# Основные алгоритмы, комментарии по десятому листку (Графы III. Остовные деревья)

Шибаев Иннокентий

April 18, 2021

## 1 По поводу теории

### 1.1 Задача построения минимального остовного дерева

Задача формулируется следующим образом – для взвешенного графа $G = (V, E)$ надо построить остовное дерево (т.е. дерево на всех вершинах графа с ребрами из $E$) с минимальным суммарным весом ребер.

К примеру на данную задачу можно смотреть как на задачу построения набора дорог между городами так чтобы из каждого города можно было добраться в каждый. При этом надо минимизировать затраты на построение дорог (в данном случае вес ребра будет стоимостью создания дороги).

#### 1.1.1 Алгоритм Краскала (Kruskal's algorithm)

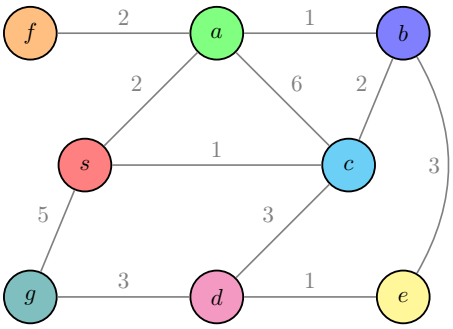
Идея этого алгоритма – в сливании леса (изначально состоящего из отдельных вершин графа) в остовное дерево. А именно, на каждом шаге алгоритм ищет ребро минимального веса, соединяющее различные компоненты и добавляет его в ответ (а так же записывает что вершины в этих компонентах теперь лежат в одной).

Алгоритм сводится к сортировке ребер по возрастанию весов, и их последовательной обработке. Единственный вопрос – как проверять что ребро соединяет разные компоненты.

Для начала рассмотрим простую реализацию – для каждой вершины будем поддерживать номер компоненты (номер наименьшей вершины в компоненте) которой она принадлежит. Изначально этот массив $C$ имеет вид $C[i] = i$ – каждая компонента состоит из своей вершины.

Каждый раз когда мы будем просматривать ребро мы будем смотреть на номера компонент (пусть $q$ и $w$, $q < w$) в которых лежат его концы – если номера $q$ и $w$ совпадают то это ребро надо отбросить. Иначе это ребро мы берем, и у всех вершин принадлежавших компоненте $w$ меняем номер компоненты на $q$ – это можно сделать линейным проходом по массиву $C$.

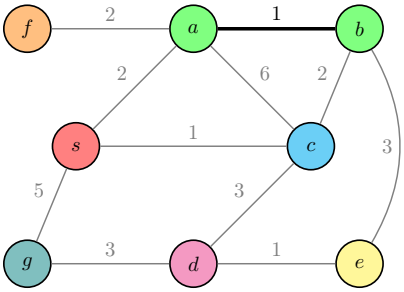
Рассмотрим пример работы этого алгоритма и затем обсудим его сложность.



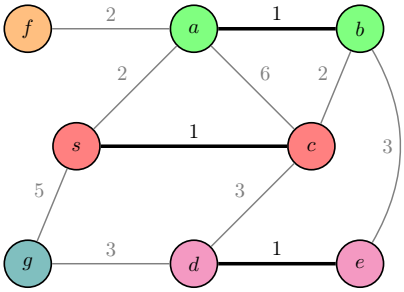

| **u** | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | ← Вершина |
|---|---|---|---|---|---|---|---|---|---|
| **C**[**u**] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |

Номер компоненты, в которой лежит вершина

В графе выше есть три ребра веса 1, они минимальны, поэтому алгоритм начинает работу с них. В начале рассмотрим ребро $(a, b)$, а затем ребра $(s, c)$ и $(d, e)$:

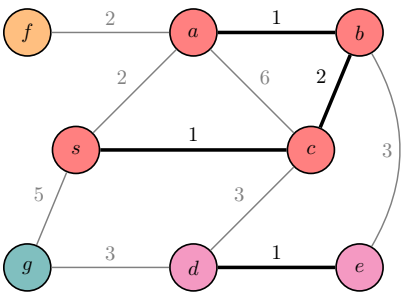



| **u** | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|---|
| **C**[**u**] | 0 | 1 | 1 | 3 | 4 | 5 | 6 | 7 |

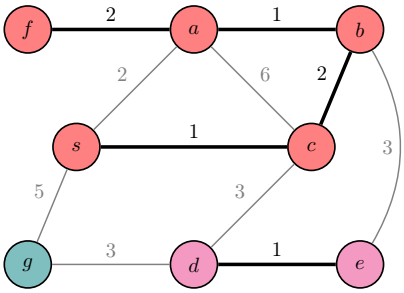



| **u** | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|---|
| **C**[**u**] | 0 | 1 | 1 | 0 | 4 | 4 | 6 | 7 |

Теперь ребра веса 1 обработаны, переходим к ребрам веса 2, рассмотрим их, допустим, в порядке $(b, c), (s, a), (a, f)$:

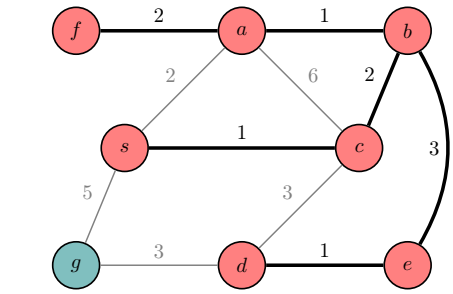



| **u** | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|---|
| **C**[**u**] | 0 | 0 | 0 | 0 | 4 | 4 | 6 | 7 |

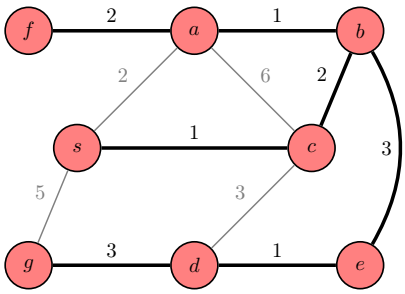



| **u** | $s$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|---|
| **C**[**u**] | 0 | 0 | 0 | 0 | 4 | 4 | 0 | 7 |

В момент когда мы обрабатывали ребро $(s, a)$ вершины $s$ и $a$ уже находились в одной компоненте, поэтому это ребро нам пришлось пропустить. Теперь надо просмотреть ребра длины 3, возьмем, к примеру, порядок $(b, e), (c, d), (d, g)$:




| u | s | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|
| C[u] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |




| u | s | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|
| C[u] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Всё, все вершины лежат в одной компоненте, далее просматривать ребра смысла уже нет. Таким образом мы построили минимальное остовное дерево для данного графа. Теперь надо обсудить сложность.

Мы просматриваем ребра в порядке возрастания их весов, так что нам нужно их отсортировать – в общем случае это мы умеем делать за $O(|E| \log |E|)$. Последнее кстати можно заменить на $O(|E| \log |V|)$ т.к. $|E| \leqslant |V|^2$.

Затем мы идем по ребрам, для каждого проверяем, лежат ли его концы в разных компонентах (обозначим сложность каждой такой проверки за $A$) и, возможно, сливаем компоненты (эту сложность обозначим за $B$). Понятно, что слияние происходит только когда мы реально добавляем ребро, и т.к. мы добавим не более $|V| - 1$ ребер можно написать что общая сложность есть

$$O(|E| \log |V| + A|E| + B|V|).$$

Теперь, для случая реализации описанного выше (массив где мы для каждой вершины пишем ее номер компоненты) сложность $A$ есть константа – нам просто надо посмотреть на то, равны ли числа в массиве $C$ для вершин $u$ и $v$. Операция же слияния описанная нами требует прохода по массиву вершин, т.е. $B = \Theta|V|$. Таким образом сложность в реализации которую мы описали есть

$$O(|E| \log |V| + |E| + |V|^2) \Rightarrow O(|E| \log |V| + |V|^2).$$

Теперь улучшим эту оценку, используя специальную структуру данных.

### 1.1.2 Система непересекающихся множеств (Disjoint-set union, DSU)

Эта структура поддерживает разбиение множества $X = \{x_1, \ldots, x_n\}$ на непересекающиеся попарно подмножества $\{s_1, \ldots, s_k\}$, и обрабатывает два типа запросов:

- $Find(x)$: Возвращает номер $i$ множества $s_i$ в котором сейчас лежит $x$;
- $Union(i, j)$: Объединяет два множества $s_i$ и $s_j$ в некоторое новое множество $s_k$.

Фактически, то что мы использовали выше это одна из возможных реализаций этой структуры данных, при этом сложность $A$ для ребра $(u, v)$ это фактически сложность двух проверок $Find(u)$ и $Find(v)$ и сравнения результатов этих проверок, а сложность $B$ это ровно сложность $Union(i, j)$.

В реализации с массивом, таким образом, мы получили $Find(u) = O(1)$ и $Union(i, j) = O(n)$. Теперь опишем улучшенный вариант реализации этой структуры с использованием деревьев.

Каждое подмножество будем хранить как дерево. Корень дерева (т.н. представитель множества) и будет определять номер подмножества. Т.е. чтобы

определить какому подмножеству принадлежит элемент $x$ надо переходить в родительскую вершину пока мы не дойдем до корня – это и будет ответ.

Слияние деревьев будем осуществлять очень простым образом – у одного из представителей выберем родителем другого.

На данный момент эта структура не очень хороша – деревья никак не балансируются, поэтому к примеру в случае когда большее множество присоединяется как потомок представителя меньшего мы можем получить цепь (допустим эл-т 1 присоединен к 2, сливаем $2-1$ и 3, получаем $3-2-1$ и т.д.)

Оказывается, чтобы этого не допустить достаточно применить одну простую эвристику:

**Эвристика 1.1.** При объединении множеств $s_i$ и $s_j$ надо подвешивать меньшее множество к большему (по кол-ву элементов, или по высоте дерева).

При этом получается что деревья высоты $h$ будут содержать порядка $2^h$ элементов, и в результате мы получим $Find(u) = O(\log n)$ и $Union(i,j) = O(1)$ (т.к. достаточно только заменить предка у одной из этих вершин $i$ или $j$).

На этом можно было бы остановиться, т.к. с точки зрения сложности для алгоритма Краскала мы уже ничего выжать не сможем – сортировка уже будет все мажорировать. Но стоит упомянуть и о другой эвристике, которая делает операции еще более быстрыми:

**Эвристика 1.2.** Эвристика сжатия путей При вызове $Find(x)$ переподвешивать все встречаемые вершины к родительской напрямую. Т.е., к примеру, для случая цепочки $s \to u \to v \to x$ у вершин $v$ и $x$ родительской надо так же выставить вершину $s$.

Оказывается что применение обеих этих эвристик делает скорость выполнения почти константной. Точнее говоря, $Find(x) = O(\alpha(x))$ или $Find(x) = O(\log^* x)$, где $\alpha(x)$ – обратная функция Аккермана (обычная функция Аккермана $A(n,m)$ это **очень** быстрорастущая функция, обратная к ней задается как $\alpha x = \min\limits_n\{A(n,n) \geqslant x\}$, и в приложениях можно считать $\alpha(x) \leqslant 4$), а $\log^* x$ – т.н. итерированный логарифм, по сути это то сколько раз надо применить логарифм чтобы превратить $x$ в 1, формально

$$\log^* x = \begin{cases} 0, & x \leqslant 1 \\ 1 + \log^*(\log x), & x > 1 \end{cases}$$

и это тоже очень медленно растущая функция. Кстати вроде как верно

$$\alpha(x) = \min_i\{\log^{\overbrace{*\cdots*}^{i-3}}(x) \leqslant i\}$$

где много звездочек определяется так же как и выше рекурсивно, и на этом мы остановимся.

Главное что надо из этого вынести – эти операции крайне эффективны, и довольно легко реализуются. И если использовать эту структуру данных для реализации алгоритма Краскала то сложность будет

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

### 1.1.3 Алгоритм Прима (Prim's algorithm)

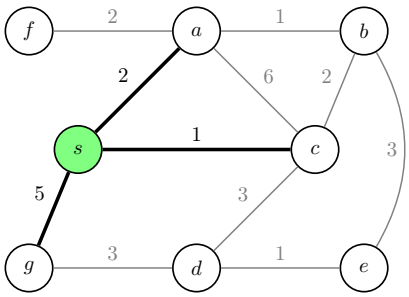
Это второй алгоритм для построения минимального остовного дерева. Идейно он заключается в постепенном наращивании остова – на каждом шаге работы мы присоединяем вершину с минимальным расстоянием до нее от вершин остова. Есть так же другой вариант этого алгоритма, который полностью повторяет алгоритм Дейкстры, с тем лишь отличием, что вместо обновлений по сумме весов используется

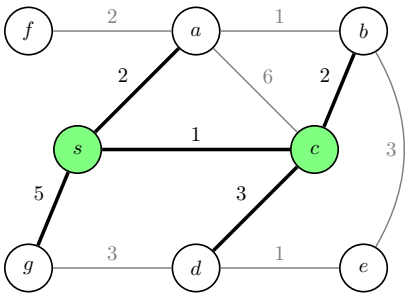
обновление минимального веса – при обработке вершины $u$ и просмотре ребра $(u, v)$ мы делаем следующее изменение относительно алгоритм Дейкстры:

$$d[v] = \min\{d[v], d[u] + w(u, v)\} \to d[v] = \min\{d[v], w(u, v)\}$$

Эти две интерпретации фактически задают один и тот же алгоритм – вершина которую мы заканчиваем обрабатывать в модифицированном алгоритме Дейкстры это как раз вершины с минимальным от нее расстоянием до остова (потому что мы, обработав все вершины остова, посмотрели все ребра из остова в эту вершину, и при этом мы стали обрабатывать ее следующей – т.е. среди не обработанных ее $d[u]$ минимально).

Опять же посмотрим на примере, будем рассматривать вариант с реализацией через алгоритм Дейкстры, и на каждом шаге сравнивать его с алгоритмом основанном на выборе ближайшей к остову вершины:

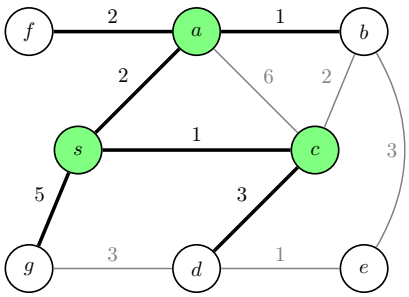


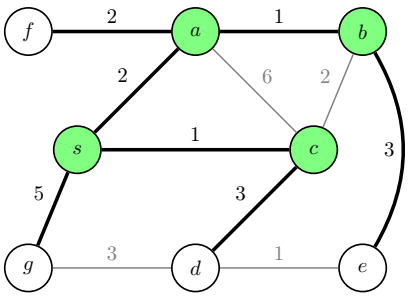




| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d[u]** | – | 2 | ∞ | 1 | ∞ | ∞ | ∞ | 5 |
| **p[u]** | – | *s* | | *s* | | | | *s* |

| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d[u]** | – | 2 | 2 | 1 | 3 | ∞ | ∞ | 5 |
| **p[u]** | – | *s* | *c* | *s* | *c* | | | *s* |

Мы начали с вершины $s$, обработали ее соседей, и перешли к необработанной вершине с минимальным до нее расстоянием – к ввершине $c$. Эта вершина так же является и ближайшей к уже взятому остову, так что пока варианты алгоритма совпадают. Длина ребра которое сейчас ведет из остова в $a$ уже равна 2, так что переподвешивание к $c$ не происходит $(6 > 2)$. Далее мы будем обрабатывать вершину $a$ (можно $b$, расстояние до них сейчас одинаково, но начнем с $a$):



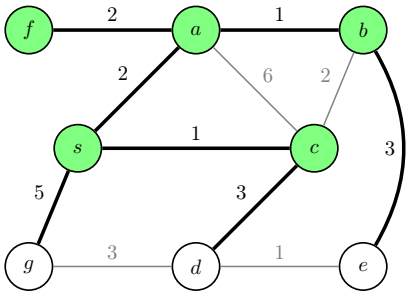

| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d[u]** | – | 2 | 1 | 1 | 3 | ∞ | 2 | 5 |
| **p[u]** | – | *s* | *a* | *s* | *c* | | *a* | *s* |

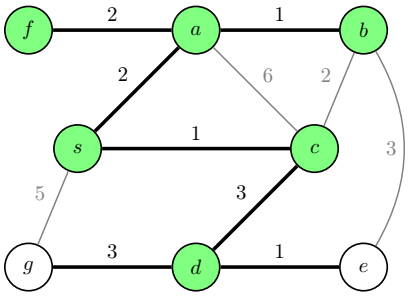
| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d[u]** | – | 2 | 1 | 1 | 3 | 3 | 2 | 5 |
| **p[u]** | – | *s* | *a* | *s* | *c* | *b* | *a* | *s* |

Заметим, что вершина $b$ обрабатывалась после вершины $a$, но ребро, соединяющее ее с будущим остовом имеет вес 1, меньше чем ребро соединяющее $a$ с $s$. В алгоритме

Дейкстры такое произойти не могло – расстояния до рассматриваемых на каждом шаге вершин монотонно не убывали. Еще можно заметить что мы, фактически, подвешиваем каждую вершину за самое легкое ребро из нее исходящее. Это не совсем так, но верно то, что для любой вершины самое легкое ребро входящее в нее будет в остовном дереве (частный случай леммы о разрезе, в которой говорится что для любого разреза в остовном дереве будет самое легкое ребро разреза).

Далее мы должны обработать вершину $f$ (тривиальный случай, так как других ребер из нее не выходит), и затем перейти либо к вершине $d$, либо к $e$, пусть к первой:

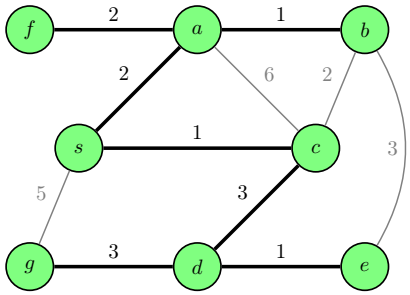



| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d**[**u**] | – | 2 | 1 | 1 | 3 | 3 | 2 | 5 |
| **p**[**u**] | – | *s* | *a* | *s* | *c* | *b* | *a* | *s* |

| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d**[**u**] | – | 2 | 1 | 1 | 3 | 1 | 2 | 3 |
| **p**[**u**] | – | *s* | *a* | *s* | *c* | *d* | *a* | *d* |

Остались только вершины $e$ и $g$, и после их обработки мы получаем




| **u** | *s* | *a* | *b* | *c* | *d* | *e* | *f* | *g* |
|---|---|---|---|---|---|---|---|---|
| **d**[**u**] | – | 2 | 1 | 1 | 3 | 1 | 2 | 3 |
| **p**[**u**] | – | *s* | *a* | *s* | *c* | *d* | *a* | *d* |

Оба алгоритма построили остовные деревья веса 13, тем не менее результаты существенно отличаются.

Теперь кратко обсудим сложность. Если использовать реализацию алгоритма Дейкстры, то наше изменение не изменит сложность, так что получаем $O(|E|\log|V|)$ или $O(|V|^2)$, в зависимости от реализации поиска вершины с минимальным $d[u]$.

Если реализовывать этот алгоритм как поиск вершины не входящей в компоненту связности расстояние до которой минимально, то мы можем просто при обработке каждой вершины добавлять ребра в очередь с приоритетами (по весу), и тогда опять же получим $O(|E|\log|V|)$. Или мы можем рассмотреть поиск подходящей вершины по массиву расстояний до них, тогда сложность будет $O(|V|^2)$.

# 2 По поводу задач

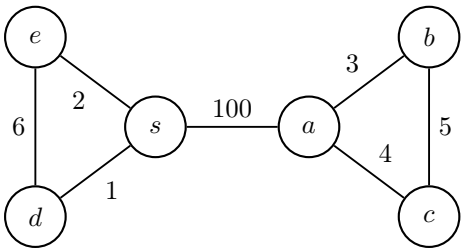
**Задача 2.1** (Задача 4 из листка 10)**.**

Дан неориентированный граф $G = (V, E)$, веса рёбер которого не обязательно различны. Для каждого из утверждений ниже приведите доказательство, если оно истинно, или постройте контрпример,если оно ложно:

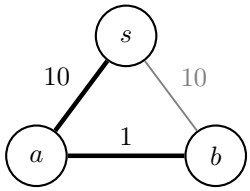
- Если в $G$ больше $|V| - 1$ ребра и самое тяжёлое ребро уникально, то это ребро не может быть частью минимального остовного дерева.
- Если в $G$ есть цикл с уникальным самым тяжёлым ребром $e$, то $e$ не может быть частью минимального остовного дерева.
- Дерево кратчайших путей, которое выдаёт алгоритм Дейкстры, является минимальным остовным деревом.
- Алгоритм Прима корректен даже при наличии в графе рёбер отрицательного веса.
- Если уменьшить вес одного ребра,входящего в минимальное остовное дерево $T$, то $T$ останется минимальным остовным деревом.

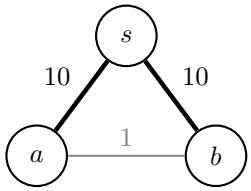
**Решение.** На семинаре мы успели разобрать часть пунктов, но на всякий случай пройдусь с начала.

- Неверно. Такое может быть если это ребро – мост, например




- Да, верно. Можно рассмотреть разрез графа в мин. остове в который предположительно входит самое тяжелое ребро – в этом разрезе будет ребро из цикла, мы можем заменить на него.
- Нет, к примеру сравните мин. остов (слева) и дерево Дейкстры (из вершины $s$, справа) на таком графе:




- Да, верно, это обсуждалось на семинаре. Более того, можно добавить любую константу ко всем весам, на результат это не повлияет. С точки зрения реализации через замену условия в алгоритме Дейкстры это происходит из-за того что мы сравниваем теперь всегда только веса ребер между собой, т.е. фактически мы делаем замену вида

$$w(u_1, v) \overset{?}{<} w(u_2, v) \Rightarrow w(u_1, v) + W \overset{?}{<} w(u_2, v) + W$$

которая, очевидно, ничего нам не меняет.

- Да, верно. Действительно, дерево $T$ было минимальным остовным, т.е. $w(T) \leqslant w(T'), \forall T'$, а теперь имеем

$$w(T) - w_0 \stackrel{?}{\leqslant} w(T') - \delta(e, T')w_0$$

где $\delta(e, T') = 1$ если $e \in T'$, и равно 0 иначе. Т.е. неравенство выше выполняется, и $T$ все еще минимальное остовное дерево.

□