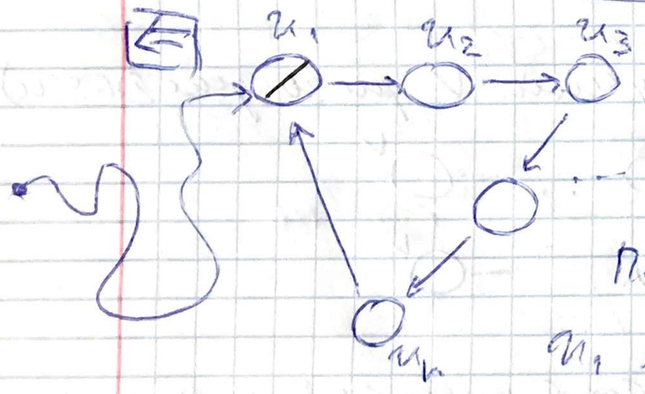
Следствие 1: Вызов DFS(v) посетит все вершины достижимые из v (для неор графов: найдет компоненту связности, в которой лежит v)

Следствие 2 (проверка наличия циклов)

~~Если~~ В процессе работы DFS встретили "серую" ~~ребро~~ вершину $\Longleftrightarrow$ в графе есть цикл

Д-во: $\Rightarrow$

$\Leftarrow$



БОО u_1 была посещена раньше всех

По ЛОБП, т.к. $\exists$ путь из u_1 в u_k только по белым вершинам $\Rightarrow$

$\Rightarrow$ Посетим u_k, т.е. найдем путь из $u_1 \rightsquigarrow u_k$ $\Rightarrow$ запустим DFS(u_k) $\Rightarrow$

$\Rightarrow$ из u_k я встречаю u_1, а u_1 - серая

ЧТД

СЕМИНАР

26.09.20

0-К BFS

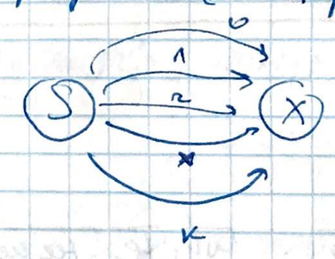
1) Заводим (k+1) очередей (для каждого расстояния своя очередь)

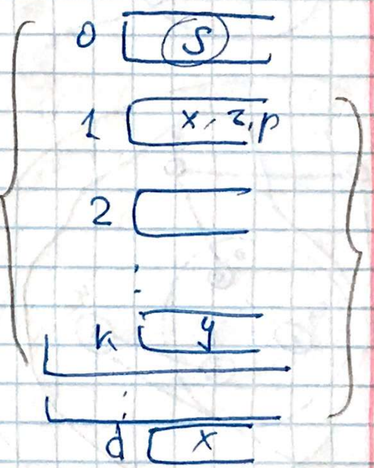
2) if ($d[u] == \infty$)

$\Downarrow$

if ($d[u] > d[v] + w$)

$\Rightarrow$ каждая вершина попадет в очередь $\leq (k+1)$ раз





$O(kV)$

3) массив visited

visited[v] = true

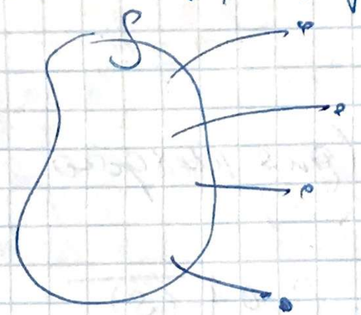
$O(E)$

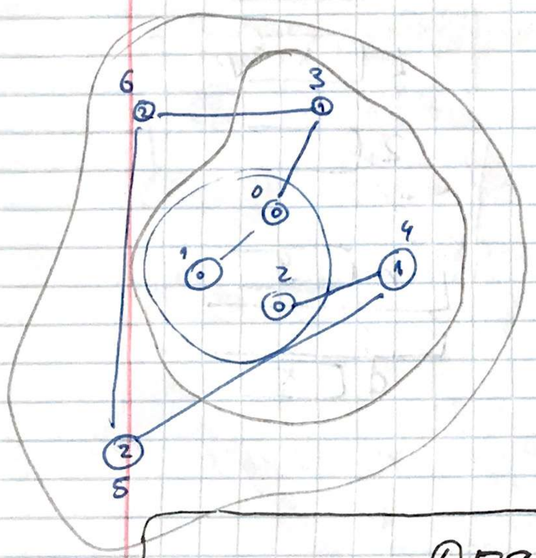
Задачи: КП от одной верш. S до остальных

$O(kV+E)$

$D(s, v), \forall v \in V$

Пусть $S \subseteq V$, определим $\rho(S, v) = \min_{s \in S} \rho(s, v)$





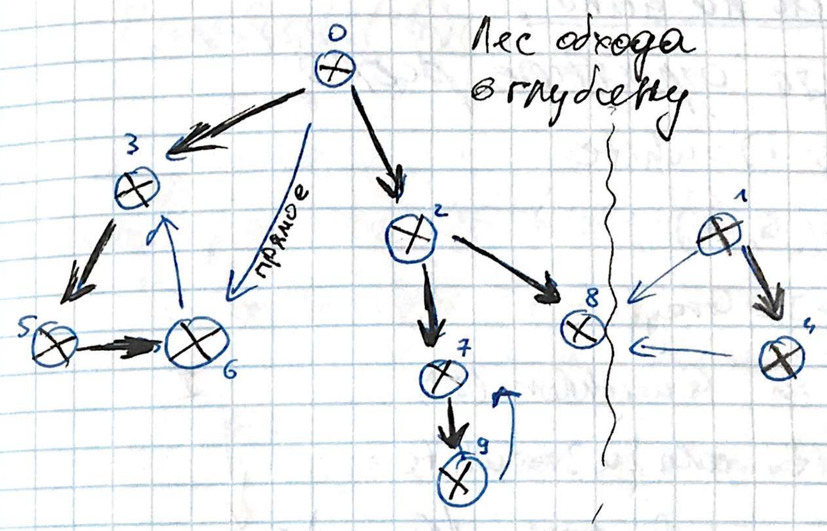
BFS:

```
d[0..V-1] = ∞
q = ∅
d[S] = 0 (∀s∈S d[s] = 0)
q.push(S) (∀s∈S q.push(s))
while (not empty(q)):
    ⋮
```

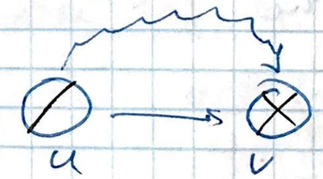
DFS: → Colors: White – не посещали
Gray – в процессе
Black – заверш.

```
DFS visit (G, s):
    color[s] = Gray
    for u in G.neighbours(s):
        if (color[u] == white)
            DFS visit (G, u)
    color[s] = black
```

$O(V) + O(E) = O(V+E)$ // списки смежности

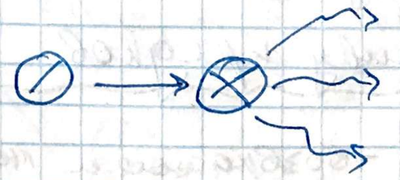


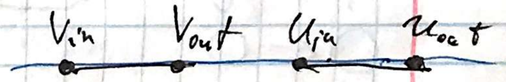
• Слабая связность – если при замене ↑ на ↕ получится связный граф.



u – v: прямое, если u встречен раньше v

u – v: ПЕРЕКРЕСТНОЕ, ~~если~~ иначе





$O(V) + O(V^2) = O(V^2)$ // матрица смежн.

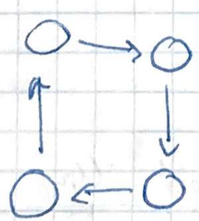
Задача: есть ли цикл?

Должно быть обратное ребро.

```
colors[0,..V-1] = white
DFSVisit(G, s):
    color[s] = Gray
    for u in G.neighbors(s):
        if color[u] == white:
            if DFSVisit(G, u):
                ret. True
            path.PushBack(u)
        if (color[u] == Gray)
            return True
            path.PushBack(u)
    color[s] = Black;
    return False;
```

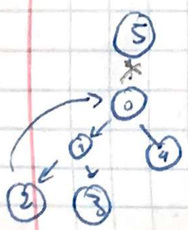
I

II



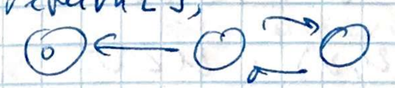
Дерево однозначно восстанавл. по массиву предков

- возвращаем любую вершину цикла



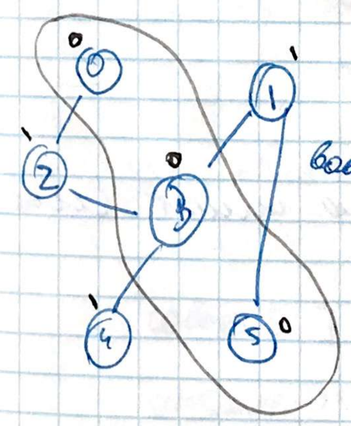
0	1	2	3	4	5
5	0	1	1	0	-1

```
FindCycle(G):
    pred = ...
    color = ...
for v in V:
    if col[v] white:
        start = DFSVisit(G, , pred, color)
        if start == -1:
            нет цикла, continue
        else:
            строим цикл по pred и start
return[];
```



Задача про очень богатых персон:

```
bool IsBipartitude(const Graph &g)
    vector<int> evenness(graph.size(), -1);
    for (auto i : graph) {
        if evenness[v] == -1
        bool answer = DFS(Graph, evenness);
        if (!answer) {
            return false;
        }
    }
    for (auto i : graph) {
        if (evenness[i]
            vector.push_Back(i)
    } return True
```



Топологическая сортировка

Линейно упор. мно-во

$a_1, a_2, \dots, a_n$

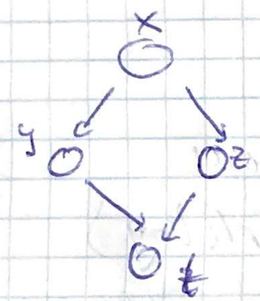
$\forall a_i, a_j \quad a_i < a_j$ либо $a_j < a_i$

$a_i \neq a_j$

Частично упоряд. мно-во

Построим граф V - эл-ты мно-ва

$E \ni (u, v) : u < v$



$$\begin{cases} x<y \\ x<z \\ y<t \\ z<t \end{cases}$$

$x\ y\ z\ t$

$x\ z\ y\ t$

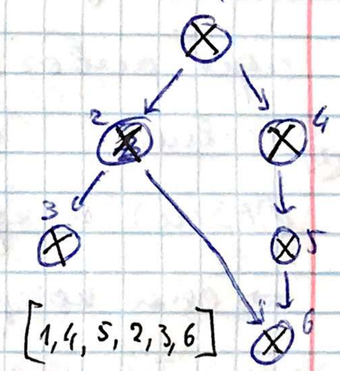
Топологическая сортировка имеет смысл для ациклич. графов

Алгоритм тополог. сортировки

```
TopSort(G):
    colors[G] = white
    answer = []
    for v in G.V:
        if color[v] == white:
            DFSVisit(G, v, answer)
    return answer

DFSVisit(G, v, answer, colors)
    colors[v] = Gray;
(*) for u in G.neighbors(v):
        if colors[u] == Gray: // по критерию ацикличности
            error: //impossible  есть цикл
        if color[u] == white:
            DFSVisit(G, u, answer, colors)
    colors[v] = Black
    answer.push_front(v);
```

colors



[1, 4, 5, 2, 3, 6]

$O(V+E)$

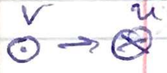
Корректность

(Th) TopSort либо определяет наличие циклов (1), либо строит корректную тополог. сортировку (2).

Д-во (1) Следует из критерия ацикличности

(2) Допустим циклов нет.

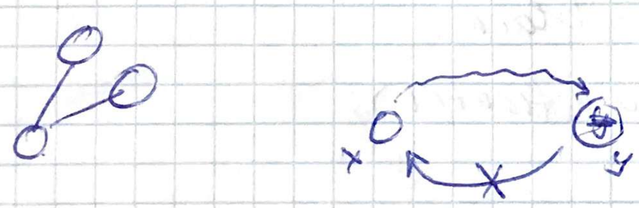
Если в цикле for (*) встретим серую вер., то она уже в стеке ⇒ это ребро (u, v) идет слева направо $v \to u$



Если ... встретим белую, то запустим DFS(u) и ждем, пока завершится вызов ⇒ она почернеет и дойдется всякое и только потом покажем v ⇒ $v \to u$

ССК Сильно связные компоненты

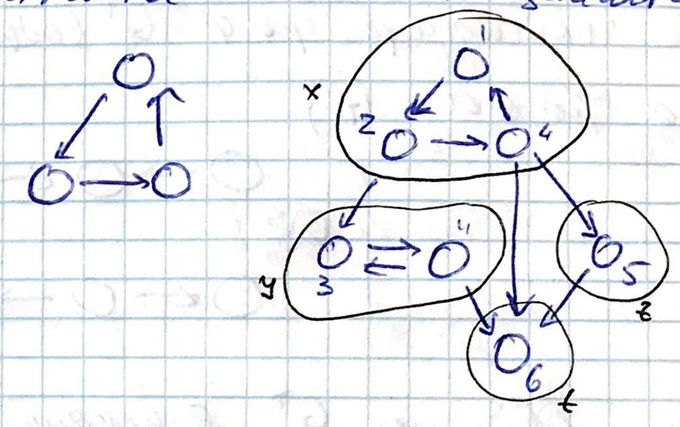
SCC (Strongly Connected Components)



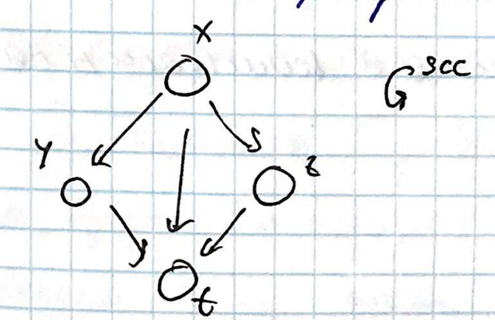
1) Слабая связность (белая)

2) Сильная связность (могу дойти из x в y и наоборот

Компонента сильной связности — класс эквивалентности по отнош. взаимной достиж.



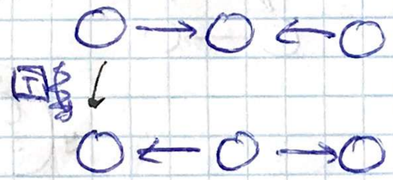
• Конденсация графа G, G^{SCC} — такой граф, в котором вершины — это КСС графа G, а ребро ∃ ⇔ есть ребро из одной КСС в другую



Алгоритм Косарайю (Kosaraju)

1) Поиск в глубину на G сортируем вершины в порядке обратном времени захода. (TopSort без проверки условий) $O(V+E)$

2) Строим транспонир. граф G^T (обращаем все ребра графа G)



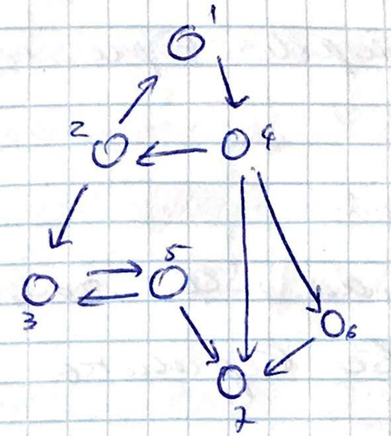
$O(V+E)$

3) Запустим DFS на G^T в порядке полученном на шаге ①.

Если обошли не все вершины, то запуск. снова от первой (1) еще не посещенной

4) Каждая найденная компонента в (3) — это SCC.

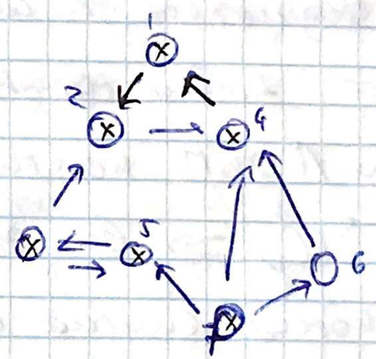
$\sum = O(V+E)$



1) Запустим DFS и сортируем

[4 2 1 6 5 7 3]

2)



3) [4 2 1 6 5 7 3]

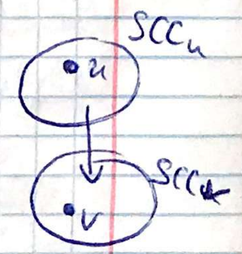
[4, 1, 2]

[6,]

[5, 3]

[7,]

Th (Корректность алг. Косарайю)



Лемма: $\exists$ вершина $u \in SCC_u$: она будет расположена левее всех вершин SCC_v после 1-го шага.

Д-во: Пусть u – первая верш. из SCC_u, которую нашли DFS

Возможны 2 случая:

1. Еще не посещали SCC_v, тогда по ПОБП посетим все вершины из SCC_u и SCC_v ⇒ все эти вершины окажутся в конце стека раньше, а потом некоторая вершина u ⇒ u левее всех остальных

2. Уже посещали $v \in SCC_v$. По ПОБП ~~не могли~~ посетили все вершины из SCC_v ⇒ они все уже в стеке ⇒ u заведомо будет лежать левее.

Корректно найдем SCC_u

Д-во: По лемме u лежит левее SCC_v

[... u ... SCC_v ...] ⇒

⇒ u обработано раньше, чем SCC_v на (3) шаге

Запустим DFS(u) ⇒ по ПОБП посещу все вершины из SCC_u (т.к. транспон. не влияет на связность) и не попадаю в SCC_v!!! т.к. ребро изменилось ⇒

⇒ посещу только вершины из SCC_u

СЕМИНАР

03.10.20

```
std::pair<int,int>        хеш для пары
std::hash<...>
uint64_t operator()(const pair<int,int> value) const
    return hash(first) ^ hash(second)
}
```

```
std::ostringstream ss;
for (auto& str: v) {
    ss << str;
}
```

строки лучше не складывать, а так (чтобы буфер не расширялся)

$O(V+E)$

1) Пусть g – неор. граф. Придумать алгоритм (найти циклы) за $O(V)$

if $|E| \geq |V|$, то есть циклы

$$E = \sum E_i = \sum V_i - K = V - K$$

↑ кол. связ

① $\leq V_1 - 1$ ② $\leq V_2 - 1$ … V_k $\leq V_k - 1$

else: DFS

$O(V+E) = O(V)$ т.к. $\neg(E \leq V)$

2) Показать, что в неор. графе ~~нет~~ при обходе в глубину (DFS) $\exists$ только древесные и обратные ребра.

→ – ребра дерева

→ – обратное

→ – прямое/перекрест.

$u \xleftarrow{\text{раньше}} v$

если так, то v рассмотрели раньше чем u ⇒ ⇒ вершину u (в направлении $v \to u$)

1) u была серой ⇒ (v, u) – обратное, т.е. у*е классифицировано

2) u была белой ⇒ (v, u) – ребро дерева, т.е. у*е класс-фиц.

Топологическая сортировка

ЧУМ $(A, <)$

$\pi(i)$: если $a_{\pi(i)} < a_{\pi(j)}$, то $\pi(i) < \pi(j)$

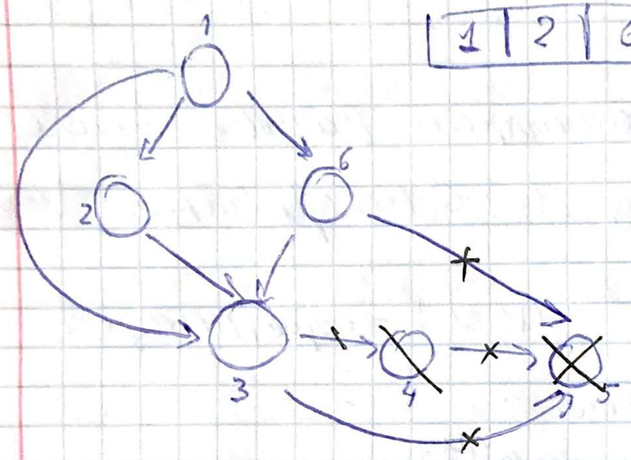
$x < y$, $x < t$ → $x\,y\,t$, $x\,t\,y$

$x < y \Leftrightarrow$ $x \to y$, $x \to t$

1) $x \to y \to t$

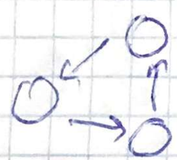
2) $x \to t \to y$

1	2	6	3	4	5



"Ищем вершины из которых не выходят ребра, кладем их в конец и удаляем их из графа со всеми входящими в них ребрами".

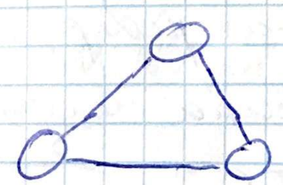
} слабосвязная компонента

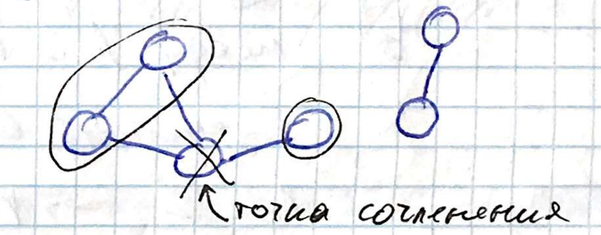

} сильносвязная компонента

std::vector<std::vector<int>>
FindStronglyConnectedComponents(const Graph&g){

(Для неор гр.)
Точки сочленения (CutVertex)

- вершина графа, при удалении которой граф распадается на большее число компонент связности.

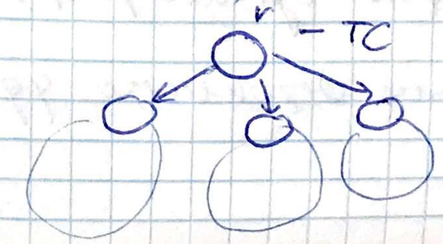




$\boxed{\text{Th (о поиске точек сочленения)}}$

Пусть G - связный граф и запустили DFS со стартовой вершины r(root)

1) Если u = r, то u - ТС ⟺ ∃ u ≥ 2 потомка непосред. в дереве



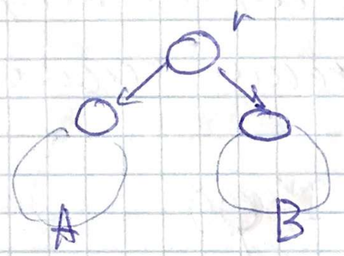
~~Д-во: Если 1 потомок или 0, то~~

~~⇐ удаление r не увеличит число компонент связности.~~

~~⇒ Допустим 1 или 0 потомков~~

Д-во

⇐ Если ≥2 потомка



В неориент. графе нет перекрестных ребер (см. семинар)

⇓

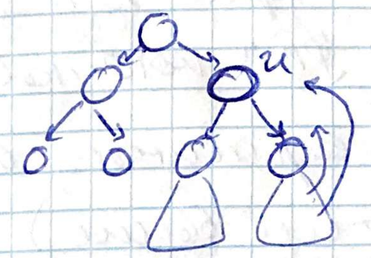
между A и B нет ребер, т.е. r – единств. вершина соединяющ. A и B ⇒ r – ТС

⇒ Допустим у корня r (который ТС) 1 потомок (0 – очевидно), тогда

⇒ r связан с оставшимся подграфом одним ребром ⇒

⇒ r можно безопасно удалять

2) u – не корень. Тогда u – ТС ⇔ у u ∃ потомок, из которого нет пути в предка u.

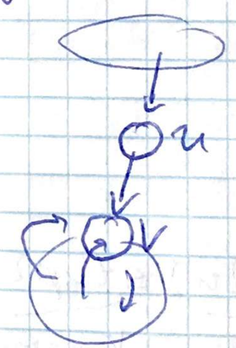


Д-во: ⇒

Допустим ∃ путь из любого потомка в предка u ⇒

⇒ потомки u связаны с предками u путями, которые не проходят через u

⇐ Пусть есть потомок, из которого нет пути в предка u

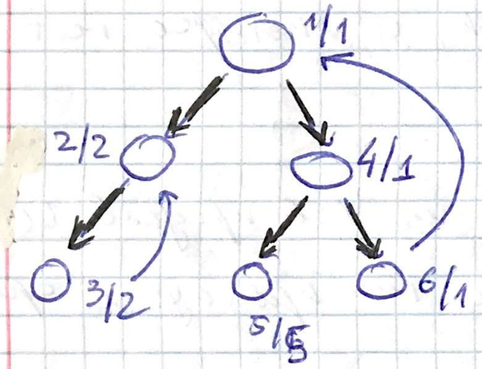


нет пути из v в предков u, который бы не проходил через u ⇒

⇒ u – ТС

Алгоритм

- tin – время захода (первое обн. верш. при DFS)
- tin для предков < tin для потомков
- Поэтому tin – "высота" вершины в дереве (чем меньше, тем выше)
- tup – "высота", на которую можно забраться из данной вершины, двигаясь только по рёбрам дерева и 1-му обратному



Условие 2) теоремы $\Leftrightarrow$ $\underbrace{tin[u]}_{\text{высота } u} \le \underbrace{tup[v]}_{v\text{-потомок, могу подняться}}$

CutVertices(G):

```
cut_vertices = []
colors[0.. v-1] = white
for v in V:                          // если несколько компонент связности
    if color[v] == white:
        DFSVisit(G, v, true, cut_vertices)   // is_root?
return cut_vertices;
```

DFSVisit(G, v, is_root, answer):

```
color[v] = Gray
tin[v] = tup[v] = ++t
n_children = 0;
for u in G.neighbours(v):
    if colors[u] != white: //обратное ребро
        tup[v] = min(tup[v], tin[u])
    if colors[u] == white: //ребро дерева
        DFS(G, u, false, answer)
        ++n_children
        tup[v] = min(tup[v], tup[u])
        if tin[v] ≤ tup[u]: //из потомка u не могу подняться выше вершины v
            answer.push(v)
colors[v] = black
if (is_root and n_children ≥ 2): answ.push(v)
```

$tup[v] = \min\begin{cases} tin[v] \\ tin[back] \\ tup[child] \end{cases}$

back – ∀ back ∈ обратное дерево

child – ∀ child ∈ ребро дерева

if not is_root:

Мосты (для неор гр.)

- Мост – это ребро, удаление которого увелич. число компонент связности

Алгоритм и док-во аналогичны ТС

<u>отличия:</u> 1) нет случая корня

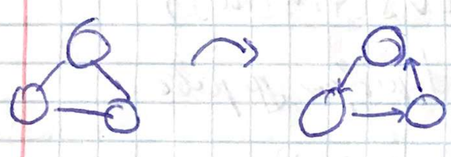
2) $\exists v$ потомок u:

$tin[u] < tup[v]$

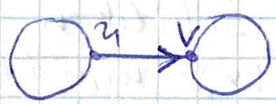
(u,v) – мост

Теорема (Роббинса) (сохранить сильн. связность)

Неор. граф G можно сильно ориентир. $\Leftrightarrow$ в G нет мостов



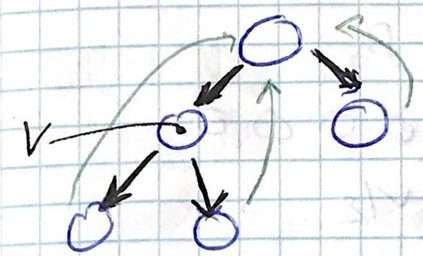
<u>Д-во:</u> $\boxed{\Rightarrow}$

Пусть есть мост 

. Ориентируем его. $\Rightarrow$ из v в u нет пути

$\boxed{\Leftarrow}$

Запустим DFS. Ориентируем ребра согласно ребрам дерева (вниз) и ребрам обратным (вверх)



$\triangle$ вершину v: $tin[v] \geq tup[\text{потомки } v] \Rightarrow$

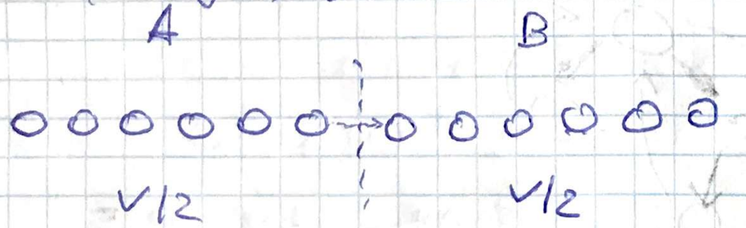
$\Rightarrow$ из любого потомка v можно добраться выше (в предка v) $\Rightarrow$ из любой вершины можно добраться до корня, а из корня в любую другую

Эйлеров путь

10.10.20

СЕМИНАР

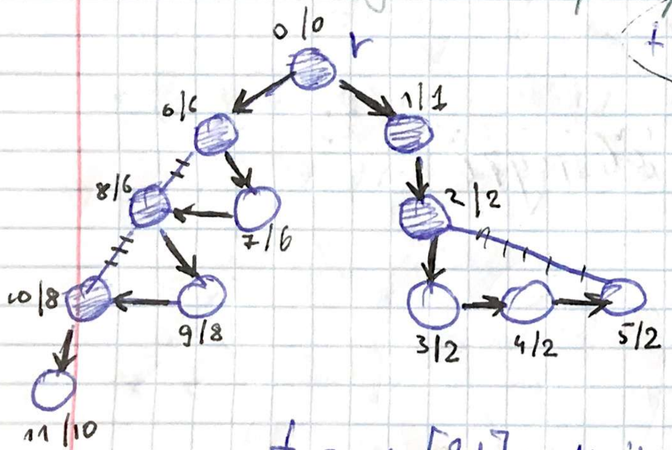
Граф ациклический. Можно ли отсортир. за $O(v \log v)$?



Быстрее, чем $O(v^2)$ тополог. сортировку не удастся.

Мосты и точки сочленения (для неор. графов)

tin / tback



$$tback[u] = \min \begin{cases} tin[u] \\ tin[b], & b \text{ ← back edge} \\ tback[w], & w \text{ ← tree edge} \end{cases}$$

```
cutVertices(G):
    colors[0,..,v-1] = white
    answer = set()
    tin[0,..,v-1] = -1
    tup[0,..,v-1] = ∞ ; // = tback
    for v in G.vertices():
        if color[v] == white : // tin == -1
            DFS(v, G, -1, colors, tin, tup, answer)
    return answer;

DFS(v, G, parent, colors, tin, tup, answer):
    colors[v] = gray
    tin[v] = tup[v] = ++t
    n_children = 0 // число потомков
    for u in G.neighbours(v):
        if colors[u] != white and parent != u :
            tup[v] = min(tup[v], tin[u])
        if colors[u] == white:
            ++n_children
            DFS(u, G, v, colors, tup, tin, answer)
            tup[v] = min(tup[v], tup[u])
            if tin[v] <= tup[u] and p != -1 :
                answer.add(v)
```

DFS от каждой КС

число потомков

Пример:

~~color[v] = black~~

parent ← not root

colors[v] = Black

if parent == -1 and n_children >= 2:

answer.Add(V)

Мосты ищутся аналогично, за исключением:

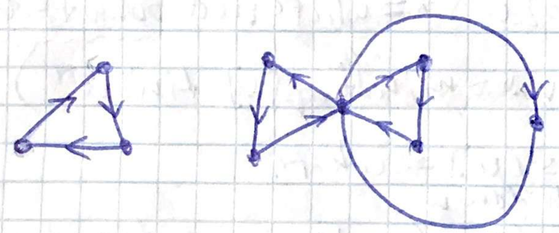
1) tin[u] < tup[v] (вместо ≤)

(тогда (u, v) - мост)

2) нет случая корня

Эйлеров цикл

• Эйлеров цикл - цикл, проход. по каждому ребру графа ровно один раз



Критерий:

Степень каждой вершины четна, граф связен

EulerCycle(G, v, answer):

for (v, u) ∈ G.E ← по ребрам

remove(v, u) ← или помечать

EulerCycle(G, u, answer)

answer.push_back((v, u))

(a, b) (b, c) (c, d) ... (z, a)

Система непересекающихся множеств (СНМ)

Disjoint-set union (DSU)

Есть набор чисел 0 ... N-1

Разбиты на непересек. мно-ва:

{0} {1, 2} {3, 4, 6} N = 7

{5}

Операции:

1) MakeSet() – создает новое 1 элементное мно-во и возвращает номер элемента

2) FindSet(x) – принимает эл-т мно-ва и возвращает номер мн-ва, в котором лежит x

$x \in A$ и $y \in A$, то FindSet(x) == FindSet(y)

иначе FindSet(x) != FindSet(y)
($x \in A$, $y \in B$)

3) Union(x, y) – объединить мно-во A ($x \in A$) и мно-во B ($y \in B$)

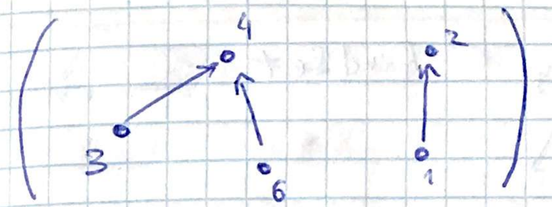
Union(2,4) → {1, 2, 3, 4, 6}

Лес непересекающихся множеств

{0}, {1, 2} {3, 4, 6} {5}

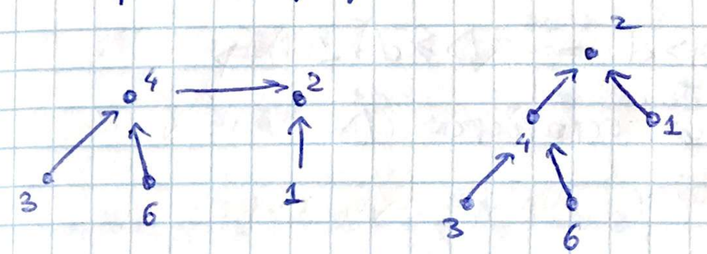
P[	0	2	2	4	4	5	4]
	0	1	2	3	4	5	6

– хранит предстовителя множества



union

Теперь 2 – предстовитель всех:



FindSet(3) 3 → 4 → 2

FindSet(1) 1 → 2

FindSet

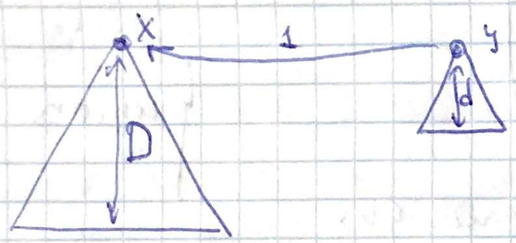
FindSet(x): O(n) – в худшем случае

```
if (P[x] == x):
    return x
return FindSet(P[x])
```

Union(x, y):

```
x = FindSet(x)
y = FindSet(y)
P[y] = x
```

FindSet



$D > d \Rightarrow D \geq d+1 \Rightarrow$

$\Rightarrow$ глубина останется D

Ранговая эвристика — подвешиваем меньшее дерево к большому

$rank[\underset{0}{r_0}\ \underset{1}{r_1}\ \underset{2}{r_2}\ \dots\ \underset{6}{r_6}]$ ← глубина

Union(x, y):

```
x = FindSet(x)
y = FindSet(y)
if (rank[x] < rank[y])
    p[x] = y
elif (rank[y] < rank[x])
    p[y] = x
else: p[x] = y
      rank[y] += 1
```

Утв size $T(v) \geq 2^{rank(v)}$ $v \in T$

База: $1 \geq 2^0$

Объединяем деревья x и y

ПИ: $size(T_x) \geq 2^{rank[x]}$

$size(T_y) \geq 2^{rank[y]}$

$size(T_x) + size(T_y) \geq 2^{r[x]} + 2^{r[y]}$ (*)

Если $r[x] > r[y]$, то $rank[T_x \cup T_y] = r[x]$

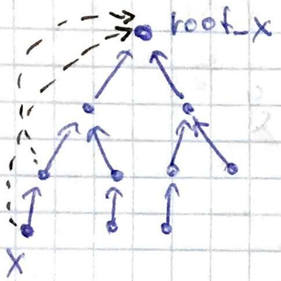
(*) $\geq 2^{r[x]}$

Если $r[x] = r[y]$, то $rank[T_x \cup T_y] = r[x]+1$

(*) $= 2^{r[x]+1}$

$\boxed{\text{Следствие: FindSet : } O(\log n)}$

Эвристика сжатия путей:



FindSet(x)

FindSet(x) ?

Один раз узнали предков за $O(\log n)$, дальше за $O(1)$

FindSet(x):

```
if ( x==p[x])
    return x;
return p[x] = FindSet(p[x])
```

Th (б/д)

При использовании ранговой эвристики и эвристики сжатия путей, амортизир. стоимость операции FindSet равна $O(\alpha(n))$, n - число элементов, α - обратная ф-я Аккермана

$\alpha(1) = 0$

$\alpha(3) = 1$

$\alpha(7) = 2$

$\alpha(61) = 3$

$\alpha(2^{2^{2^{65536}}} - 3) = 4$

10^{81} - число атомов в видимой вселенной

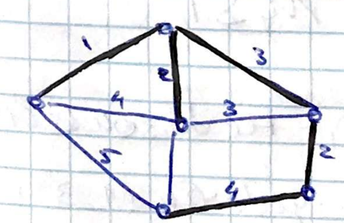
Минимальное остовное дерево

Minimum Spanning Trees (MST)

Дано: неор. связный взвешенный граф G

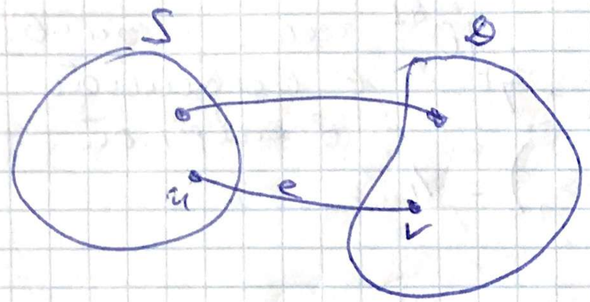
$w: E \to \mathbb{R}$ - вес ребра

Нужно: найти дерево с ~~сум~~ min суммой весов ребер



Опр • Разрезом графа G называется пара множеств (S, D): $S \subseteq V$

$D = V \setminus S$



e пересекает разрез (S, D), если $u \in S$, $v \in D$

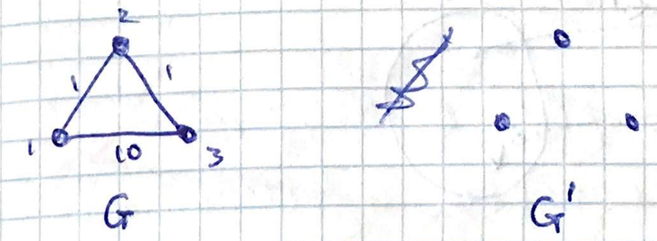
Опр • Безопасное ребро

Пусть $G' \subseteq T \subseteq G$ – подграф некоторого минимального остовного дерева T

Ребро e безопасно для G', если $G' \cup \{e\}$ – подграф некоторого MST

e войдет в какой-то MST

$(1,3)$ – не безопасно, т.к. добавл. $(1,3)$ – не будет дерева

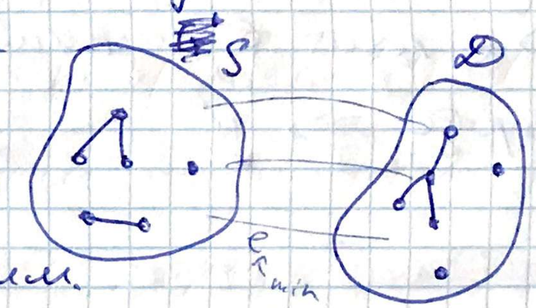


$(2,3)$ – не безопасно

Th (о безопасном ребре)

Пусть $G' \subseteq T$ – подграф некоторого MST T. (S, D) – разрез G: ни одно из ребер G' не пересекает его.

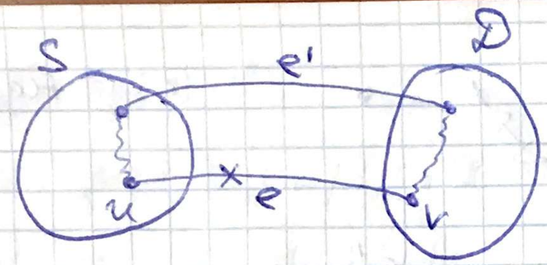
Тогда e – безопасно для G', если e минимальное по вес ребро, пересекающее разрез



Д-во:

$\triangle\ T: G' \subseteq T$.

Допустим e не лежит в T. Тогда на пути из u в v есть другое ребро e', которое пересекает разрез.



Добавим в T ребро e ⇒ получим цикл.

Удалим e'. Граф останется связным.

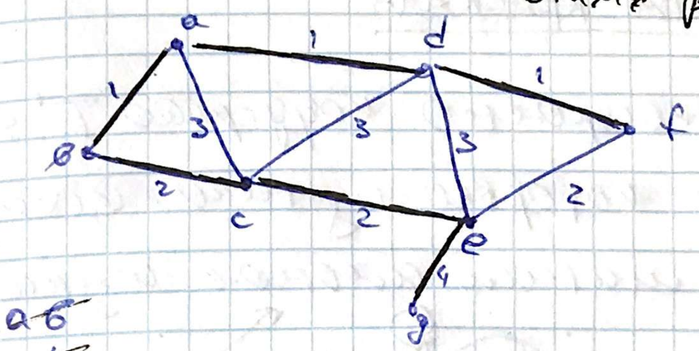
$w(e) \le w(e')$ ⇒ вес дерева не увеличился ⇒ e лежит в минимальном остовном дереве ⇒ ✓

Алгоритм Прима

1. Сортируем ребра по ↑ веса $O(E \log E)$
2. Берем минимальное ребро, проверяем есть ли цикл, если нет то добавляем в MST ← проверка цикла ? $O(\alpha(V))$
3. Повторяем 2 пока не построим дерево.

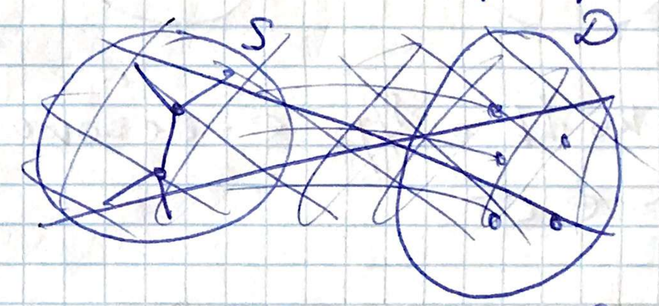
если (u,v) соедин. вершины из одного мно-ва, то есть цикл. Проверка цикла – DSU

Время работы: $O(E \log E)$

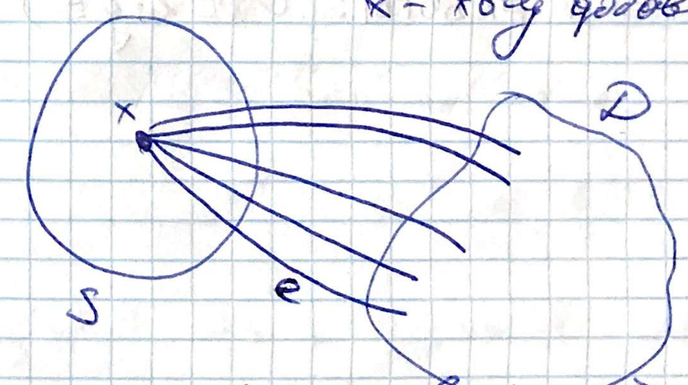


~~ab~~
~~ad~~
~~df~~
~~bc~~
~~ce~~
ef ← skip
ac skip
cd skip
~~eg~~de skip
~~eg~~

Корректность следует из th о безопасном ребре:



x – хочу добавить



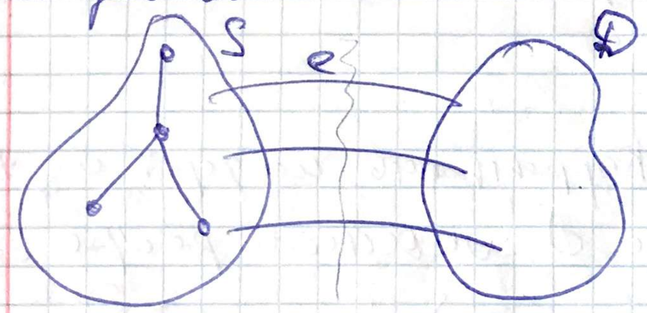
e – min по условию ⇒

⇒ по th о б/р оно безопасно ⇒

⇒ можно добавить в MST

Алгоритм Краскала

Пусть уже построено поддерево $T' \subseteq T$.
Δ все ребра, которые из него исходят и добавим минимальное из них.



каждое ребро ищем с помощью пирамиды ($O(\log E)$)

по th о б/р e - безопасно $\Rightarrow$
$\Rightarrow e \in T$

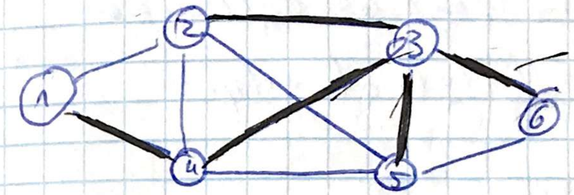
Время работы: $O(E \log E)$

Семинар

17.10.20

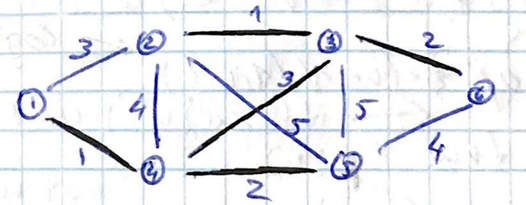
MST (Minimum Spanning Tree)

• Остовный подграф G' графа G - подграф : $V' = V$

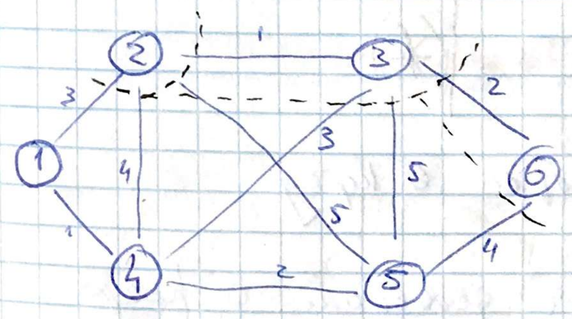


остовное дерево

Граф G связен и взвешен.



Алгоритм Прима:



вершины	ребра
2	
3	(2, 3)
6	(3, 6)
4	(3, 4)
1	(4, 1)
5	(4, 5)

~~MST(G)~~

Prim (G):

```
used [0,.. V-1] = false // вершины, добав в дерево
used [0] = true.
current = 0 // текущая вершина
answer = [ ] // list of edges
heap = { }
heap.insert (G.neighbours(current))
while (answer.size() < V - 1):
    (v, u, w) = heap.ExtractMin()        ← logE
    if (used[u]) continue
    used[u] = true
    answer.push((p[u], u))
    heap.insert(G.neighbours(u))         ← logE
return answer
```

v, u, weight

not used neighbours

(*)

E раз добавляем ребра в кучу

O(E logE) — E раз добавляем в heap

O(E logE) - асимптотика

Улучшение:

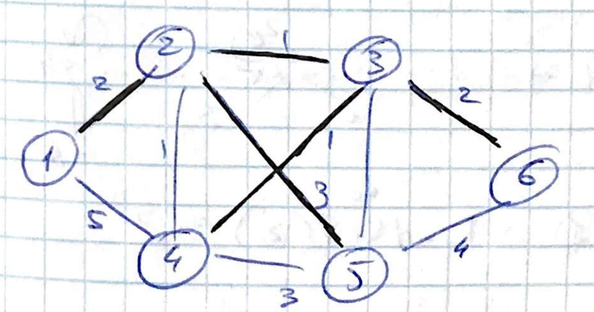
добавляем конца ребра (а не само ребро), вес ребра по которому идем

~~d[0..V-1] = ∞~~ // оценка расст. до ост. дерева

p[0..V-1] = ?

```
(*): for (auto v: g.neighbours(u)):
        if (!used[v] && d[v] > (u,v).weight):
            ~~d[v] = (u,v).weight~~
            p[v] = u
            heap.push((v, weight))
```

Алгоритм Краскала



(3,4)
(2,3)
(1,2)
(3,6)
(2,5)

kruskal (G):

$E\log E \rightarrow$ Sort(G.edges) // by weight

DSU dsu(n)

for e in G.Edges:

if e.u and e.v in different components: $O(\alpha(n))$

?

answer.push(e), dsu.Union(u, v)

dsu.FindSet(e.u) != dsu.FindSet(e.v)

Можно DFS $O(V)$

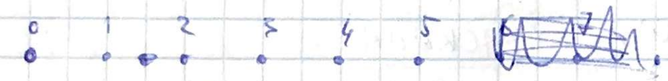
$\Downarrow$

$O(EV) \sim V^3$

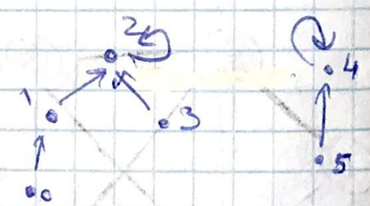
$$O(E\log E + E\alpha(n)) = O(E\log E)$$

Disjoint_set Union (DSU)

1) MakeSet() - новое одноэл. мн-во
2) FindSet(x) - возвращает номер мн-ва, в котором лежит x
3) Union(x, y) - объединить A и B: $x \in A$, $y \in B$



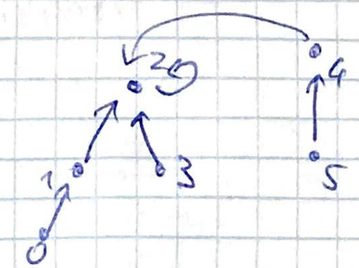
$P[\,1\ \ 2\ \ 2\ \ 2\ \ 4\ \ 4\,]$



FindSet(2) = 2

FindSet(0) → FindSet(1) → FindSet(2) = 2

Union(1, 5) → Union(2, 4) → P[4] = 2



Ранговая эвристика: rank[ ]

FindSet: $O(n) \rightarrow O(\log n)$ — глубина дерева

Если объедин. x и y; rank[x] > rank[y], то меньшим y к x → rank[x] меняется

Если rank[x] == rank[y], то rank[x] увелич. на единицу

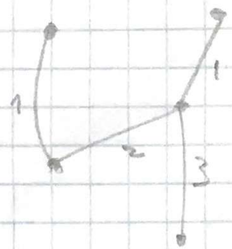
Эвристика сжатия путей:

FindSet: $O(\log n) \rightarrow O(\alpha(n))$

```
FindSet(x):
    if x == P[x]
        return x;
    return P[x] = FindSet(P[x])
```

Алгоритм нахожд. подход хеш-ф. из универ. сем. ф. для статич. хеш таблицы

II уровня



```
class DSU {
    using elem_t = int;
    std::vector<elem_t> predecessors;
    vector<size_t> ranks_;
public:
    DSU(size_t n_sets)
        : predecessors(n_sets), ranks_(n_sets, 0) {
    for (elem_t i=0; i<n_sets; ++i) {
        predecessors_[i]=i;
    }
    }

    elem_t MakeSet() {
        predecessors_.pushBack(predes.size())
        ranks.pushBack(0);
    }   return pred.size()-1;

    elem_t FindSet(elem_t x) const {
        if (predes[x]==x) { return x }
        predes[x]=FindSet(pedec[x])
    }   return pred[x];
```

mutable в объявлении

```
void Union(elem_t x, elem_t y) {
    x=FindSet(x);
    y=FindSet(y);
    if (ranks_[x] < ranks_[y]) {
        pred[x]=y
    } else if (      >      ) {
        pred[y]=x
    } else {
        ++rank[x];
        pred[y]=x;
    }
}
```

Исправление косяка на пред. лекции

Алгоритм Краскала

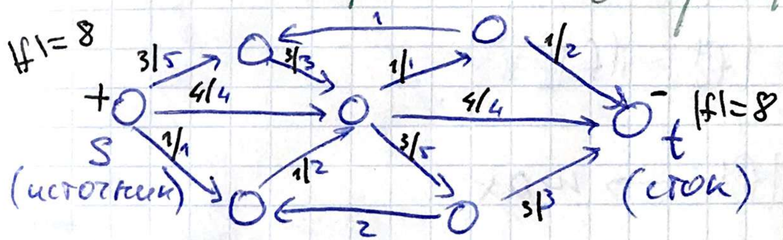
Use сортировка ребер + DSU

Алгоритм Прима

Пирамида

22.10.20

Потоки в графах



кол-во проходящ. жидкости max сколько проводят

<u>Задача:</u> Максимизировать поток из s в t.

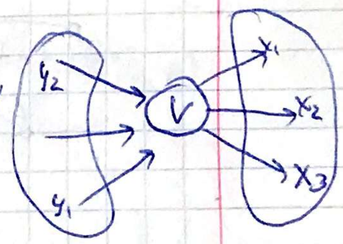
$(u, v) \in E$

$\exists c : E \to \mathbb{R}_+$ – пропускная способность ребра

$f(u, v)$ – величина потока из u в v

1) $0 \le f(u, v) \le c(u, v)$

2) $\forall v \neq s, t \quad \sum_{x \in V} f(v, x) = \sum_{y \in V} f(y, v)$ //



закон сохранения массы, заряда

Опр 1: $|f|_+ = \sum_{x\in V} f(s,x) - \sum_{y\in V} f(y,s)$

Величина потока

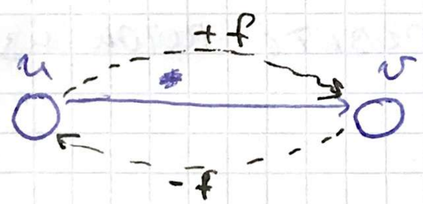
$\uparrow$ производится в S $\qquad$ $\uparrow$ сколько входит в S

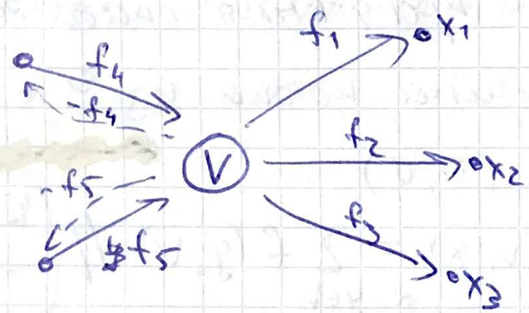
Опр 2: $|f|_- = \sum_{x\in V} f(x,t) - \sum_{y\in V} f(t,y)$

$\uparrow$ входит в t $\qquad$ $\uparrow$ выходит из t

Утверждение: $|f|_+ = |f|_-$

Задача: $|f| \longrightarrow \max$





Было:

$\sum_{x\in V} f(v,x) = \sum_{y\in V} f(y,v)$

$f_1 + f_2 + f_3 = f_4 + f_5$

Стало:

$\sum_{x\in V} f(v,x) = f_1 + f_2 + f_3 - f_4 + f_5$

Опр f – поток в G, если $f: V\times V \to \mathbb{R}$

1) $\forall x,y \quad f(x,y) = -f(y,x)$
2) $\forall x,y \quad f(x,y) \le c(x,y)$ // если $(x,y)\notin E$ $\quad c(x,y)=0$
3) $\forall v \ne s,t \quad \sum_{x\in V} f(v,x) = 0$

Опр $|f|_+ = \sum_{x\in V} f(s,x)$

$|f|_- = \sum_{y\in V} f(y,t)$

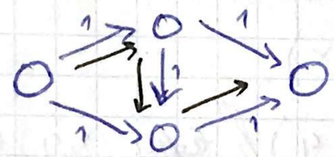
Утв: $|f|_+ = |f|_-$

Д-во $0 = \sum_x \sum_y f(x,y) = \sum_y f(s,y) + \sum_y f(t,y)$

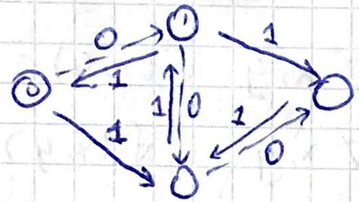
$+ \sum_{x\ne t,s} \overbrace{\sum_y f(x,y)}^{=0} = \sum_y f(s,y) + \sum_y f(t,y)$

$\Downarrow$

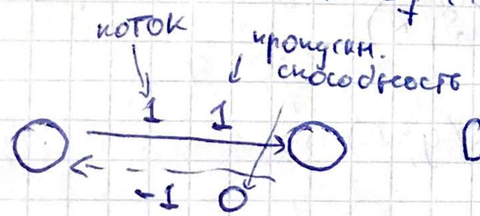
$\sum_y f(s,y) = -\sum_y f(t,y) = \sum_y f(y,t)$



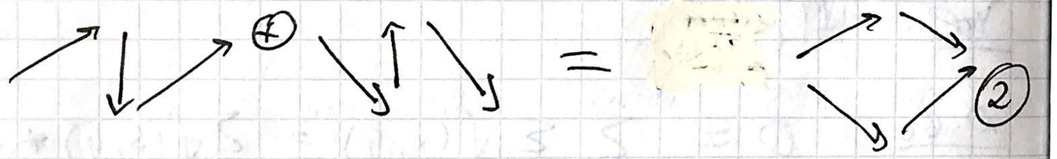
$f=2$



<u>Остаточная сеть</u> G_f (для потока f графа G) – это граф G_f: $C_f(x,y) = C(x,y) - f(x,y)$



$C_f = 0 - (-1) = 1$



<u>Лемма</u> (Арифметика потоков)

① Пусть f – поток в G, а f' – поток в G_f, тогда $f+f'$ – поток в исходном графе G, причем $|f+f'| = |f| + |f'|$

<u>Д-во</u>: Проверим по опред, что $f+f'$ – поток

1) $\forall x,y \quad f(x,y) + f'(x,y) = -f(y,x) - f'(y,x)$

(т.к. f и f' – потоки)

2) Так как f' – поток в $G_f \Rightarrow f' \le \underbrace{c-f}_{c_f} \Rightarrow$

$\Rightarrow f' + f \le c$

3) $\forall v \ne s,t \quad \sum\limits_{y \in V} (f(v,y) + f'(v,y)) = 0$

$$\sum f(v,y) + \sum f'(v,y)$$

$$\sum f(v,y) = 0, \quad \sum f'(v,y) = 0$$

$$|f+f'| = \sum_{x \in V} (f(s,x) + f'(s,x)) = \sum_{x \in V} f(s,x) + \sum_{x \in V} f'(s,x)$$

$$\sum_{x \in V} f(s,x) = |f|, \quad \sum_{x \in V} f'(s,x) = |f'|$$

② Пусть f_1 – поток в G и f_2 – поток в G, тогда $f_1 - f_2$ – поток в G_{f_2}

<u>Д-во</u> Аналогично.