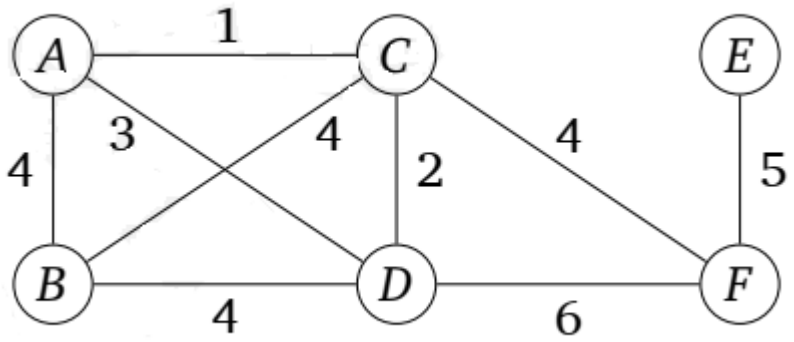
Семинар 12

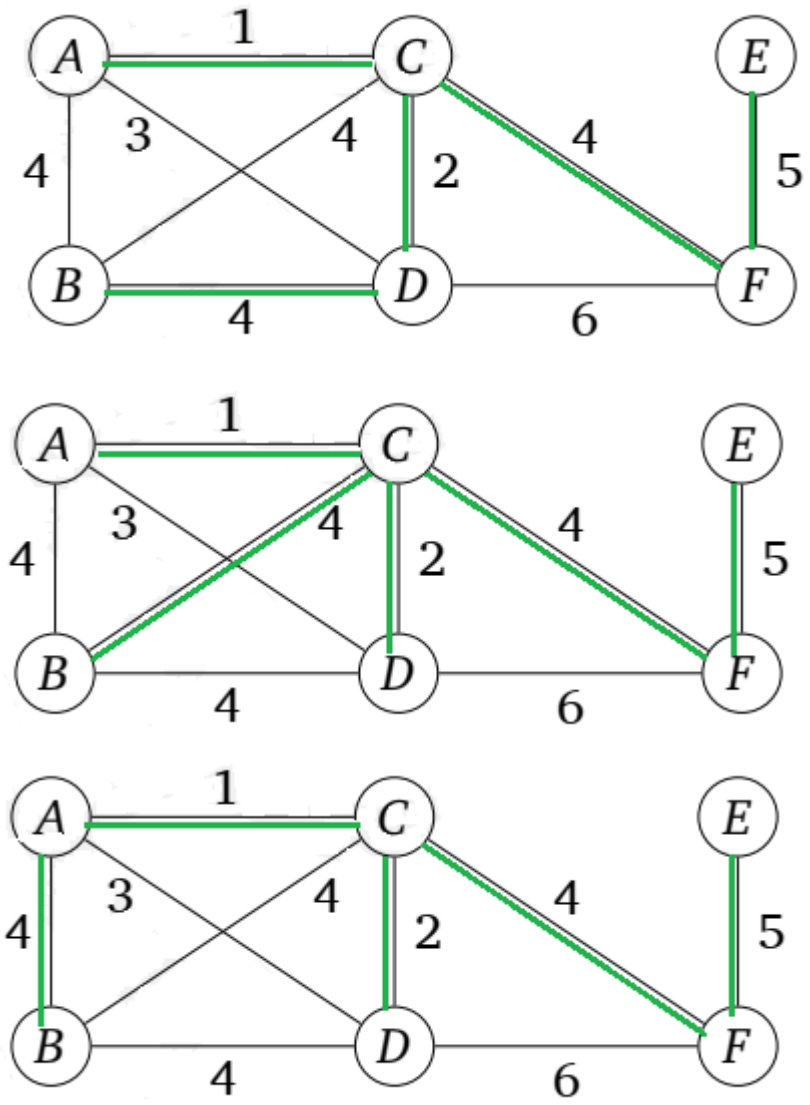
Построение минимального остовного дерева с помощью жадных алгоритмов

Пример



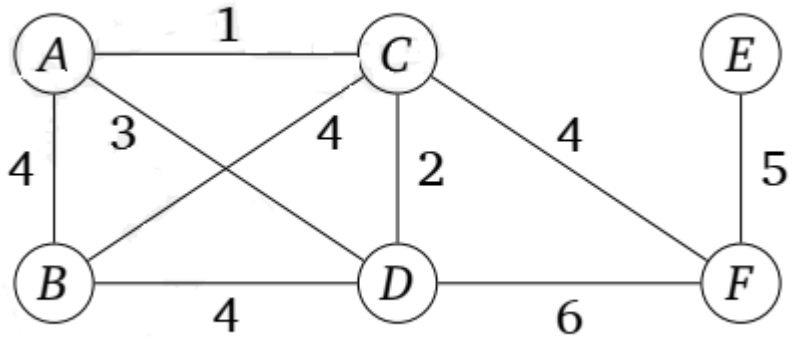
Вес остовного дерева?

Минимальные остовные деревья



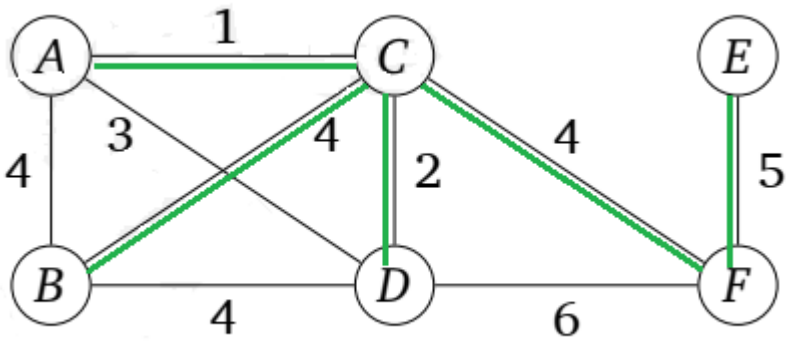
Алгоритм Крускала

- В начале ребра отсортировывают в порядке возрастания весов.
- На каждом шаге выбираем очередное ребро из отсортированного списка и добавляем его в остовное дерево. Добавляемые ребра не должны создавать циклов.



AC, CD, AD, BC, BD, CF, EF, DF
1 , 2, 3, 4, 4, 4, 5,

AC, CD, AD, BC, BD, CF, EF, DF
1 , 2, 3, 4, 4, 4, 5,



Количество ребер в остовном дереве=$|V|-1$
5=6-1

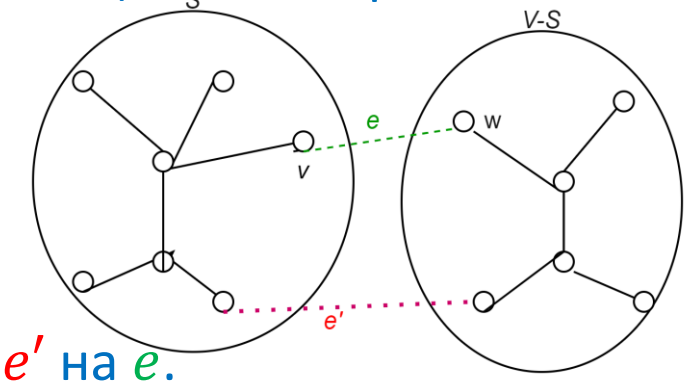
Свойство разреза

Корректность алгоритма следует из свойств разреза.

Разрез (cut) – множество ребер, соединяющих 2 непересекающихся подмножества вершин графа S и $V - S$. Разрезом также называют линию, проходящую через все рёбра разреза графа.

Пусть X – это множество ребер, входящих в MST, ни одно ребро из X не соединяет S с $V - S$.

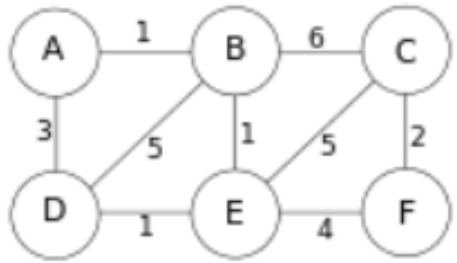
Свойство разреза: Пусть e – ребро с наименьшим весом, пересекающее некоторый разрез S и $V - S$. Тогда $X \cup \{e\}$ является частью некоторого MST.



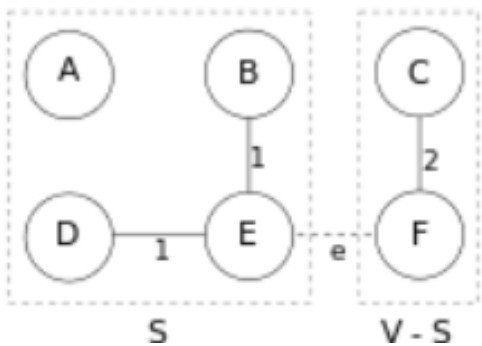
Доказательство: Пусть T – MST, X содержит ребра из T.

Если $e \in$ T, то все доказано. Если $e \notin$ T, то *v* и *w* соединены некоторым путем в T. Вместе с e получим цикл. В цикле есть некоторое e', которое тоже пересекает линию разреза. Заменим e' на e.

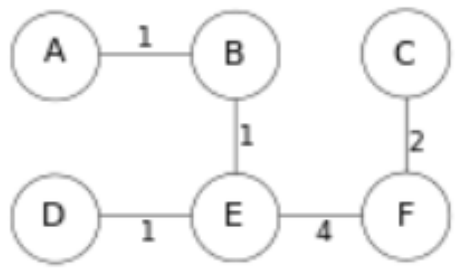
Получим связный граф с тем же числом ребер $\Rightarrow$ дерево $\tilde{T}$, его вес не больше веса T, но T- минимальное. Следовательно, $X \cup \{e\}$ является частью MST. Ч.т.д.



The cut:



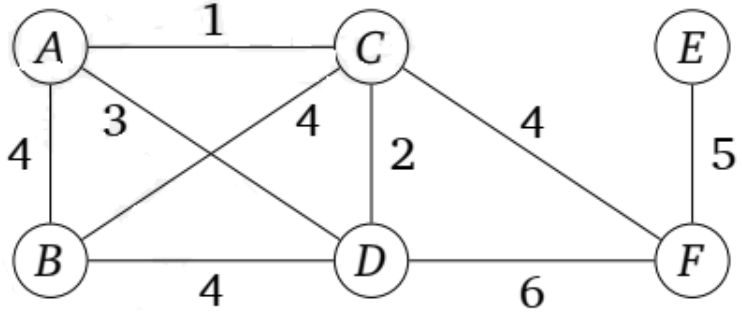
MST T:



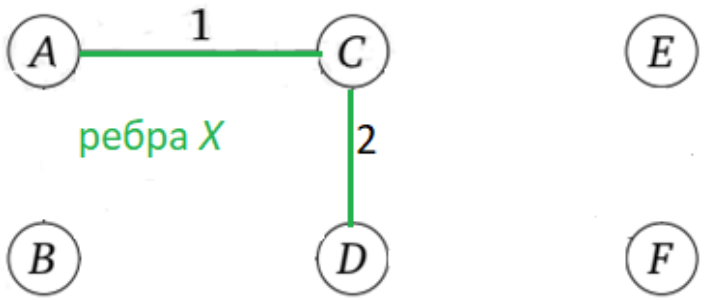
Корректность алгоритма Крускала

В каждый момент выбранные ребра $X \epsilon E_T$ образуют лес. Этот лес является частью остовного дерева. Пусть k связных компонент (деревьев леса) уже построено и e – ребро наименьшего веса, выбранное на очередной итерации. e связывает компоненты E_{T_1} и E_{T_2} из вершин $S = V_1$ в вершины $V - S = V - V_1$, $e \notin X$. Имеем разрез. По свойству разреза e будет принадлежать некоторому MST вместе с предшествующими ребрами X.

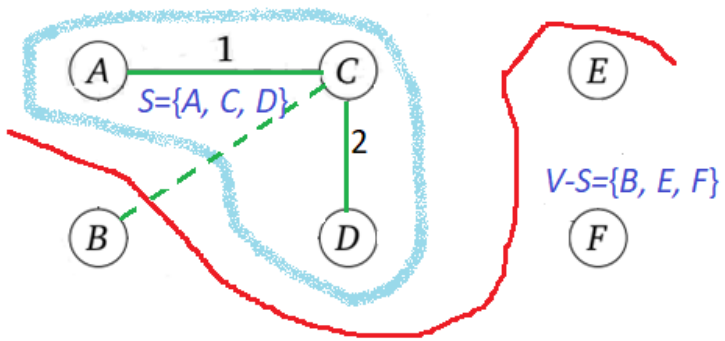
а)

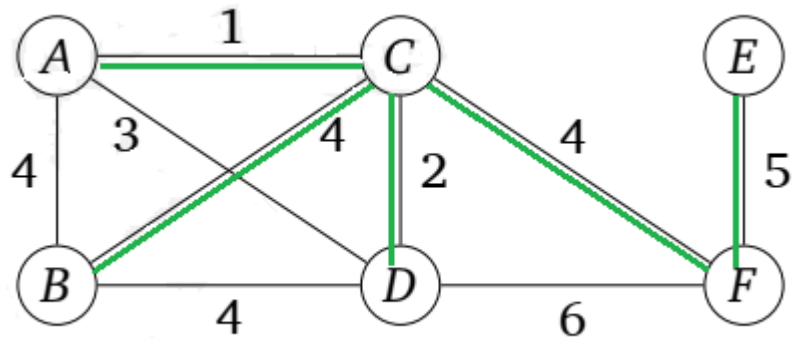


б)



в)





Псевдокод (простая реализация)

KRUSKAL

Вход: связный неориентированный граф $G = (V, E)$, представленный в виде списков смежности, и стоимость c_e для каждого ребра $e \in E$.

Выход: ребра минимального остовного дерева графа G.

```
// Предобработка
T := ∅
отсортировать ребра E по стоимости  // напр., сортировкой
                                    // слиянием MergeSort
// Главный цикл
for каждый e ∈ E, в неуменьшающемся порядке стоимости do
    if T ∪ {e} является ациклическим then
        T := T ∪ {e}
return T
```

$O(|E|log|V|)$ $(|E| < |V|(|V| - 1))$

$|E|$ итераций

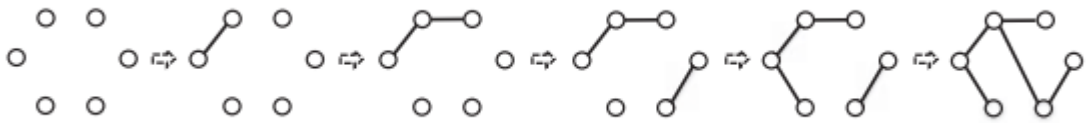
Проверка для ребра $e = (v, w)$, например, с помощью поиска в ширину в построенном дереве T пути $v \to w \Rightarrow O(|V|+|E|)$. Если путь есть, то ребро e циклическое – не добавляем.

$$\boldsymbol{O(|V| \cdot |E|)}$$

Оценку можно улучшить, если использовать структуру UNION_FIND

Структура Union_find

- сначала имеем n непересекающихся множеств (отдельных компонент связности). Каждый объект в своей компонент связности.



Операции : - инициализировать из объектов V;

- найти (find) объект в множестве и вернуть представителя этого множества;
- объединить (union) непересекающиеся множества, содержащие объекты x и y, в одно. Вернуть объединенное множество.

Алгоритм Крускала (на основе структуры UNION_FIND)

АЛГОРИТМ KRUSKAL (НА ОСНОВЕ СТРУКТУРЫ ДАННЫХ UNION-FIND)

Вход: связный неориентированный граф $G = (V, E)$, представленный в виде списков смежности, и стоимость c_e для каждого ребра $e \in E$.

Выход: ребра минимального остовного дерева G.

```
// Инициализация
T := ∅
U := Инициализировать(V) // структура данных Union-Find
отсортировать ребра E по стоимости // например, с помощью
// сортировки слиянием MergeSort
// Главный цикл
for каждый (v, w) ∈ E, в неубывающем порядке стоимости do
    if Найти(U, v) ≠ Найти(U, w) then
        // в T нет v-w-пути, поэтому можно добавить (v, w)
        T := T ∪ {(v, w)}
        // обновить из-за слияния компонент
Объединить(U, v, w)
return T
```

$O(|V|)$

$O(|E|log|V|)$ $(|E| < |V|(|V| - 1)$

Если v и w не лежат в одной связной компоненте (ребро (v, w) не циклическое) $2 \cdot O(log|V|)$ раза для каждого ребра

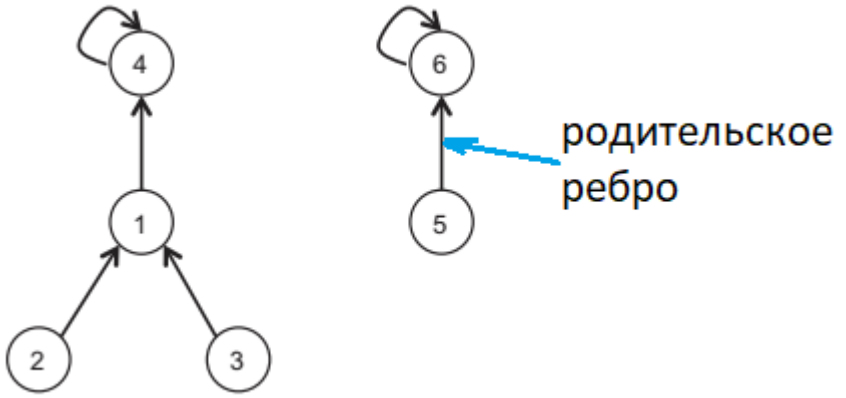
$O(log|V|)$ не больше, чем $n - 1$ раз

$O(|V|)$+ $O(|E|log|V|)$+ $2|E| \cdot O(log|V|) + O(|E|)$=$O((|V| + |E|)) \cdot log|V|$

Реализация структуры Union-Find

Массив объектов визуализируется в виде ориентированных деревьев (родительских графов)

Индекс объекта *x*	Родитель(*x*)
1	4
2	1
3	1
4	4
5	6
6	6



Операции «инициализировать» и «найти»

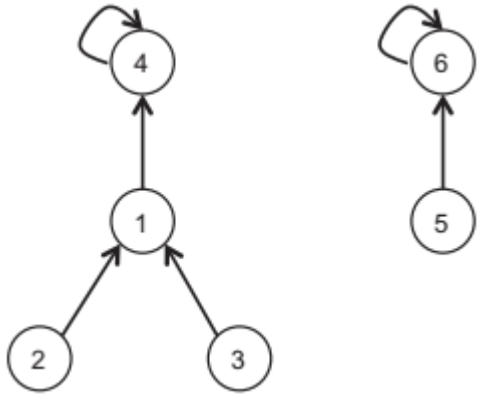
ИНИЦИАЛИЗИРОВАТЬ

Для каждого $i = 1, 2, ..., n$ инициализировать *родитель*(i) в i.

Сложность O(n)

НАЙТИ

1. Начиная с позиции x в массиве, многократно пройти по родительским ребрам до достижения позиции j, где *родитель*(j) = j.
2. Вернуть j.



Пример: Найти (U, 3)=4
Прошли путь, равный глубине дерева (3→1→4→4)

Каково время выполнения операции Найти в зависимости от числа n объектов?

а) $O(1)$

б) $O(\log n)$

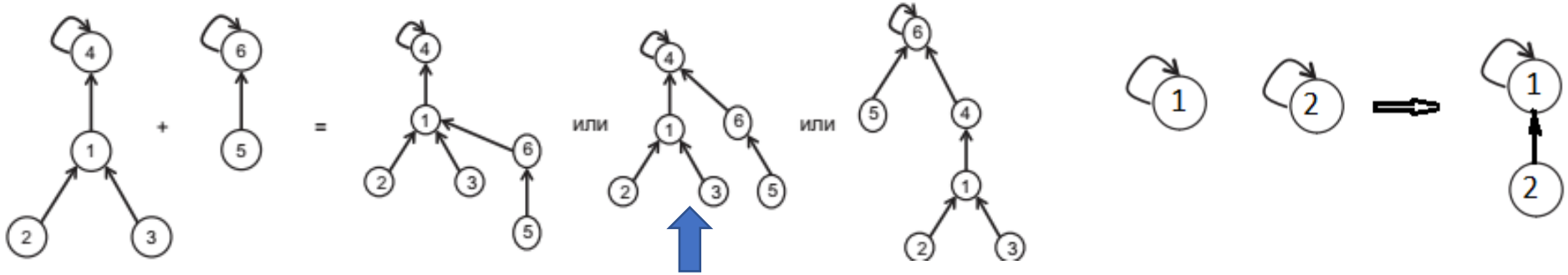
в) $O(n)$

г) для ответа недостаточно информации

Ответ зависит от способа реализации операции «объединить»

Операция «Объединить»

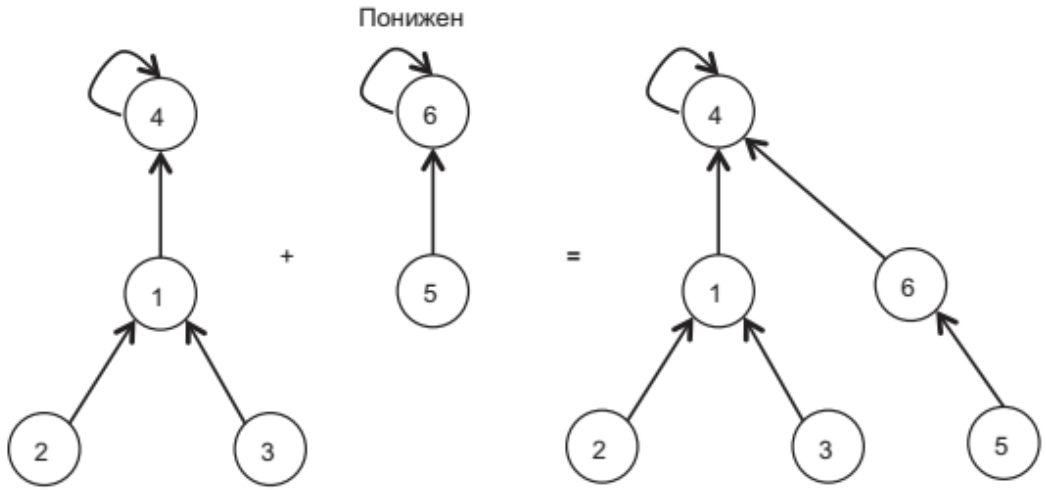
Что лучше?



Объединить(U, v, w): если v и w находятся в разных компонентах связности, лучше реализовать так, чтобы высота получившегося дерева не изменилась или увеличилась минимально. Понизим узлы в дереве меньшей глубины. В корне хранят высоту дерева.

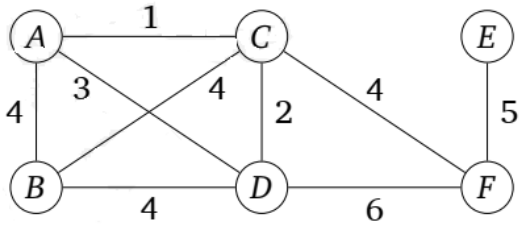
ОБЪЕДИНИТЬ

1. Вызвать операцию Найти дважды, локализовав позиции i и j корней родительских графовых деревьев, содержащих соответственно x и y. Если $i = j$, то вернуть. *// в одной компоненте связности*
2. Если rank (i) > rank(j), то установить
   родитель(j) := *родитель(i)*
   иначе установить *родитель*(i) := *родитель (j)*
   если rank(i)=rank(j) то установить rank(j):=rank(j)+1

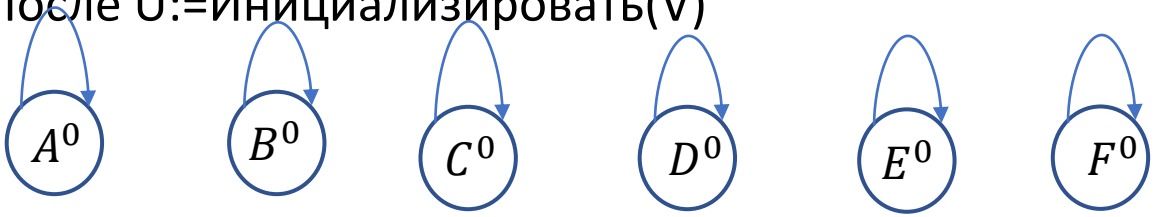


Т.к. каждый раз сливаются деревья примерно одного размера, то количество включенных вершин по крайней мере удваивается. Количество объектов $n \geq 2^x$, *поэтому глубина* $x \leq \log_2 n$. Для «Найти» *O(log n)*

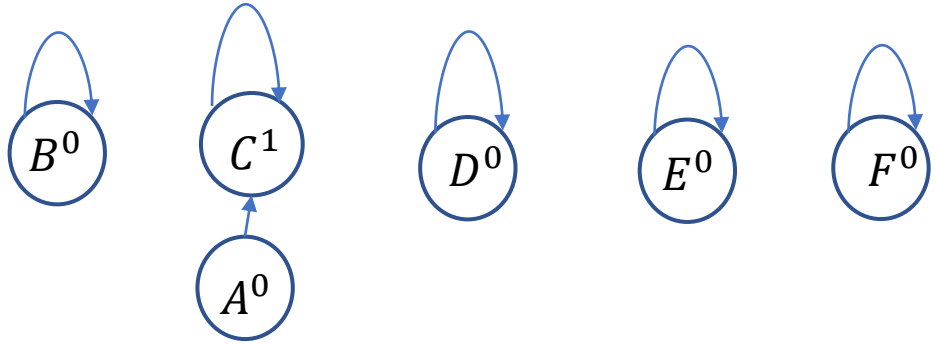
Последовательность операций для системы непересекающихся множеств



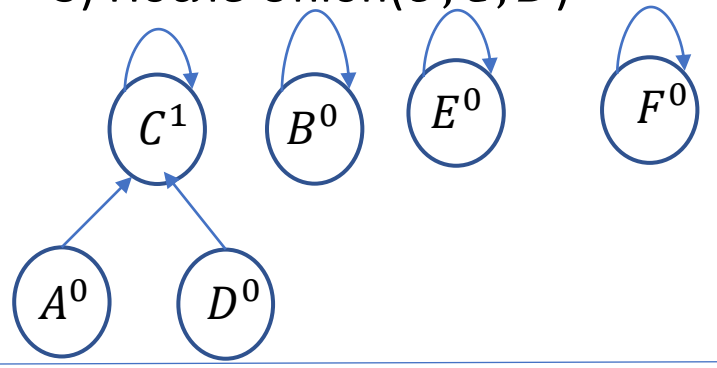
1) После U:=Инициализировать(V)



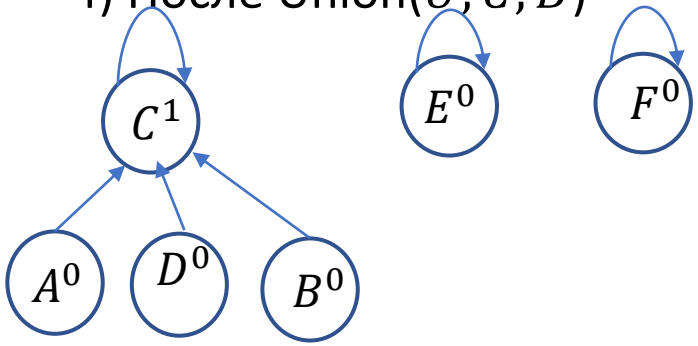
2) После Union(U, A, C)



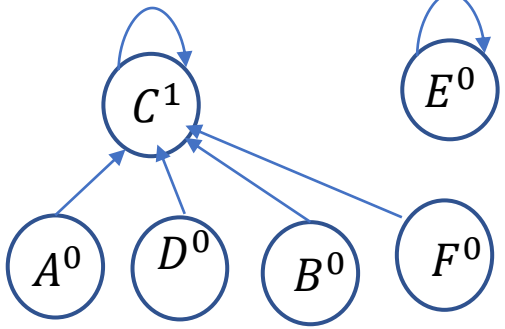
3) После Union(U, C, D)



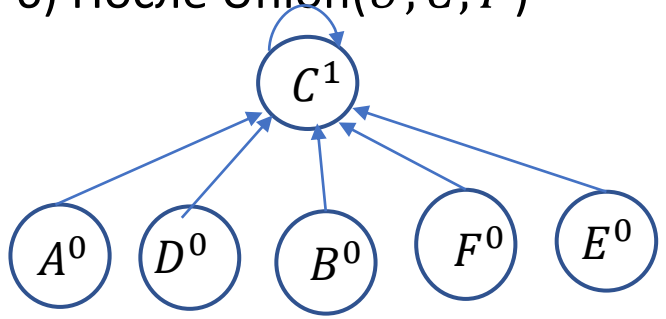
4) После Union(U, C, B)



5) После Union(U, C, F)



6) После Union(U, C, F)



Задание

Как изменить алгоритм Крускала, чтобы найти максимальное остовное дерево?

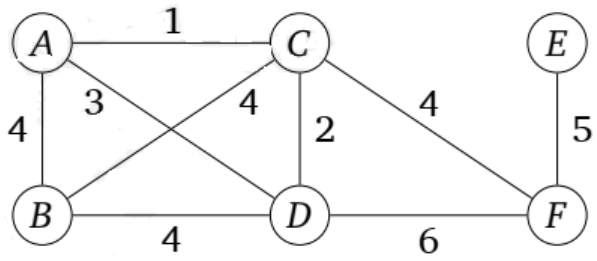
Алгоритм Прима

Сначала в пустое дерево добавляют произвольную вершину. При добавлении каждой следующей вершины выбирают ребро, инцидентное вершинам, уже включенным в MST, причем его стоимость должна быть наименьшей из стоимостей инцидентных ребер. После того, как все вершины добавлены, алгоритм останавливается. Работает подобно алгоритму Дейкстры, но вместо минимальной дейкстровой оценки ребра выбирает ребро из X в V-X с минимальной стоимостью.

Доказательство корректности на основе свойства разреза. Самостоятельно.

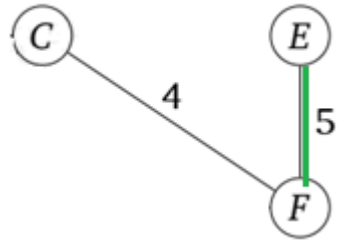
Пример.

1) Берем вершину E

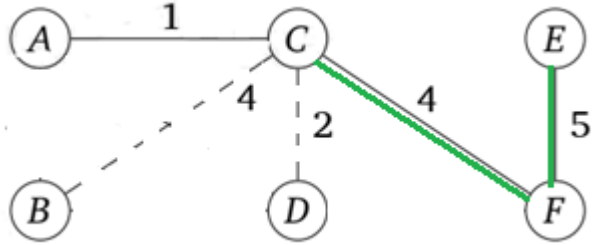


2)

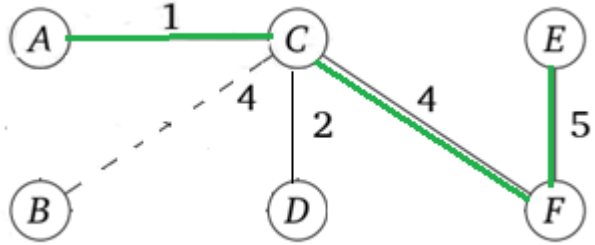
3)



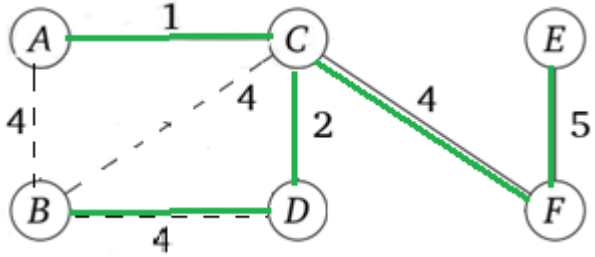
4)



5)



6)



S \ V	A	B	C	D	E	F
{}	∞/-	∞/-	∞/-	∞/-	0/-	∞/-
E	∞/-	∞/-	∞/-	∞/-		5/E
E,F	∞/-	∞/-	4/F	6/F		
E,F,C	1/A	4/C		2/C		
A,C,E,F		4/C		2/C		
A,C,D,E,F		4/C				
A,C,D,B,F,E						

Вес MST=1+4+4+2+0+5=16

Всего 3 варианта MST

Псевдокоды алгоритма Прима

PRIM

Вход: связный неориентированный граф $G = (V, E)$, представленный в виде списков смежности, и стоимость c_e для каждого ребра $e \in E$.

Выход: ребра минимального остовного дерева графа G.

```
// Инициализация
X := {s} // s — это произвольно выбранная вершина
T := ∅ // инвариант: ребра в T охватывают X
// Главный цикл
while существует ребро (v, w), где v ∈ X, w ∉ X do
  (v*, w*) := минимально-стоимостное такое ребро
  добавить вершину w* в X
  добавить ребро (v*, w*) в T
return T
```

При простой реализации получим $O(|V| \cdot |E|)$

PRIM (НА ОСНОВЕ КУЧИ)

Вход: связный неориентированный граф $G = (V, E)$, представленный в виде списков смежности, и стоимость c_e для каждого ребра $e \in E$.

Выход: ребра минимального остовного дерева графа G.

```
// Инициализация
1  X := {s}, T = ∅, H := пустая куча , key(s):=0
2  for каждый v ≠ s do
3      if существует ребро (s, v) ∈ E then
4          key(v) := c_sv, winner(v) := (s, v)
5      else      // v не имеет пересекающих инцидентных ребер
6          key(v) := +∞, winner(v) := NULL
7      Вставить v в H
   // Главный цикл
8  while H не является пустым do
9      w* := Извлечь минимум(H)
10     добавить w* в X
11     добавить winner(w*) в T
       // обновить ключи для поддержки инварианта
12     for каждое ребро (w*, y) с y ∈ V − X do
13         if c_w*y < key(y) then
14             Удалить y из H
15             key(y) := c_w*y, winner(y) := (w*, y)
16             Вставить y в H
17 return T
```

$O((|V| + |E|)) \cdot log|V|$