Жадные алгоритмы (greedy algorithms)

Общая идея: мы оптимизируем что-то итеративно

На каждом шаге делаем локально оптимальный шаг

Доказываем, что в результате получим глобальный оптимум

Непрерывный рюкзак

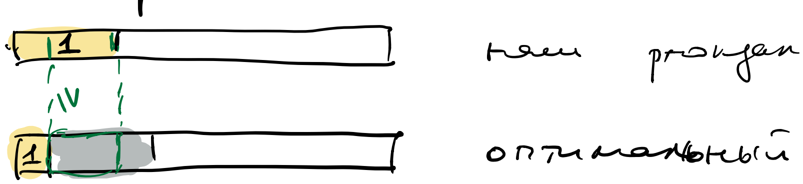
Товары $1..n$

Вес $w_1 \dots w_n$ Размер рюкзака W

Стоимость $v_1 \dots v_n$

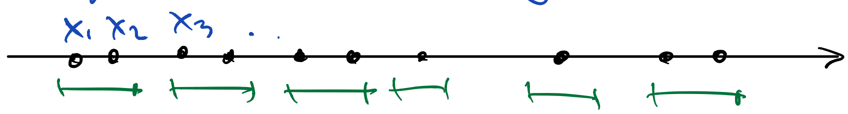
Алгоритм: упорядочиваем по $d_i = \frac{v_i}{w_i}$ (удельная стоимость) и набиваем рюкзак начиная с самого дорогого.

Док-во: $\exists$ есть оптимальное решение, не содержащее весь самый дорогой товар.

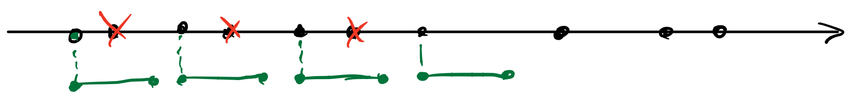


∃ опт. заполнение, кот. включает 1 товар полностью.

Покрытие точек единичными отрезками



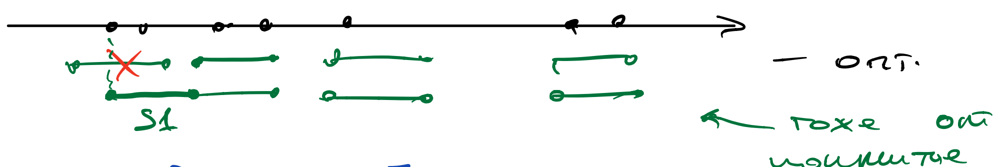
Построить покрытие отрезками длины 1 минимального размера



$S1 = [x_1, x_1+1]$

<u>Утв:</u> ∃ опт. покрытие содержащее S1.

▷ ∡ опт. покрытие. И ∡ самый левый отрезок.



$S1 = [x_1, x_1+1]$

◁

Задача о выборе заявок

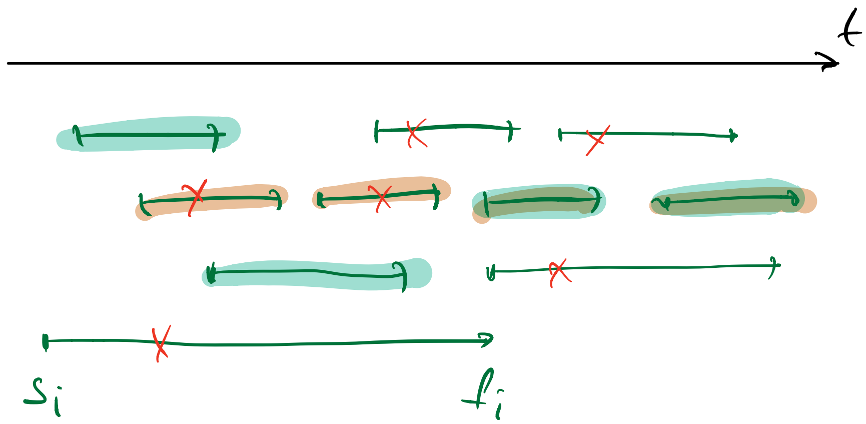
Набор заявок: $\{[s_i, f_i)\}_{i=1}^{n}$

время начала

время конца

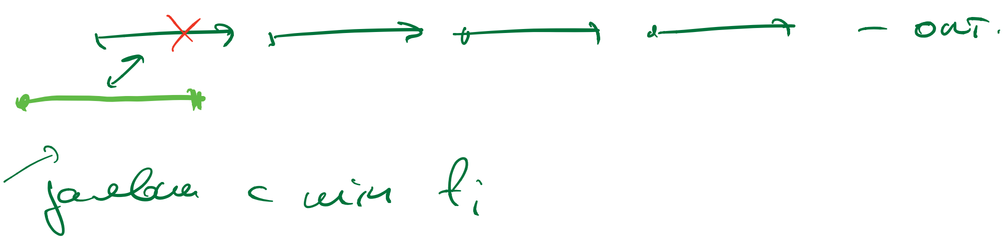
Найти подмножество непересек. заявок

максимального размера



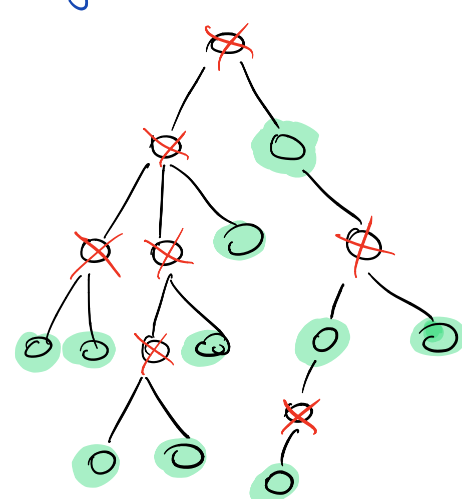
Жадный шаг: удовлетв. заявку, которая заканчивается раньше всех

Докажем, что $\exists$ опт. решение, включающее заявку с min f_i.



Задача о max нез. мн-ве в деревьях

max $|IS|$



Жадный шаг:
берём все листья

Кодирование Хаффмена

Дана строка в алфавите Σ

Задача: закодировать её так, что бы длина кода была min.

$\equiv$ Однозначно декодируемый код — код, кот. всегда можно однозначно декодировать (инъекция)

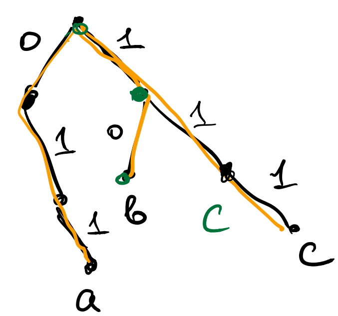
$\forall x, y : x \neq y$ выполняется $C(x) \neq C(y)$

$\equiv$ Префиксный код (prefix-free code) — $\forall a, b \in \Sigma$ код $C(a)$ не является префиксом $C(b)$.

Утв: $\forall$ префиксный код — однознач. дек.

a	011
b	10
c	111

$\longrightarrow$



бор

0 1 1 0 1 1 1 0 1 1 1 1 1 1

$\underset{a}{\longrightarrow}\underset{a}{\longrightarrow}\underset{b}{\longrightarrow}$

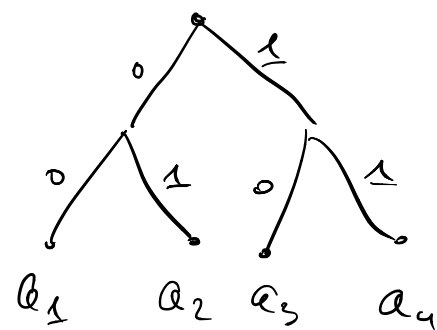
Th: Достаточно ограничиться только префиксными кодами

(∀ одн. дек можно переделать в префиксн)

Задача: оптимальный префиксный код.

\# вхожд. a_i

Σ	f_i	d_i
a_1	20	2
a_2	10	2
a_3	30	2
a_4	5	2



Задача: $\sum_{a_i \in \Sigma} f_i \cdot |c(a_i)| \to \min$

длина кода

$L(T) = \sum f_i \cdot d_i \to \min$

глубина a_i

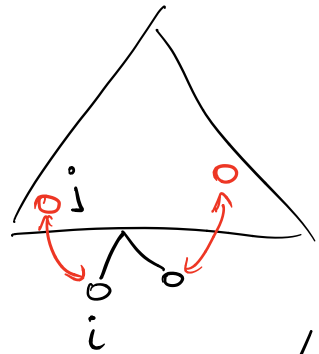
Задача: найти дерево, кот. min

Утв: В опт. дереве нет одиноких листьев

$\Rightarrow$ на нижн. уровне как минимум 2 листа

Утв: Две вершины с min частотами находятся на нижнем уровне

▷



$f_j < f_i$

$d_j < d_i$

$L(T) = \sum f_i \cdot d_i \to \min$

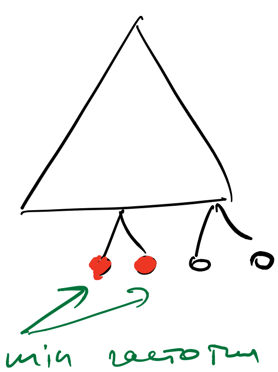
$$\Delta L = -f_i \cdot d_i + f_i \cdot d_j - f_j \cdot d_j + f_j \cdot d_i$$

$$= d_j \underbrace{(f_i - f_j)}_{>0} - d_i \underbrace{(f_i - f_j)}_{>0}$$

$$= \underbrace{(d_j - d_i)}_{<0} \underbrace{(f_i - f_j)}_{>0} < 0$$

◁

Утв: $\exists$ опт. дерево, в кот. две вершины с min частотами образуют "вилку"

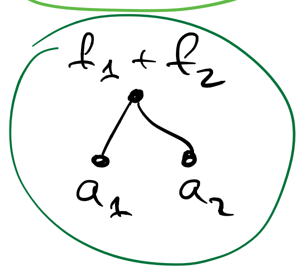


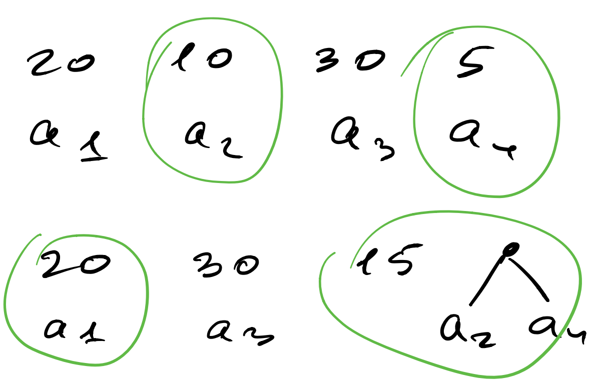
Алгоритм Хаффмана:

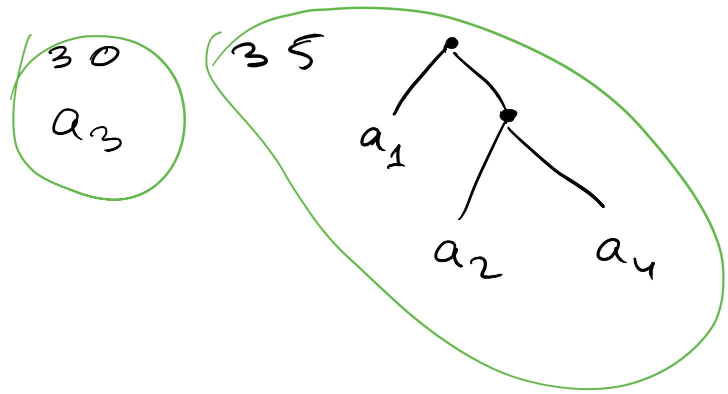
```
P = make_priority_queue()
for i = 1 to n:
    P.insert((f_i, a_i))              O(n log n)

while P.size() > 1:
    (a, T1) = P.extract_min()
    (b, T2) = P.extract_min()
    P.insert((a+b), /\ )
                   T1 T2
return P.extract_min()
```

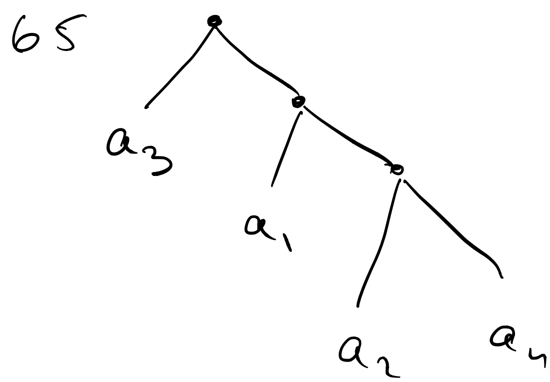
f_1 $\quad$ f_2 $\quad\quad$ f_n

a_1 $\quad$ a_2 ...... a_n — листья









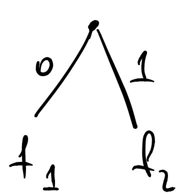
$\rightarrow f_1 \quad f_2 \quad \dots \quad f_n \leftarrow$ оптимальное

$\downarrow \qquad\qquad\qquad \Uparrow$

$\rightarrow (f_1 + f_2) \quad f_3 \dots f_n \rightarrow$ опт дерево

Док-во:

1. База: $f_1 \quad f_2$

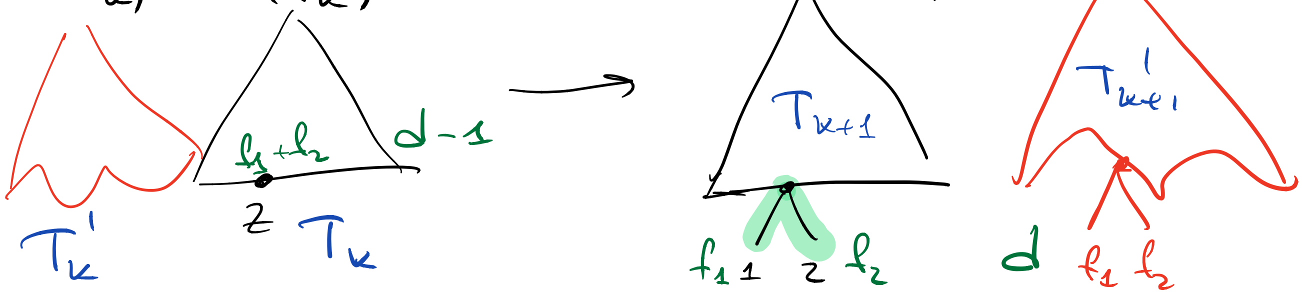


2. Пусть для $\forall$ набора из k частот мы построим опт. дерево

3. Покажем, что из него можно построить оптимальное дерево для $k+1$ частоты

$$\underbrace{(f_1+f_2)}_{z}\ f_3 \ \ldots \ f_{k+1}$$

$$C(T_k') < C(T_k) \qquad\qquad C(T_{k+1}) > C(T_{k+1}')$$



$$C(T) = \sum \underbrace{f_i}_{\text{частота}} \cdot \underbrace{|c_i|}_{\text{длина кода}} = \underline{\sum \underbrace{f_i}_{\text{частота}} \cdot \underbrace{d_i}_{\text{глубина}}}$$

$$\ldots + (f_1+f_2)\cdot(d-1) + \ \ldots \ + f_1 \cdot d + f_2 \cdot d +$$

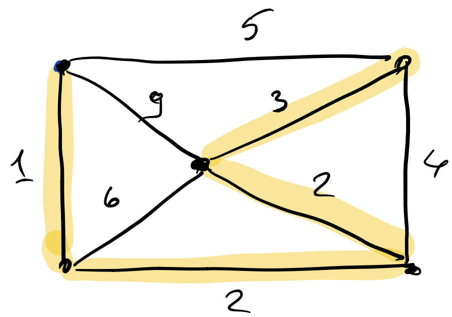
$$\boxed{\Delta = f_1 + f_2}$$

$$C(T_k) + \Delta = C(T_{k+1})$$

$$C(T_k') + \Delta = C(T_{k+1}')$$

$\lhd$

Минимальное остовное дерево



Вход: взвеш. граф
Выход: MST

minimal spanning tree

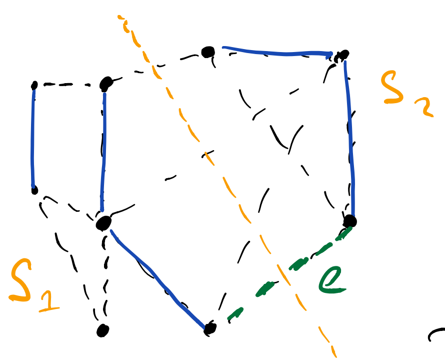
Лемма (Свойство разреза)

J M — MST

$G = (V, E)$

$\measuredangle$ $T \subset M$

$\measuredangle$ $S_1 \sqcup S_2 = V$ — разрез: в T нет ребра перес. разрез

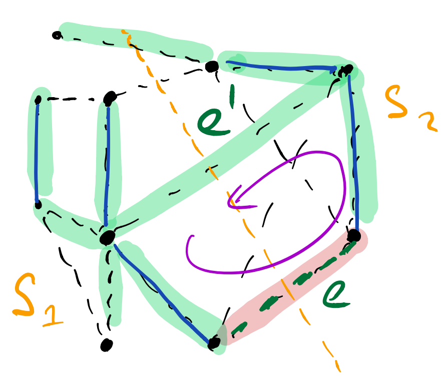


Утв: J e — min ребро перес. разрез

Тогда $\exists$ M' — MST:

$T \cup \{e\} \subset M'$

Док-во:



J M' не $\exists$

Тогда $\exists$ M^* — MST:

$e \notin M^*$, $T \subset M^*$

$\measuredangle \ T^* = M^* \cup \{e\}$

дерево

$w(e) \leqslant w(e')$

$\measuredangle \ \tilde{M} = M^* \cup \{e\} \setminus \{e'\}$

1. $\tilde{M}$ - дерево
2. Вес $\tilde{M} \leq$ вес M^*
3. Следовательно $\hat{M}$ — MST

Причём $T \subset \tilde{M}$ и $e \in \tilde{M}$

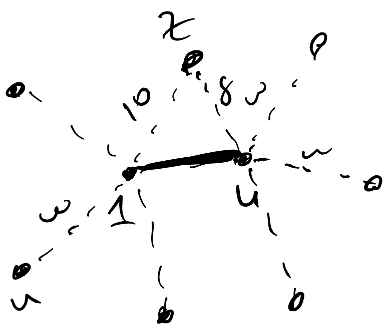
Противоречие □

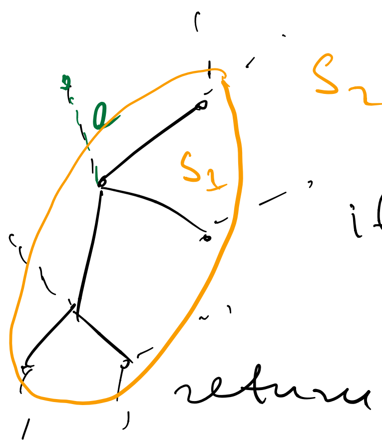
Алгоритм Прима

```
def Prim(V, E):
    for v ∈ V:
        dist[v] = ∞
        prev[v] = 0
    dist[1] = 0
    T = []
    Q = make_priority_queue()
    Q.insert(0, 1)
```

i

```
while Q.size() > 0:
    v = Q.extract_min()
    for (v, u) ∈ E:
        if dist[u] = ∞:
            Q.insert(w(v,u), u)
            dist[u] = w(v, u)
            prev[u] = v
        else if dist[u] > w(v,u):
            dist[u] = w(v,u)
            prev[u] = v
            Q.decrease_key(u, w(v,u))
    if v ≠ 1:
        T.append((prev[v], v))
return T
```





Время работы:

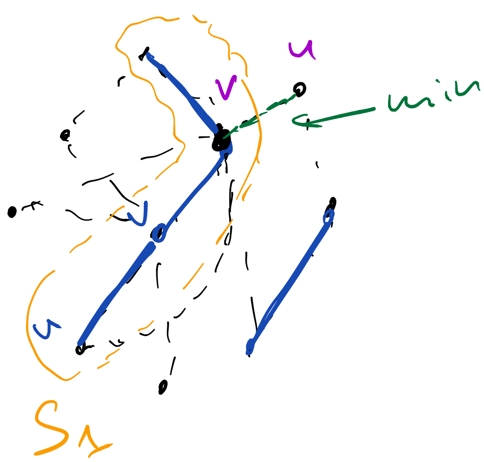
$$O\left((V+E)\cdot \log V\right) = O(E\log V)$$

Алгоритм Краскала (Kruskal)

АТД Disjoint Sets — система непересек. множеств СНМ

- make_set(v)
- union(v, u)
- find(v)

```
def Kruskal(V, E):
    for v ∈ V:                         O(V)
        make_set(v)
    T = []
    Sort(E)  ←  O(E)
    for (u, v) ∈ E:                     O(E)
        if find(v) ≠ find(u):
            Union(v, u)  ←  O(log* V)
            T.append((u, v))
    return T            Итого: O(E · log* V)
```



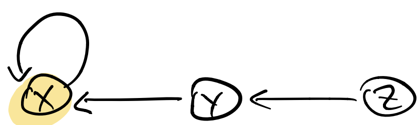
Множество ⟷ компоненты связности

Система непересекающихся множеств

- make_set
- find ← ?
- union

В каждом множестве выберем представителя

Представитель = идентификатор

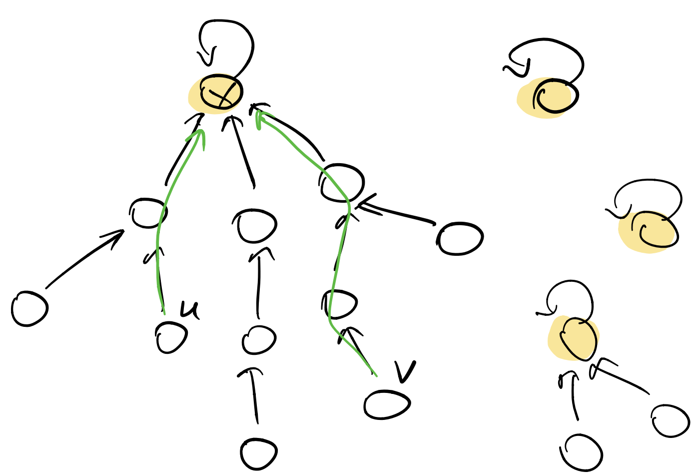



```
def make_set(x):
    parent[x] = x
```

$O(1)$

```
def find(x):
    while parent[x] ≠ x:
        x = parent[x]
    return x
```

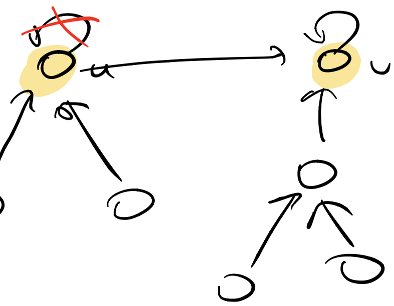
$O(n)$

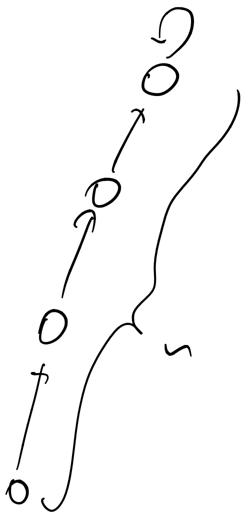


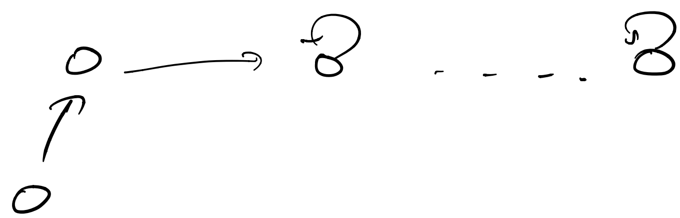
∄ ребро (u,v)

```
# u, v - представители, u ≠ v
def union(u, v):
    parent[u] = v
```

O(1)







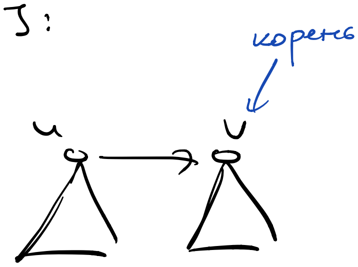
Эвристика рангов

```
def make_set(x):
    parent[x] = x
    rank[x] = 0
```

O(1)

```
def union(u, v):
    if rank[u] < rank[v]:
        parent[u] = v
    else if rank[u] > rank[v]:
        parent[v] = u
    else:
        parent[u] = v
```

O(1)



$rank[v] \mathrel{+}= 1$

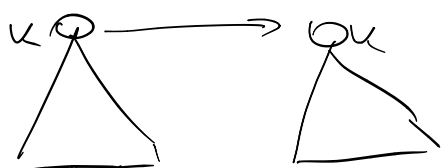
<u>Утв.</u> При исп. эвр. рангов find работает за $O(\log n)$

Лемма: В дереве с корнем ранга k не менее 2^k вершин.

▷ База: $k=0 \Rightarrow 1$ эл-т

Предположим, что верно от 0 до k

Докажем для $k+1$



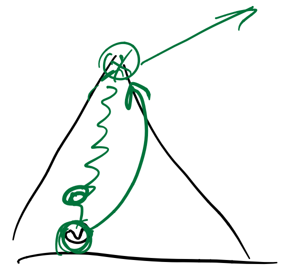
$\geqslant 2^k + \geqslant 2^k \quad \geqslant 2 \cdot 2^k = 2^{k+1}$ ◁

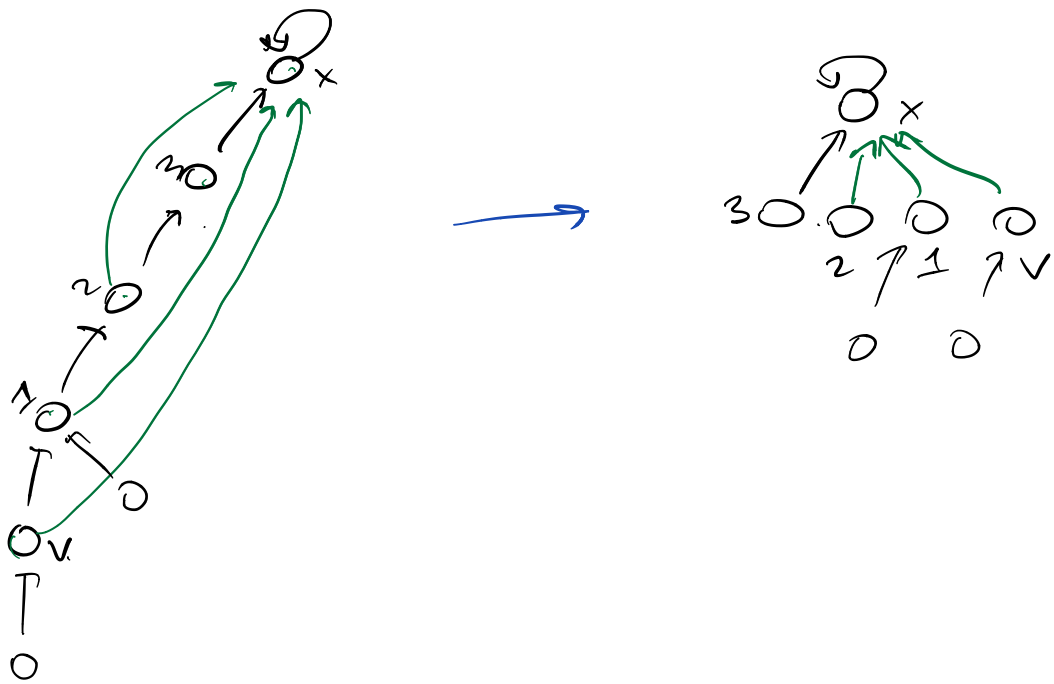
Следствие: $\max k$: $n \geqslant 2^k$

$$k \leq \log n$$

$\Rightarrow$ Утв. доказано, т.к сложность find пропорциональна max глубине деревьев

Эвристика сжатия путей






```
def find(x):
    if parent[x] ≠ x:
        parent[x] = find(parent[x])
    return parent[x]
```

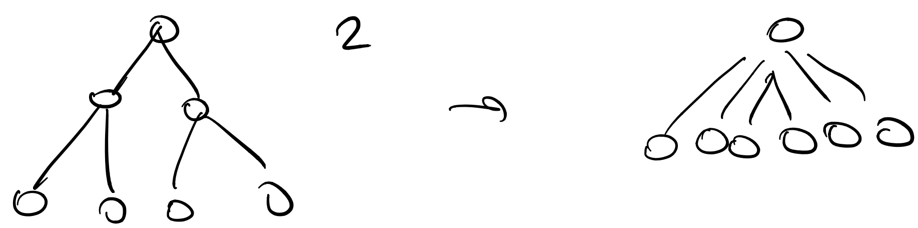
Утв СНМ с рангами и сжатием путей имеет амортизированную сложность find $O(\log^* n)$
(Если запросов find больше n, то в среднем стоимость 1 запроса не более $O(\log^* n)$)

Замечание 1: ранг ≠ высота

Замечание 2: В дереве ранга k не менее 2^k вершин.

Лемма 1. В $\forall$ цепочке find ранги возрастают.

(Если x - не корень, то $rank[x] < rank[parent[x]]$)



Замечание 3: Если вершина перестаёт быть корнем, то её ранг больше не уменьшается.

Разобьем отрезок $[1 \dots \log n]$ на отрезки вида $[k+1, 2^k]$

$$\overset{0}{[1]},\ \overset{1}{[2]},\ \overset{2}{[3,4]},\ \overset{3}{[5,\dots,16]},$$

$$\overset{4}{[17,\dots,2^{16}]},\ \overset{5}{[2^{16}+1,\dots,2^{2^{16}}]},\ \dots$$

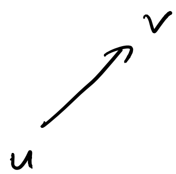
$$\dots,\ [k+1,\dots,2^k]\ \dots\ [\dots \log n]$$

$2^{2^{16}} = 2^{65536} \approx 10^{20000}$ $\qquad 10^{80}$

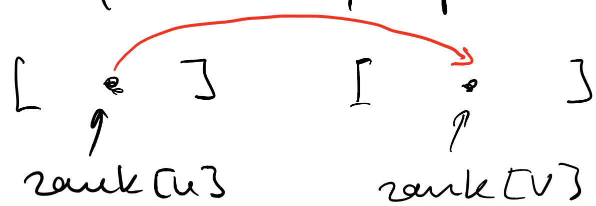
$10^3 \approx 2^{10}$

$\boxed{\text{На практике } \log^* n \le 5}$

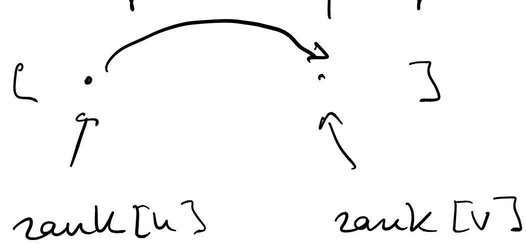
Рёбра 2х типов:



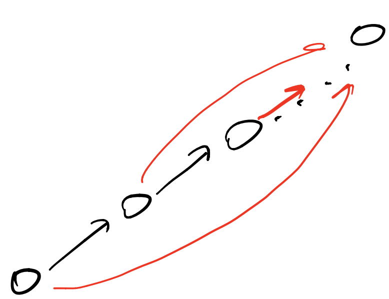
1. Красное ребро



2. Чёрное ребро

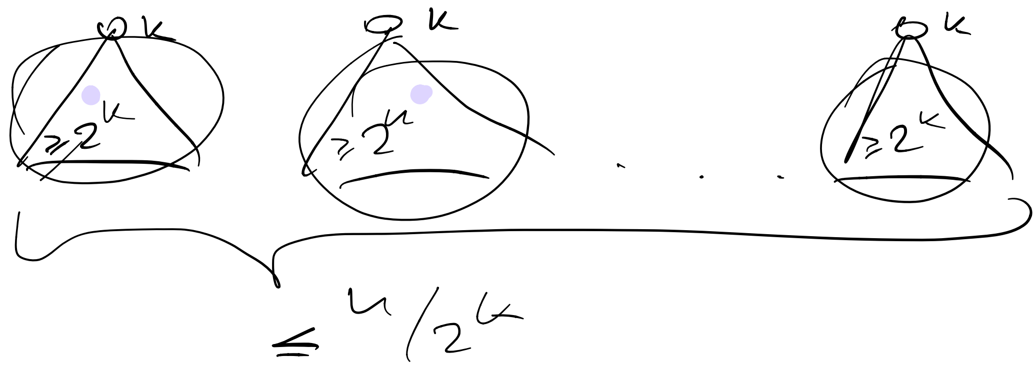


Лемма 2: В $\forall$ цепочке рёбер $\le \log^* n$ красных рёбер

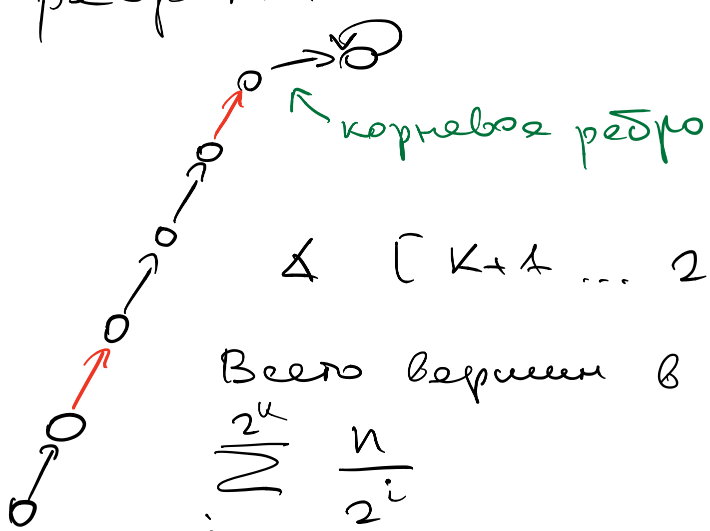


$\triangleright$ Ранг в цепочке $\uparrow\uparrow$, а количество интервалов $\le \log^* n$ $\triangleleft$

Лемма 3: Вершин с рангом k не более, чем $n/2^k$



Сколько переходов по чёрным рёбрам?



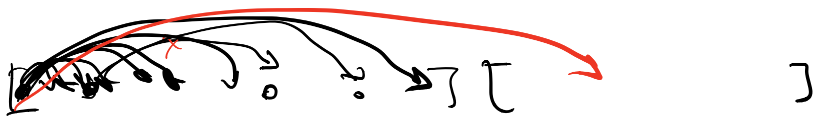
$\Delta\ [k+1 \ldots 2^k]$

Всего вершин в интервале:

$$\sum_{i=k+1}^{2^k} \frac{n}{2^i}$$

Всего переходов по чёрным рёбрам в этом интервале:

$$\le \sum_{i=k+1}^{2^k} \frac{n}{2^i} \cdot 2^k = n \cdot 2^k \cdot \sum_{i=k+1}^{2^k} \frac{1}{2^i} \le n \cdot 2^k \cdot \frac{1}{2^k} = n$$



<u>Следствие:</u> Всего переходов по чёрным рёбрам $\leq n \cdot \log^* n$

число интервалов

J выполнено m операций find.

Всего переходов:

$$m \cdot O(\log^* n) + O(n \cdot \log^* n) + O(m)$$

красные рёбра — чёрные рёбра — корневые рёбра

При $m \geq n$ в сумме $O(m \log^* n)$

На каждый запрос $O(\log^* n)$