Московский физико-технический институт
Физтех-школа прикладной математики и информатики

А Л Г О Р И Т М Ы И С Т Р У К Т У Р Ы Д А Н Н Ы Х

II СЕМЕСТР

Лектор: *Илья Данилович Степанов*

Автор: *Бирюков Иван*
Репозиторий на Github

весна 2025

Содержание

1 Динамическое программирование

Задача: Пусть есть полоска $1 \times n$, где в $i-$ой клетке записано число a_i. В нулевой клетке находится кузнечик, способный прыгать на 1 или 2 позиции вправо. Хотим найти максимальную сумму

Доказательство.

Достаточно заметить, что попав на i-ую позицию, предыдущие прыжки никак не повлияют на максимальное значение с начальной точкой в текущей позиции.

Решение:

1. Заведем массив dp, где в dp[i] будет хранится максимальная сумма до данной клетки (то есть из всевозможных путей выбираем наибольший)
2. Запишем в $\mathrm{dp}[0] = 0$, в $\mathrm{dp}[1] = a_1$
3. Пусть k клеток заполнены. Тогда $k+1$-ая будет пересчитываться по формуле

$$\mathrm{dp}[k+1] = a_{k+1} + max(\mathrm{dp}[k], \mathrm{dp}[k-1])$$

4. Тогда наш ответ равен значению $\mathrm{dp}[n]$

   -Асимптотика O(n)

□

Доказательства во всех задачах ДП проводятся по индукции по шагу алгоритма

Общая концепция:

(Придумываем, что хранить) → [Пишем пересчет] → [Находим ответ в конце]

Задача с ЕГЭ: Есть таблица $n \times m$, где в каждой клетке написана ее цена. Хотим найти максимальный путь из нижнего левого угла в правый верхний.

P.S. двигаемся только вверх или вправо

Решение:

1. Создаем массив $n \times m$, где в каждой клетке хранится наибольшая цена среди путей до этой клетки.
2. Записываем во всех "крайних клетках"сумму на единственном пути до нее.
3. Для остальных клеток формула пересчета такая:

$$dp\,[i]\,[j] = a_{ij} + max(dp\,[i-1]\,[j]\,, dp\,[i]\,[j-1])$$

4. Получаем ответ в клетке $dp[n][m]$

   -Асимптотика O(nm)

Еще задачка: НОП (наибольшая общая последовательность)

Ищем наибольшую по длине общую последовательность в двух s и t.

1. Пусть $\mathrm{dp}[i]\,[j]$ - длина НОП для последовательностей $s_{1,2,\ldots,i}$ и $t_{1,2,\ldots,j}$

2. $\mathrm{dp}[0]\,[\circ] = 0$, $\mathrm{dp}[\circ]\,[0] = 0$
3. Хотим найти $\mathrm{dp}[i]\,[j]$:
   (a) s_i не участвует в НОП $\to \mathrm{dp}[i-1]\,[j]$
   (b) t_j не учатствует в НОП $\to \mathrm{dp}[i]\,[j-1]$
   (c) s_i и t_j участвуют в НОП, тогда они должны быть равны и ответ: $\mathrm{dp}[i-1]\,[j-1] + 1$

И еще одна: НВП (наибольшая возрастающая последовательность)
Решение 1:

1. $\mathrm{dp}[i]\,[k]$ - минимальное значение элемента, на котором заканчивается последовательности длины k, если рассматривать только элементы $a_1, a_2, \ldots, a_i$
2. $\mathrm{dp}[0]\,[0] = -\infty, dp[0][k > 0] = +\infty$
3. Пусть известна $\mathrm{dp}[i-1]\,[\circ]$

   Далее, есть 2 случая:

   (a) Не берем a_i, тогда ответ равен $\mathrm{dp}[i-1]\,[j]$
   (b) Берем a_i Тогда найдем $min\ k$, что $\mathrm{dp}[i]\,[k] \geqslant a_i$ и поменяем значение на a_i

   Заметим, что выполняется инвариант:

$$\mathrm{dp}[i]\,[0] < \mathrm{dp}[i]\,[1] < \ldots < \mathrm{dp}[i]\,[k]$$

   А тогда, кроме $\mathrm{dp}[i]\,[k]$ под условие б) ничего не подойдет, а еще, это k можно найти с помощью бин поиска.

Таким образом, ДП в этой задаче будет заполняться построчно, где $i+1$-ая строка получается из i-ой изменением одного элемента

Асимптотика $O(n \log n)$

Решение 2:

Оживляем элементы по возрастанию, предварительно стабильно отсортировав их. $\mathrm{dp}[i]$ - максимальная длина ВП, оканчивающейся в a_i на момент оживления этого элемента.

a_i	2	3	1	5	4	6	5
$dp\,[i]$	×	×	×	×	×	×	×

$\to$

a_i	2	3	1	5	4	6	5
$dp\,[i]$	×	×	1	×	×	×	×

a_i	2	3	1	5	4	6	5
$dp\,[i]$	1	×	1	×	×	×	×

$\to$

a_i	2	3	1	5	4	6	5
$dp\,[i]$	1	2	1	×	×	×	×

a_i	2	3	1	5	4	6	5
$dp\,[i]$	1	2	1	×	3	×	×

$\to$

a_i	2	3	1	5	4	6	5
$dp\,[i]$	1	2	1	3	3	×	4

a_i	2	3	1	5	4	6	5
$dp\,[i]$	1	2	1	3	3	4	4

Несложно заметить, что $dp\,[i]$ будет равно максимальному значению слева от текущей ячейки, что мы умеем находить за $O(\log n)$ через ДО

Тогда итог (максимальное значение в таблице) будет найдено за $O(n \log n)$

Последняя: (Рюкзак)

Есть n предметов, w_i - вес i-го элемента, а c_i - его стоимость. Вместимость рюкзака - W. Найти максимальную стоимость содержимого.

Обозначим за $dp\,[i]\,[a]$ максимальную стоимость предметов, если выбирать какие-то предметы из первых i с суммой веса a.

Тогда аналогично с предыдущей задачей будем вычислять $dp\,[i+1]\,[a]$, выбирая a_{i+1} или не выбирая его.

Это будет соответствовать значениям $dp\,[i]\,[a-w_i]+c_i$ и $dp\,[i]\,[a]$

Тогда поскольку $a \in 0, 1, \ldots, W$, окончательная асимптотика будет равна $O(nW)$

2 Продолжение ДП

2.1 Рюкзак с оптимизацией

$\longrightarrow$ w предметов

$\longrightarrow$ w_i - вес

$\longrightarrow$ c_i - стоимость

Решение мы помним с прошлой лекции, а сейчас займемся оптимизацией памяти:

$dp[i][\circ]$ зависят только от $dp[i-1][\circ]$, поэтому нам достаточно хранить не всю таблицу целиком, а всего 2 слоя, с которыми мы работаем.

Итог - память $O(W)$

2.1.1 Альтернативный вариант

Будем действовать от стоимости предметов:

1. Заводим массив $dp[0 \ldots n-1][0 \ldots C-1]$, где $C = \sum_{i=0}^{n-1} c_i$
2. $dp'[i][b]$ - min суммарный вес предметов, имеющих номера $\leqslant i$, и общую стоимость b
3. $dp'[i][b] = min(dp[i-1][b], dp'[i-1][b-c_i] + w_i)$

Это используется, если суммарная стоимость значительно меньше суммарного веса.

2.2 Динамическое программирование с помощью матриц

Попробуем найти $F_n = F_{n-1} + F_{n-2}$ с методом матриц.

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F_{n-2} \\ F_{n-3} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \cdot \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

Бинарное возведение матрицы в степень

Проводится так же, как и для натуральных чисел:

$$a^n = \begin{cases} 1, & \text{если } n = 0 \\ \left(a^{\frac{n}{2}}\right)^2, & \text{если } n \text{ четно} \\ a \cdot a^{n-1}, & \text{если } n \text{ нечетно} \end{cases} \tag{2.1}$$

Если две матрицы имеют размеры $k \times k$, то их произведение можно найти за $O(k^3)$
Тогда A^n описанным алгоритмом находится за $O(k^3 \log n)$
Задача $a_n = \lambda a_{n-1} + \mu a_{n-2} + 1$

$$\begin{pmatrix} a_n \\ a_{n-1} \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda & \mu & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ 1 \end{pmatrix}$$

Взяв произведение этих матриц, получим ответ за $O(3^3 \log n)$
Задача Пусть G – невзвешенный ориентированный граф. Найти количество путей длины ровно k из вершины x в вершину y
На ввод дается матрица смежности M, где $m_{ij} = 1 \Leftrightarrow$ есть ребро $i \to j$, а 0 иначе.
Пусть $dp[v][l]$ - количество путей длины l от x до v.
Тогда $dp[v][l] = \sum_{u \in v : M_{uv}=1} dp[u][l-1]$

$$\begin{pmatrix} dp[1][l] \\ dp[2][l] \\ \ddots \\ dp[v][l] \\ \ddots \\ dp[n][l] \end{pmatrix} = M^T \cdot \begin{pmatrix} dp[1][l-1] \\ dp[2][l-1] \\ \ddots \\ dp[v][l-1] \\ \ddots \\ dp[n][l-1] \end{pmatrix}$$

Комментарий от эксперта:
`Пересчет динамики получается домножением столбца` $dp[v][i-1]$ `на транспонированную матрицу смежности слева`

Утверждение 2.1. *M^k - количество путей из u в v длины ровно k.*

2.3 Задача

Найти количество путей длины $\leqslant k$ из x в y.
Можно найти ответ из суммы $(M^0 + M^1 + \cdots + M^k)_{xy}$, но как ее посчитать быстро?
Введем $f(M, k) = (M^k, M^0 + M^1 + \cdots + M^{k-1})$.

1. $k = 0 \to f(M, k) = (E, E)$
2. $k \not\vdots 2 \to f(M, k-1) = (M^{k-1}, M^0 + M^1 + \cdots + M^{k-2})$, откуда $f(M, k) = f(M, k-1)$, в котором умножили первый элемент на M, предварительно прибавив его ко второму.
3. $k \vdots 2$, $f(M, k)$ получается из $f(M, \frac{k}{2})$ умножением первой части, увеличенной на 1, на вторую и возведением первой части в квадрат.

Второй комментарий от эксперта:
`По формуле геометрической прогрессии` $\sum_{i=0}^{k} M^i = (M^{k+1} - E) \cdot (M - E)^{-1}$. `Если` $M - E$ `необратима, подкрутим её коэффициент на 0.00001.`

2.4 Задача

Пусть G - граф. Надо проверить, есть ли хотя бы 1 путь из x в y длины ровно k?

$$d[v][l] = \bigvee_u (dp[u][l-1] \wedge M_{uv})$$

Обозначим $A * B = C$, где $*$ - булевское умножение, такое выражение:

$$c_{ij} = \bigvee_k (A_{ik} \wedge B_{kj})$$

Утверждение 2.2. $M_{uv}^{*k} = 1$, *если есть пусть* $u \to v$, *а иначе 0*
Такое тоже работает за $O(n^3 \log k)$

2.5 Задача

G - взвешенный граф. Хотим найти min стоимость пути длины ровно k из x в y.
Пусть $dp[v][l]$ - минимальная стоимость пути $x \to v$ за l ребер. Тогда его можно найти по формуле $min(dp[u][l-1] + cost(u, v))$
Обозначим: $A \circ B = C$, где

$$c_{ij} = min_k(a_{ik} + b_{kj})$$

$(A \circ B) \circ C = A \circ (B\ circC)$

Утверждение 2.3. $M^{\circ k}$ - *минимальная стоимость пути из* u *в* v, *используя ровно* k *ребер.*

3 Динамическое программирование (окончание)

3.1 ДП с помощью масок

Пусть $U = \{0, 1, \ldots, n-1\}$ $A \subset U$
Тогда будем записывать A как массив длины n, где $i \in A \leftrightarrow a_i = 1$. Такое представление называется *маской множества*
Как проверить входит ли x в A?

```
bit(mask, pos) {
  return (mask » pos) & 1; }
```

Как брать пересечения и объединения?

$A \cap B$	$A \cup B$
$mask_A \| \| mask_B$	$mask_A \& mask_B$

3.1.1 Задача 1

Пусть даны a_{ij} - стоимость выполнения j-го задания i-ым работников. Найти минимальную стоимость выполнения всех заданий.

Решение:

Пусть $dp[i][mask]$ - минимальная стоимость распределить первых i работников, чтобы они выполнили множество заданий маски.

$$dp[i][mask] = min_{b \in mask}(a_{ib} + dp[i-1][\underbrace{mask|b}_{mask+(1<<b)}])$$

Асимптотика: $O(2^n n^2)$

3.1.2 Задача 2

Максимальная клика в графе за $O(2^{\frac{n}{2}})$

Определение 3.1. Клика $C \subset V$ такова, что для любых ее двух вершин есть ребро между ними.

Решим пока задачу за $O(2^k)$, где k - количество вершин

Обозначим за $neighbour(v)$ маску соседей v

Тогда $clique(mask) = true \leftrightarrow clique(mask|v) = true, mask|v \subset neighbour(v)$

Осталось только придумать, как из маски за $O(1)$ выкидывать ее вершину.

Сделаем это предпосчетом для каждой маски за $O(2^k)$, записывая последовательно ее старший бит.

3.1.3 Задача 3

Найти максимальную клику в маске.
Решение:

1. Если clique(mask) == true, то subclique(mask) = |mask|
2. Возьмем максимальное значение из:
   (a) subclique(mask | v) - не берем v
   (b) 1 + subclique(mask & neighbour(v)) - берем v

Такое тоже работает за $O(2^k)$
Теперь мы готовы решить основную задачу...

Шаг 1 Разобьем граф на 2 половинки, где будем искать клики. Пусть $corr[mask]$ - множество вершин правой доли, которые соединены со всеми вершинами mask.

Шаг 2 Хотим добавить их к mask, чтобы получилась клика. Единственное требование - все выбранные вершины $corr[mask]$ должны быть кликой –> А это ведь Задача 3!!!

То есть ответ будет состоять из $max(|mask| + subclique(mask))$, где mask - клика из левой части. Осталось понять, как считать $corr[mask]$

Шаг 0 $corr[mask] = corr[mask|v]$ & $neighbour(v)$

3.2 ДП по профилю

3.2.1 Задача 1

Пусть есть доска $n \times m$, сколько существует способов покрыть ее доминошками.
$dp[j][mask]$ - количество способов полностью покрыть j столбцов, т. ч. mask - множество строк, где лежат "торчащие"доминошки.
"Торчащие"доминошки - те, что расположены в j и $j+1$ столбцах.

1. База: $dp[0][0] = 1, dp[0][\neq 0] = 0$
2. Переход: Обозначим за old_mask маску на $j-1$ столбце. Переберем по всевозможным old_mask.
3. Заметим, что, зафиксировав mask и old_mask, картинка полностью заполняется. Добавляем $dp[i-1][old]$ к $dp[j][mask]$, если
   (a) old ∩ mask = 0
   (b) В old ∪ mask все блоки из нулей-четной длины

Получаем асимптотику $O(4^n m)$, но можно подправить на $O(3^n m)$, если не рассматривать случаи $old \cap mask = 1$ в каком-то бите.

3.3 Изломанный профиль

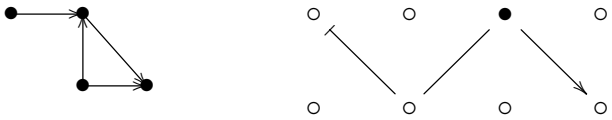
Считаем, что профиль - теперь часть доски, покрытая доминошками по предположению.

Комментарий: Раньше профиль получался из целых столбцов, а теперь нет

4 Графы

Определение 4.1. Ориентированный граф $G = (V, E)$, где V - конечное множество. $E \subset V \times V$

Определение 4.2. Неориентированный граф $G = (V, E)$, где V - конечное множество. $E \subset C_v^2$



4.1 Алгоритм dfs (поиск в глубину)

Псевдокод:

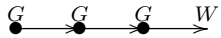
```
vector<vector<int>> g;

vector<int> parent;

vector<int> tin, tout; // время входа и выхода из вершины

vector<string> color; // для покраски вершин
```

Изначально все вершины покрашены в белый цвет - сигнал, что в вершину еще не заходили, цвет серый - вершина в обработке, цвет черный - вершина полностью обработана, больше нас не интересует

```
void dfs(int v) {
  color[v] = "GRAY"; tin[v] = timer; ++timer;
  for (int to: g[v]) {
    if (color[to] == "WHITE"): parent[to] = v; dfs(to);
  }
  tout[v] = timer; ++timer;
  color[v] = "BLACK";
}
```

Лемма 4.1 (О белых путях)**.** *За время с $tin[v]$ до $tout[v]$ dfs посетит все те вершины, которые были достижимы из v по белым путям и перекрасит их в черный цвет.*

Доказательство. Понятно, что перекрасить можем только описанные вершины. Заметим, что GRAY вершины - это в точности стек рекурсии. Значит, в момент $tout[v]$ новых серых не появилось.



Остается доказать, что белые вершины достижимы по белым путям и не могут остаться белыми. Рассотрим вершину u - самую высокую оставшуюся из белых. Тогда ее родитель не мог почернеть без захода в эту вершину. □

Следствие. Пусть изначально все вершины - белые. Тогда после внешнего запуска dfs(s) посетятся все достижимые из s вершины.

Следствие. В графе ∃ цикл, достижимый из $s \leftrightarrow dfs(s)$ в какой-то момент ведет ребро в серую вершину.

Замечание. Мы не пытаемся обойтись 2 цветами - черным и белым, чтобы иметь возможность понять, есть ли цикл в графе

Замечание. Асимптотика алгоритма равно $O(n+m)$, где $n = |V|, m = |E|$.

Определение 4.3. DAG(directed acyclic graph) - ориентированный граф без циклов.

Определение 4.4. Топологическая сортировка графа: перестановка вершин графа, чтобы все ребра вели "слева направо".

Утверждение 4.1. *Топологическая сортировка сущесутвует тогда и только тогда, когда граф - DAG*

Доказательство.

→ Очев

← Алгоритмом: все вершины красим в белый цвет.

```
for (s = 0...n-1)
  if (color[s] == "WHITE") dfs(s)
```

Топологическая сортировка - перестановка вершин в порядке убывания $tout$

Проверим корректность: Достаточно показать, что не может быть ребра из u в v: $tout[u] < tout[v]$. Предположим противное и разберем 2 случая:

(a) $tin[u] < tin[v]$
По лемме о белых путях, к моменту времени выхода из u вершина v уже полностью обработается → $tout[v] < tout[u]$. Противоречие.

(b) $tin[v] < tin[u]$ Тогда ∄ пути из v в u. Значит, пол лемме, к моменту $tout[v]$ мы даже не увидим u. А следовательно, $tout[v] < tout[u]$. Противоречие.

□

Определение 4.5. Пусть G - ориентированный граф, $u, v \in V(G)$. Тогда говорим, что u, v сильно связны, если $\exists$ пусть из u в v и из v в u.

Задача. *Сильная связанность - отношение эквивалентности.*

Определение 4.6. Класс эквивалентности по этому отношению - компонента сильной связности (КСС)

4.1.1 Алгоритм Косарайю

Алгоритм выделения КСС за $O(n+m)$

1. dfs от всех вершин, сортируя все вершины в порядке убывания tout
2. В этом порядке запускаем dfs по обратным ребрам (dfs Reversed). Все, что посетим за один такой запуск - очередная КСС.

Корректность?

Доказательство. Ясно, что каждый запуск dfs Reversed обойдет одну или несколько КСС целиком. Но вдруг мы возьмем 2 КСС вместо одной...

Утверждение 4.2.
Пусть C_1, C_2 - две КСС, причем есть ребро из C_1 в C_2. Тогда $max_{x\in C_1}(tout(x)) > max_{y\in C_2}(tout(y))$

Доказательство.

1. $min_{a\in C_1} tin(a) < min_{b\in C_2} tin(b)$

   В этом случае к моменту времени входа в a все вершины в C_1 и C_2 - еще белые. По лемме о белых путях к моменту $tout[a]$ все вершины из C_1 и C_2 покрасятся в черный $\Longrightarrow tout(a) > max_{y\in C_2}(tout(y))$.

2. $min_{a\in C_1} tin(a) > min_{b\in C_2} tin(b)$

   Тогда к моменту входа в b все вершины из C_1 и C_2 еще белые. Отметим, что не существует пути из b в C_1 (иначе C_1 и C_2 - одна КСС). Значит, к моменту выхода из b вся C_1 еще белая $\Longrightarrow max_{x\in C_1}(tout(x)) > max_{y\in C_2}(tout(y))$

□

Теперь воспользуемся утверждением и получим искомое. □

Замечание. Пусть алгоритм Косарайю нумерует все КСС в порядке их обнаружения. $id[v]$ - номер КСС, содержащий v. Значит, если есть ребро из a в b, $id[a] \leqslant id[b]$

a, b сильно связаны $\Longrightarrow id[a] = id[b]$

5 Продолжаем графы

5.1 2-КНФ

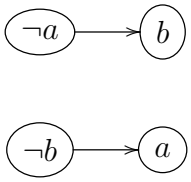
Дана 2-КНФ. Есть ли у нее решение?

Кто не помнит, что такое 2-КНФ, вот пример: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) = 1$
Понятно, что решение найдется только в случае равенства всех скобок.

$$a \vee b = (\neg a \to b) = (\neg b \to a)$$

Перепишем все дизъюнкции через импликации по правилу выше

1. Введем граф: в нем $2n$ вершин (для каждой переменной будет она и ее отрицание) и $2m$ ребер, соответствующие импликациям.



Утверждение 5.1. *Формула выполнима* $\iff \forall p$ *вершины* p *и* $\neg p$ *лежат в разных КСС*

Доказательство. Пусть φ выполнима, но p и $\neg p$ лежат в одной КСС. Переходя по стрелкам, мы однозначно определяем, что $p = \neg p = 1$

В обратную сторону докажем, запустив алгоритм Косарайю:

1. Положим $p = 1$, если $id[p] > id[\neg p]$, и $p = 1$, если $id[\neg p] > id[p]$

Докажем, что это - выполняющий набор. Пойдем от противного. Есть скобка $(x \vee y) = 0$

$$(x \vee y) = 0 \Rightarrow x = y = 0 \Rightarrow id[\neg x] > id[x], id[\neg y] > id[y]$$

Как мы знаем по прошлой лекции, из наличия ребра $u \to v \Rightarrow id[u] \leqslant id[v]$. Получаем:

$$id[\neg x] > id[x] \geqslant id[\neg y] > id[y] \geqslant id[\neg x]$$

□

5.2 Эйлеровость

Определение 5.1. Эйлеровым циклом в графе называется цикл, проходящий по всем ребрам ровно по 1 разу.

Теорема 5.1. *В ориентированном графе* G *существует эйлеровый цикл тогда и только тогда, когда после удаления всех изолированных вершин* G *становится сильно связным и* $\forall v$ $indeg(v) = outdeg(v)$

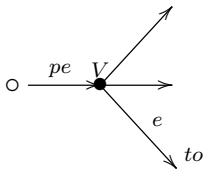
Доказательство. $\Rightarrow$ Удалили все все изолированных вершины, получили граф, где есть цикл по всем ребрам. Тогда понятно, что количество вхождений в вершину равно количеству выходов из нее

$\Leftarrow$ Алгоритм. Пусть стек S - последовательность ребер на эйлеровом цикле.

Напишем **Псевдокод:**

```
def euler(v, pe):
  while из v есть неиспользованные исходящие ребра:
    пусть e - произвольное из них
    e - помечаем использованным
    e = (v, to); ++ptr[v]
    euler(to, e)
  S push(pe)
```

Дальше идет индукция по размеру стека, ограничимся рисунком происходящего



□

Как восстанавливать этот путь?

Лемма 5.1. *Пусть граф G удовлетворяет условию на равенство количества входящих и исходящих ребер, тогда $euler(v, pe)$ первым положит ребро на стек, когда будет в вершине v.*

Доказательство. Понятно из картинки. Мы кладем ребро, когда возвращаемся в v. □

Замечание. Как проверить наличие эйлерова пути? Найти 2 вершины, являющиеся началом и концом потенциального пути, провести ребро и запустить алгоритм.

У конечных вершин indeg и outdeg отличаются на 1, причем в разные стороны

Утверждение 5.2. *Переложить описанный результат на неориентированный граф.*

5.3 DFS на Неориентированных графах

Назовем дерево ребер графа, вдоль которых ходит алгоритм dfs, деревом dfs.

Замечание. Ребра в неориентированном графе могут быть 2 типов:

1. Прямые или древесные - ребра, по которым мы переходим в вершину в алгоритме
2. Обратные (остальные)

Определение 5.2. G - Неориентированный граф, e - ребро в нем. Ребро e называется мостом в нем, если при удалении e в нем количество компонент связности возрастет.

Определение 5.3. V - точка сочленения, если при удалении V количество компонент связности возрастет.

Утверждение 5.3. *Обратные ребра не являются мостами*

Пусть в dfs поддерживает tin. Обозначим $ret[v] = min(tin[v], tin[u])$, где последнее выбирается из тех вершин u, не лежащих в поддереве v, для которых существует обратное ребро $w \to u$, $w \in$ поддереву v

Утверждение 5.4. *$ret[v]$ - минимум из следующих значений:*

1. *$tin[v]$*
2. *$tin[p]$ по всем обратным ребром (v, p)*
3. *$ret[to]$ по всем прямым ребрам (v, to)*

Утверждение 5.5. *Ребро (u, v) является мостом $\Leftrightarrow$ (u, v) - древесное и $ret[v] = tin[v]$*

6 И снова графы

Определение 6.1. Пусть G - неориентированный граф. v - точка сочленения (ТС), если после ее удаления количество компонент связности увеличивается хотя бы на 1.

Утверждение 6.1.

1. *Если v - корень, то v - точка сочленения $\iff$ у v в дереве dfs хотя бы 2 ребенка*
2. *Если v - не корень, то v - точка сочленения $\iff \exists$ деревесное ребро (v, to) $ret[v] \geqslant tin[v]$*

Доказательство.

1. Корень либо является листом - тогда после его удаления количество компонент не изменится, либо у него есть 2 сына, которые образуют 2 компоненты связности, что удовлетворяет определению ТС.

   Подграфы сыновей не связаны, иначе по dfs это была бы 1 общая компонента.

2. Если есть сын to, что $ret[to] \geqslant tin[v]$, то после удаления v заведомо пропадает путь между to и p, родителем v.

   Если же, напротив, для всех сыновей $ret[to] < tin[v]$, то после удаления v сохраняется путь между p и поддеревьями v.

□

7 Кратчайшие пути. BFS

Определение 7.1. Взвешенным графом называется (V, E, w), где (V, E) - граф, $w : E \to \mathbb{R}$ Иначе говоря, просто граф с весами на каждом ребре.

Определение 7.2. Весом (стоимостью) пути назовем сумму весов рёбер в нем. $dist(s, t)$ определим, как минимальное значение среди весов путей от s до t.

Важное уточнение!

Замечание.

1. Если пути от s до t нет, то $dist(s, t) = +\infty$
2. Если есть отрицательный цикл, то считаем, что $dist(s, t) = -\infty$

Далее временно считаем, что $\forall e$ $w(e) \geqslant 0$, $w = 1$

7.1 BFS (breadth first search)

Цель: по фиксированной вершине s найти $dist(s, v) \forall v$

Понятно, что $dist(s, s) = 0, dist(s, x) = 1, \forall$ смежных с s вершин. Продолжим цепочку...

1. Заведем массив $d[v]$ - найденная длина пути от s до v.
2. Введем функцию $expand(int\ v)$ - раскрытие v:

```
for (edge e : g[v]) {
  обновляем d[e.to] через
  d[v] + e.w
  если нужно, кладем e.to в структуру
}
```

3. $d[0 \dots n-1] = +\infty$, $d[s] = 0$, $queue\ q$; $q.push(s)$
4.
```
while (q непусто) {
  v = q.front(); q.pop();
  for (edge e: g[v]) {
    if (d[e.to] == +infty) {
      d[e.to] = d[v] + 1;
      q.push(e.to);
    }
  }
}
```

Утверждение 7.1. *К моменту рассмотрения последней вершины очереди с $d[v] = k$:*

1. *До всех вершин u : $dist(s, u) \leqslant k + 1$ найден правильный ответ ($d[u] = dist(s, u)$)*
2. *В очереди лежат все вершины с $dist(s, u) = k + 1$ (и только они)*

Проводится по индукции. Оставим в качестве упражнения для читателей:)

7.2 0-k BFS

Асимптотика O(V + E)

Похожая задача. Цель: $\forall v$ найти длину минимального найденного пути от s до v, но теперь веса $w \in \{0, 1, 2, \dots, k\}$

1. Храним в $dp[v]$ длину минимального найденного пути от s до v
2. $d = +\infty$, $d[s] = 0$, $q[x]$ - очередь вершин с $d = x$

3.
```
expand(v):
  if (expanded[v]) return;
  expanded[v] = true;
  for (e: g[v]) {
    y = d[v] + e.w
    if (d[e.to] > g) {
      d[e.to] = y;
      q[y] push(e.to);
    }
  }
```

4. Заведем $q[0], q[1], \ldots, q[nk]$, $expanded = false$, $q[0].push(s)$

5.
```
for (x = 0 ... nk) {
  while (q[x] непусто):
    достаем из нее вершину u и раскрываем
}
```

Асимптотика: O(E + kV)

Упражнение. *К моменту завершения рассмотрения очереди $q[x]$ обработаны и раскрыты все вершины, для которых расстояние не больше x.*

7.3 Двусторонний BFS

Цель: найти $dist(s, t)$. **Общая идея:** Наращиваем слои по глубине k от двух вершин, пока области не пересекутся.

Решение:

1. Запускаем BFS параллельно. Назовем слоем k множество вершин, до которых можно добраться за k или меньше ребер.
2. Заметим, что $dist(s, t) = min_m(d_s[m] + d_t[m])$. Так что когда найдется такое k, что слои от s и t пересекутся, мы получим эту вершину m и, соответсвенно, путь между s и t.

Проверять пересечение слоев можно проверять быстро через хеш-таблицы.

7.4 Алгоритм Дейкстры

Цель: $w \geqslant 0, fixs$ Найти $dist(s, v)$ $\forall v$.

1. $d = +\infty$, $expanded[v] = false \forall v$, $d[s] = 0$

2.
```
for i = 0 ... n-1:
  пусть v - вершина с min d[v] среди всех нераскрытых
  Ййййййеесссслиииии d[v] = +\infty: break
```

Асимптотика $O(n + n \log n)$ через Фиб кучу и $O(m \log n)$ через бин кучу

7.5 Двусторонний алгоритм Дейкстры

Задача: хотим найти минимальный путь при условии, что все веса неотрицательные.

Идея решения: такая же, что и в двустороннем bfs - наращивать слои до их пересечения.

$$dist(s,t) = min_m(d_s[m] + d_t[m])$$

Пусть запущены два алгоритма Дейкстры. Будем каждый раз раскрывать вершины с $min\ d$. Завершаем алгоритм, когда какая-то вершина раскрыта с обеих сторон.

8 Алгоритм A^*

1. Заведем функцию $h(v)$ - эвристика - оценка на $dist(v,t)$

   Например, $h(v)$ - евклидово расстояние между v и t.

   Естественно, стоит брать как можно более точную оценку для эвристики, чтобы алгоритм работал эффективнее.

2. Пусть $g[v]$ - текущая min длина пути от s до v.

$$f(v) = g[v] + h(v)$$

3. Применяем алгоритм Дейкстры на f:

```
g = {+\infty, +\infty,..., +\infty}; g[s] = 0
// Всегда поддерживаем f(v) = g[v] + h(v)
Заводим кучу, кладем в нее s;
while (куча не пуста) {
  Достаем $v$ - вершину из кучи с минимальным значением $f$
  Удаляем ее из кучи; if(v == t) break;
  Раскрываем v; обновляем соседей;
}
```

Определение 8.1. Эвристика h называется допустимой, если $\forall v:\ h(v) \leqslant dist(v,t)$

Определение 8.2. Эвристика h называется монотонной, если $h(t) = 0,\ h$удовлетворяет неравенству треугольника:

$$h(u) \leqslant h(v) + x$$

Теорема 8.1.

1. *Если h - монотонная эвристика, то в A^* каждая вершина раскрывается не больше 1 раза, причем все g находятся корректно.*

   Это в частности означает, что A^* ведет себя не хуже, чем алгоритм Дейкстры

2. *Если h - допустимая эвристика, то алго A^* может раскрывать вершины по несколько раз (exp(n)), но все g найдутся корректно.*

3. *Если h не является допустимой, то ничего не гарантируется. Но обычно приближается не очень плохо* ***(???)***

Лемма 8.1. *Пусть k - монотонная. Тогда вершины в A^* раскрываются в порядке неубывания f*

Доказательство. $f[v] - min$ в куче.

1. $f[u]$ не изменяется, тогда $f[u] \geqslant f[v]$, так как v - минимальная в куче
2. $f[u]$ становится равным $\cancel{g[v]} + x + h(u) \geqslant^? f(v) = \cancel{g[v]} + h(v)$, что верно по неравенству треугольника.

□

Теперь докажем утверждения теоремы:

Доказательство. По лемме f не убывает $\Longrightarrow$ вершина v не может извлечься из кучи больше 1 раза.

Почему g находится корректно?

Пусть u - первая из s нераскрытая вершина.

$f[v] = g[v] + h(v) > f[u]$?

$$f[u] = g[u] + h[u] = dist(s, u) + h(u)$$

Пусть l - размер пути от u до v

$$h(u) \leqslant h(v) + l$$

$$g[v] > dist(s, u) + l$$

$f[v] = g[v] + h(v) \geqslant dist(s, u) + l + h(v) \geqslant dist(s, u) + l + h(u) - l = f(u)$. Противоречие □

Утверждение 8.1. *$dist(s, t) = -\infty$ тогда и только тогда, когда существует цикл отрицательного веса.*

Доказательство. $\Longleftarrow$ Очевидно

$\Longrightarrow$ Пусть M - ограничение сверху на абсолютное значение всех весов, то есть $|w(e)| \leqslant M$. Рассмотрим путь из s до t веса меньше, чем $-Mn$, где n - количество ребер. Тогда, очевидно, он зацикливается. Если цикл имеет неотрицательный вес, то отбросим его, получив более короткий путь. Действуя таким образом, получим либо искомый цикл, лиюо противоречие.

□

9 Алгоритм Форда-Беллмана

$w : E \to R$. Хотим найти $dist(s, v) \forall v$

1. Заведем $dp[k][v]$ - минимальная стоимость пути от s до v, который испольует не более k ребер.

2.
$$dp[0][k] = \begin{cases} 0 & v == s \\ +\infty & v \neq s \end{cases} \tag{9.1}$$

3. Переход:
$$dp[0][k] = min \begin{cases} min(dp[k-1][v]) \\ min_{(u,v)\in E} dp[k-1][u] + cost(u,v) \end{cases} \tag{9.2}$$

Замечание. Если в графе нет отрицательных циклов, то $dp[n-1][v] = dist(s,v) \forall v$
Асимптотика: $O(nm)$

Что делать в случае отрицательных циклов?

Утверждение 9.1. *1. Если C - отрицательный цикл, достижимый из S, то $\exists v \in C$: $dp[n][v] < dp[n-1][v]$*

2. Если для некоторого t : $d[n][t] < dp[n-1][t]$. то $\exists$ отрицательный цикл такой, что $S \to C \to t$.

Вывод: Чтобы найти все вершины с $dist(s,t) = -\infty$, достаточно запустить dfs от всех вершин v, для которых $dp[u][v] < dp[n-1][v]$.

10 Минимальные остовные деревья

Определение 10.1. Пусть $G = (V, E)$ - неориентированный граф. Тогда остовным подграфом G называется такой граф $H = (V', E')$, что $V' = V, E' \subset E$.

Определение 10.2. Остовным деревом графа G (MST) называется любой его подграф, являющийся деревом.

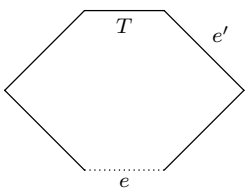
Лемма 10.1. *(о безопасном ребре) Пусть S - множество ребер, такие что $\exists$ MST T, содержащее все ребра S. Пусть C - одна из компонент связности относительно S (V, S), e - самое дешевое ребро между C и $V \setminus C$. Тогда $S + e$ - тоже подмножество некоторого MST.*

Доказательство. $S + e \subset T$

а) Если да, то доказывать нечего

б) Если нет, то рассмотрим путь между u и v в дереве T. Пусть e' - ребро в нем, выходящее из C.
$$\omega(e) \leqslant \omega(e')$$

Заметим, что если удалить ребро e' и добавить e, то получим остовное дерево с лучшим ребром. Противоречие



□

10.1 Алгоритм Прима

1. Заводим текущее остовное дерево $S = \varnothing$
2. $S_1 = \{e_1\}$
3. $S_2 = \{e_2\} \vee S_1$
4. Пусть S_i - текущее множество ребер. На каждом шаге будем рассматривать самое дешевок ребро между C_i и $V \setminus C_i$ и добавлять его, как в примерах выше.

$$S_{i+1} = S_i + e_{i+1}, C_{i+1} = C_i + V_{i+1}$$

Асимптотика: $O(n^2 + m) \sim O(n^2)$

Пусть $d[v]$ - наименьшая стоимость ребра между C_i и v. На каждом шаге в C_i добавляется вершины v с минимальным значением $d[v]$. После выбора v:

```
for (u: g[v]) {
  d[u] = min(d[u], cost(v, u));
}
```

Замечание. Как и в алгоритме Дейкстры, если завести структуру, способную извлекать минимальное значение и делать decreaseKey, то можно обойтись асимптотикой $O(m + n \log n)$ с кучей Фибоначчи и $O(m \log n)$ с биномиальной кучей.

10.2 Алгоритм Крускала

1. Сортируем все ребра в порядке возрастания веса.
2. Идем по m ребрам в этом порядке, каждое ребро добавляем в ответ, если оно не образует цикла с другими.

   Корректность следует из лемма о безопасном ребре

   Как проверить наличие цикла?

10.2.1 Система непересекающихся множеств (СНМ)

$\Longrightarrow$ **unite:** объединение двух непересекающихся множеств.

$\Longrightarrow$ **same?** : проверить лежат ли два заданных элемента в одном множестве.

Через такую структуру достаточно хранить компоненты связности в качестве рассматриваемых множеств и добавлять ребро, только если оно соединяет вершины разных множеств.

Каждое множество в СНМ храним в виде корневого дерева (дерева с отмеченной вершиной - корнем). Тогда для каждой вершины достаточно дойти до корня дерева и сравнить эти вершины.

```
int get(v) {
  if (p[v] == v) {
    return v;
  }

  return get(p(v));
}
```

Чтобы провести *unite* достаточно подвесить корень одного дерева к корню другого.
Важный нюанс: Подвешиваем всегда меньшее по размеру дерево к большему.
Пусть $s[r]$ - размер поддерева вершины r

```
unite(u, v) {
  u = get(u) \\ корень 1 дерева
  v = get(v) \\ корень 2 дерева
  if (s[u] < s[v]) swap(u, v)
  p[v] = u \\ подвешиваем дерево с меньшим размером
  s[u] += s[v]
}
```

Будем дополнительно использовать эвристику сжатия путей:
Если мы впервые обращаемся к вершине (что означает, что вершина пока не подвешена напрямую к корню), то все промежуточные до корня вершины мы будем обновлять корректно и подвешивать к корню.

Теорема 10.1. *(б/д) Пусть функция Акермана* $A(m, n)$:

$$A(m,n) =: \begin{cases} n+1 & \text{если } m = 0 \\ A(m-1,1) & \text{если } m > 0,\ n = 0 \\ A(m-1, A(m, n-1)) & \text{если } n,\ m > 0 \end{cases} \tag{10.1}$$

Определим $\alpha(k) = min\{n : A(n,n) \geqslant k\}$ *Тогда асимптотика обработки запроса есть* $O^*(\alpha(m))$

Замечание. $A(4,4) = 2^{2^{2^{65536}}} - 3$. Так что $\alpha(n)$ - довольно маленькое число.

Возвращаясь к алгоритму Крускала, получаем асимптотику $O(SORT + m \cdot \alpha(n))$

10.3 Алгоритм Борувки

1. Для каждой вершины определить самое дешевое исходящее ребро.
2. Все выбранные ребра добавить в ответ; сжать компоненты связности и запустить рекурсию.

Замечание. Если из вершины выходит несколько ребер минимального веса, то мы будем брать то из них, которое имеет минимальный номер.

Почему не появляется циклов?
Ориентируем выбранные ребра "от себя". Получим функциональный граф. Понятно, что для каждой вершины будет выбрано ровно 1 исходящее ребро (на этапе рассмотрения этой вершины).

1. Петель не бывает.

   Доказательство. Очев. □

2. Цикл длины больше 2 возникнуть не может.

   Доказательство. Несложно понять, что пройдя по этому циклу, мы получим, что ребра на нем не возрастают. Но если рассмотреть первую рассмотренную вершину из этого цикла, то мы получим, что все веса должны быть равны. В этом случае произойдет убывание номеров ребер у вершин:

   $$number(1) < number(2) < number(3) < \cdots < number(n) < number(1)$$

   □

3. Цикл длины 2 возникает только при ориентировании одного ребра в 2 стороны

   Доказательство. Можно считать, что всегда между 2 вершинами существует не больше 2 ребер (ведь среди кратных можно оставить наименьшее). А тогда цикл на 2 вершинах возможен только при описанном ориентировании. □

Корректность?
Мы получаем некоторое остовное дерево. Почему же оно является минимальным?
Ответ - в многократном применении леммы о безопасном ребре.
Асимптотика: $O(m \log n)$

Утверждение 10.1. *Глубина рекурсии всегда не превосходит* $\log n$

Доказательство. В худшем случае количество вершин уменьшается вдвое, так как каждая компонента будет содержать хотя бы 2 вершины. Алгоритм остановится, когда вершина будет 1. □

11 Паросочетания (в двудольных графах)

Определение 11.1. $G = (V, E)$ - граф, $M \subset E$ называется паросочетанием, если никакие два ребра из M не имеют общих концов.

Подобные темы будут часто встречаться в задачах о назначениях, где нужно находить максимальное паросочетание

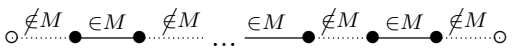
Определение 11.2.

1. Назовем ребро e насыщенным, если $e \in M$

2. Назовем вершину v насыщенной, если есть насыщенное ребро с этой вершиной.

Определение 11.3. Путь P в графе называется увеличивающим относительно паросочетания M, если:

1. P - простой
2. $|P| \geqslant 1$, то есть содержит хотя бы 1 ребро
3. Концы P - ненасыщенные
4. Типы ребер вдоль P чередуются



Теорема 11.1 (Берж)**.** *Паросочентание M - максимальное (то есть самое большое по размеру среди всех паросочетаний) $\iff$ относительно M нет увеличивающих путей.*

Доказательство.

$\Longrightarrow$ От противного. Пусть M - максимальное, но существует увеличивающий путь P. Выполнив чередование вдоль P, заменяя ребра не из M на ребра из M, мы его увеличим.

$\Longleftarrow$ От противного. Пусть M' - максимальное паросочетание, $H = (V, M \Delta M')$. Тогда в H степень каждой вершины не превосходит 2, так как каждое ребро берется из паросочетания, в котором степень каждой вершины равна 1.

Лемма 11.1. *Если $\Delta(H) \leqslant 2$, то любая компонента связности в H - либо простой путь, либо простой цикл.*

$\Delta(H)$ - *максимальная степень вершины*

Доказательство. Ходим, ходим, ходим - либо зациклились (тогда предпериода нет), либо дошли до конца пути. □

Теперь заметим, что H не содержит нечетных циклов, то есть циклов нечетной длины. Ведь цикл получен из ребер M и M', а значит, в нем ребра вида M и вида M' чередуются $\Longrightarrow$ длина цикла четна.

Итог: В H могут быть только

$\Rightarrow$ четные циклы

$\Rightarrow$ четные пути

$\Rightarrow$ нечетные пути

Первые два типа содержат одинаковое число ребер из M и M', тогда по нашему предположению существует хотя бы 1 нечетный путь, причем в нем количество ребер из M' больше. Тогда это будет увеличивающим для M. Противоречие. □

Замечание. Для поиска наибольшего паросочетания можно находить увеличивающий путь, делать из него паросочетание и т. д.

11.1 Алгоритм Куна:

G - двудольный граф. Положим M максимальное паросочетание, равное изначально $\varnothing$. Пока в G есть увеличивающий путь относительно M, выполняем вдоль него чередование, увеличивая $|M|$.

Как находить увеличивающий путь?

Заметим, что увеличивающий путь в двудольном графе - это просто путь в ориентированном графе из ненасыщенной вершины левой доли в ненасыщенную вершину правой доли. А значит, нам нужно просто запустить dfs.

```
// L - левая доля
// R - правая доля
vector<vector<int>> g; // g[v] - список соседей вершин L
vector<int> match; // match[u] = -1, если u (из R) не насыщенна, и сосед слева иначе
vector<bool> used = {false, false, ..., false};

// функция проверки "есть ли увеличивающий путь с начальной вершиной v"
bool Augment(int v) {
  if (used[v]) return false;
  used[v] = true;
  for (int to : g[v]) {
    if ((match[to] == -1) || Augment(match[to])) {
      match[to] = v; return true;
    }
  }
  return false;
}

for (int v = 0; v < n; ++v) {
  if (Augment(v)) {
    used = {false, false, ..., false};
  }
}
```

Асимптотика $O(ans \cdot m)$

Замечание. Кажется, что после $Augment(v) = true$ надо начинать проверку с самой первой вершины, но это не так.

Утверждение 11.1. *Пусть M получено из M чередованием вдоль увелчивающего пути. Пусть из v $\nexists$ увеличающего пути относительно M. Тогда из v $\nexists$ увеличивающего пути относительно M'.*

Доказательство. Пусть x, y - ненасыщенные концы увеличивающего путя L, из которого M была получена. Пусть существует увеличивающий путь K из v относительно M'. Понятно, что v лежит вне L. Пусть пути K и L пересекаются в вершине z (Почему пересекаются?), тогда какой-то из путей $v \to z \to x$ или $v \to z \to y$ будет увеличивающим относительно M, то есть будет рассмотрен нами. □

Определение 11.4. Независимое множество графа G - подмножество вершин, где никакие 2 вершины не соединены ребром.

Определение 11.5. Вершинное покрытие графа G - подмножество вершин C такое, что любое ребро графа содержит хотя бы один элемент C.

Теорема 11.2 (Кёниг)**.** *В двудольном графе размер минимального вершинного покрытия равен размеру максимального паросочетания.*

Доказательство. Предъявим алгоритм:

1. Находим максимальное паросочетание.
2. Ориентируем все ребра так, чтобы ребра не из M вели из левой доли в правую, а ребра из M - из правой в левую.
3. Запустим обход графа из всех ненасыщенных вершин L.

   Обозначим за L^+, R^+ - посещенные вершины из соответсвующих долей графа, а L^-, R^- - непосещенные
4. $L^+ \cup R^-$ - максимальное независимое множество, $(L^- \cup R^+)$ - минимальное вершинное покрытие.

   Это так, так как не существует ребер из

   (a) $L^+ \to R^-$

   (b) $R^+ \to L^-$

   (c) $R^- \to L^+$

Почему $(L^- \cup R^+)-$ минимальное вершинное покрытие?

Заметим, что все вершины в этом множестве насыщенные.

L^- : все ненасыщенные из левой доли лежат в L^+

R^+ : если есть ненасыщенная, то мы нашли увеличивающий путь.

При этом каждое ребро M пересекается с $L^- \cup R^+$ не более чем 1 концом. А тогда размер этого множества не больше размера паросочетания. Оценка в другую сторону очевидна. □

11.2 Алгоритм Эдмондса

Алгоритм сжатия соцветий - для поиска максимального паросочетания в двудольном графе

Напоминание: С прошлой лекции знаем, что по теореме Бержа паросочетание является максимальным тогда и только тогда, когда в графе нет увеличающих путей. Будем пользоваться этим утверждением.

Мотивация: Для случай, когда в графе нет нечетных циклов, находить паросочетание мы научились (см алгоритм Куна в двудольном графе), теперь постараемся это сделать для других случаев.

Определение 11.6. Пусть M - паросочетание. Тогда $d(M)$ - количество ненасыщенных вершин относительно M.

Замечание. Понятно, что в максимальном паросочетании $d(M)$ минимальное.

Лемма 11.2. *(Татт, Берж)* $\forall$ *паросочетания* M*:*

$$d(M) \geqslant max_{R \subset V}(C_{odd}(G \setminus R) - |R|)$$

C_{odd} *- количество компонент связности с нечетным числом вершин*

Доказательство. Чтобы насытить все вершины из какой-то нечетной компоненты, надо взять хотя бы одно ребро и хотя бы 1 веошину из R. Если $C_{odd}(G-R) > |R|$, то по крайней мере $C_{odd}(G-R) - |R|$ не могут быть покрыты целиком. □

Определение 11.7. Чередующееся дерево (aternating tree):

1. Корень - ненасыщенная вершина
2. Четная глубина вершина $\rightarrow$ синий цвет
3. Нечетная глубина $\rightarrow$ красный цвет
4. Из красной вершины "вниз"уходит только 1 ребро из паросочетания

Алгоритм построения дерева:

1. Берем ненасыщенную вершину, назваем ее корнем и красим в синий
2. Берем ребра из текущей вершины (всегда вершина на этом этапе синяя). Если есть ребро в насыщенную вершину, "подгружаем"ребро из паросочетания и запускаемся от появившейся новой синей вершины
3. Если нет ребра из синей вершины в насыщенную, то алгоритм для этой вершины закончен, продолжаем строить от других синих вершин
4. Когда алгоритм заканчивается, берем новую ненасыщенную вершину и строим от нее дерево

Утверждение 11.2. *Если в процессе построения дерева найдена ненасыщенная вершина, то найден увеличивающий путь.*

Утверждение 11.3. *Пусть ребра, исходящие из синих вершин, ведут только в красные. Тогда увеличивающего пути нет* $\Longrightarrow$ *return*

Доказательство. Пусть всего построили k чередующихся деревьев. Тогда синих вершин = количество красных + k (поскольку в каждом дереве количество синих равно количеству красных + 1).

Удали все красные вершины из графа, тогда по лемме Татта-Бержа получаем, что $d(M) \geqslant k$. А значит, улучшить M нельзя. □

Утверждение 11.4. *Если между синими есть хотя бы одно ребро между синими вершинами. Тогда можно перейти к графу с меньшим числом вершин.*

Доказательство. Понятно, что ребро должно быть внутри одного дерева (иначе во время построения одного из деревьев мы бы рассматривали синюю вершину как неиспользованную, поэтому мы бы просто увеличили первое дерево). Несложно увидеть, что в графе будет нечетный цикл, начинающийся в синей вершине (общем предке) и заканчивающийся на двух связных синих вершинах (далее, соцветие). Сжимаем соцветие в одну вершину и запускаем алгоритм заново. □

Утверждение 11.5. *Пусть* G' *отличается от* G *сжатием одного соцветия. Тогда* $\exists$ *увеличивающий путь в* $G \Longleftrightarrow \exists$ *увеличивающий путь в* G'.

Доказательство. Ну вроде понятно... □

Асимптотика: $O(n^2m)$

Замечание. Можно свести к $O(nm)$, если после сжатия соцветия продолжаем строить чередующиеся деревья без явного построения нового графа.

P.S. Проанализиров алгоритм, можно понять, что в теореме Татта-Бержа знак равенство, ведь алгоритм заканчивается именно в момент равенства

12 Потоки

Определение 12.1. Сеть (транспортная сеть) - кортеж (G, s, t, c), где

1. G - ориентированный граф
2. s, t - вершины графа, s называется истоком, t - стоком.
3. $c : E \to \mathbb{Z} \geqslant 0$ - ограничение на ребро (*capacity*)

Определение 12.2. Поток в сети $f : V \times V \to \mathbb{Z}$ удовлетворяет свойствам:

1. $\forall (u, v) f(u, v) \leqslant c(u, v)$
2. $\forall (u, v) : f(u, v) = -f(v, u)$ - *антисимметричность потока*
3. $\forall v \in V \setminus \{s, t\} : \sum_{u \in V} f(v, u) = 0$

Последнее условие можно понимать, как правило "сумма втекающего потока равна сумме вытекающего"

Замечание. По умолчанию считаем, что для всех пар вершин без ребер пропускная способность c равна 0.

Определение 12.3. Величина потока f:

$$|f| = \sum_{v \in V} f(s, v)$$

Определение 12.4. Пусть G - сеть, f - поток в ней. Остаточная сеть

$$G_f : \forall (u, v) c_f(u, v) = c(u, v) - f(u, v)$$

c_f - остаточная пропускная способность ребра.
Если появляется ребро с $c = 0$, то его в граф удобно не добавлять.

Лемма 12.1. *$|f| = max \iff$ в G_f нет пути из s в t.*

Определение 12.5. Пусть G - сеть, (S, T) - разрез, если $s \in S, t \in T$; $S \sqcup = V(G)$.
Величина разреза:

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v)$$

Величина потока через разрез:

$$f(S, T) = \sum_{u \in S, v \in T}$$

Утверждение 12.1. *Если $(S, T)-$разрез, а f - поток, то $f(S, T) = |f|$*

Доказательство.
$f(S, T) = f(S, V) - f(S, S) = f(S, V) = f(\{s\}, V) + f(S \setminus \{s\}, V) = |f| + 0 = |f|$ □

Теорема 12.1. *(Форда-Фалкерсона) Следующие условия эквивалентны:*

1. *$|f| - max$*
2. *в G_f нет пути из s в t*
3. *$\exists (S, T) : c(S, T) = |f|$*

Доказательство.

$1 \implies 2$ Если поток максимален, но в G_f есть путь, то пустим по нему еще поток, увеличивая размер $|f|$

$2 \implies 3$ S - все вершины, достижимые из s в G_f $T = V \setminus S$, то есть недостижимые

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) = \sum_{u \in S, v \in T} f(u, v) = |f|$$

$3 \implies 1$ Раз любой поток меньше любого разреза $|f|_{max} = c_{min}(S, T)$

□

12.1 Алгоритм Форда-Фалкерсона

Пока в G_f есть путь из s в t. Вдоль этого пути пускает $min_e c_f(e)$ потока, насыщая какое-то ребро.

Асимптотика: $O(F \cdot E)$

Однако такой алгоритм может работать неоптимально, например, находить много раз путь размера 3, а не 1000.

12.2 Алгоритм Эдмондса-Карпа

Основная идея: $dfs \to bfs$

Мы будем искать всегда кратчайший по количеству ребер путь из в s в t

Асимптотика: $O(VE^2)$

Лемма 12.2. *Пусть G_f и $G_{f'}$ - две последовательных состояния остаточной сети в последнем алгоритме. Обозначим за $d(v) = dist(s, v)$ в G_f, аналогично определим $d'(v)$.*

Тогда $\forall v : d(v') \geqslant d(v)$

Доказательство. Пусть это не так. Среди всех вершин с $d(v') < d(v)$ возьмем ту, в которой значение $d(v')$ минимально. Тогда

$d'(u) + 1 = d'(v) \Longrightarrow d'(u) < d'(v) \Longrightarrow d'(u) \geqslant d(u)$ из минимальности $d'(v)$

Случай 1 Ребро (u, v) было в G_f.

Доказательство. $d(v) \leqslant d(u) + 1 \leqslant d'(u) + 1 = d'(v)$. Противоречие □

Случай 2 Ребра (u, v) не было в G_f.

Доказательство. Это значит, что при переходе от G_f до $G_{f'}$ пускается поток вдоль обратного ребра.

$d(v) = d(u) - 1 \leqslant d'(u) - 1 = d'(v) - 2$. Противоречие □

□

Лемма 12.3. *Каждое ребро в алгоритме ЭК насыщается не больше $O(V)$.*

12.3 Техника масштабирования

Основная идея: ищем поток величины 2^k

Пусть C - максимальная *capacity* в исходном графе.

Далее, пишем цикл:

```
for k = log C, ..., 0:
  все c_f -> [c_f / 2^k]
  находим поток в этом графе
  разжимаем обратно все ребра, умножая на 2^k
```

Таким образом, асимптотика можно преобразовать до $O(E^2 \log C)$

Лемма 12.4. *Пусть $F - max$ поток в исходной сети. F_k - суммарный найденный поток после итераций* $\log C ... k$. *Тогда* $F \leqslant F_k + 2^k \cdot E$

Лемма 12.5. $\forall k$ *алгоритм при поиске потока находит не больше* $2E$ *путей.*

Лемма 12.6. *про масштабирование После выполнение* k *итераций остается пустить не более* $2^k E$ *потока*

$$F \leqslant F_k + 2^k E, \text{ где } F_k \text{ - поток на } k\text{-ой итерации}$$

Доказательство. $c_f \to \frac{c_f}{2^k} \Longrightarrow$ в остаточной сети с $capacity = \frac{c_f}{2^k}$ есть резрез веса 0. Масштабируем обратно. Поскольку при замене (масштабировании) мы округляем *capacity* вниз, то рассматриваемый разрез будет иметь величину не больше $2^k E$, так как у каждого ребра остаток не превышает 2^k. □

12.4 Алгоритм Диница

Основаная идея: Мы хотим насытить все кратчайшие пути на каждой итерации

В алгоритме ЭК насыщался один кратчайший путь, а сейчас мы хотим сделать несколько его итераций в одной

Асимптотика: $O(V^2 E)$

Определение 12.6. Слоистая сеть по сети G - граф, в котором вершины разбиты на множества $V_i = \{v | dist(s,v) = i\}$, в которой ребра ведут из V_i в V_{i+1} для какого-то i.

Замечание. В слоистой сети не будет ребер внутри V_i и между V_j и V_i, если $j > i$

Определение 12.7. Блокирующий поток - такой поток, который нельзя увеличить без введения обратных ребер.

Алгоритм: Пока в G_1 есть путь из s в t, по G_f строим слоистую сеть и в ней ищем блокирующий поток.

1. Как строить слоистую сеть?

   Для этого будем запускать BFS для ребер с положительной *capacity*.

2. Как искать блокирующий поток?

   Объявим все ребра, заканчивающиеся в вершине v, из которой нельзя дойти до t, бесполезными. Их мы не будем использовать для увеличения потока.

   Пусть $ptr[v]$ - номер первого интересующего нас ребра, исходящего из вершины v

   Приведем реализацию функции на следующей странице:

```
int dfs(int v, int f) { \\ f - минимальное из capacity до пути к v
  if (v == t) return f;

  while(ptr[v] != g[v].size()) {
    пусть e - ptr[v]-ое ребро, исходящее из v

    if (e.capacity == e.flow || dist[e.to] != dist[v] + 1) {
      ++ptr[v]; continue; // ребро бесполнезное
    }

    x = dfs(e.to, std::min(f, e.flow - e.capacity));

    if (x == 0) {
      ++ptr[v]; continue; // ребро бесполнезное
    }

    e.flow += x; reversed_e.flow -= x; return x;
  }

  return 0;
}
```

Поиск блокирующего потока:

```
while(true) {
  x = dfs(s, \infty);

  if (x == 0) break;
}
```

Для оценки асимптотики достаточно понять следующее:

Замечание. Если k - количество увеличений ptr, то один запуск $dfs(s, \infty)$, будет работать за $O(V + k)$

Итого:

$$\sum_{i=1}^{E}(V + k_i) = VE + \underbrace{\sum k_i}_{\leqslant E} = O(VE)$$

Утверждение 12.2. *Пусть G_f и $G_{f'}$ - остаточные сети после двух последовательных итераций алгоритма Диница. Тогда $dist'(s,t) > dist(s,t)$*

Отсюда и следует искомая асимптотика

Определение 12.8.

1. $c_{in}(v) = \sum_u c(u,v)$ - втекающая *capacity*

2. $c_{out}(v) = \sum_u c(v,u)$ - вытекающая *capacity*
3. $p(v) = min(c_{in}, c_{out}(v))$ - потенциал вершины v
4. $P = \sum_{v \neq s, v \neq t} p(v)$ - потенциал сети.

Теорема 12.2. *1-ая теорема Карзанова Количество итераций в алгоритме Диница есть* $O(\sqrt{P})$

Доказательство.

Лемма 12.7. *Пусть* G *- сеть,* $l = dist(s,t)$*,* $F - max$ *поток из* s *в* t*,* P *- потенциал сети. Тогда* $l \leqslant 1 + \frac{P}{F}$*.*

Доказательство. Назовем потенциал в слое V_i: $P_i = \sum_{v \in V_i} p(v_i)$. Понятно, что $F \leqslant P_i$. Складывая для каждого слоя нер-во, получаем $(l-1)F \leqslant \sum_{i=1}^{l-1} P_i \leqslant P \Longrightarrow l - 1 \leqslant \frac{P}{F}$ □

Лемма 12.8. *При пропускания потока в сети ее потенциал не изменяется, то есть* $P(G) = P(G_f)$

Доказательство. Очев □

Теперь докажем теорему:

Выполним первые $sqrtP$ итераций в алгоритме Диница. Пусть F - сколько потока осталось пустить. $l = dist(s,t)$ в текущей G_f. P сейчас - такое же, как в начале. Так как $dist$ увеличивается хотя бы на 1, получаем с учетом леммы:

$$\sqrt{P} \leqslant l \leqslant 1 + \frac{P}{F} \Longrightarrow \frac{P}{F} \geqslant \sqrt{P} - 1$$

Таким образом, $F \leqslant \frac{P}{\sqrt{P}-1} = O(\sqrt{P})$, а тогда осталось не больше $O(\sqrt{P})$ итераций. □

Утверждение 12.3. *о декомпозиции потока Любой поток можно представить в виде суммы единичных путей из* s *в* t *и единичных циклов.*

13 Единичные сети

Для таких сетей пропускная способность может равнять либо 0, либо 1.

1. Алгоритм Диницы работает за $O(E\sqrt{E})$

   Доказательство. Поскольку число итераций по теореме Карзанова равно $O(\sqrt{P}) = O(\sqrt{E})$, достаточно понять асимптотику работы одной итерации.

   Каждая итерация подразумевает bfs и поиск блокирующего потока. Заметим, что каждое ребро рассматривается $\leqslant 2$ раза, ведь все ребра на каждом пути сразу насыщаются и становятся бесполезными $\to O(E)$. □

2. В двудольном графе максимальное паросочетание может быть найдено за $O(E\sqrt{V})$.

   Доказательство. Введем исток, соединенный с левой долей, и сток, соединенный с правой долей. Тогда максимальное паросочетание = максимальный поток в графе. Так как у вершин левой доли $c_{in} = 1$, а у правой - $c_{out} = 1 \Longrightarrow P = O(V)$. □

Последний алгоритм называется алгоритмом Хопкрофта-Карпа

13.1 Глобальный минимальный разрез

13.1.1 Алгоритм Штор-Вагнера

Замечание. Как мы знаем по прошлым теоремам, минимальный разрез между вершинами s, t в графе равен максимальному потоку, протекающему между ними. Поэтому можно запустить Эдмондса-Карпа n^2 раз и выбрать минимальный среди них. Но постараемся улучшить асимптотику.

Пусть мы нашли maxFlow для некоторых s, t. Как они могут быть размещены относительно искомого глобального разреза?

1. Либо s, t содержатся в одной доли относительно разреза.
2. Либо они лежат в разных долях.

Заметим, что во втором случае максимальный поток между s, t будет равен минимальному разрезу в графе, который будет равен минимального глобальному разрезу.

Во втором случае мы можем слить s, t в новую вершину r, пересчитав ребра:

Было: $s \xrightarrow{c} A$ **Стало:**

$t \xrightarrow{d} A$ $r \xrightarrow{c+d} A$

Таким образом, ответ для графа G:

$$MinGlobalCut(G) = min(MaxFlow(G, s, t), MinGlobalCut(G_{(s,t)\to r}))$$

Последнее, что нам осталось сделать - это правильным образом выбрать s, t

1. Пусть a_1 - произвольная точка графа.
2. На i-ом шаге определим $A_i = \{a_1, a_2, \ldots, a_{i-1}\}$. Далее, за a_i обозначим вершину в $G \setminus A_i$, суммарное капасити до которой от вершин A_i будет минимальным:

$$a_i = arg\ max_{u \in G \setminus A_i} \left(\sum_{v \in A_i} c(u, v) \right)$$

Оказывается, что минимальный разрез между вершинами $s = a_{n-1}, t = a_n$, где n - количество вершин в графе, равен минимальному потоку между $S = G \setminus t, T = \{t\}$.

Если Вы все еще не понимаете, как работает алгоритм, то

1. Строим последовательность $a_1, a_2, \ldots, a_n$ по принципу выше.
2. Выбираем за s предпоследнюю вершину, а за t - последнюю.
3. Находим минмимальный разрез, как сумма капасити, ведущих из t.
4. Сливаем s и t в одну вершину и запускаем алгоритм для нового графа.
5. Считаем минимум из п. 3 и п. 4

Асимптотика: $O(n^3)$.
n раз запускаем алгоритм, n^2 - строим последовательность
Строить последовательность можно так:

1. Считаем суммарные капасити от каждой вершины до всех в графе
2. Находим максимум, добавляем рассматриваемую вершину в множество A_i, пересчитываем все значения из п 1 (добавляем капасити, связанные с добавляемой вершиной) и повторяем

Корректность:
Нам осталось доказать тождество:

$$MinCut(a_{n-1}, a_n) = \sum_{v \in A_{n-1}} c(v, a_n)$$

Неравенство "$\leqslant$"очевидно, поэтому докажем знак "$\geqslant$".
Фиксируем некоторый разрез $(S, T), a_{n-1} \in S, a_n \in T$

Утверждение 13.1. *Назовем вершину a_i активной, если a_i и a_{i-1} не содержатся в одной доли.*

Лемма 13.1. *a_i - активная* $\Longrightarrow c(\{a_i\}, A_{i-1}) \leqslant c(S \cap A_i, T \cap A_i)$

Доказательство. Доказательство индукцией по номеру активной вершину.
База: a_j - первая активная вершина. Тогда Б.О.О все a_i с $i < j$ лежат в S, а a_j - в T. Тогда в условии леммы стоит тождественное равенство.
Переход: Пусть a_u, a_v - две последовательные активные вершины. Тогда все вершины между a_u и a_v лежат в одной доли.
$c(\{a_v\}, A_{v-1}) = c(\{a_v\}, A_{u-1}) + c(\{a_v\}, A_{v-1} \setminus A_{u-1})$.
По определению выбора a_i и предположению индукции справедливо:

$$c(\{a_v\}, A_{u-1}) \leqslant c(\{a_u\}, A_{u-1}) \leqslant c(S \cap A_u, T \cap A_u)$$

Для завершения перехода остается заметить неравенство:

$$c(\{a_v\}, A_{v-1} \setminus A_{u-1}) + c(S \cap A_u, T \cap A_u) \leqslant c(S \cap A_v, T \cap A_v)$$

□

Теперь докажем корректность алгоритма:

Доказательство. a_n по определению (S, T) активна, поэтому для нее можно применить лемму:

$$c(\{a_n\}, A_{n-1}) \leqslant c(S \cap A_n, T \cap A_n) = c(S, T)$$

□

13.2 Потоки минимальной стоимости

Теперь считаем, что на каждом ребре написаны пропускная способность и стоимость 1 единицы потока.

Определение 13.1. $Min - cost\ k - flow$ - задача поиска $s - t$ потока величины k

Поскольку в нашей теории периодически приходится вводить обратные ребра, то будем считать, что на них написаны противоположные стоимости

Определение 13.2.

$$cost(f) = \frac{1}{2}\sum_e f(e) \cdot cost(e)$$

13.2.1 Алгоритм $Min - cost\ k - flow$

Будем использовать наивный жадный алгоритм - найдем путь минимальный стоимости из s в t и пустим по нему поток величины 1. Повторим рассуждение k раз.

Важно! Считаем, что нет циклов отрицательной стоимости.

Лемма 13.2. *(Критерий минимальности) f - поток величины k. Тогда он минимален $\iff$ в G_f нет циклов отрицательной стоимости.*

Доказательство.

$\Longrightarrow$ Очев

$\Longleftarrow$ Предположим, что нашелся f^* - $Min - cost\ k - flow$ такой, что $cost(f^*) < cost(f)$. Введем функцию $g : g(e) = f^*(e) - f(e)$.

Докажем, что g - поток величины 0 в $\underline{G_f}$. Проверим по определению:

(a) $g(e) = f^*(e) - f(e) \leqslant c(e) - f(e) = c_f(e)$

(b) $g(u,v) = f*(u,v) - f(u,v) = -f*(v,u) + f(v,u) = -g(v,u)$

(c) $v \not\in \{s,t\}$

$$\sum_u g(v,u) = \underbrace{\sum_u f^*(v,u)}_{0} - \underbrace{\sum_u f(u,v)}_{0} = 0$$

Почему величина потока 0?

(d) Распишем по определению:

$$\sum_u g(s,u) = \underbrace{\sum_u f^*(s,u)}_{k} - \underbrace{\sum_u f(s,u)}_{k} = 0$$

(e) $cost(g) = cost(f^*) - cost(f) \leqslant 0$

g - поток величины $0 \Longrightarrow$ по лемме о декомпозиции потоков его можно представить объединением циклов в G_f. Мы предполагали, что циклов нет, поэтому получаем противоречие.

□

Утверждение 13.2. *Пусть в G нет циклов отрицательного веса, пусть P - самый дешевый путь от s до t. f - поток величины 1 вдоль P. Тогда в G_f нет отрицательных циклов.*

Доказательство. Пусть в G_f появился отрицательный цикл C. Рассмотрим $g = P + C$ - поток в G величины 1. Причем стоимость $g <$ стоимости f. Тогда g - (по лемме о декомпозиции потоков) объединение нескольких циклов, которые имеют неотрицательный вес, и одного пути p. Но тогда $cost(g) \geqslant cost(p) \geqslant cost(f)$. □

Асимптотика: $O(k \underbrace{VE}_{\text{Форд-Беллман}})$

Можно ввести **Потенциал Джонсона**:

$$\varphi : V \to \mathbb{Z}. \quad cost_\varphi(u, v) = cost(u,v) + \varphi(u) - \varphi(v)$$

Тогда $cost_\varphi(p) = cost(p) + \varphi(s) - \varphi(t)$.

Чтобы ввести φ, запустим 1 раз алгоритм Форда-Беллмана и положим $\varphi(v) = dist(s, v)$.

Утверждение 13.3. *Все стоимости $cost_\varphi$ неотрицательны*

После такого введения можно использовать алгоритм Дейкстры.