Обход в глубину. Деревья. Задача о минимальном остовном дереве. Алгоритмы Краскала, Прима-Ярника. Алгоритм Дейкстры.

Руслан Назирович Мокаев

Математико-механический факультет,
Санкт-Петербургский государственный университет

Санкт-Петербург, 09.04.2024

Содержание лекции

- Неориентированные графы.
- Обход графа в глубину. Поиск пути. Дерево кратчайших путей.
- Дерево и лес. Теорема об эквивалентных определениях дерева.
- Задача о минимальном остовном дереве. Лемма о безопасном ребре.
- Алгоритмы Краскала, Прима-Ярника, их обоснования.
- Дерево кратчайших расстояний. Алгоритм Дейкстры.

Напоминалка

Обозначим $2_V := \{\{u, v\} : u, v \in V\}$

Определение: **Неориентированным графом** называют $G = (V, E)$, где $V \neq \emptyset$ – множество вершин, $E \subseteq 2_V$ – множество ребер.

Пути, простые пути, циклы, простые циклы, подграфы и порожденные подграфы определяются так же, как для орграфов, с поправкой на замену $V \times V$ на 2_V.

Определение: Вершины $v_1, v_2 \in V$ называются **смежными**, если $v_1 v_2 \in E$.

Определение: Вершины v_1 и v_2 графа G называются **связанными**, если в графе существует путь между ними. Граф называется **связным**, если любые две его вершины связаны. Очевидно, связанность вершин – отношение эквивалентности, и все вершины графа по этому отношению разбиваются на классы эквивалентности – множества попарно связанных вершин. Эти классы эквивалентности мы будем называть **компонентами связности** графа G. Будем называть **компонентами** графа G подграфы, индуцированные на его компонентах связности.

Замечание: Компонента связности – это максимальное по включению связное множество вершин графа. Часто под компонентами связности графа G подразумевают максимальные связные подграфы этого графа, которые мы называем просто компонентами.

Обход в глубину

1. хранится список (массив или словарь) непосещенных и посещенных вершин, изначально все вершины непосещены;
2. отмечаем некоторую начальную вершину v;
3. запускаем алгоритм из вершины v:
   - 3.1 помечаем v как посещенную;
   - 3.2 для каждой непосещенной вершины u такой, что есть ребро vu, запускаем рекурсивно алгоритм из u;
4. повторяем шаг 3 пока не закончатся вершины.

Используется для решение большого числа задач на графах, например:

- проверка неорграфа на связность и поиск компонент связности;
- проверка орграфа на сильную связность и поиск компонент сильной связности;
- проверка наличия пути между вершинами u и v и его поиск;
- проверка графа на ацикличность и поиск цикла;
- топологическая сортировка орграфа (такой порядок вершин $V(G)$, задаваемый в виде функции $\varphi : V(G) \to \overline{1 : |V(G)|}$, что $\forall\ (u, v) \in E(G)$ выполняется $\varphi(u) < \varphi(v)$).

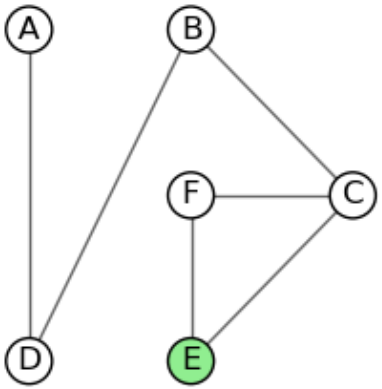
Пример: поиск пути обходом в глубину

Дан граф $G = (V, E)$ и две вершины $s, t \in V$. Существует путь из s в t?

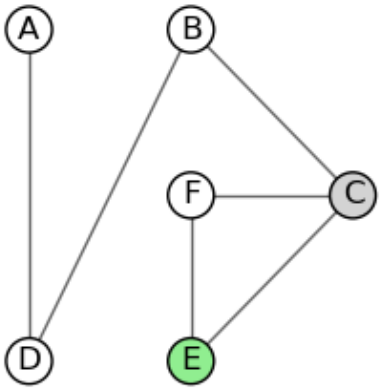
1. храним список непосещенных и посещенных вершин, изначально все вершины непосещены;
2. запускаем алгоритм из вершины s:
3. на очередной итерации вызываем поиск из текущей вершины v и передаем в вызове информацию о посещенных вершинах:
   - 3.1 проверяем является ли текущая вершина t; если да, то путь существует;
   - 3.2 отмечаем v как посещенную;
   - 3.3 для каждой непосещенной вершины u такой, что есть ребро vu, запускаем рекурсивно алгоритм из u;
4. повторяем шаг 3 пока не закончатся вершины.

Если путь существует, то в какой-то момент обход прекратится с положительным ответом.

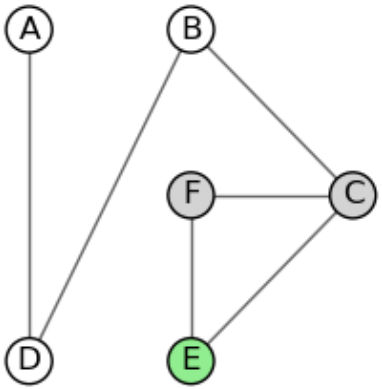
Для получения самого пути необходимо для каждой вершины в обходе $v \neq s$ хранить информацию (список/массив) о вершине, из которой во время обхода пришли в v: для ребра на пути (u, v) будет верно $p[v] = u$. Тогда искомый путь будет $(s, \ldots, p[p[t]], p[t], t)$.



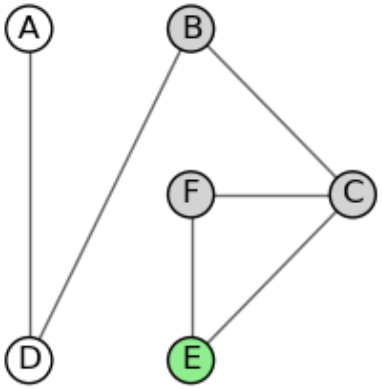
Текущая вершина - E



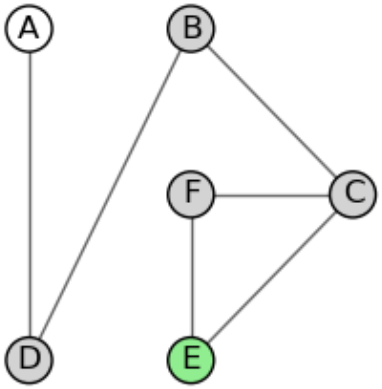
Текущая вершина - C



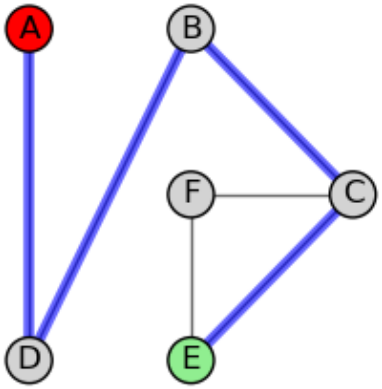
Текущая вершина - F



Текущая вершина - B



Текущая вершина - D

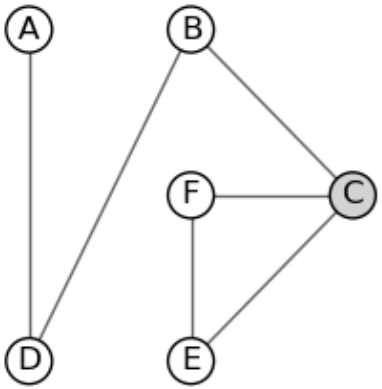


Текущая вершина - A

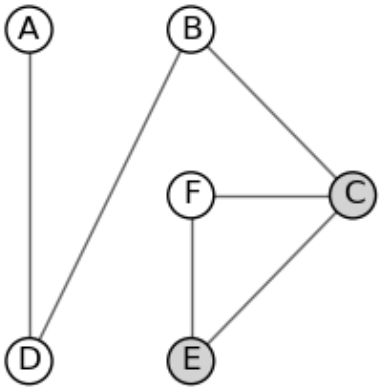
Пример: проверка графа на связность

Выполняется обходом в глубину с небольшой модификацией: каждая запущенная итерация обхода возвращает количество посещенных вершин.

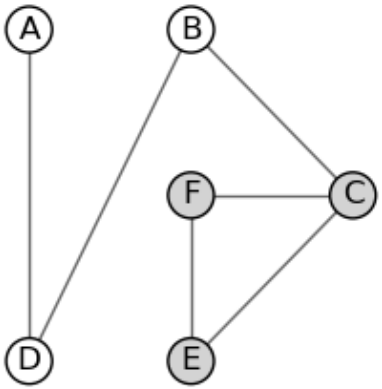
1. храним список непосещенных и посещенных вершин, изначально все вершины непосещены;
2. отмечаем некоторую начальную вершину v;
3. запускаем алгоритм из вершины v с информацией о посещении вершин:
   - 3.1 помечаем v как посещенную; инициализируем количество посещенных в данной итерации вершин единицей;
   - 3.2 для каждой непосещенной вершины u такой, что есть ребро vu, запускаем рекурсивно алгоритм из u; и складываем результаты;
   - 3.3 после рекурсивного вызова из всех u возвращаем количество посещенных вершин;
4. Если результат запуска из вершины v равен количеству вершин в графе, то граф связный!



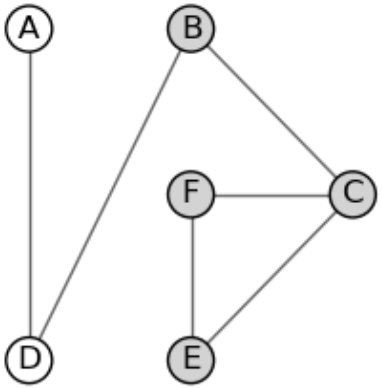
Текущая вершина - C



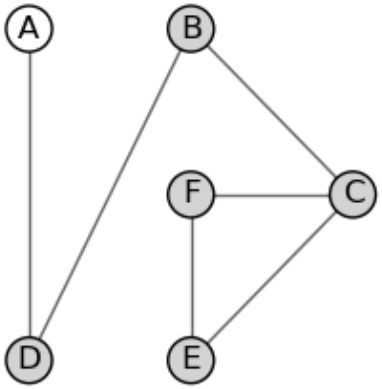
Текущая вершина - E



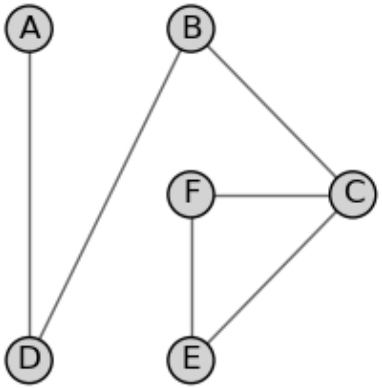
Текущая вершина - F



Текущая вершина - B



Текущая вершина - D



Текущая вершина - A

Дерево и лес

Определение: Связный граф без циклом называется **деревом**. Произвольный граф без циклов называют **лесом**.

Теорема (эквивалентные определения):

1. G – дерево.
2. G – граф, в котором $\forall\ v_1, v_2 \in V$ существует единственный простой путь между ними.
3. G – связный граф, в котором $|V| = |E| + 1$.
4. G – граф без циклов, в котором $|V| = |E| + 1$.
5. G – граф без циклов, в котором при добавлении ребра между несмежными вершинами образуется один простой цикл.

Док-во:

$1 \Rightarrow 2$: из связности следует существование пути между любыми двумя вершинами. Такой путь единственный, иначе нарушится ацикличность. Если это единственный путь непростой, то значит путь заходит в одну из вершин дважды, что противоречит ацикличности.

$2 \Rightarrow 3$: Из существования простых путей между любыми двумя вершинами следует связность. Докажем по индукции по количеству вершин.
База для $|V(G)| = 2$ очевидна. **Переход**: пусть верно для графов с менее, чем k вершинами. Пусть в графе ровно k вершин. Удалим одно ребро.

Тогда нарушится связность и образуется ровно две компоненты с k_1 и $k-k_1$ (если компонент больше, то между каким-то вершинами не существовало простого пути). Для обеих компонент справедливо предположение индукции. Тогда в исходном графе ребер $(k_1 - 1) + (k - k_1 - 1) + 1 = k - 1$.

$3 \Rightarrow 4$: Если граф связен и ребер на одно меньше, чем вершин, то в нем нет циклов. Если добавлять ребра в граф по одному и в какой-то момент образовался цикл, то в цикле количество вершин и ребер совпадает, поэтому ребер останется меньше, чем вершин. Поэтому какую-то вершину не удастся "присоединить" к графу. Значит циклов нет.

$4 \Rightarrow 5$: Пусть в графе G ровно k компонент связности. Каждая компонента это связный граф и, к тому же, ацикличный (по условию). Значит каждая компонента – дерево и справедливо $|V_i| = |E_i| + 1, i \in 1 : k$. Сложим по всем компонентам и получим $|V| = |E| + k$. По условию $|V| = |E| + 1$, значит есть только одна компонента связности, то есть граф связный. Значит исходный граф является деревом и между любыми двумя вершинами есть единственный простой путь. Значит при добавлении одного ребра образуется ровно один простой цикл.

$5 \Rightarrow 1$: Необходимо показать связность. Если при добавлении в G ребра между (u, v) произвольными вершинами u и v появляется один простой цикл, то следовательно в графе G между u и v существует простой путь. Отсюда следует связность.

Остовные деревья

Определение: Подграф $H \leq G$ называют **остовом** G, если $V(H) = V(G)$.

Утверждение: У каждого графа существует остовный лес (а у связного графа – остовное дерево).

Док-во: Покажем для связного графа. Если в графе есть цикл, то можно удалить из этого цикла ребро. Граф, очевидно, останется связным. Продолжим такие действия до тех пор, пока циклы не исчезнут (с каждым шагом уменьшается количество рёбер, изначально оно конечно). В результате мы получим связный граф без циклов, являющийся остовным подграфом исходного графа, то есть, остовное дерево этого графа.

Утверждение: $G = (V, E)$ – связный граф, то существует нумерация вершин $v_1, v_2, \ldots, v_n : \forall\ i \in \overline{1:n}\ G_i = G[\{v_1, \ldots, v_i\}]$ – связный.

Док-во: База: граф из одной вершины связен по определению. Переход: пусть $G_1, \ldots, G_i$ построены и связны. Пусть $v \in V \setminus \{v_1, \ldots, v_i\}$ (если разность пуста, то $i = n$ и утверждение уже доказано). В силу связности G существует путь $x_0 x_1 \ldots x_m$, где $x_0 = v, x_m = v_1$.
$s := \max\left\{j \in \overline{0:(m-1)}\ :\ x_j \notin \{v_1, \ldots, v_i\}\right\}$.
Тогда положим $v_{i+1} = x_s$. Очевидно, что G_{i+1} связный. $\square$

Утверждение: Пусть T – дерево. Тогда $\exists$ такая нумерация вершин $V(T) = \{v_1, \ldots, v_n\}$, что $\forall\ i \in \overline{1:n}\ \exists!\ j \in \overline{1:(i-1)}$ такое, что $v_i v_j \in E(T_i)$, где $T_i = T[\{v_1, \ldots, v_i\}]$.

Док-во: Так как T связный, то существует такая нумерация, что все T_i связные. В частности, т.к. T_i связный, то существует $\geq$ одна вершина в $\{v_1, \ldots, v_{i-1}\}$, смежная с v_i. Если есть две вершины в $\{v_1, \ldots, v_{i-1}\}$, смежные с v_i, то т.к. T_{i-1} связный $\Rightarrow$ цикл в T (?!).
Значит такая вершина ровно одна.

Утверждение: Связный граф на n вершинах дерево $\Leftrightarrow$ в нем $n-1$ ребро.

Док-во: $\Rightarrow$) : следует из нумерации
$\Leftarrow$) : граф G : $|V(G)| = n, |E(G)| = n-1$. Т.к. граф связный, то он имеет остовное дерево T : $|V(T)| = n, |E(T)| = n-1 \Rightarrow G = T \Rightarrow G$ – дерево.

Минимальное остовное дерево

Лемма о безопасном ребре

$w: E(G) \to \mathbb{R}_+$ – веса ребер. Вес графа $w_G = \sum_{e \in E} w(e)$

Определение: Минимальным остовным деревом называется остовное дерево с минимальным весом.

Лемма: Пусть $\mathcal{T}$ – множество всех минимальных остовных деревьев связного графа G. $T \in \mathcal{T}, X \subset E(T)$.
Пусть $\varnothing \neq S \subseteq V(G), Q = \{uv : u \in S, v \in V(G) \setminus S\}$, причем $X \cap Q = \varnothing$.
Выберем $e \in Q: \ w(e) = \min_{q \in Q} w(q)$. Тогда $\exists T' \in \mathcal{T}: \ X \cup \{e\} \subseteq E(T')$.

Док-во: Если $e \in E(T)$, $T' = T$. Иначе $T + e$ содержит цикл (т.к. дерево – максимальный граф без циклов).
Тогда $\exists e' \in E(T): \ e' \in Q$ и $e' \notin X$, т.к. $X \cap Q = \varnothing$.
$T' = T + e - e'$ – остовное дерево (вершины мы не удаляли, связность не нарушалась, так как удалили ребро из цикла).
Т.к. $w(e) = \min_{q \in Q} w(q)$, то $w(e) \leq w(e')$ и
$w(T') = w(T) + w(e) - w(e') \leq w(T) \Rightarrow$ т.к. T – минимальное остовное дерево, то T' – тоже минимальное остовное дерево. $\square$

Замечания: такое ребро называется **безопасным**. Разбиение $V(G)$ на два множества $S \subseteq V(G)$ и $V(G) \setminus S$ называется **разрезом**. Множ-во Q из леммы – множество ребер, **пересекающих** разрез $\langle S, V \setminus S \rangle$.

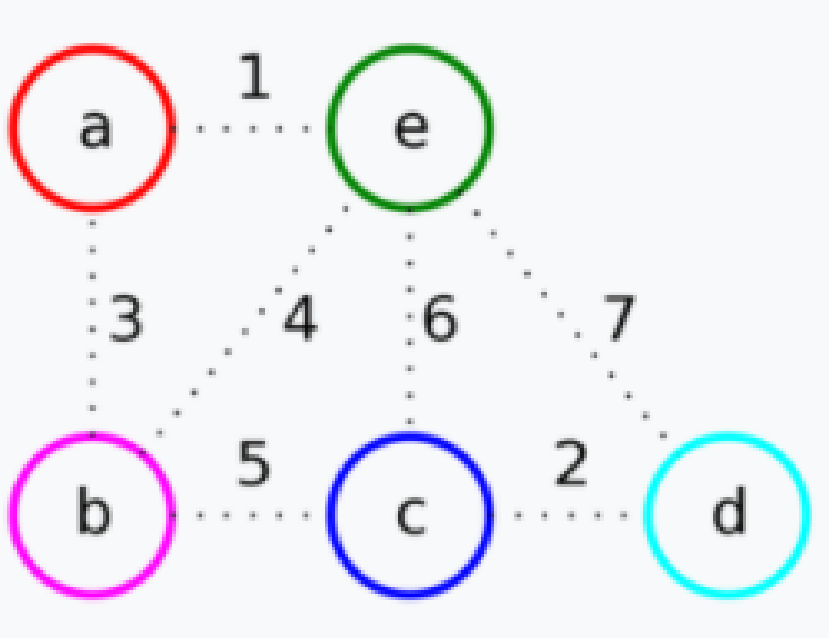
Алгоритм Краскала

Итеративно строим $F \leq G$, на каждом шаге достраиваем F до МОД.

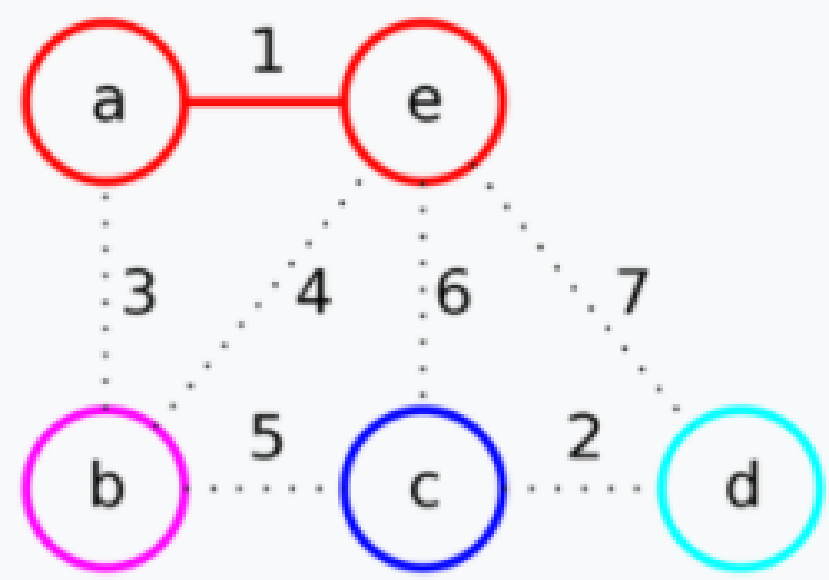
Алгоритм (Краскала) и обоснование:

1. Добавим все вершины G в F: $V(F) = V(G)$.
2. Обходим ребра $E(G)$ в порядке неубывания весов ребер:
   - Если у ребра e вершины из одной компоненты связности графа F, то добавление его в остов приведет к возникновению цикла в этой компоненте связности. Не включаем в F.
   - e соединяет вершины из разных компонент связности F. Значит существует разрез $\langle S, V(F) \setminus S \rangle$: одна из компонент связности составляет одну его часть, а оставшаяся часть графа – вторую. Получается, что e – минимальное ребро, пересекающее этот разрез $\Rightarrow$ по лемме e – безопасное и его можно добавить в F.
   - На последнем шаге ребро соединит две оставшиеся компоненты связности и полученный подграф будет минимальным остовным деревом графа G.

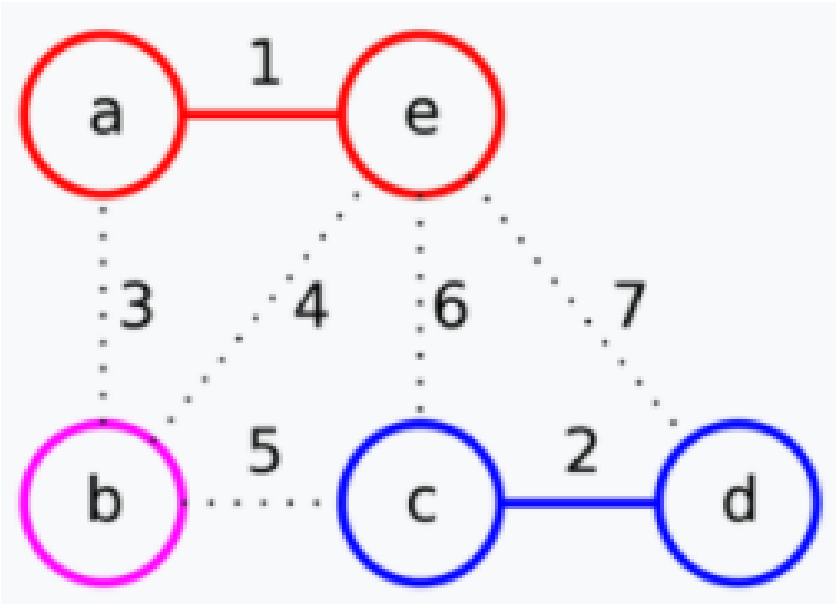
Пример работы алгоритма Краскала



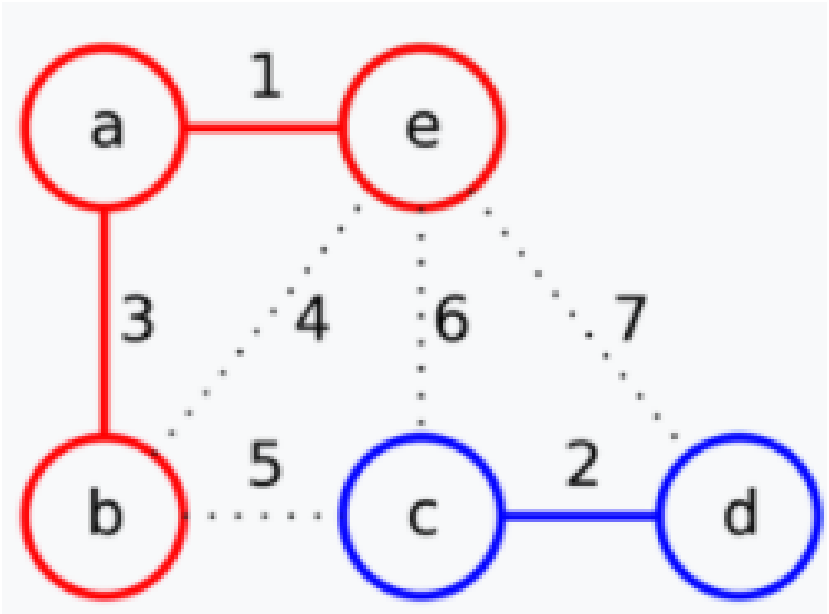
Пример работы алгоритма Краскала



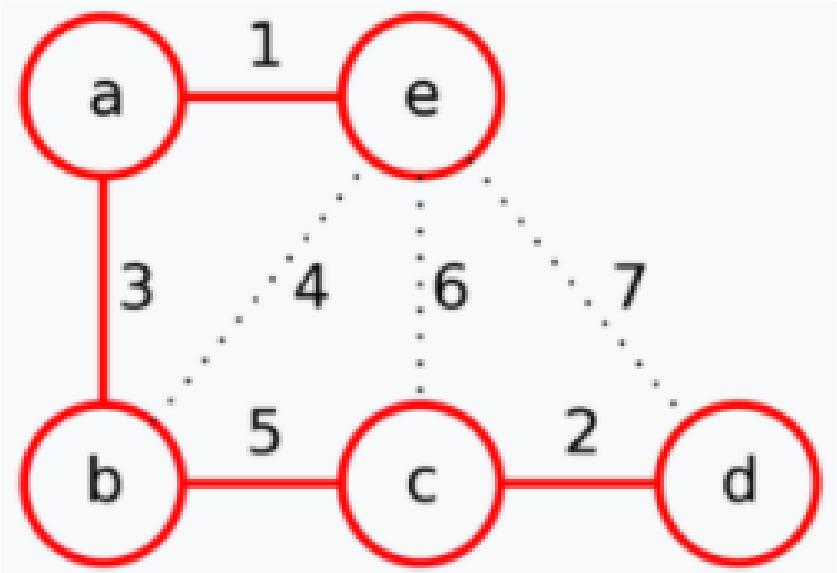
Пример работы алгоритма Краскала



Пример работы алгоритма Краскала



Пример работы алгоритма Краскала



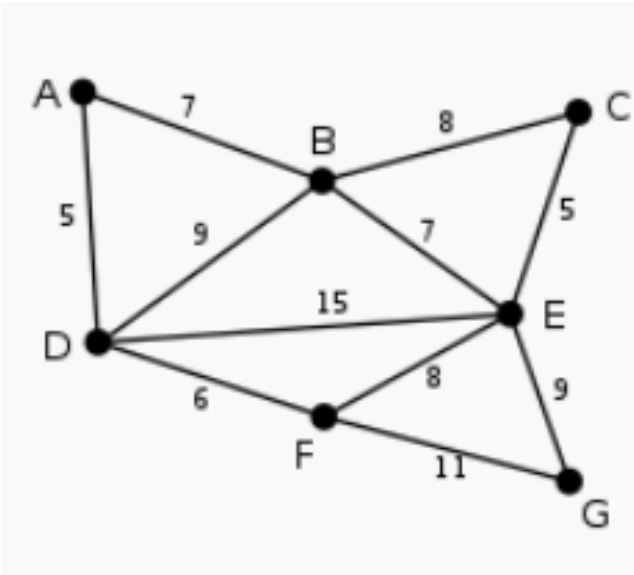
Алгоритм Прима-Ярника

Последовательное построение поддерева F ответа для графа G.
Хранится приоритетная очередь Q из вершин $G \setminus F$, ключ для вершины v – это $\min\limits_{u \in V(F), uv \in E(G)} w(uv)$ – вес минимального ребра из F в $G \setminus F$.

Также для каждой вершины хранится $u = p(v)$ – вершину, на которой достигается минимум в определении ключа. Дерево F поддерживается неявно, его ребра – пары $(v, p(v))$, где $v \in G \setminus \{r\} \setminus Q$ (r – корень F).

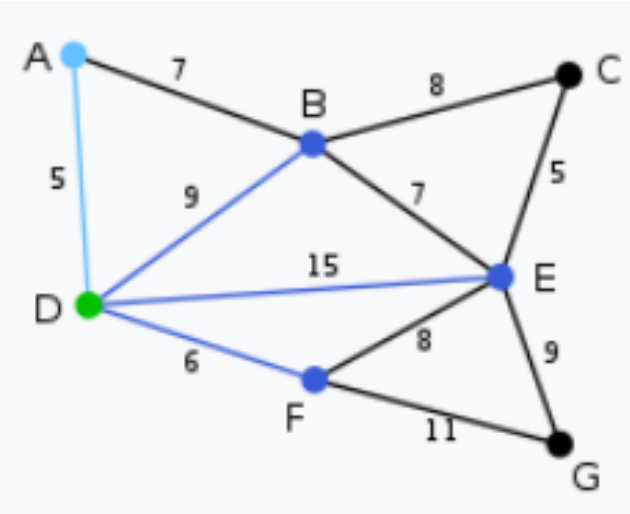
1. F пусто, все ключи имеют значение $+\infty$
2. Выбирается произвольная вершина r, её ключу присваивается значение 0.
3. На очередном шаге алгоритма (пока Q не пусто) извлекается v – минимальная вершина из Q
   - Пробегаемся по всем ребрам $vu \in E(G)$ и, если $u \in Q$ и её ключ $> w(v, u)$, то обновляем вершину с минимумом для u ($p(u) = v$)
   - значение ключа u равно $= w(vu)$. «Релаксация» ребра vu.
   - В Q обновляем ключ для u.
   - В ответ добавляется ребро $(v, p(v))$.

Пример работы алгоритма Прима-Ярника



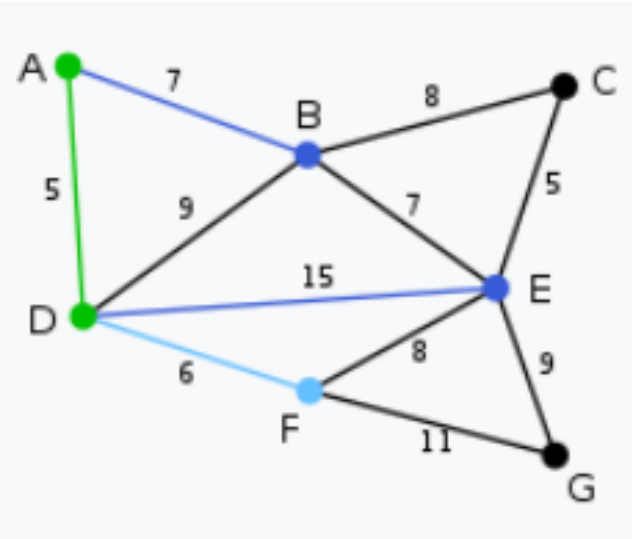
$V(F_{gr})$: $\emptyset$
Рёбра $(u, v), u \in V(F_{gr}), v \in V(G) \setminus V(F_{gr}) : \emptyset$
$V(G) \setminus V(F_{gr}) : A, B, C, D, E, F, G$

Пример работы алгоритма Прима-Ярника



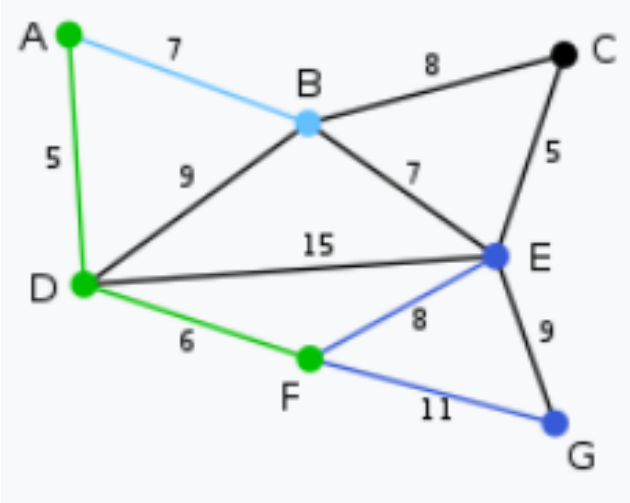
$V(F_{gr})$: D
Рёбра (u, v):
$(D, A) = \mathbf{5}, (D, B) = 9,$
$(D, E) = 15, (D, F) = 6$
$V(G) \setminus V(F_{gr}) : A, B, C, E, F, G$
Добавляем в ответ ребро (D, A)

Пример работы алгоритма Прима-Ярника



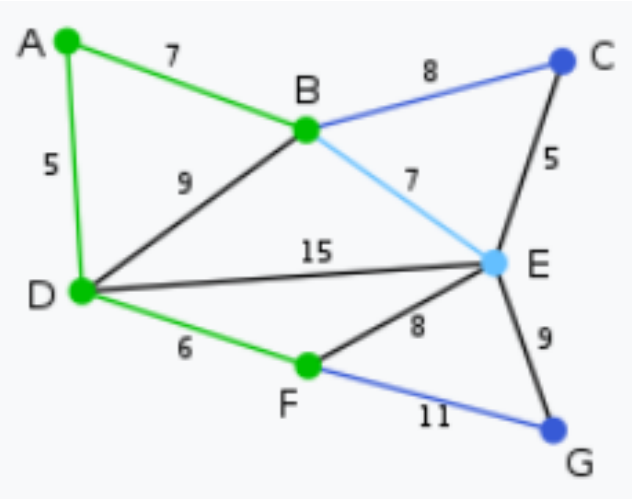
$V(F_{gr})$: D, A
Рёбра (u, v):
$(D, B) = 9, (D, E) = 15,$
$(D, F) = \mathbf{6}, (A, B) = 7$
$V(G) \setminus V(F_{gr}) : B, C, E, F, G$
Добавляем в ответ ребро (D, F)

Пример работы алгоритма Прима-Ярника



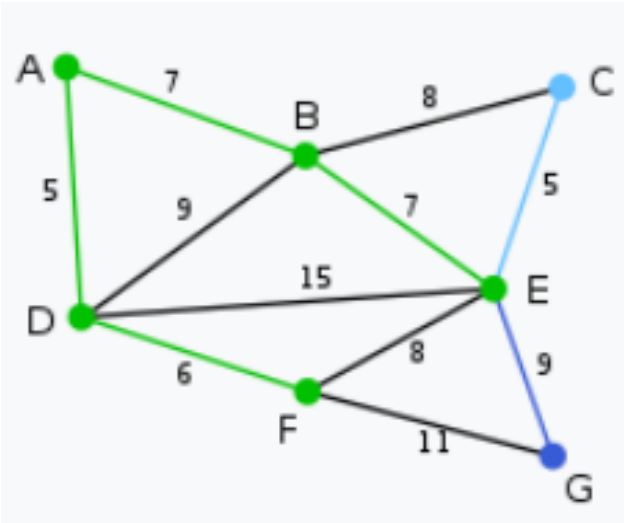
$V(F_{gr})$: D, A, F
Рёбра (u, v):
$(D, B) = 9, (D, E) = 15,$
$(A, B) = \mathbf{7}, (F, E) = 8, (F, G) = 11$
$V(G) \setminus V(F_{gr}) : B, C, E, G$
Добавляем в ответ ребро (A, B)

Пример работы алгоритма Прима-Ярника



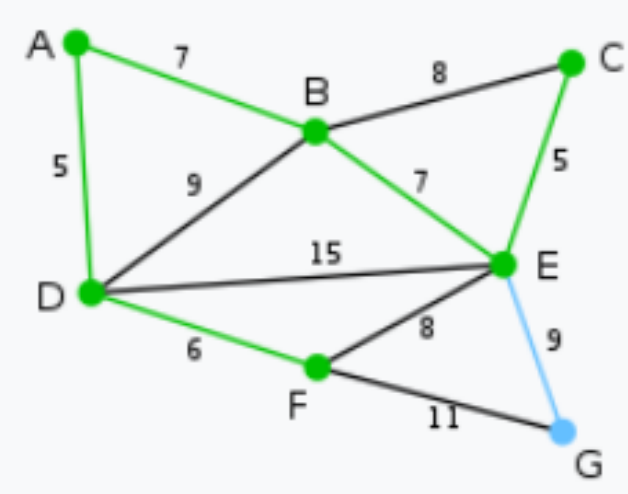
$V(F_{gr})$: D, A, F, B
Рёбра (u, v):
$(B, C) = 8, (B, E) = \mathbf{7},$
$(D, B) = 9, (D, E) = 15,$
$(F, E) = 8, (F, G) = 11$
$V(G) \setminus V(F_{gr}) : C, E, G$
Добавляем в ответ ребро (B, E)

Пример работы алгоритма Прима-Ярника



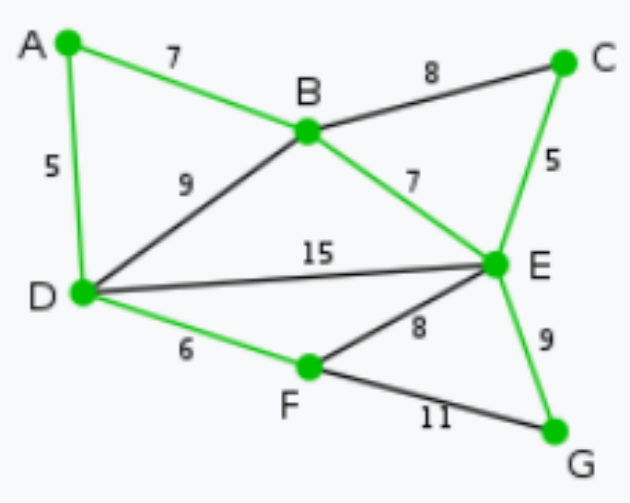
$V(F_{gr})$: D, A, F, B, E
Рёбра (u, v):
$(B, C) = 8, (E, C) = \mathbf{5}$,
$(E, G) = 9 (F, G) = 11$
$V(G) \setminus V(F_{gr}) : C, G$
Добавляем в ответ ребро (E, C)

Пример работы алгоритма Прима-Ярника



$V(F_{gr})$: D, A, F, B, E, C
Рёбра (u, v):
$(F, E) = 8, (F, G) = 11$
$V(G) \setminus V(F_{gr}) : G$
Добавляем в ответ ребро (E, G)

Пример работы алгоритма Прима-Ярника



$V(F_{gr})$: D, A, F, B, E, C, G
Рёбра (u, v): $\varnothing$
$V(G) \setminus V(F_{gr}) : \varnothing$
Минимальное остовное дерево выбрано!

Дерево кратчайших путей на взвешенном графе

Пусть дан взвешенный граф $G = (V, E)$, веса всех ребер заданы неотрицательной функцией $w : E \to \mathbb{R}_+$, и зафиксирована некоторая вершина s. Необходимо найти кратчайшие пути (если есть) до остальных вершин в графе.

Определение: Получающийся подграф кратчайших путей в случае связного графа будет деревом. Такое дерево называется **дерево кратчайших путей**.

В рамках построения дерева поддерживаются структуры:

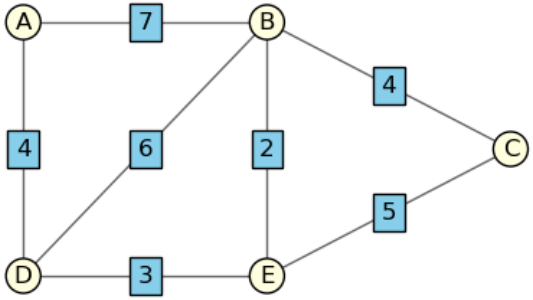
- список непосещенных вершин,
- список текущих кратчайших расстояний,
- список предков: для каждой вершины $v \neq s$ будем хранить $p(v)$ – предпоследнюю вершину на кратчайшем пути от s до v.

Алгоритм Дейкстры

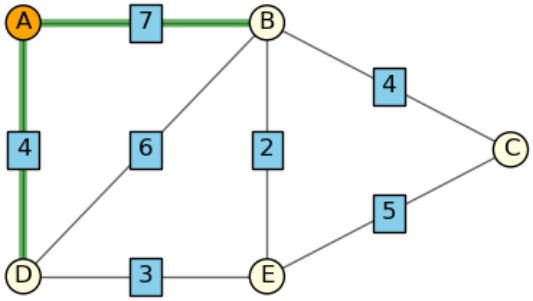
1. инициализируем список текущих кратчайших расстояний $d[V]$ – у всех вершин, кроме s, значение $+\infty$; у вершины s это значение будет 0;
2. в списке непосещенных вершин выбирается вершина v с кратчайшим расстоянием до s; вершина v помечается как посещенная; если на какой-то итерации кратчайшее расстояние равно $+\infty$, то все непосещенные вершины недостижимы из s и алгоритм можно останавливать;
3. просматриваются непосещенные вершины v_i, до которых можно добраться по ребрам из вершины v, и, если $d[v_i]$ – текущее расстояние от s до v_i превышает значение $d[v] + w(vv_i)$, то $d[v_i] := d[v] + w(vv_i)$ и обновляется значение $p[v_i] = v$; в противном случае они не меняются;
4. второй и третий шаги повторяются, пока все вершины графа не будут посещены.

Кратчайший путь от вершины s до вершины v будет $(s, \ldots, p[p[v]], p[v], v)$ и он восстанавливается из списка предков. Дерево кратчайших путей, по аналогии с деревом обхода в глубину, строится как $G_d = (V(G), E_d)$, где $E_d = \{(u, v) \mid u = p[v], p[v] \neq \varnothing, v \in V(G)\}$.

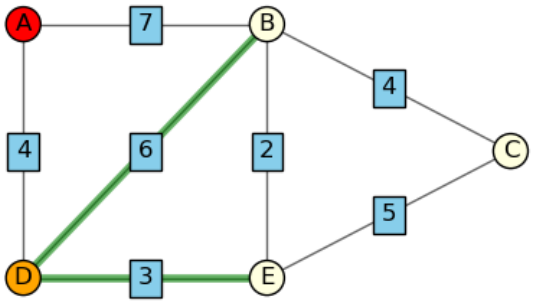
Пример



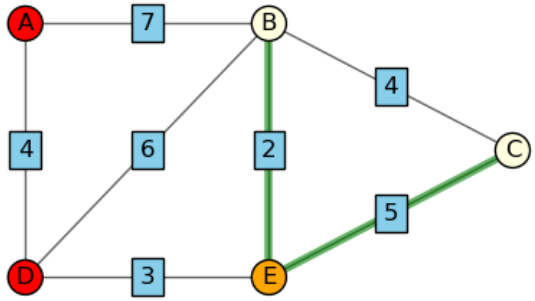
Вершина v	$d[v]$	$p[v]$
A	0	
B	∞	
C	∞	
D	∞	
E	∞	



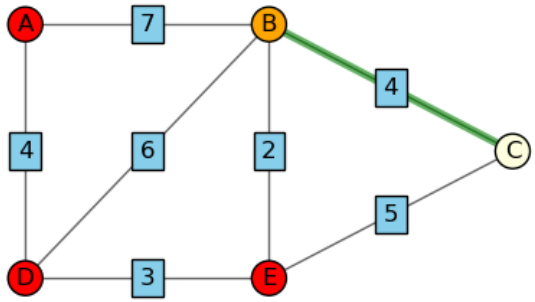
Вершина v	$d[v]$	$p[v]$
A	0	
B	7	A
C	∞	
D	4	A
E	∞	



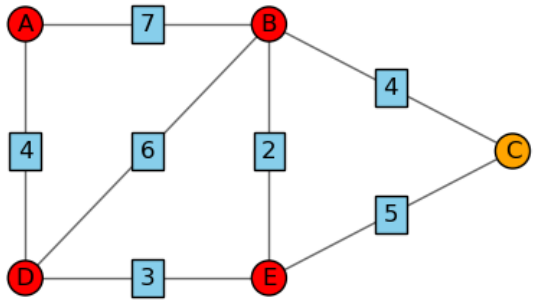
Вершина v	$d[v]$	$p[v]$
A	0	
B	7	A
C	∞	
D	4	A
E	7	D



Вершина v	$d[v]$	$p[v]$
A	0	
B	7	A
C	12	E
D	4	A
E	7	D



Вершина v	$d[v]$	$p[v]$
A	0	
B	7	A
C	11	B
D	4	A
E	7	D



Вершина v	$d[v]$	$p[v]$
A	0	
B	7	A
C	11	B
D	4	A
E	7	D