Минимальные остовные деревья

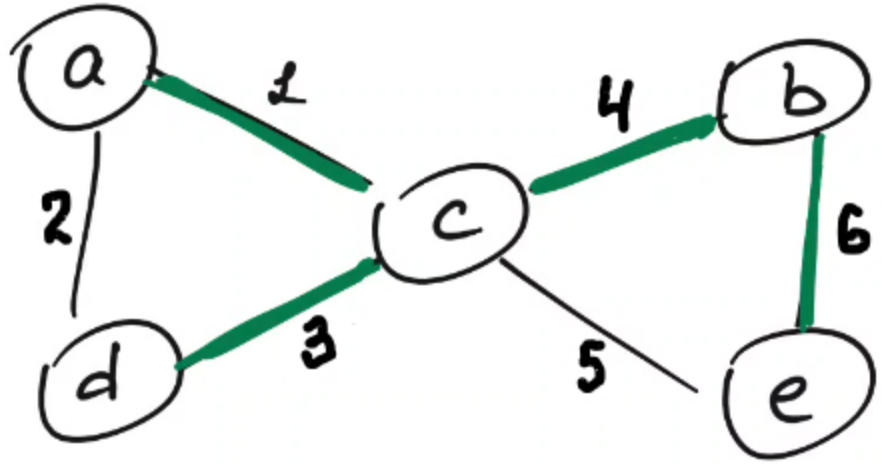
Minimum spanning trees

Определения

Дан неориентированный граф $G = (V, E)$ с весовой функцией на ребрах $w : E \to \mathbb{R}$

Опр. Если подграф $G' \subset G$ содержит все вершины графа G, то такой подграф называется *остовным*.

Опр. *Минимальное остовное дерево* - остовный подграф-дерево, который имеет наименьший вес (сумму весов ребер) среди всех остовных деревьев.

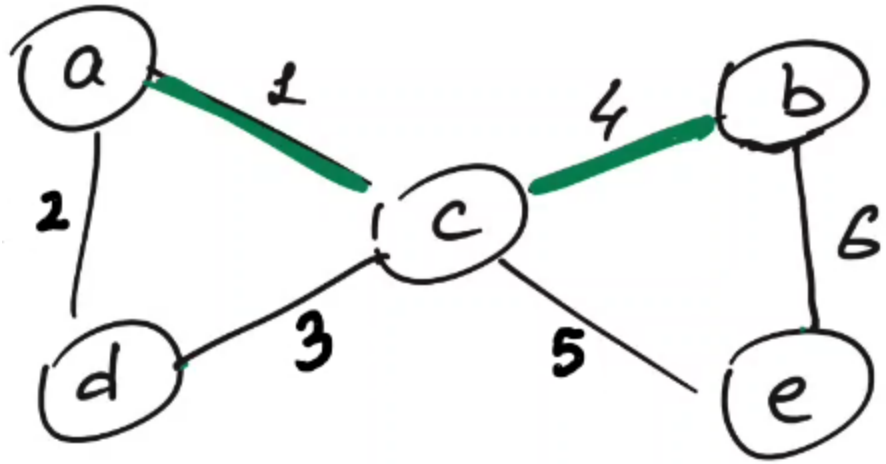


Зеленый подграф является остовным деревом, но не минимальным.

Определения

Пусть $G' = (V', E')$ - подграф некоторого MST T графа $G = (V, E)$.

Опр. Ребро $vu \in E \setminus E'$ называется *безопасным*, если при его добавлении в граф G' он останется подграфом некоторого (возможно другого) MST T' графа G.



Для зеленого подграфа безопасными являются ребра ad и ce.

Поиск MST: план

Будем стартовать с пустого графа, а затем последовательно добавлять безопасные ребра пока не построим минимальное остовное дерево.

Поиск MST: лемма о безопасном ребре

Опр. *Разрезом* графа называется разбиение вершин графа на два непересекающихся множества

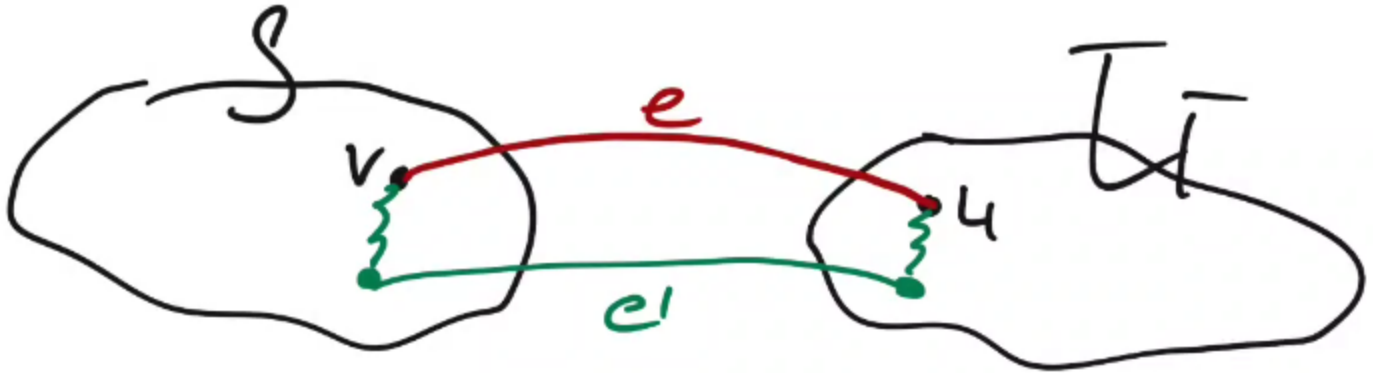
Опр. Ребро vu *пересекает* разрез (S, U), если $v \in S$, а $u \in U$.

Теорема (лемма о безопасном ребре).
Пусть G' - подграф некоторого MST T графа G, а (S, U) - разрез графа G такой, что ребра G' не пересекают его. Тогда ребро минимального веса среди пересекающих разрез будет безопасным для G'.

Теорема (лемма о безопасном ребре).
Пусть G' - подграф некоторого MST T графа G, а (S, U) - разрез графа G такой, что ребра G' не пересекают его. Тогда ребро минимального веса среди пересекающих разрез будет безопасным для G'.

Доказательство.

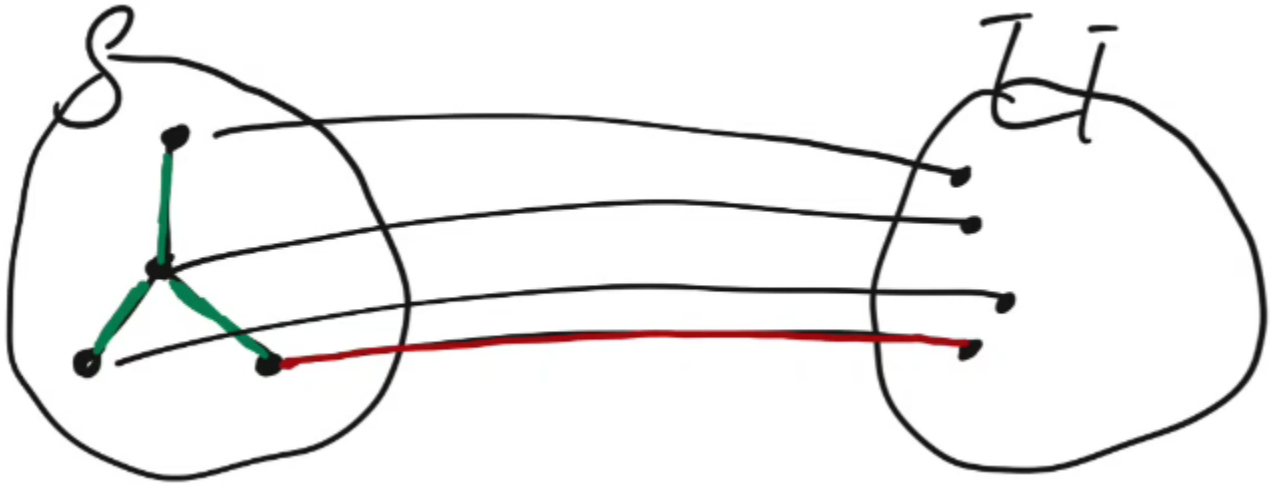
Пусть $e = (v, u)$ - минимальное ребро пересекающее разрез. Рассмотрим путь из v в u в дереве T. Он пересечет разрез по некоторому ребру e'. По условию $w(e) \leq w(e')$. Заменим ребро e' на e, связность графа сохранится. Так как T минимальное, то это значит, что $w(e) = w(e')$. То есть $e = e'$, либо e' спокойно можно заменить на e. В любом случае e принадлежит некоторому MST. ■



Алгоритм Прима

Алгоритм Прима (Prim, 1957)

0. Инициализируем разрез (S, U): $S = \{0\}, U = V \setminus \{0\}$.
1. Ищем минимальное ребро среди пересекающих разрез (S, U)
2. Добавляем ребро в MST, а конец ребра переносим из U в S.
3. Повторяем 1-2 пока $S \neq V$



Алгоритм Прима: реализация с пирамидой

```
def Prim(G):
  dist = [0, inf ..., inf]  # мин. вес ребра, ведущего в данную вершину
  prev = [None, ..., None]  # начало минимального ребра
  heap.insert(..., (dist[v], v), ...)  # Все вершины из U в пирамиде
  while heap is not empty:
    v = heap.ExtractMin()
    if prev[v] is not None:
      AddToMst(prev[v], v)
    for (v, u) in G.neighbors(v):
      if u in heap and w(v, u) < dist[u]:
        prev[u] = v
        dist[u] = w(v, u)
        heap.DecreaseKey(u, w(v, u))
```

При использовании бинарной пирамиды $O(E \log V)$

При использовании фибоначчиевой пирамиды $O(E + V \log V)$

Алгоритм Прима: реализация с массивом

```
def Prim(G):
  dist = [0, inf ..., inf]  # мин. вес ребра, ведущего в данную вершину
  prev = [None, ..., None]  # начало минимального ребра
  S = {}
  while |S| != |V|:
    v = argmin(dist)
    S.insert(v)
    dist[v] = inf  # вершина больше не понадобится
    if prev[v] is not None:
      AddToMst(prev[v], v)
    for (v, u) in G.neighbors(v):
      if u not in S and w(v, u) < dist[u]:
        prev[u] = v
        dist[u] = w(v, u)
```

Сложность: $O(E + V^2)$

Алгоритм Прима: сравнение реализаций

Граф/Алгоритм	Бинарная пирамида	Фибоначчиева пирамида	Массив
Разреженый ($E \sim V$)	$O(V \log V)$	$O(V \log V)$	$O(V^2)$
Плотный ($E \sim V^2$)	$O(V^2 \log V)$	$O(V^2)$	$O(V^2)$

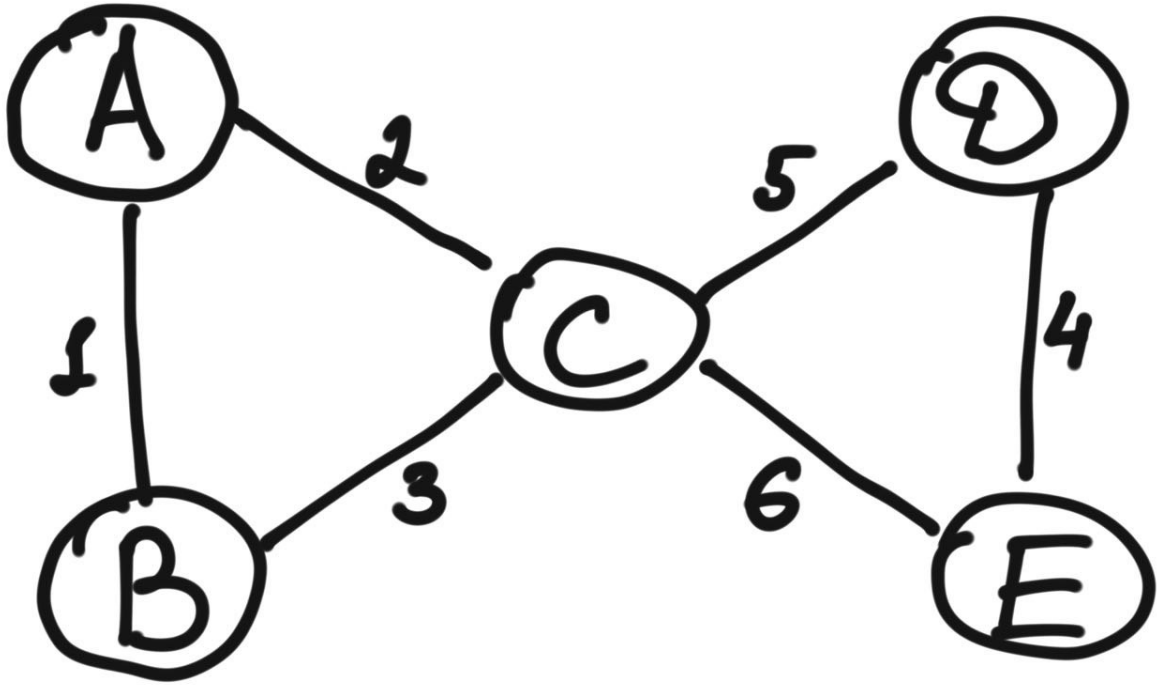
Вывод: для разреженных графов лучше выбирать бинарную пирамиду, для плотных - массив.

Фибоначчиева пирамида на практике никогда не лучше (большая константа).

Алгоритм Краскала

Алгоритм Краскала (Kruskal, 1956)

1. Сортируем ребра по весу. $O(E \log E) = O(E \log V)$
2. Последовательно проходимся по ребрам, если ребро не образует цикл (соединяет вершины из разных компонент), то добавляем в ответ. $O(E)$



Алгоритм Краскала

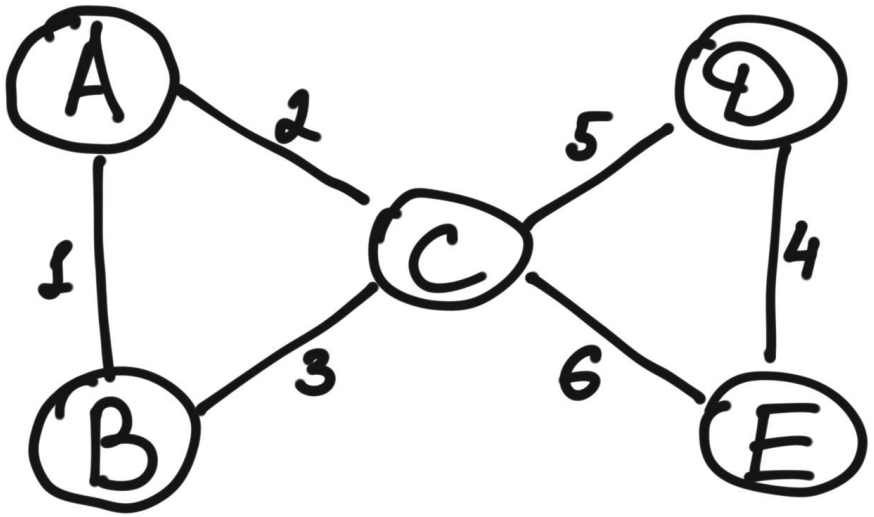
```
def Kruskal(G):
  sorted_edges = Sort(G.E, key=w)
  dsu = DSU(G.V)  # V одноэлементных множеств
  for (v, u) in sorted_edges:
    if (dsu.FindSet(v) != dsu.FindSet(u)):
      dsu.Union(v, u)
      AddToMst((v, u))
```

Формально, $O(E \log E)$, но $E \lesssim V^2$, поэтому $O(E \log V)$

Алгоритм Борувки

Алгоритм Борувки (Borůvka, 1926)

0. Инициализируем множество деревьев $T = \{\{0\}, \{1\}, ..., \{V-1\}\}$ (V одноэлементных дерева)
1. Для каждого дерева из T ищем минимальное ребро, связывающее это дерево с другим.
2. Добавляем найденные ребра, обновляем деревья.
3. Повторяем 1-2 пока $|T| > 1$



Алгоритм Борувки

```
def Boruvka(G):
  dsu = DSU(G.V)  # V одноэлементных множеств
  while dsu.SetCount() > 1:
    min_edge = [None, ... None]
    for (v, u) in G.E:
      component_v = dsu.FindSet(v)
      component_u = dsu.FindSet(u)
      if component_v != component_u:
        if w(v, u) < w(min_edge[component_v]):
          min_edge[component_v] = (v, u)
          if w(v, u) < w(min_edge[component_u]):
            min_edge[component_u] = (v, u)
    for (v, u) in min_edge:
      if (v, u) is None: continue
      AddToMst(v, u)
      dsu.Union(v, u)
```

Минимум среди ребер ищем суммарно за E, всего итераций $\leq \log V$, так как число компонент на каждом шаге уменьшается минимум вдвое. Итог $O(E \log V)$