Алгоритмы и структуры данных

Лекция 14

Графы. Компоненты связности. Специальные элементы. MST.

Сергей Леонидович Бабичев

Поиск компонент связности

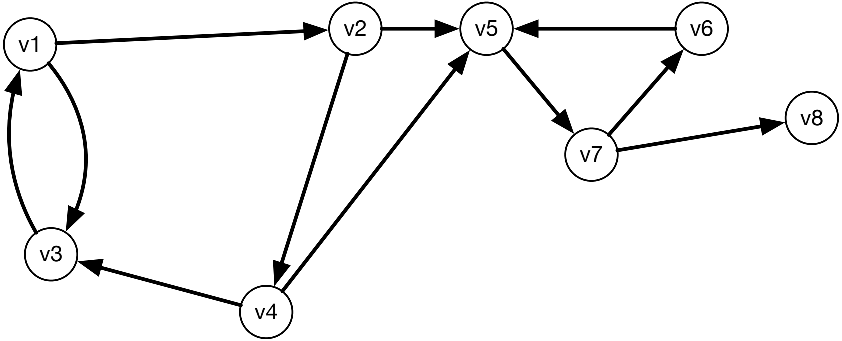
Поиск компонент связности

- Для неориентированных графов: запустив поиск `BFS` или `DFS`.
- Все выкрашенные по завершении поиска вершины образуют компоненту связности.
- Выбирается произвольным образом необработанная вершина и алгоритм повторяется, формируя другую компоненту связности.
- Алгоритм заканчивается, когда не остаётся необработанных вершин.

Поиск компонент связности

- Для ориентированных графов: результаты зависят от порядка обхода вершин.
- **Компонента сильной связности ориентированного графа:** максимальное по размеру множество вершин, взаимно достижимых друг из друга.

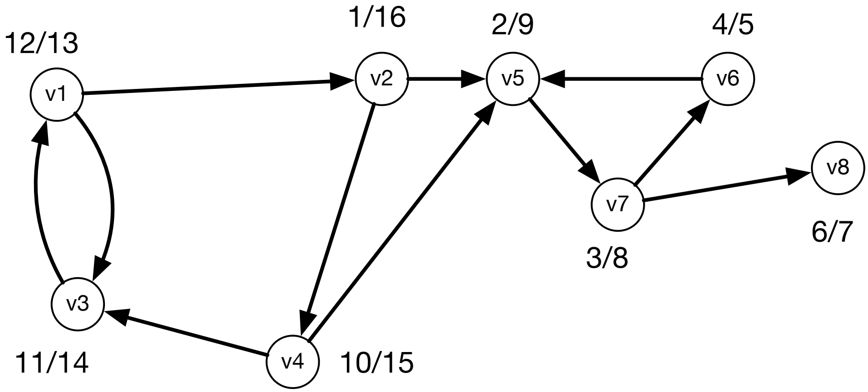
Поиск компонент связности: алгоритм Косарайю



Поиск компонент связности: алгоритм Косарайю

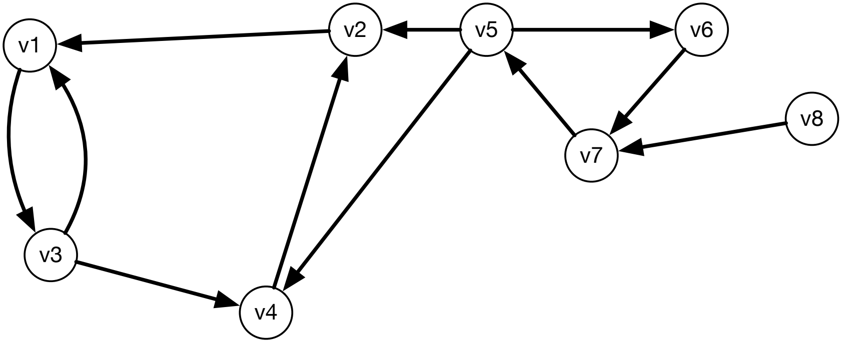
- Проведём полный DFS поиск.
- В алгоритме полного DFS не специфицировано, с какой вершины начинается поиск $\to$ можно выбрать произвольную.

Обход с вершины v_2:



Поиск компонент связности: алгоритм Косарайю

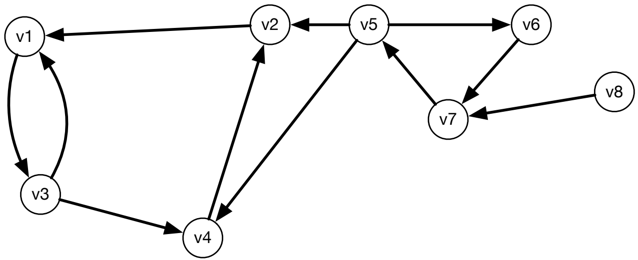
- Заменим направления всех рёбер (перевернём все стрелки).
- Каждое ребро $u \to v$ заменяется на $v \to u$.



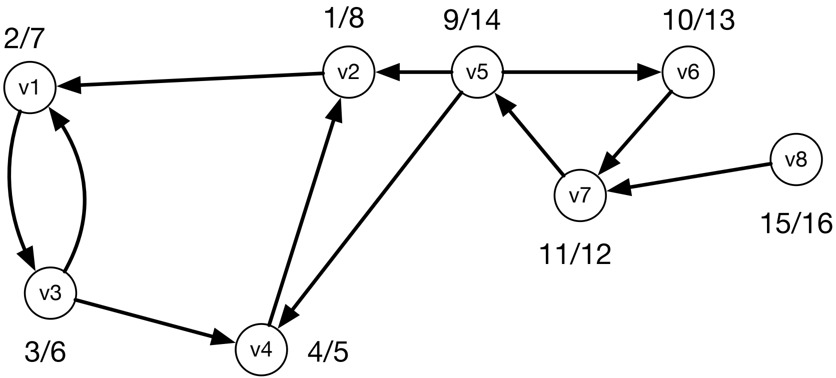
Поиск компонент связности: алг. Косарайю

- Обходим ещё раз. Начальная вершина — из необработанных, у которой наибольшие значение времени выхода.
- Обход из вершины 2 покрасил вершины v_1, v_2, v_3 и v_4.
- Остались непокрашенные вершины v_5, v_6, v_7 и v_8.
- Повторяем, пока останутся непокрашенные вершины.

Номер	1	2	3	4	5	6	7	8
Вход/выход	12/13	1/16	11/14	10/15	2/9	4/5	3/8	6/7

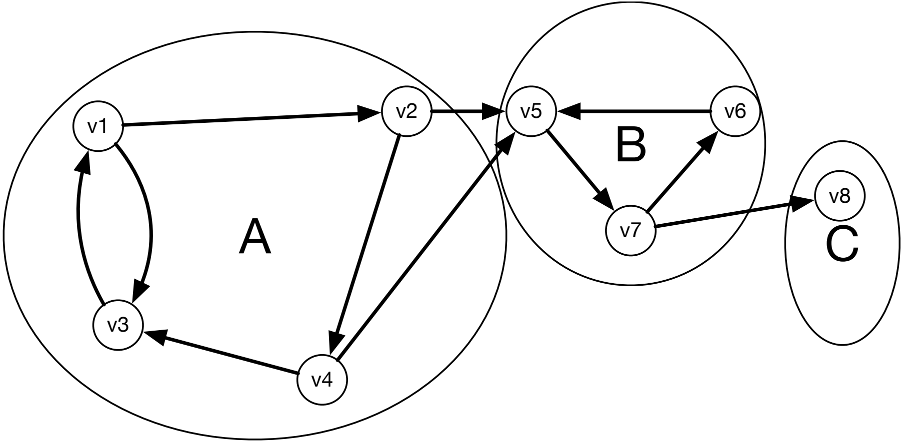


Поиск компонент связности: алгоритм Косарайю



- Каждый «малый» проход алгоритма DFS даст нам вершины, которые принадлежат одной компоненте сильной связности.

Поиск компонент связности: алгоритм Косарайю



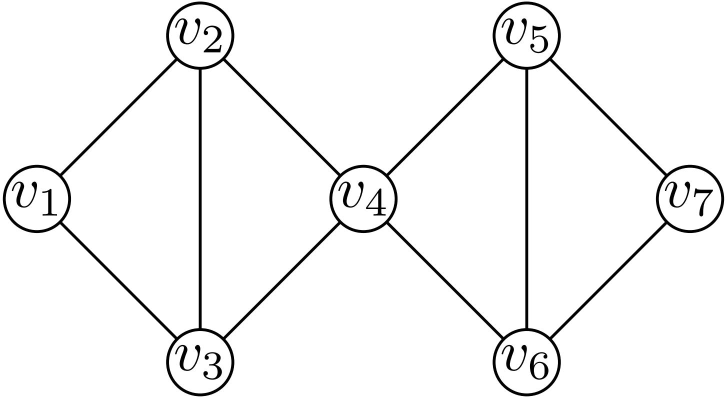
- Рассматривая компоненту сильной связности как единую мета-вершину, мы получаем новый граф, который называется *конденсацией* исходного графа или *конденсированным графом*.

Поиск специальных элементов графа

- **Точка сочленения (junction point)** — такая вершина графа, удаление которой вместе с исходящими из неё рёбрами, приводит к увеличению числа компонент связности графа.
  Синонимы: **точка раздела**, **точка артикуляции**, **разделяющая вершина**, **cut vertex**, **articulation vertex**.
- **Блок** — связный непустой граф, не содержащий точек сочленения. Другое название — **компонент двусвязности**.
- **Мост (bridge)** — такое ребро графа, удаление которого приводит к увеличению числа компонент связности графа.

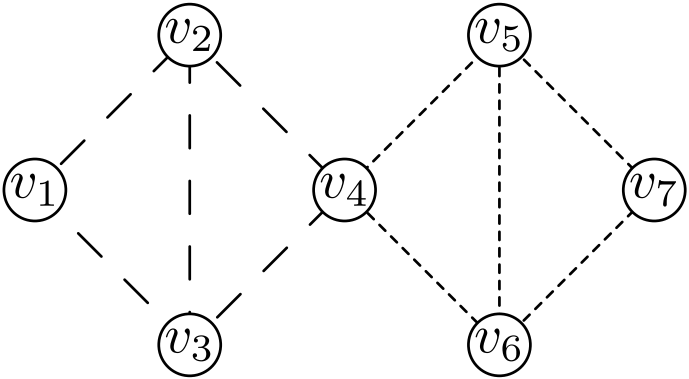
- Для любого моста (u, v) вершины u и v являются точками сочленения.
- Пара различных рёбер e_1 и e_2 удовлетворяют **отношению** R **на множестве рёбер** E если существует простой цикл, который содержит эти рёбра.
- **Любое ребро** e **находится в отношении** R **с самим собой**
- R — *отношение эквивалентности*.
- Множество всех рёбер графа E можно разбить на непересекающиеся множества таким образом, что каждое ребро попадает ровно в одно из них — множество E разбито на *классы эквивалентности* относительно R.

Поиск точек сочленения: вывод свойств



Поиск точек сочленения: вывод свойств

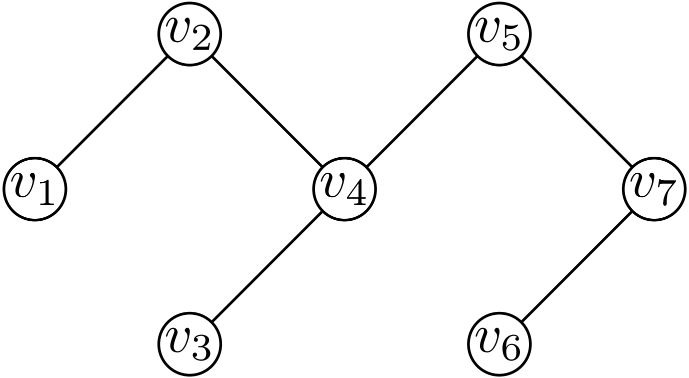
Выкрасим рёбра в один и тот же цвет, если они принадлежат одному классу, и в разный — если они принадлежат разным классам.



Вершина 4 — единственная, к которой подходят рёбра одного цвета. Удаление этой вершины и её рёбер приведёт к графу с двумя компонентами связности.

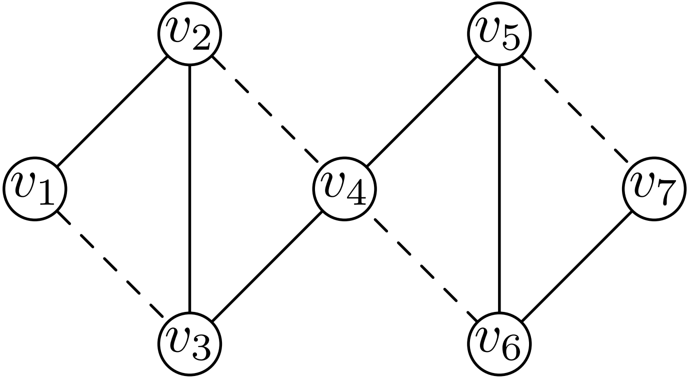
Поиск точек сочленения

- Наивный поиск — удалять вершину с рёбрами и проверять связность: $T = O(|V| \cdot (|V| + |E|))$.
- Быстрее: использовать DFS.
- Для связного графа DFS даст дерево обхода. Вот одно из них:



Поиск точек сочленения

Рёбра из дерева обхода назовём *прямыми*, а оставшиеся — *обратными*.

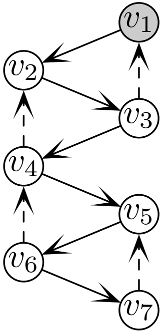


Поиск точек сочленения

- Если при обходе DFS мы попытаемся попасть в уже выкрашенную в серый или чёрный цвет вершину, то мы обнаружили цикл.
- Добавим массив l, который в вершине u будет принимать минимальное значение из всех $d[v]$, где v пробегает по концам всех обратных рёбер, начинающихся в поддеревьях с корнем в u.

Поиск точек сочленения

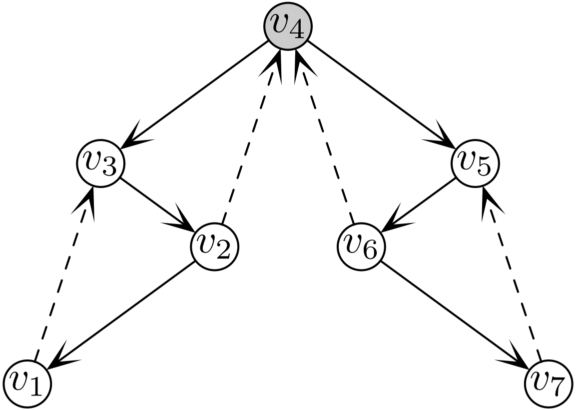
- Более наглядно: граф нарисован с его обходом в ориентированном виде, расположив узлы дерева в порядке их прохождения сверху вниз. Сплошные стрелки будут формировать порядок обхода (прямые рёбра), пунктирные — обратные рёбра.



- Рассмотрим какую-либо вершину, из которой начинается поддерево. Если не имеется обратных рёбер, которые её «перепрыгивают», то есть ведут в вершины, находящиеся выше рассматриваемой, значит эта вершина находится ровно в одном классе эквивалентности и не является точкой сочленения.
- Мы можем «перепрыгнуть» вершину 2 (есть обратное ребро $3 \to 1$), вершину 3 (есть обратное ребро $4 \to 2$), вершины 5 и 6. Вершина 7 — терминальная в дереве обхода и не является точкой сочленения.

Поиск точек сочленения

- Корень обхода — исключение их правил.
- Если от этой вершины исходит по меньшей мере два прямых ребра, то при обходе нам приходилось возвращаться в эту вершину — мы сформировали класс эквивалентности — и приступили к формированию другого.



Поиск мостов

- Несколько утверждений:
  - Мост является и компонентом двусвязности.
  - Ребро тогда и только тогда является мостом, когда оно не принадлежит ни одному простому циклу.
  - Каждый класс эквивалентности по отношению R либо компонента двусвязности, либо мост.
  - Обратные рёбра мостами являться не могут.
- Допустим, что прямое ребро $e = (u, v)$ является мостом в связном графе G, причём v — дочерняя вершина u.
- Тогда при его удалении граф распадётся на две компоненты связности, в одной из которых останется вершина u, а другая будет определяться поддеревом дерева обхода, начинающимся в вершине v.
- Это означает, что из v не имеется обратных рёбер в первую компоненту связности.
- Вспомним, что массив l в обходе `DFS-junction` в алгоритме нахождения точек связности уже содержит нужную нам информацию.

Нахождение мостов

```
procedure DFS-BRIDGE(u : Vertex)
    c[u] ← grey
    time ← time + 1
    d[u] ← time
    l[u] ← time
    for all v ∈ Adj[u] do
        if c[v] = white then
            DFS-bridge(v)
            l[u] ← min(l[u], l[v])
            if l[v] > d[u]  &  l[v] ⩾ d[v] then
                Register (u, v) as bridge
            end if
        else
            l[u] ← min(l[u], d[v])
        end if
    end for
    c[u] ← black
    time ← time + 1
    f[u] ← time
end procedure
```

Остовные деревья

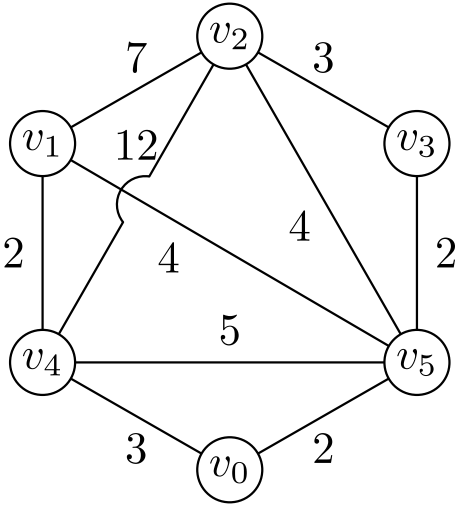
Остовное дерево: ещё немного терминов

- С точки зрения теории графов **дерево** есть ациклический связный граф.
- Множество деревьев называется **лесом (forest)** или **бором**.
- **Остовное дерево** связного графа — подграф, который содержит все вершины графа и представляет собой полное дерево.
- **Остовный лес** графа — лес, содержащий все вершины графа.

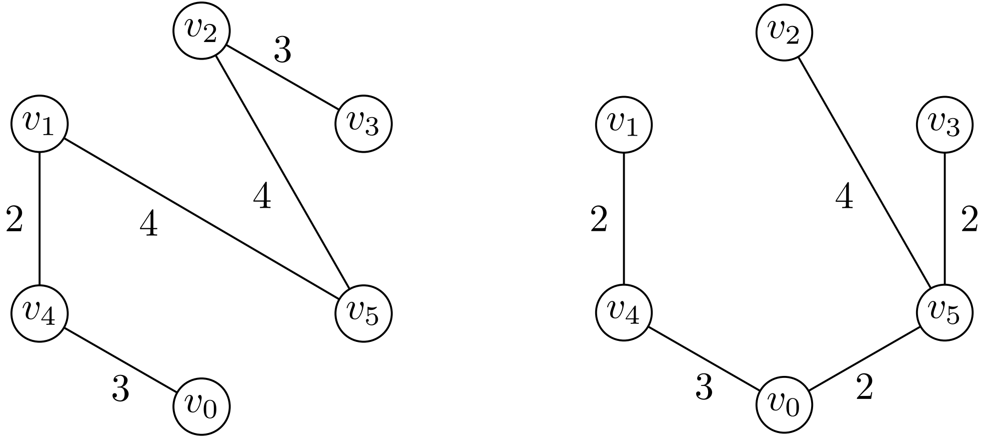
Минимальное остовное дерево

- Построение остовных деревьев — одна из основных задач в компьютерных сетях.
- Решение задачи — как спланировать маршрут от одного узла сети до других.
- Для некоторого типа узлов в передаче сообщений недопустимо иметь несколько возможных маршрутов. Например, если компьютер соединён с маршрутизатором по Wi-Fi и Ethernet одновременно, то в некоторых операционных системах сообщения от компьютера до маршрутизатора не будут доходить из-за наличия цикла.
- Построение остовного дерева — избавление от циклов в графе.

Остовные деревья



Остовные деревья



Минимальное остовное дерево

- `MST` — Minimal Spanning Tree.
- Минимальное остовное дерево взвешенного графа есть остовное дерево, вес которого (сумма его всех рёбер) не превосходит вес любого другого остовного дерева.
- Именно минимальные остовные деревья больше всего интересуют проектировщиков сетей.
- **Сечение графа** — разбиение множества вершин графа на два непересекающихся подмножества.
- **Перекрёстное ребро** — ребро, соединяющее вершину одного множества с вершиной другого множества.

Свойства MST

- **Лемма.** Если T — произвольное остовное дерево, то добавление любого ребра e между двумя вершинами u и v создаёт цикл, содержащий вершины u, v и ребро e.

Свойства MST

- **Лемма.** При любом сечении графа каждое минимальное перекрёстное ребро принадлежит некоторому MST-дереву и каждое MST-дерево содержит перекрёстное ребро.
- **Доказательство** от противного. Пусть e — минимальное перекрёстное ребро, не принадлежащее ни одному MST и пусть T — MST дерево, не содержащее e. Добавим e в T. В этом графе есть цикл, содержащий e и он содержит ребро e', с весом, не меньшим e. Если удалить e', то получится остовное дерево не большего веса, что противоречит условию минимальности T или предположению, что e не содержится в T.

Свойства MST

- **Следствие.** Каждое ребро дерева MST есть минимальное перекрёстное ребро, определяемое вершинами поддеревьев, соединённых этим ребром.

Свойства MST

- **Лемма (без доказательства).** Пусть имеется граф G и ребро e. Пусть граф G' есть граф, полученный добавлением ребра e к графу G. Результатом добавления ребра e в `MST` графа G и последующего удаления максимального ребра из полученного цикла будет `MST` графа G'.
- Эта лемма выявляет рёбра, которые не должны входить в `MST`.

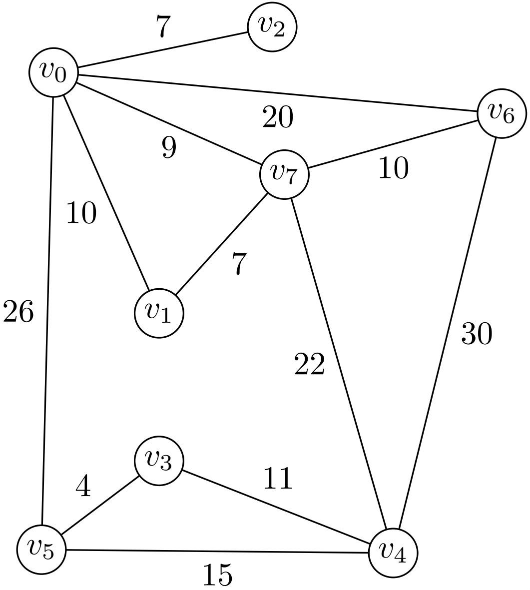
Алгоритмы поиска MST

Алгоритм Прима

1. Используется сечение графа на два подграфа — древесных вершин и недревесных вершин.
2. Выбираем произвольную вершину. Это — MST дерево, состоящее из одной древесной вершины.
3. Выбираем минимальное перекрёстное ребро между MST множеством и недревесным множеством.
4. Повторяем операцию до тех пор, пока все вершины не окажутся в дереве.

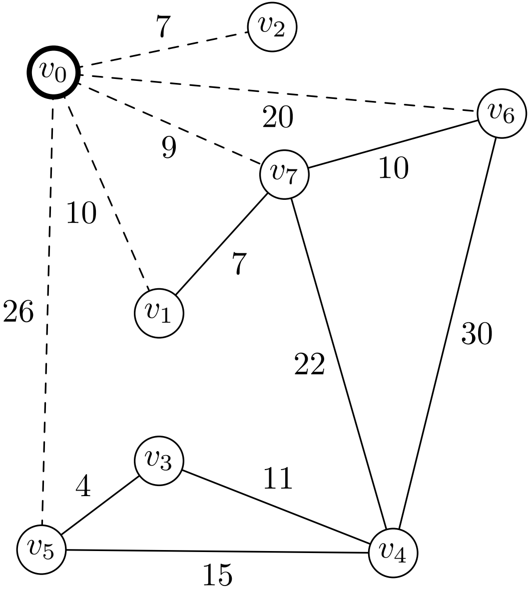
Алгоритм Прима

Исходный граф.



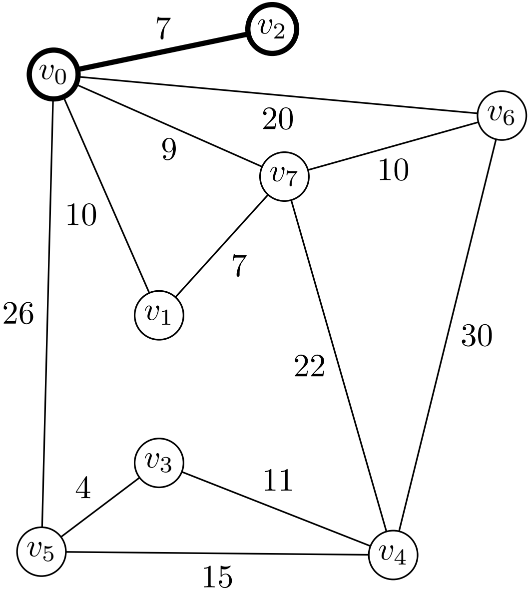
Алгоритм Прима

Вершина `0` — корневая. Переводим её в `MST`. Проверяем все веса из `MST` в не `MST`.



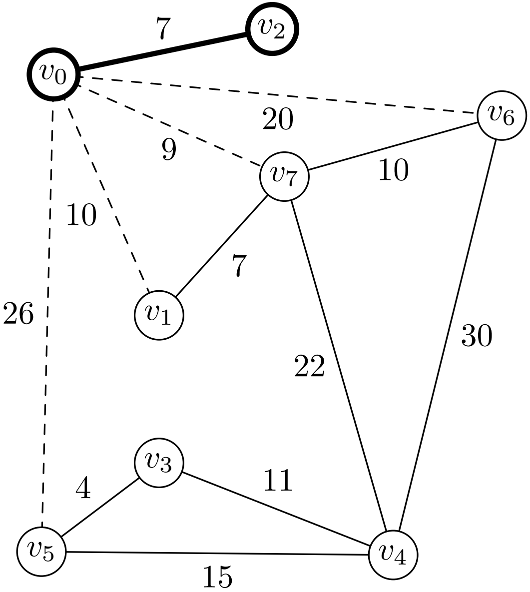
Алгоритм Прима

(0-2) самое лёгкое ребро. Переводим вершину 2 и ребро (0-2) в MST.



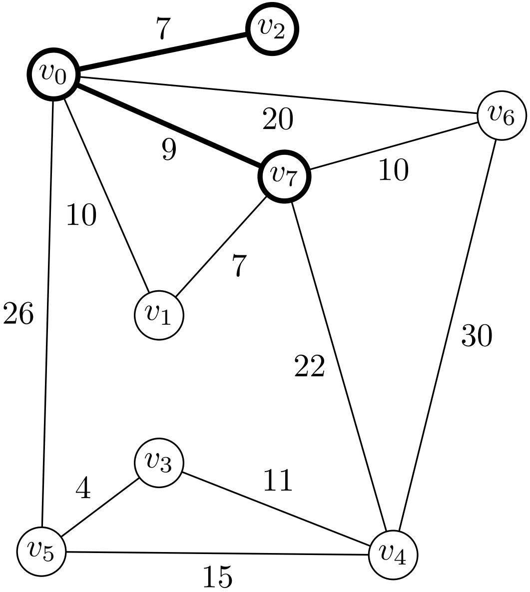
Алгоритм Прима

Отмечаем все рёбра из MST в не MST.



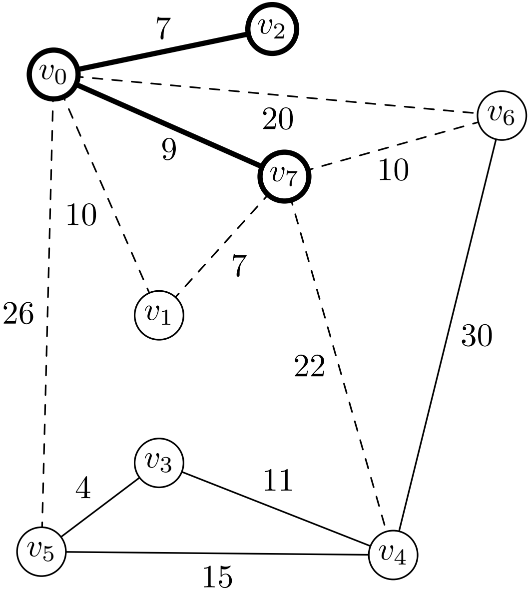
Алгоритм Прима

Переносим вершину 7 и ребро (0–7) в MST.



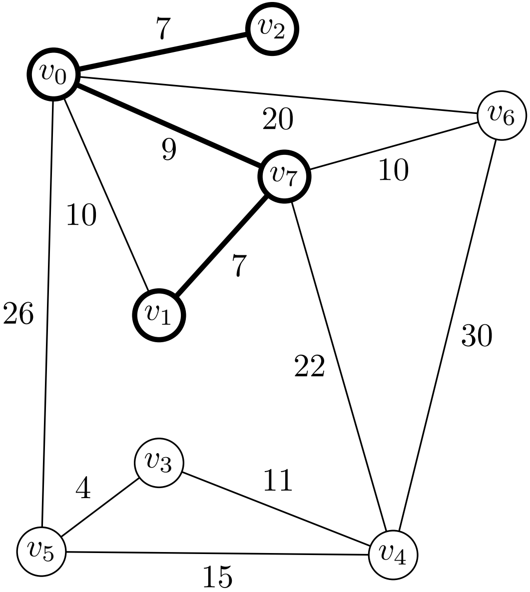
Алгоритм Прима

Отмечаем все рёбра из MST в не MST.



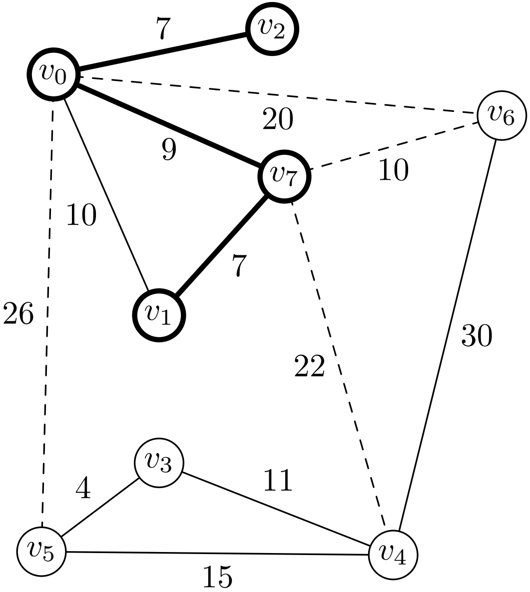
Алгоритм Прима

Переносим вершину 1 и ребро (1-7) в `MST`.



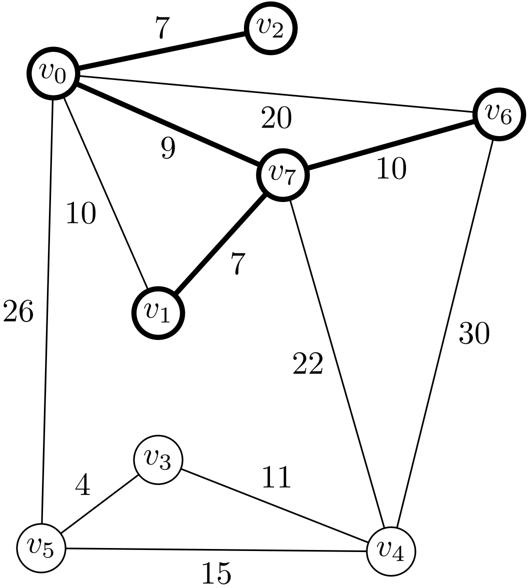
Алгоритм Прима

Отмечаем все рёбра из MST в не MST.



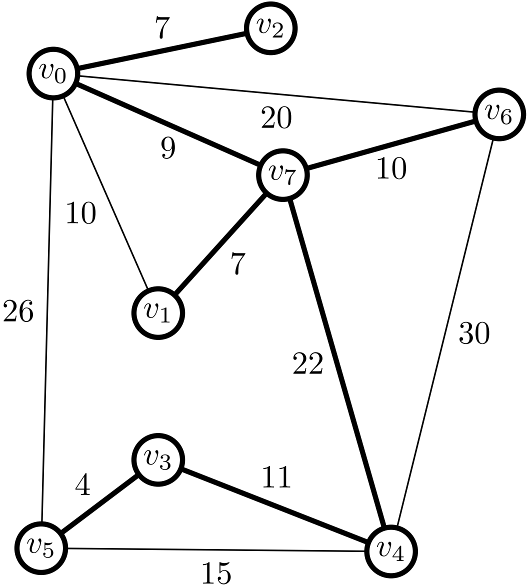
Алгоритм Прима

Переносим вершину 6 и ребро `(7-6)` в `MST`.



Алгоритм Прима

Заключительная позиция: все вершины в `MST`.



Алгоритм Прима

- В данном виде алгоритм не очень эффективен.
- На каждом шаге мы забываем про те рёбра, который уже проверяли.
- Введём понятие **накопителя**.
- Накопитель содержит множество рёбер-кандидатов.
- Каждый раз в `MST` включается самое лёгкое ребро.

Алгоритм Прима

Более эффективная реализация алгоритма Прима

1. Выбираем произвольную вершину. Это — `MST` дерево, состоящее из одной вершины. Делаем вершину текущей.
2. Помещаем в накопитель все рёбра, которые ведут из этой вершины в не `MST` узлы. Если в какой-либо из узлов уже ведёт ребро с большей длиной, заменяем его ребром с меньшей длиной.
3. Выбираем ребро с минимальным весом из накопителя.
4. Повторяем операцию до тех пор, пока все вершины не окажутся в дереве.

Алгоритм Прима

- Алгоритм Прима — обобщение поиска на графе.
- Накопитель представляется очередью с приоритетами.
- Используется операция «извлечь минимальное».
- Используется операция «увеличить приоритет».
- Такой поиск на графе называется PFS — поиск по приоритету.
- Сложность алгоритма $O(|E|\log|V|)$.