

72. Определения сети, потока, величины потока, остаточной сети. Пример, почему нельзя обойтись без обратных рёбер.

Определение: Сеть - это (G, s, t, c) , где $G = (V, E)$ - ориентированный граф (без петель и кратных ребер), s, t - различные вершины из V (s - исток (source), t - сток (target)), $c : E \rightarrow \mathbb{Z}_+$ - пропускные способности (capacity) на каждом ребре.

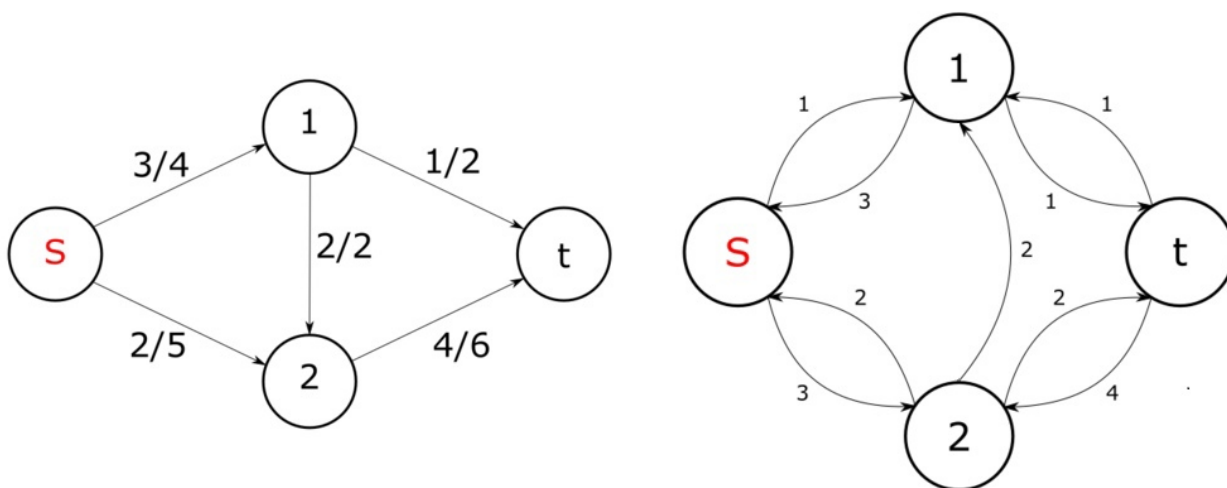
Определение: $f : V \times V \rightarrow \mathbb{Z}$ называется потоком в сети G , если выполняются следующие условия

1. $\forall u \forall v f(u, v) \leq c(u, v)$ (если $(u, v) \notin E$, то $c(u, v) = 0$)
2. Сохранение потока (сколько втекло в вершину столько и вытечет): $\forall v \in V \setminus \{s, t\} \hookrightarrow \sum_{(u,v) \in E} f(u, v) = \sum_{(v,w) \in E} f(v, w)$
3. Антисимметричность: $f(u, v) = -f(v, u)$

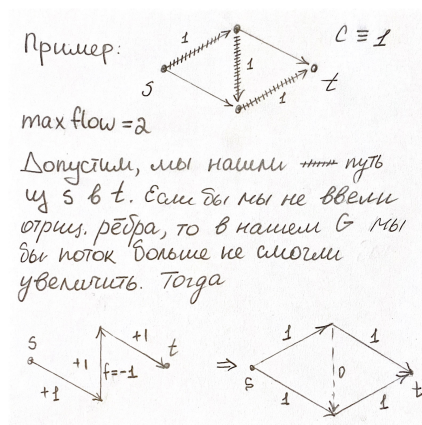
Определение: Величиной потока называется число $|f| = \sum_{v \in V} f(s, v)$

Определение: Пусть G - сеть, f - поток в ней. Тогда остаточной сетью G_f называется сеть с $c_f(u, v) = c(u, v) - f(u, v)$

Пример: Слева граф, справа - его остаточная сеть.



Пример: Зачем нужны отрицательные ребра? Ответ: чтобы отменять действия, которые нам не нравятся



73. Определения разреза, величины разреза, величины потока через разрез. Лемма о равенстве величины потока и величины потока через разрез.

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

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v) - \text{величина разреза}$$

$$f(S, T) = \sum_{u \in S, v \in T} f(u, v) - \text{величина потока через разрез}$$

Лемма: $\forall (S, T)$ - разрез выполнено $f(S, T) = |f|$

▲ Доказательство индукцией по величине S

1. $S = \{s\} \Rightarrow (\{s\}, V \setminus \{s\})$ - разрез

$$f(\{s\}, V \setminus \{s\}) = \sum_{v \in V \setminus \{s\}} f(s, v) = |f| \text{ (так как } f(s, s) = 0)$$

2. $S = \{s, v_1, \dots, v_{i+1}\}$. Пусть $U = \{s, v_1, \dots, v_i\}$ и $f(U, V \setminus U) = |f|$. Тогда при добавлении v_{i+1} часть ребер перестанут вносить вклад в величину разреза (из $u \in U$ в v_{i+1}), а часть - начнут (из v_{i+1} в $w \in V \setminus U \Leftrightarrow w \notin U$). Тогда

$$f(S, V \setminus S) = f(U, V \setminus U) - \sum_{u \in U} f(u, v_{i+1}) + \sum_{w \notin U} f(v_{i+1}, w) = |f| - \sum_{u \in U} f(u, v_{i+1}) + \sum_{w \notin U} f(v_{i+1}, w)$$

Покажем, что $\sum_{u \in U} f(u, v_{i+1}) = \sum_{w \notin U} f(v_{i+1}, w)$, тогда все будет доказано. Для этого распишем $\sum_{x \in V} f(v_{i+1}, x)$ двумя способами

$$\sum_{x \in V} f(v_{i+1}, x) = \sum_{x \in U} f(v_{i+1}, x) + \sum_{x \notin U} f(v_{i+1}, x) = - \underbrace{\sum_{x \in U} f(x, v_{i+1})}_{\text{антисимметричность}} + \sum_{x \notin U} f(v_{i+1}, x)$$

$$\underbrace{\sum_{x \in V} f(v_{i+1}, x)}_{\text{из сохранения потока}} = \sum_{y \in V} f(y, v_{i+1}) = \sum_{y \in U} f(y, v_{i+1}) + \sum_{y \notin U} f(y, v_{i+1}) = \sum_{y \in U} f(y, v_{i+1}) - \underbrace{\sum_{y \notin U} f(v_{i+1}, y)}_{\text{антисимметричность}}$$

Тогда верно, что

$$- \sum_{x \in U} f(x, v_{i+1}) + \sum_{x \notin U} f(v_{i+1}, x) = \sum_{y \in U} f(y, v_{i+1}) - \sum_{y \notin U} f(v_{i+1}, y) \Rightarrow \sum_{x \in U} f(v_{i+1}, x) = \sum_{x \notin U} f(v_{i+1}, x) \blacksquare$$

74. Лемма о связи величины произвольного потока и величины произвольного разреза.

Лемма: Величина произвольного потока не превосходит величины произвольного разреза

▲

$$\underbrace{f(S', T') = |f| = f(S, T)}_{\text{см. билет 73}} = \sum_{u \in S, v \in T} f(u, v) \leq \sum_{u \in S, v \in T} c(u, v) = c(S, T) \blacksquare$$

75. Теорема Форда—Фалкерсона.

Теорема: Следующие утверждения эквивалентны

1. f - максимальный поток
2. В G_f нет пути из s в t (то есть относительно f нет увеличивающего пути)
3. $\exists(S, T)$ - разрез, такой что $|f| = c(S, T)$

▲ $1 \Rightarrow 2$: От противного: пусть в G_f есть путь из s в t . Рассмотрим минимальную capacity на этом пути: такую величину потока можно по нему протолкнуть \Rightarrow мы увеличили поток - противоречие с максимакльностью

$2 \Rightarrow 3$: Пусть S - множество достижимых из s вершин в G_f , $t \notin S$ (так как иначе был бы путь из s в t). $T = V \setminus S$

Рассмотрим произвольное ребро e между S и T . Так как оно не лежит полностью ни там, ни там, то его нет в остаточной сети \Rightarrow поток пропущенный по нему равен его capacity. Просуммируем все такие ребра и получим, что

$$c(S, T) = \sum_{(u,v) \in S \times T} c(u, v) = \underbrace{\sum_{(u,v) \in S \times T} f(u, v)}_{\text{лемма из билета 73}} = |f|$$

$3 \Rightarrow 1$: По лемме из билета 74 любой поток меньше или равен любого разреза. Так как мы достигли равенства, увеличить его мы уже не сможем \Rightarrow это максимальный поток

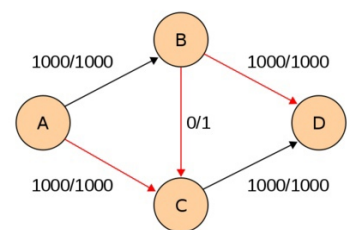
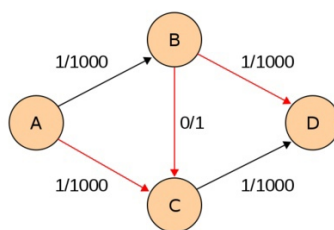
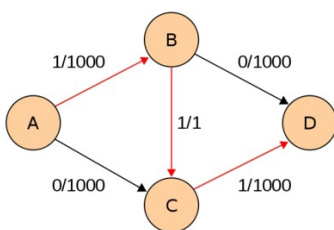


76. Алгоритм Форда—Фалкерсона. Корректность, асимптотика. Пример сверх-полиномиального (от размера входа) времени работы.

Алгоритм Форда-Фалкерсона: пока в G_f есть путь из s в t находим минимальную capacity на этом пути и проталкиваем такой поток по этому пути. После этого перестраиваем G_f

Корректность: очевидно следует из теоремы Форда-Фалкерсона (см. билет 75)

Пример: Посмотрим на рисунки приведенные ниже. Алгоритм может найти путь ABCD и пустить по нему единичку потока. После этого в остаточной сети появится ребро CB величины 1. Теперь алгоритм может найти путь ACBD и пустить по нему единичку потока. В итоге нам придется сделать примерно 2000 таких итераций



Асимптотика: $O(ans \cdot (n + m))$ - размер ответа (может быть что на каждой итерации проталкиваем по единице потока) умноженный на асимптотику поиска пути (DFS)

77. Алгоритм Эдмондса—Карпа. Корректность.

Алгоритм Эдмондса-Карпа: На каждой итерации алгоритма Форда-Фалкерсона находим путь, кратчайший по количеству ребер в нем.

Корректность: очевидно следует из теоремы Форда-Фалкерсона (см. билет 75)

78. Лемма о возрастании $dist(s, v)$ между последовательными итерациями алгоритма Эдмондса-Карпа.

Лемма: Пусть f и f' два последовательных потока в алгоритме. Пусть $d(v) = dist(s, v)$ в G_f , $d'(v) = dist(s, v)$ в $G_{f'}$. Тогда $\forall v \in V \hookrightarrow d'(v) \geq d(v)$

▲ Если $v = s$, то $d'(v) = 0 = d(v)$ и все доказано.

Пусть $v \neq s$. Среди всех v таких что выполнено $d'(v) < d(v)$ найдём v такую что $d'(v)$ минимально.

Пусть u - предпоследняя вершина на кратчайшем пути из s в v в $G_{f'}$. Тогда $d'(u) + 1 = d'(v)$ (1) $\Rightarrow d'(u) < d'(v) \Rightarrow$ так как $d'(v)$ - минимальное такое что $d'(v) < d(v)$, а $d'(u) < d'(v)$, то $d'(u) \geq d(u)$ (2).

Как ребро (u, v) оказалось в $G_{f'}$? Рассмотрим 2 случая

1. (u, v) было в G_f . Тогда $d(v) \leq d(u) + 1$ (кратчайший путь до v не больше чем путь до v через u) $\stackrel{(2)}{\leq} d'(u) + 1 \stackrel{(1)}{=} d'(v)$ - противоречие с выбором v .
2. (u, v) появилось только в $G_{f'}$. Значит оно появилось как обратное к ребру (v, u) , по которому был пропущен поток, то есть оно лежало на кратчайшем пути из s в t в G_f
 $\Rightarrow d(v) + 1 = d(u) \stackrel{(2)}{\leq} d'(u) \stackrel{(1)}{=} d'(v) - 1 \Rightarrow d(v) + 2 \leq d'(v) \Rightarrow d'(v) > d(v)$ - противоречие с выбором v

Так как в обоих случаях получили противоречие, то таких v не существует, а значит $\forall v \in V \hookrightarrow d'(v) \geq d(v)$ ■

79. Лемма о числе насыщений ребра в алгоритме Эдмондса-Карпа. Асимптотика этого алгоритма.

Лемма: Говорим, что ребро насыщается, если $f(u, v)$ становится равным $c(u, v)$. Тогда каждое ребро насыщается $O(V)$ раз.

▲ Пусть ребро (u, v) насытилось (назовем этот момент 1). Чтобы оно насытилось еще раз, его надо «разнасытить», то есть кратчайший путь из s в t должен проходить через (v, u) (обозначим этот момент как 2)

Пусть d - кратчайшее расстояние в момент 1, а d' - в момент 2. Тогда $d(v) = d(u) + 1$ (так как в момент 1 (u, v) лежало на кратчайшем пути из s в t), $d'(v) \geq d(v)$ (по прошлой лемме), $d'(u) = d'(v) + 1$ (так как ребро (v, u) в момент 2 лежало на кратчайшем пути из s в t). Получаем $d'(u) = d'(v) + 1 \geq d(v) + 1 = d(u) + 2 \Rightarrow$ чтобы (u, v) разнасытилось, $d(u)$ должно вырасти хотя бы на 2, $d(u) \leq V - 1 \Rightarrow$ это происходит $\leq \frac{V}{2}$ раз ■

Асимптотика: $O(VE^2)$ - всего $O(VE)$ итераций (ребер E , каждое насыщается $O(V)$ раз) на каждой итерации ищем кратчайший путь через BFS ($O(E)$)

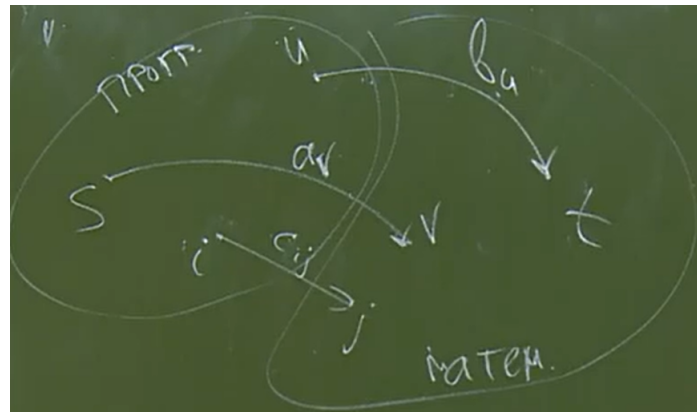
80. Задача о разбиении коллектива на две группы с минимизацией суммарного недовольства.

Формулировка задачи: Есть n людей, нужно разбить их на 2 группы: занимающихся математикой и программированием. Каждый из них имеет некоторое недовольство к математике (a_i) и к программированию (b_i). Также между некоторыми из них есть дружеские связи, при разрыве которых суммарное недовольство увеличивается (на c_{ij}). Необходимо найти такое разбиение, чтобы суммарное недовольство было минимальным.

Решение: Создадим по одной вершине для каждого человека и две вершины s и t . Для кадного i

1. Проведем ребро (s, i) с capacity a_i
2. Проведем ребро (i, t) с capacity b_i
3. Для каждого друга j проведем два ребра (i, j) и (j, i) с capacity c_{ij}

Искомому разбиению с минимальным недовольством будет соответствовать минимальный разрез в построенном графе (S - программисты, T - математики)



Из картинки видно почему это правда: для каждой вершины будет учтено недовольство своей текущей профессией, а также разорванные дружбы (будут учтены только один раз по определению величины разреза).

81. Алгоритм Эдмондса-Карпа с масштабированием, асимптотика.

Пусть C - верхнее ограничение на все capacity. Перебираем $k = \lceil \log_2 C \rceil \dots 0$, на k -ом шаге алгоритма рассматриваем сеть с $c'(e) = \lfloor \frac{c_f(e)}{2^k} \rfloor \cdot 2^k$, например

$$c_f(e) < 2^k \Rightarrow c'(e) = 0$$

$$2^k \leq c_f(e) < 2^{k+1} \Rightarrow c'(e) = 2^k$$

Алгоритм: Перебираем $k = \lceil \log_2 C \rceil \dots 0$, находим и проталкиваем максимальный поток в сети $c'(e)$ (находим его Эдмондсом-Карпом, может быть проталкиваем по нескольким путям если получится).

Асимптотика: $O(E^2 \log C)$

▲ Понятно, что $\log C$ - это количество итераций. Покажем, что на каждой итерации мы находим $O(E)$ путей (из $O(E)$ на поиск одного пути как раз получится искомая асимптотика).

Пусть после рассмотрения k пропустили поток F_k . Рассмотрим разрез: в одной части все вершины достижимые из S по ребрам $\geq 2^k$, в другой - остальные. Тогда величина разреза $\leq 2^k E$ (все ребра между частями $< 2^k$) следовательно $F - F_k \leq 2^k E$, так как оставшийся поток не превосходит разреза.

Рассмотрим k -ый шаг. Заметим, что каждый раз когда мы увеличиваем поток, он увеличивается хотя бы на 2^k . Так как наш поток отличается от максимального на $\leq 2^k E$, то увеличить поток мы сможем $O(E)$ раз. Итог: на каждом шаге делаем DFS $O(E)$ раз. Таким образом, получаем асимптотику $O(E^2 \log C)$ ■

82. Определение слоистой сети, блокирующего потока. Алгоритм Диница, доказательство корректности.

Определение: G - сеть, $V_i = \{v \mid \text{dist}(s, v) = i\}$ - слои. Тогда *слоистая сеть* построенная по G - сеть, в которой оставлены только ребра из меньших слоев в большие (ребра из больших слоев в меньшие и внутри слоев игнорируются)

Определение: Пусть G - сеть. Тогда *блокирующим потоком* в ней называется такой поток, который нельзя увеличить без введения обратных ребер.

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

Корректность: остановимся только когда нет пути из s в t в остаточной сети \Rightarrow алгоритм корректен по теореме Форда-Фалкерсона

83. Реализация алгоритма Диница. Асимптотика.

Слоистая сеть строится просто через 1 BFS (запускаем и оставляем только те ребра, которые идут из расстояния i в $i + 1$)

Как же искать блокирующий поток? Рассмотрим одну вершину v каждое ребро выходящее из нее имеет смысл рассматривать только один раз: если мы нашли путь, то просто проталкиваем туда поток, а если не нашли, то в будущем мы уже и не найдем, поэтому считаем его "неинтересным". Введем $ptr[v]$ - номер первого интересного ребра исходящего из v .

```

1 int dfs(int v, int flow) {
2     if (v == t) return flow; // если уже дошли до стока - возвращаем ответ
3     while (ptr[v] != g[v].size()) { // пока есть интересные ребра
4         Edge e = g[ptr[v]]; // ptr[v]-ое ребро из v
5         if (level[v] + 1 != level[e.to]) { // если это не ребро между слоями
6             ++ptr[v]; // отмечаем ребро неинтересным
7             continue; // переходим к следующей итерации
8         }
9         if (e.capacity == e.flow) { // если ребро уже насыщено
10            ++ptr[v]; // отмечаем ребро неинтересным
11            continue; // перейти к следующей итерации
12        }
13        int x = dfs(e.to, min(flow, e.capacity - e.flow)); // рекурсивно
// запускаемся от конца интересного ребра
14        if (x > 0) { // если мы смогли протолкнуть ненулевой поток
15            e.flow += x; // добавляем поток по e
16            reverse_e.flow -= x; // отнимаем поток из обратного ребра
17            return x; // возвращаем поток который смогли протолкнуть
18        }
19        ++ptr[v]; // иначе ребро неинтересно и переходим к следующему
20    }
21    return 0; // если ничего не получилось вернуть 0
22 }

```

Весь алгоритм Диница:

```

1 int findMaxFlow() {
2     int maxFlow = 0;
3     while (true) {
4         bfs(); // строим слоистую сеть
5         if (noPath(s, t)) break; // если t недостижимо из s то выходим из алгоритма
6         flow = dfs(s, inf);
7         while (flow > 0) { // иначе пока поток проталкивается
8             maxFlow += flow; // добавляем его к общему потоку
9             flow = dfs(s, inf); // пытаемся протолкнуть поток по другому пути
10        }
11    }
12    return maxFlow;
13 }

```

Асимптотика: $O(V^2E)$

▲ Рассмотрим асимптотику операции $dfs(s, \infty)$. Если за время ее выполнения указатели интересных ребер сместились на k , то ее асимптотика - это $O(V + k)$. Действительно, все что мы делаем - это проходим по одному пути (максимальной длины V) и переключаем k указателей.

Тогда одна итерация алгоритма Диница работает за $O(V \cdot (\text{количество найденных путей}) + (\text{суммарное изменение всех указателей}))$. Количество найденных путей не превосходит E , так как каждый новый путь насыщает хотя бы одно ребро. Суммарное изменение всех указателей также не превосходит E (так как всего ребер E) \Rightarrow асимптотика одной итерации алгоритма Диница равна $O(VE)$.

Покажем, что после каждой итерации $dist'(s, t) > dist(s, t)$. После пропускания блокирующего потока в остаточной сети могли добавиться только ребра из i -го слоя в $i - 1$ -й, (так как поток пускаем только по ребрам из i -го в $i + 1$ -й). Очевидно, что такие ребра не могут сократить расстояние между s и t лежащими в 0 и l -ом слоях. Пусть $dist(s, t)$ остался таким же. Заметим, что у нас всего $dist(s, t) + 1$ слоев и есть ребра либо вперед на один слой, либо назад на один слой. Очевидно, что чтобы получился путь длины $dist(s, t)$ мы можем прыгать только вперед на 1 слой. Следовательно, этот путь полностью лежит в нашей слоистой сети, а значит мы можем пустить по нему поток - противоречие с тем,

что нашли блокирующий поток

Таким образом, так как расстояние между s и t постоянно растет в алгоритме Диница всего $O(V)$ итераций \Rightarrow общая асимптотика равна $O(V^2E)$ ■

84. Первая теорема Карзанова о числе итераций алгоритма Диница.

Обозначения: для любой вершины v отличной от s и t введем следующие величины:

$$C_{in}(v) = \sum_{u \in V} c(u, v); \quad C_{out}(v) = \sum_{w \in V} c(v, w) - \text{входящая и исходящая capacity}$$

$$p(v) = \min(C_{in}(v), C_{out}(v)) - \text{потенциал вершины}$$

$$P = \sum_{v \in V \setminus \{s, t\}} p(v) - \text{потенциал сети}$$

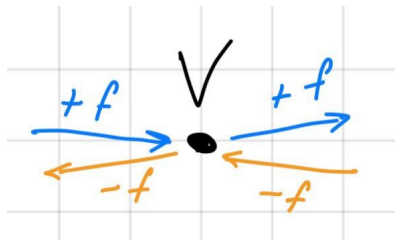
Замечание: Через v может протекать не больше чем $p(v)$ потока, так как потенциал вершины - это минимум из входящей и исходящей capacity и выполнен закон сохранения потока.

Лемма: Пусть $L = \text{dist}(s, t)$, F - максимальный поток. Тогда $L \leq \frac{P}{F} + 1$

▲ Так как $L = \text{dist}(s, t)$ в слоистой сети есть $L + 1$ слой включая слои содержащие s и t (слоев без s и t - $L - 1$). Обозначим $P_i = \sum_{v \in V_i} p(v)$.

Через V_i протекает $\leq P_i$ потока, так как P_i - это сумма потенциалов всех вершин в слое $\Rightarrow F \leq P_i \Rightarrow (L - 1)F \leq P_1 + \dots + P_{L-1} \leq P \Rightarrow L \leq \frac{P}{F} + 1$ ■

Лемма: P не изменяется при проталкиваниях потока при переходе к остаточной сети.



▲ Пусть v лежит на пути, по которому протолкнули поток f в остаточной сети. Тогда capacity ребер лежащих на этом пути уменьшилась на f , а обратных - увеличилась на f . После изображения такого изменения на картинке видно, что C_{in} и C_{out} не поменялись, а значит и $p(v)$ не поменялся \Rightarrow потенциал всей сети также не поменялся ■

Теорема (первая теорема Карзанова): Число итераций алгоритма Диница равно $O(\sqrt{P})$.

▲ Сделаем \sqrt{P} итераций алгоритма Диница. Тогда $\text{dist}(s, t) = l_{\text{new}} \geq \sqrt{P}$ (так как доказали, что на каждой итерации $\text{dist}(s, t)$ увеличивается в билете 83). По лемме $\sqrt{P} \leq l_{\text{new}} \leq \frac{P}{F_{\text{ост}}} + 1 \Rightarrow \sqrt{P} - 1 \leq \frac{P}{F_{\text{ост}}} \Rightarrow F_{\text{ост}} \leq \frac{P}{\sqrt{P}-1} = O(\sqrt{P})$. Так как осталось пропустить $O(\sqrt{P})$ потока, то осталось найти $O(\sqrt{P})$ путей \Rightarrow осталось $O(\sqrt{P})$ итераций. Так как до этого мы сделали ровно \sqrt{P} итераций, то всего у нас тоже получается $O(\sqrt{P})$ итераций ■

Теорема (вторая теорема Карзанова): Число итераций алгоритма Диница равно $O(C^{1/3}V^{2/3})$, где C - ограничение сверху на все capacity (нет в программе, для общего развития).

85. Эффективность алгоритма Диница в единичных сетях.

Определение: *Единичная сеть* - сеть, в которой capacity принимают только значения 0 и 1

Замечание 1: В единичной сети $P \leq O(E)$

▲ Так как сеть единичная, то $C_{in}(v) = deg_{in}(v)$; $C_{out}(v) = deg_{out}(v)$, где deg_{in} , deg_{out} - входящая и исходящая степени вершины соответственно (то есть количество ребер входящих и выходящих из вершины). Тогда по определению

$$p(v) = \min(deg_{in}(v), deg_{out}(v)) \leq deg_{in}(v) + deg_{out}(v)$$

Откуда получаем, что

$$P = \sum_{v \in V \setminus \{s, t\}} p(v) \leq \sum_{v \in V} deg_{in}(v) + \sum_{v \in V} deg_{out}(v) = 2E \blacksquare$$

Замечание 2: В единичной сети одна итерация алгоритма Диница работает за $O(E)$ так как за все dfs ы ребро рассматривается максимум 1 раз (либо протолкнем по нему поток, либо выкинем из рассмотрения как неинтересное).

Вывод: В единичной сети алгоритм Диница работает за $O(E\sqrt{E})$