Лекция 6

Жадные алгоритмы. Задача о составлении расписания.
Покрывающие деревья.

Как работают жадные алгоритмы?

- Жадные алгоритмы конструируют решение итеративно, посредством последовательности близоруких решений и надеются, что в конце концов получится правильное решение.
- Как правило:

- легко придумать один или несколько жадных алгоритмов;
- легко проанализировать время выполнения алгоритма;
- трудно установить правильность (часто они не являются правильными).

Задача планирования

Есть n работ. l_j - длительность j-ой работы.

Можно составить расписание (план) работ $n!$ способами. Для расписания σ обозначим $C_j(\sigma)$ – срок завершения j-ой работы. Каждая j-ая работа, завершенная в срок $C_j(\sigma)$ имеет вес w_j.

Целевая функция $$\sum_{j=1}^{n} w_j \cdot C_j(\sigma) \xrightarrow[\text{по всем } n! \text{ перестановкам}]{} min$$

Пример: $l_1 = 1,\ l_2 = 2,\ l_3 = 3$. Тогда сроки завершения работ 1, 3, 6.

Пусть веса $w_1 = 3,\ w_2 = 2,\ w_3 = 1$. Тогда при последовательном выполнении работ (1, 2, 3).

Целевая функция: $3 \cdot 1 + 2 \cdot 3 + 1 \cdot 6 = 15$ (это минимум)

Задача минимизации суммы взвешенных сроков завершения

- Вход: множество n работ с положительными длинами $l_1,\ l_2,\ \dots, l_n$ и положительными весами $w_1,\ w_2,\ \dots, w_n$
- Выход: последовательность работ, которая минимизирует $\sum_{j=1}^{n} w_j \cdot C_j(\sigma)$
- Разработка жадного алгоритма.

Частные случаи:

1) Если все длины работ являются идентичными, то лучше сначала планировать работы большего веса.

2) Если все веса являются идентичными, то лучше сначала планировать более короткие работы.

Как совместить эти 2 эмпирических правила?

2 жадных алгоритма:

- *Предложение 1* ($\mathrm{Greedydiff}$): планировать работы в убывающем порядке величин $w_j - l_j$.
- *Предложение 2*($\mathrm{Greedyratio}$): планировать работы в убывающем порядке величин $\frac{w_j}{l_j}$.

Что выбрать?

Пример:

	Работа №1	Работа №2
длительность	$l_1 = 5$	$l_2 = 2$
вес	$w_1 = 3$	$w_2 = 1$

$\mathrm{Greedydiff}$: 3-5<1-2 (Сначала №2, потом №1)

Целевая функция $1 \cdot 2 + 3 \cdot (5 + 2) = 23$

$\mathrm{Greedyratio}$: $\frac{3}{5} > \frac{1}{2}$ (Сначала №1, потом №2)

Целевая функция $3 \cdot 5 + 1 \cdot (2 + 5) = 22$

$22 < 23 \Rightarrow$ Отвергаем $\mathrm{Greedydiff}$. Докажем правильность $\mathrm{Greedyratio}$.

Теорема: Для любого множества положительных весов $\boldsymbol{w_1},\ \boldsymbol{w_2},\ \dots,\boldsymbol{w_n}$ и множества положительных длин $\boldsymbol{l_1},\ \boldsymbol{l_2},\ \dots,\boldsymbol{l_n}$ алгоритм **Greedyratio** выводит расписание с минимально возможной суммой взвешенных работ.

Доказательство: Примем 2 допущения:

(1) Работы проиндексированы в порядке $\frac{w_1}{l_1} \geq \frac{w_2}{l_2} \geq \cdots \geq \frac{w_n}{l_n}$ (жадное расписание)

(2) Между отношениями нет совпадений $\frac{w_i}{l_i} \neq \frac{w_j}{l_j}$ при $j \neq i$

Предположим обратное: путь расписание σ алгоритма $Greedy\mathrm{ratio}$ не является оптимальным. σ^* - оптимальное расписание. Сконструируем еще одно расписание σ', которое будет лучше, чем σ^*.

Пара i,j образуют последовательную инверсию, если $i > j$, но i обрабатывается раньше j.

Лемма: Каждое расписание, отличное от жадного имеет, по крайней мере, одну последовательную инверсию

Доказательство леммы: используем правило контрапозиции: $A \to B$ эквивалентно $\neg B \to \neg A$

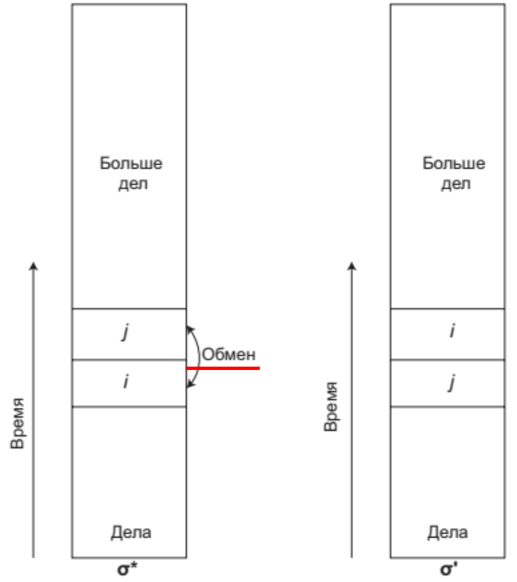
$\overbrace{\text{Не жадное расписан.}}^{A} \to \overbrace{\text{Есть посл. инверсии}}^{B} \equiv \overbrace{\text{Нет посл. инверсий}}^{\neg B} \to \overbrace{\text{Жадное расписан.}}^{\neg A}$

Индексы в возрастающем порядке, всего *n* работ⇒между индексами не может быть скачков, равным 2 и больше. Следовательно, σ – жадное расписание. Ч.т.д.

Пусть σ^* - оптимальное расписание (не жадное) ⇒

в нем есть последовательная инверсия (i, j)

После обмена сроки завершения работ, отличных от *i* и *j* остаются без изменений; срок завершения *i* увеличивается, срок завершения *j* уменьшается



а) До обмена б) После обмена

Получение нового расписания σ′ из предположительно оптимального расписания σ* путем обмена работами в последовательной инверсии (при *i* > *j*)

После обмена сроки завершения работ, отличных от i и j остаются без изменений; срок завершения i увеличивается, срок завершения j уменьшается

Значение целевой функции: $\sum_{k=1}^{n} w_k \cdot C_k(\sigma') = \sum_{k=1}^{n} w_k \cdot C_k(\sigma^*) + \underbrace{w_i l_j - w_j l_i}_{< 0}$

$$\frac{w_i}{l_i} < \frac{w_j}{l_j} \Rightarrow w_i l_j < w_j l_i \Rightarrow$$

Получили расписание σ', которое лучше оптимального σ^* - противоречие. **Ч.т.д.**

Дополнение: в случае совпадений $\frac{w_i}{l_i} = \frac{w_j}{l_j}$ утверждение теоремы верно (без доказательства)

Сложность **Greedyratio**: $O(n \log n)$

Минимальные остовные деревья

Задача о минимальном остовном дереве (MST)-minimum spanning tree

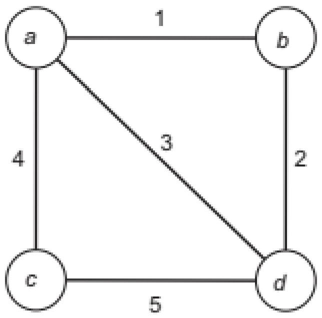
ЗАДАЧА О МИНИМАЛЬНОМ ОСТОВНОМ ДЕРЕВЕ (MST)

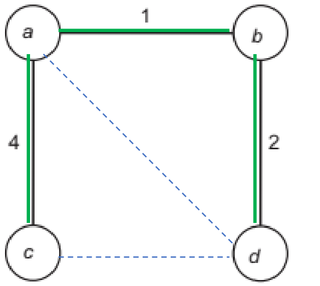
Вход: связный неориентированный граф $G = (V, E)$ и вещественная стоимость c_e для каждого ребра $e \in E$.

Выход: остовное дерево $T \subseteq E$ графа G с минимально возможной суммой $\sum_{e \in T} c_e$ реберных стоимостей[1].

Пример:

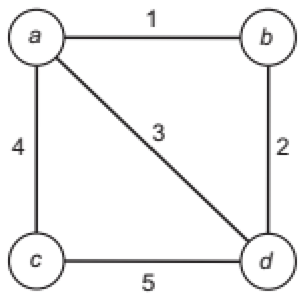
Какова минимальная сумма реберных стоимостей остовного дерева следующего ниже графа? (Каждое ребро помечено его стоимостью.)

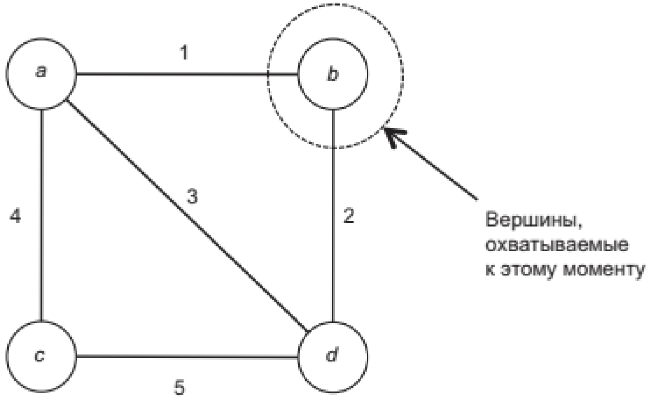




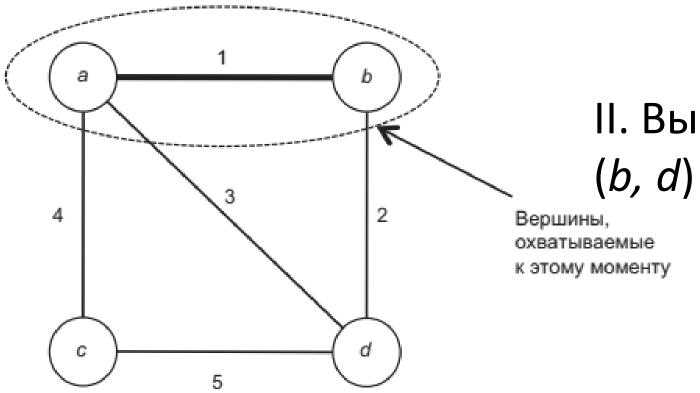
$$\sum c_l = 1 + 2 + 4 = 7$$

Алгоритм Прима

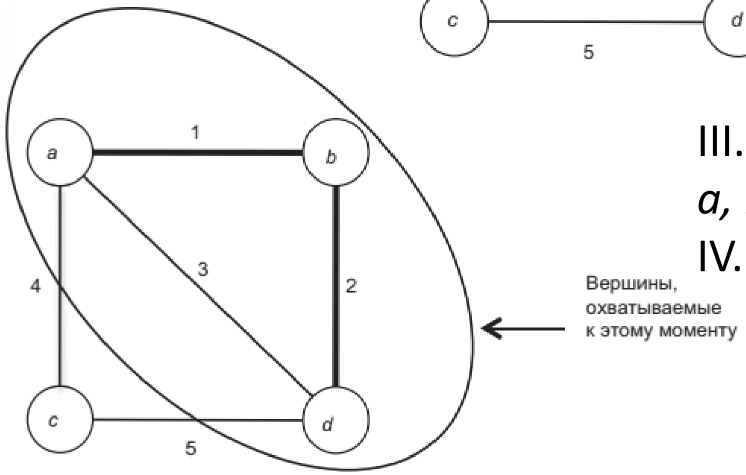




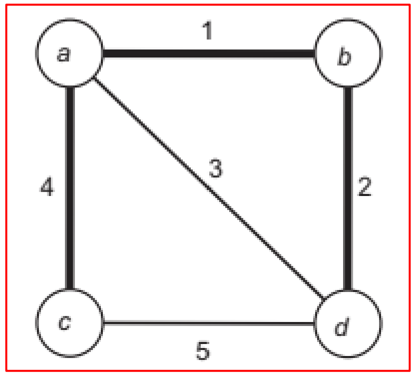
I.
1) Берем любую вершину, например, *b*. Добавляем *b* в остов.
2) Выбираем самое дешевое ребро, инцидентное b. (*a, b*)добавляем



II. Выбираем самое дешевое из ребер, инцидентные *a* и *b*.
(*b, d*) – добавляем.



III. Самое дешевое из не включённых ребер, инцидентных *a, b, d* – это (*a, d*) – циклическое – пропускаем.
IV. Следующее самое дешевое – это (*a, c*). Добавляем его.



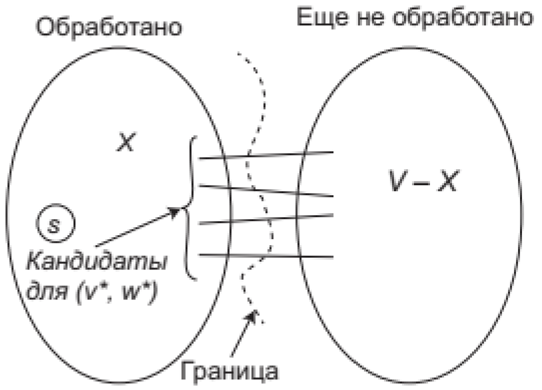
Псевдокод алгоритма Прима

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
```



Каждая итерация алгоритма Прима выбирает одно новое ребро, которое проходит из X в $V - X$

Каждая вершина из X – $|V|$ раз

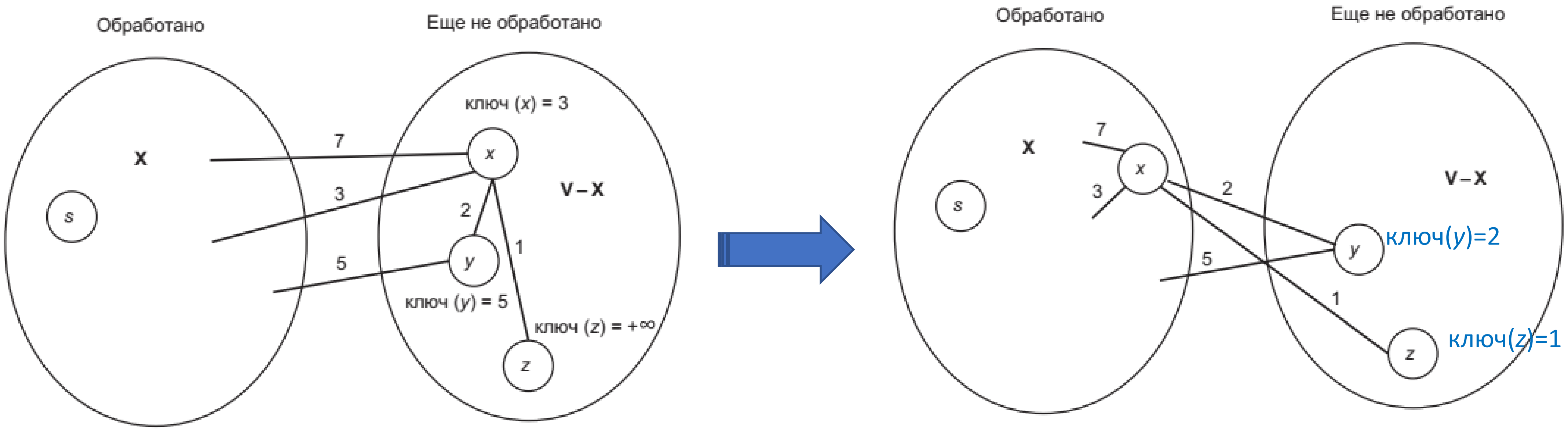
Исчерпывающий поиск по всем ребрам $|E|$ раз

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

Ускорение посредством двоичных куч

- Объекты в куче – еще необработанные вершины.

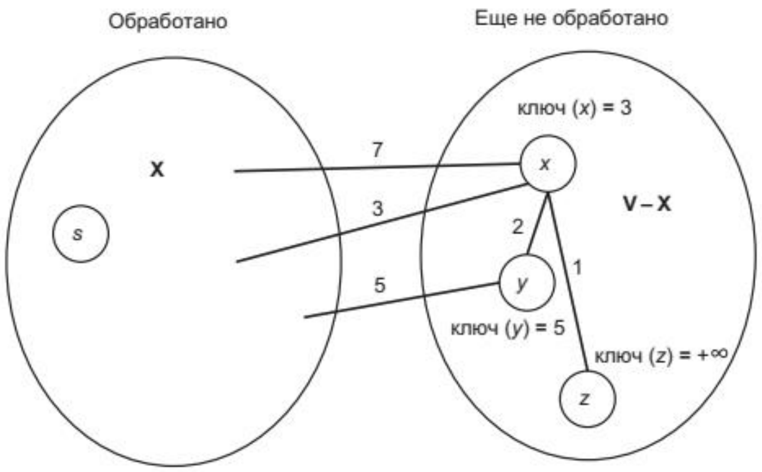
Ключ вершины $w \in V - X$ есть минимальная стоимость ребра (v, w), где $v \in X$ (либо $+\infty$, если такого ребра не существует).



Ключ вершины $w \in V - X$ определяется как минимальная стоимость ребра (v, w), где $v \in X$ (либо $+\infty$, если такого ребра не существует)

Ребра формы (v, x), где $v \in X$, всасываются в X и больше не пересекают границу (как и в случае с ребрами со стоимостями 3 и 7). Другие ребра, инцидентные x, (x, y) и (x, z), частично выдергиваются из $V - X$ и теперь пересекают границу. Как для y, так и для z эти новые инцидентные пересекающие ребра дешевле, чем все их старые. С целью поддержания инварианта оба их ключа должны быть обновлены: ключ y из 5 в 2, а ключ z из $+\infty$ в 1.

Псевдокод Прима на основе Кучи

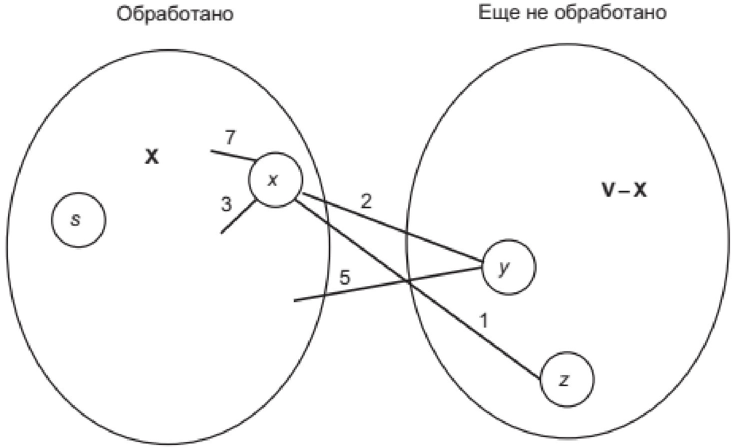


PRIM (НА ОСНОВЕ КУЧИ)

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
$X := \{s\}$, $T = \emptyset$, $H :=$ пустая куча , key(s):=0
for каждый $v \neq s$ **do**
    if существует ребро $(s, v) \in E$ **then**
        $key(v) := c_{sv}$, $winner(v) := (s, v)$ $O(|E|)$
    else // v не имеет пересекающих инцидентных ребер
        $key(v) := +\infty$, $winner(v) := NULL$
    Вставить v в H $O(|V|log|V|)$
// Главный цикл
while H не является пустым **do**
    $w^* :=$ Извлечь_минимум(H) $O(|V|log|V|) + O(|V|)$
    добавить w^* в X
    добавить $winner(w^*)$ в T $O(|E|log|V|) + O(|E|)$
    // обновить ключи для поддержки инварианта
    for каждое ребро (w^*, y) с $y \in V - X$ **do**
        if $c_{w^*y} < key(y)$ **then**
            Удалить y из H
            $key(y) := c_{w^*y}$, $winner(y) := (w^*, y)$
            Вставить y в H
return T



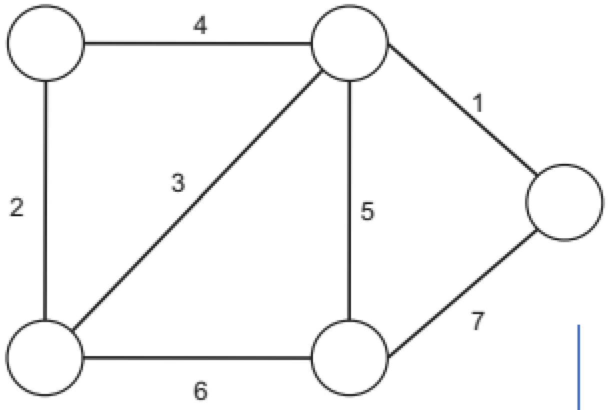
Всего $O((|V| + |E|)log|V|)$

Теорема: Для каждого связного графа G=(V, E) и вещественных реберных стоимостей алгоритм PRIM возвращает минимальное остовное дерево (MST). (Без доказательства, в к.р. необходимо привести)

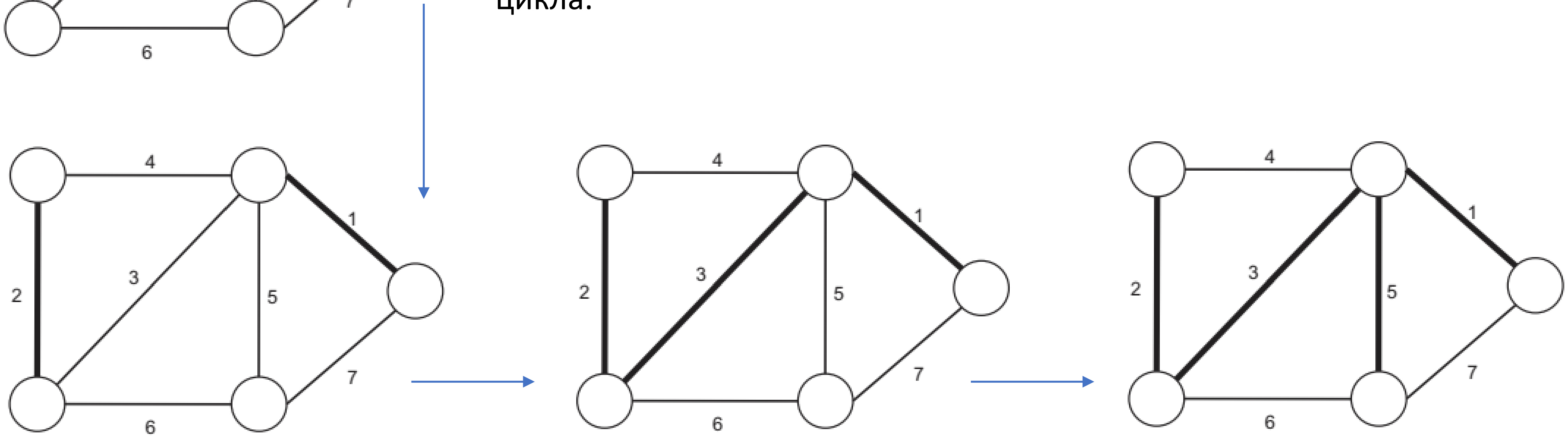
Алгоритм Крускала

Рассмотрим ребра в возрастающем порядке стоимостей.

1, 2, 3, 4, 5, 6, 7



Сначала $\mathrm{T}\ (MST) = \emptyset$
Каждый раз добавляем очередное ребро из списка, чтобы не было цикла.



4 ребра для связного графа с 5 вершинами. Все!

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 $v \rightarrow w \Rightarrow O(|V| + |E|)$. Если путь есть, то ребро e циклическое – не добавляем.

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

Оценку можно улучшить, если использовать структуру UNION_FIND

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

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

Теорема: Для каждого связного графа G=(V, E) и вещественных реберных стоимостей алгоритм Крускала возвращает минимальное остовное дерево (MST). (Без доказательства, в к.р. необходимо привести)