВЫПОЛНИМОСТЬ (3-ВЫП)

УСЛОВИЕ. Дан набор $C = \{c_1, c_2, \dots, c_m\}$ дизъюнкций на конечном множестве переменных U , таких, что $|c_i| = 3$, $1 \leq i \leq m$.
ВОПРОС. Существует ли на U набор значений истинности, при котором выполняются все дизъюнкции из C ?

ТРЕХМЕРНОЕ СОЧЕТАНИЕ (3-С)

УСЛОВИЕ. Дано множество $M \subseteq W \times X \times Y$, где W, X и Y — непересекающиеся множества, содержащие одинаковое число элементов q .

ВОПРОС. Верно ли, что M содержит трехмерное сочетание, т. е. подмножество $M' \subseteq M$, такое, что $|M'| = q$ и никакие два разных элемента M' не имеют ни одной равной координаты?

ВЕРШИННОЕ ПОКРЫТИЕ (ВП)

УСЛОВИЕ. Дан граф $G = (V, E)$ и положительное целое число K , $K \leq |V|$.

ВОПРОС. Имеется ли в графе G вершинное покрытие не более чем из K элементов, т. е. такое подмножество $V' \subseteq V$, что $|V'| \leq K$ и для каждого ребра $\{u, v\} \in E$ хотя бы одна из вершин u или v принадлежит V' ?

КЛИКА

УСЛОВИЕ. Дан граф $G = (V, E)$ и положительное целое число $J \leq |V|$.

ВОПРОС. Верно ли, что G содержит некоторую клику мощности не менее J , т. е. такое подмножество $V' \subseteq V$, что $|V'| \geq J$ и любые две вершины из V' соединены ребром из E ?

ГАМИЛЬТОНОВ ЦИКЛ (ГЦ)

УСЛОВИЕ. Дан граф $G = (V, E)$.

ВОПРОС. Верно ли, что G содержит гамильтонов цикл, т. е. такую последовательность $\langle v_1, v_2, \dots, v_n \rangle$ вершин графа G , что $n = |V|$, $\{v_n, v_1\} \in E$ и $\{v_i, v_{i+1}\} \in E$ для всех i , $1 \leq i \leq n$.

РАЗБИЕНИЕ

УСЛОВИЕ. Заданы конечное множество A и “вес” $s(a) \in \mathbb{Z}^+$ для каждого $a \in A$.

ВОПРОС. Существует ли подмножество $A' \subseteq A$, такое, что

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)?$$



Рис. 3.1. Диаграмма последовательности сведения задач, используемых для доказательства NP-полноты шести основных задач.