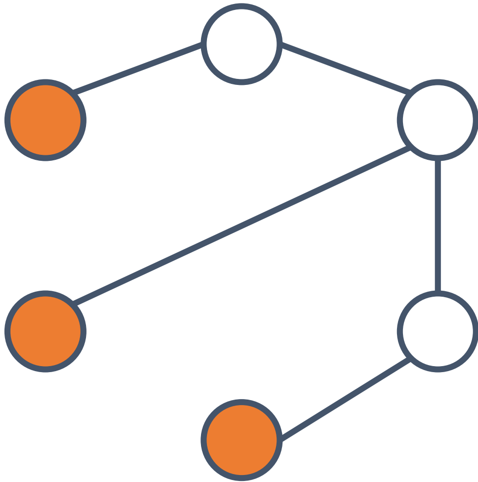
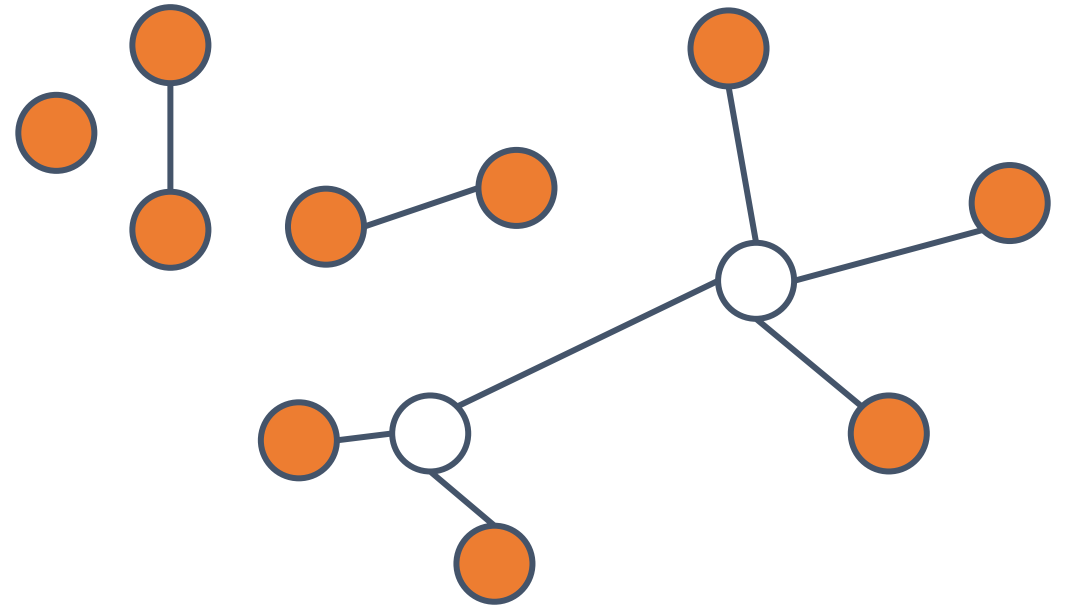
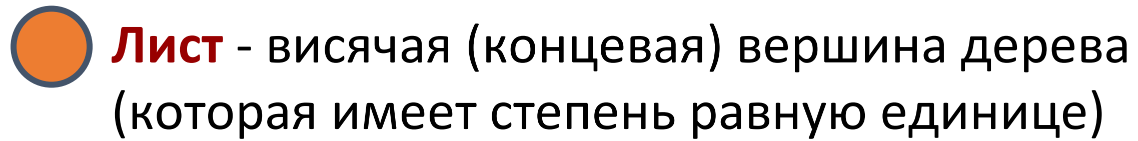
Деревья

Дерево - связный ациклический граф



Лес – граф, являющийся набором непересекающихся деревьев





Лист - висячая (концевая) вершина дерева (которая имеет степень равную единице)

Есть и в лесу, и на дереве

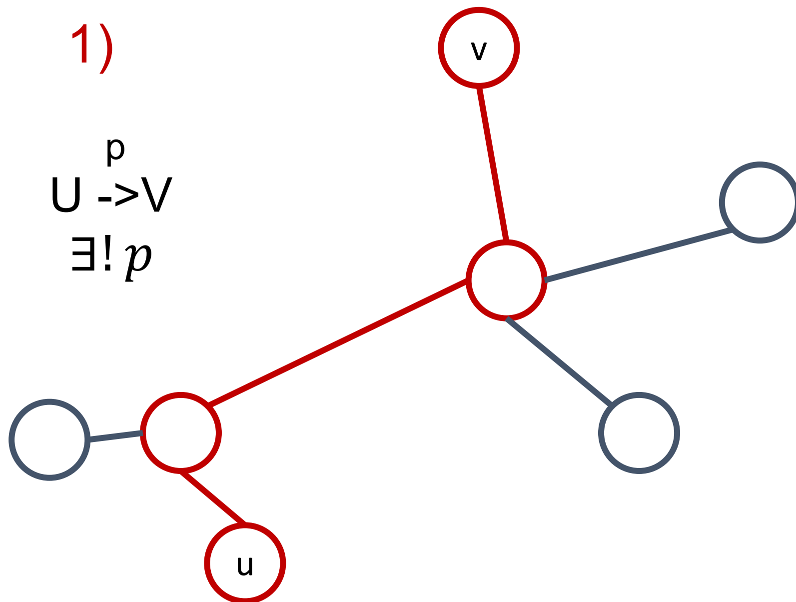
Деревья

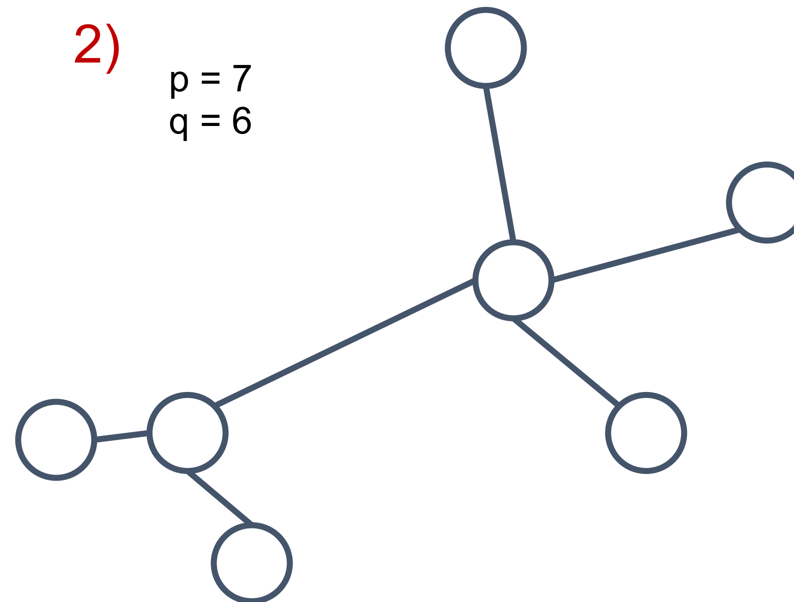
Дерево - связный ациклический граф
Лес - граф, являющийся набором непересекающихся деревьев.

Деревья

Пусть G - дерево тогда :

1) Любые две вершины графа G соединены единственным простым путем
2) G - связен и ацикличен, причем p = q+1, где p - количество вершин, а q - количество ребер
3) G - ацикличен и при добавлении любого ребра для несмежных вершин появляется один простой цикл.
4) Любое ребро дерева - это мост
5) Дерево это двудольный граф

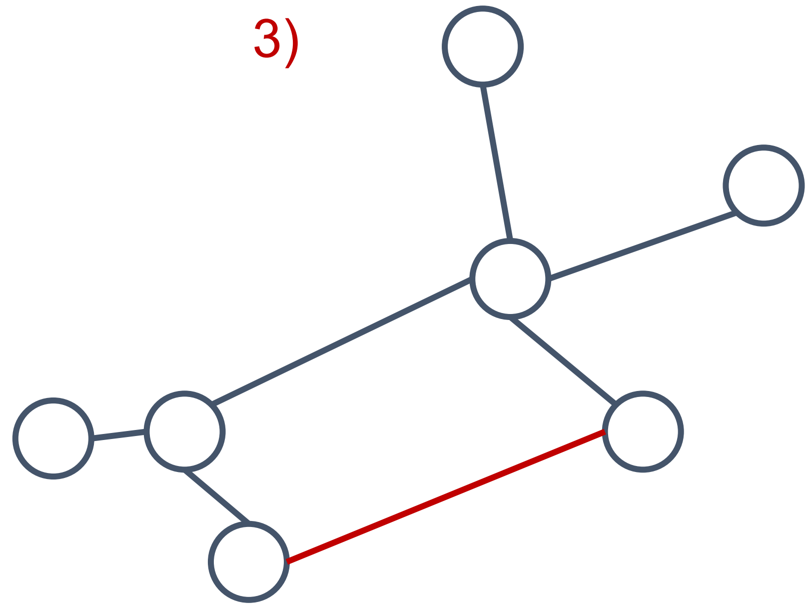


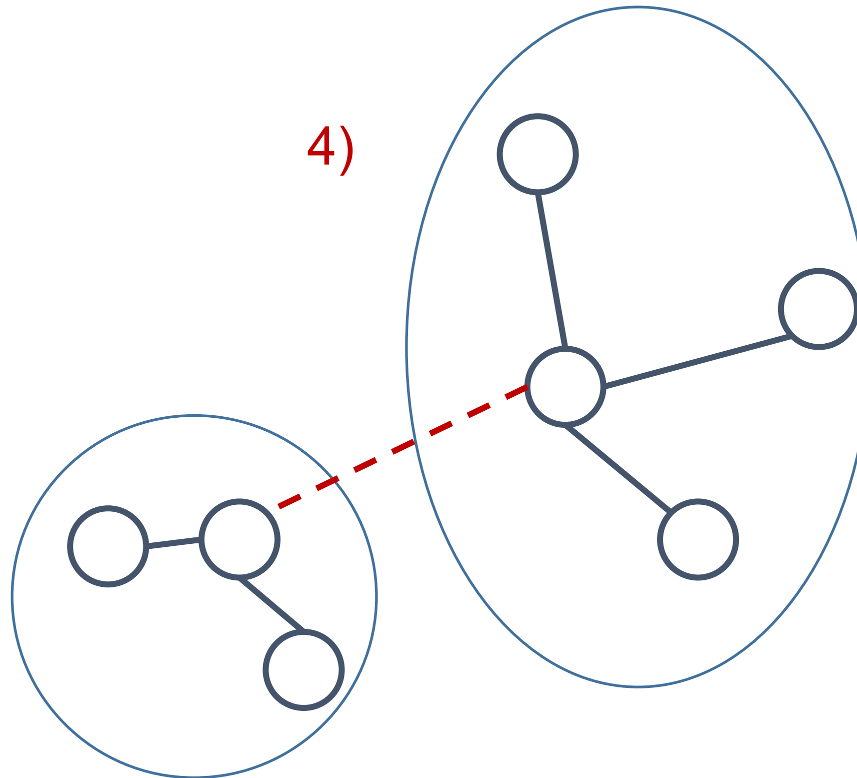


Деревья

Пусть G - дерево тогда :

1) Любые две вершины графа G соединены единственным простым путем
2) G - связен и ацикличен, причем $p = q+1$, где p - количество вершин, а q - количество ребер
3) G - ацикличен и при добавлении любого ребра для несмежных вершин появляется один простой цикл.
4) Любое ребро дерева - это мост
5) Дерево это двудольный граф

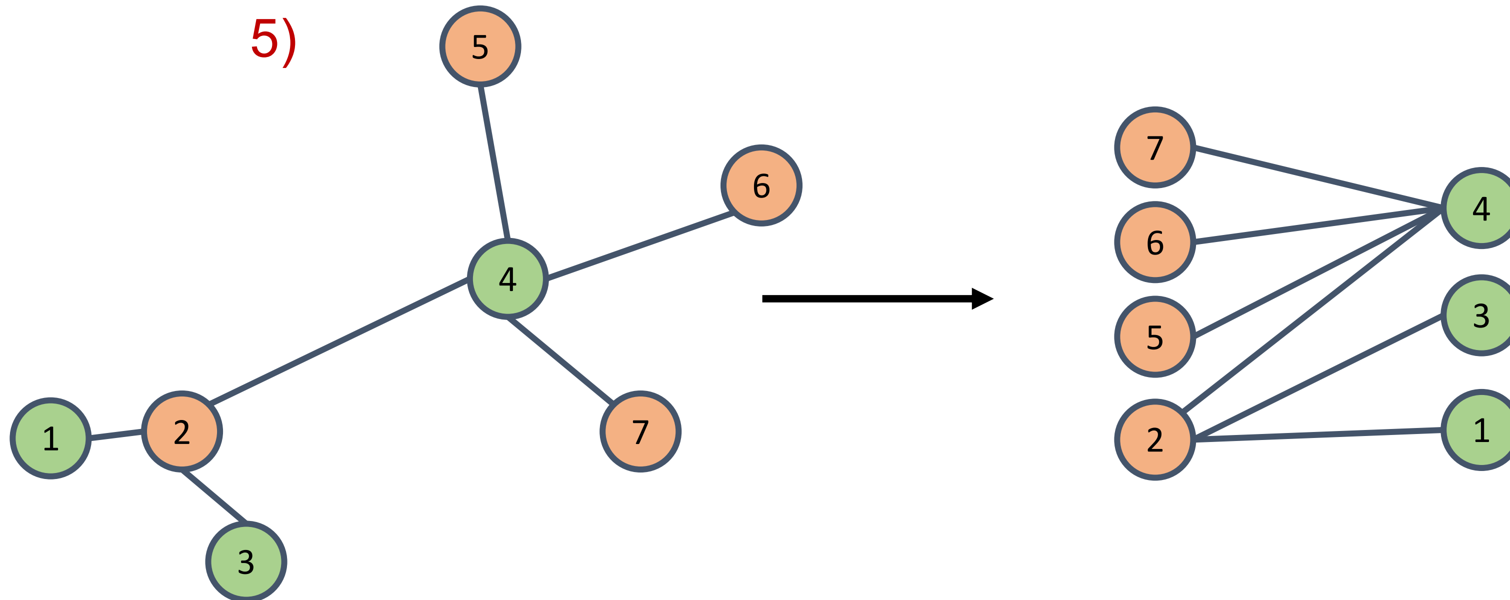




Деревья

Пусть G - дерево тогда :

1) Любые две вершины графа G соединены единственным простым путем
2) G - связен и ацикличен, причем $p = q+1$, где p - количество вершин, а q - количество ребер
3) G - ацикличен и при добавлении любого ребра для несмежных вершин появляется один простой цикл.
4) Любое ребро дерева - это мост
5) Дерево это двудольный граф

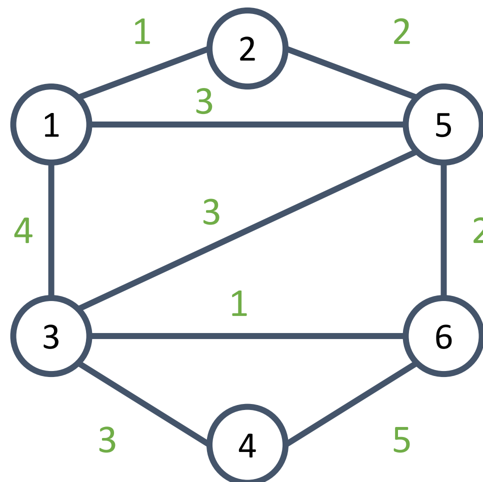


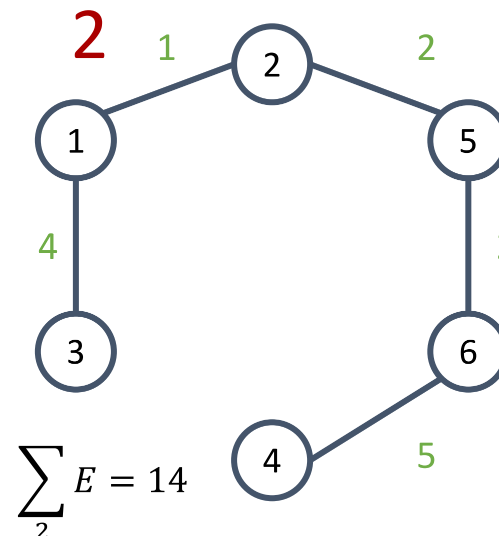
Минимальное остовное дерево

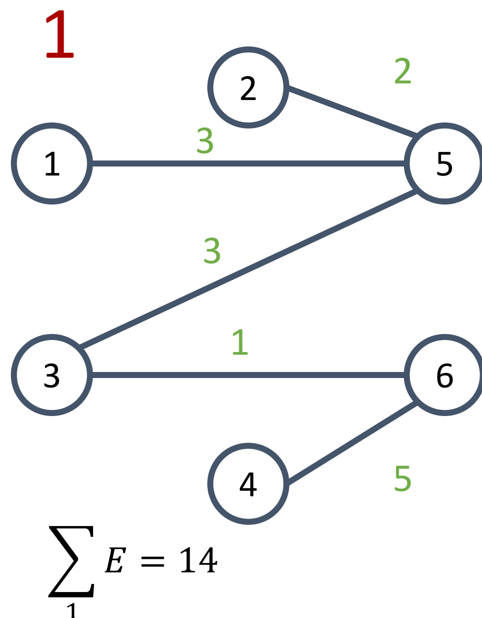
Остов графа (англ spanning tree) - связный и ацикличный частичный граф исходного графа.
(частичный граф - тот же самый граф, только меньше ребер, а если ацикличный - то это дерево, таким образом, остов графа - это его скелет в виде дерева)

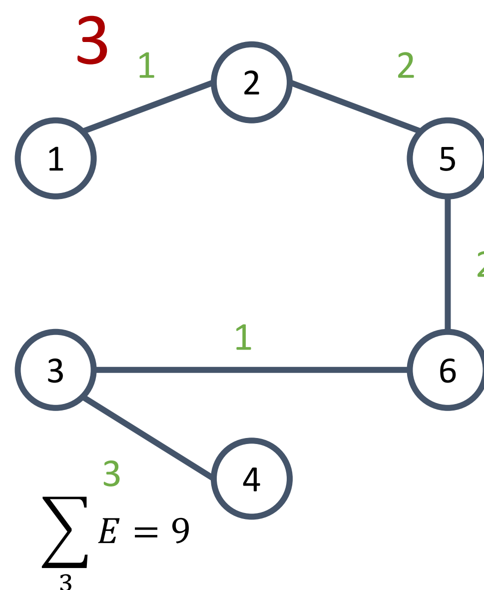
Пример:
Строим остов
6z = M - n + k
= 9- 6 + 1 = 4

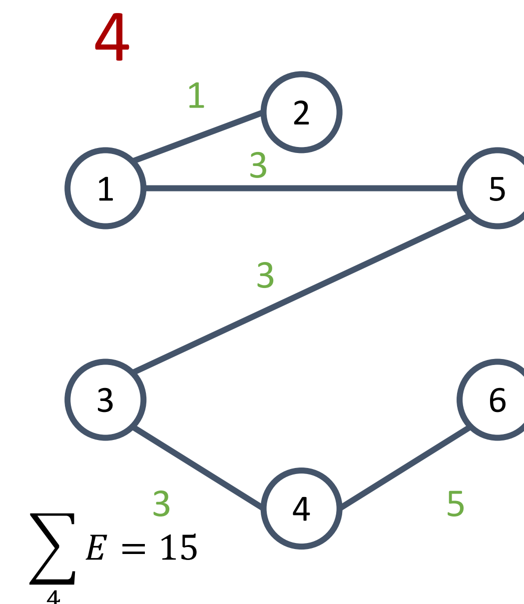
Цикломатическое число = 4











Минимальное остовное дерево

Минимальное остовное дерево (англ. minimum spanning tree далее MST) графа G = (V, E) - это его ациклический связный подграф, в который входят все его вершины, обладающий минимальным суммарным весом ребер.

Примечание: Заметим, что граф может содержать несколько минимальных остовных деревьев.

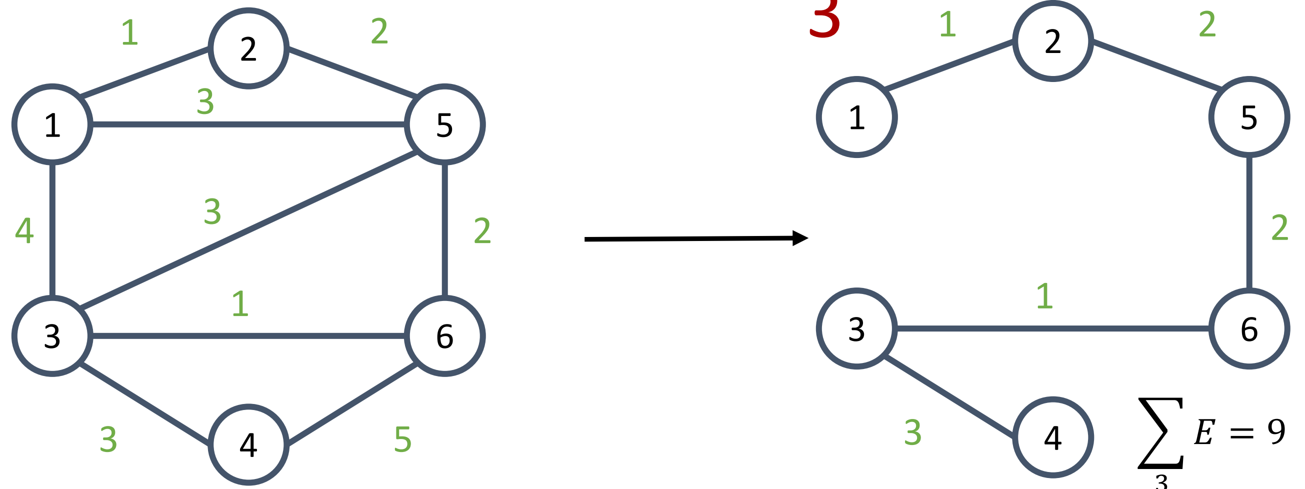
Пример:

Для графа G(E, V) MST – это наименьшее по $\sum E$ остовное дерево. Для G существует 4 остовных дерева из них MST это 3 дерево.

Для точного ответа необходимо построить все возможные остова графа

↓

Это долго

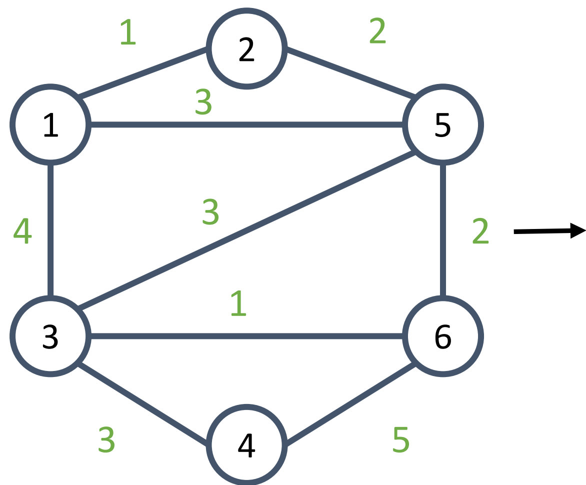


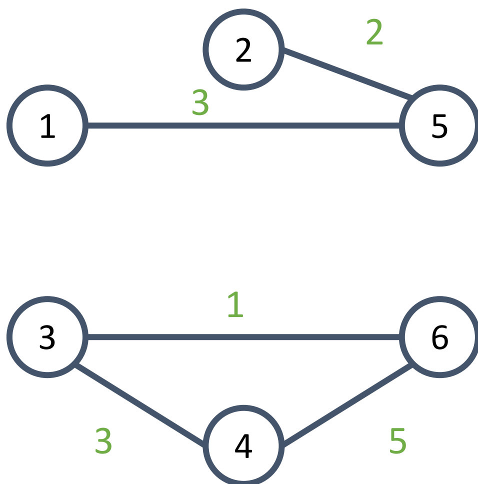
Минимальное остовное дерево

Мы знаем z = 4 (цикломатическое число) нужно удалить ребер, чтобы получить остов

будем удалять ребра с наибольшим весом?

нельзя удалять любые ребра: нужно сохранить связность графа,
чтобы получить остовное дерево а не лес, как на примере





Нужно удалять ребра с наибольшим весом при это
следить за связностью графа
(использовать dfs на каждом шаге не эффективно)

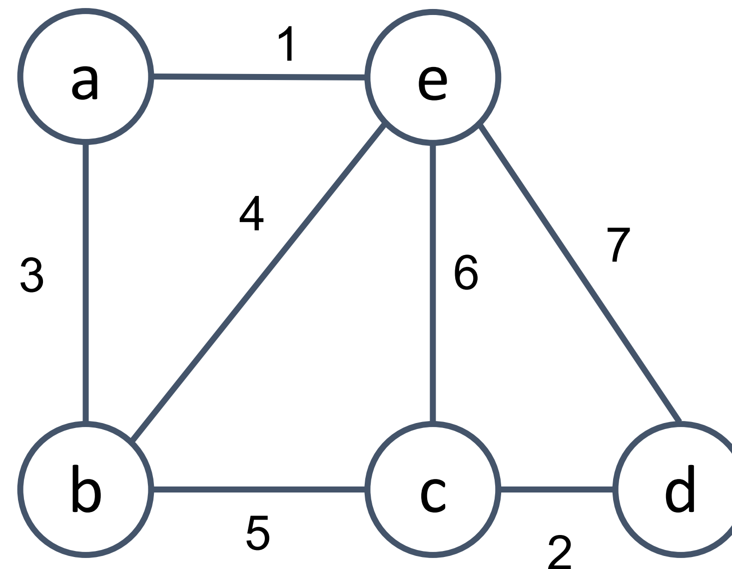
Как построить MST быстро и не
использовать много памяти (например,
красить все остовы)?

Примечание: для любого взвешенного графа,
может быть несколько разных MST
А как получить все возможные MST для G?

Алгоритм Краскала Для поиска MST

Идея:

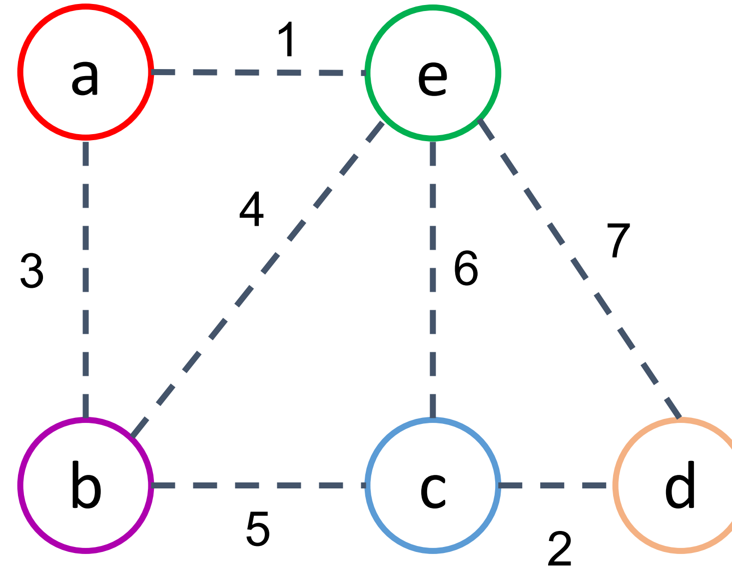
1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?							

Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST



Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$

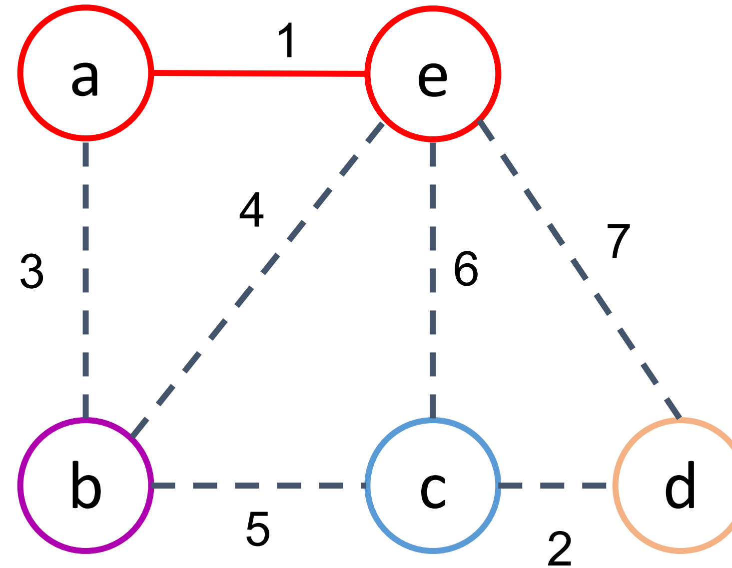
Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?							

Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:
1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+						

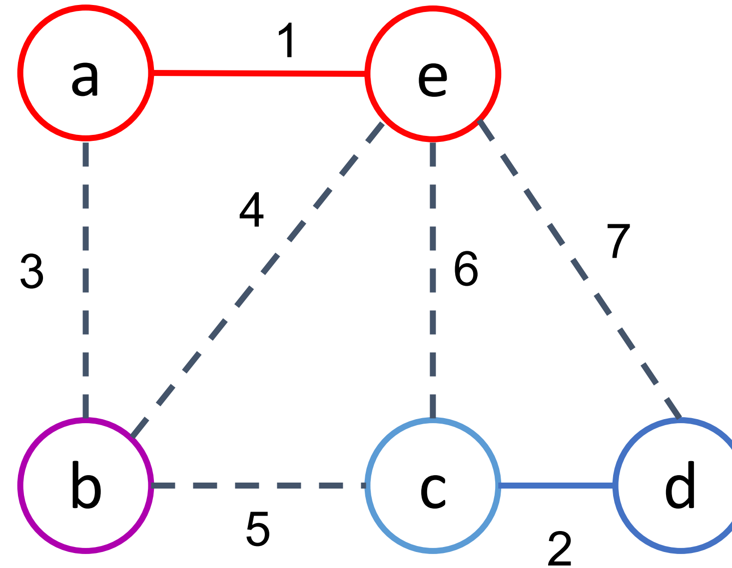
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+					

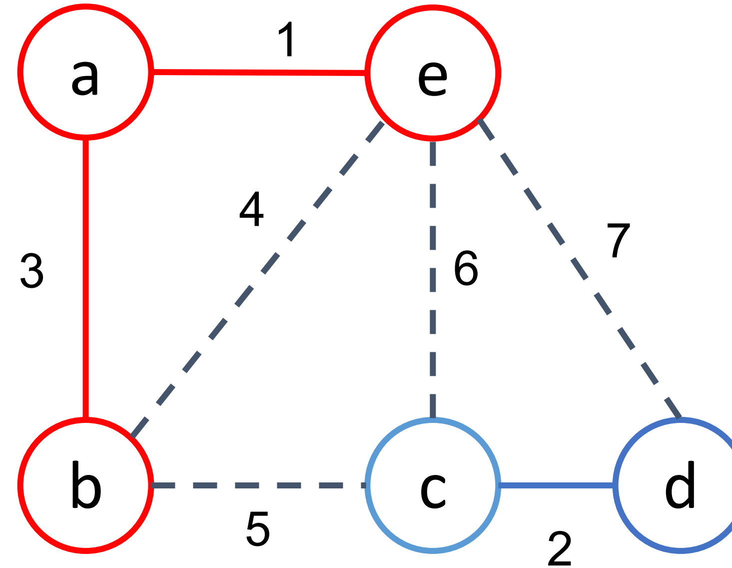
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+	+				

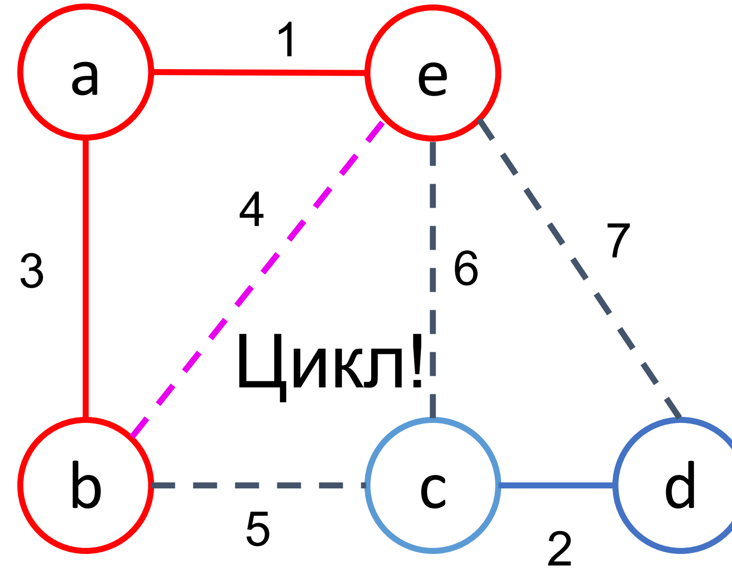
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+	+				

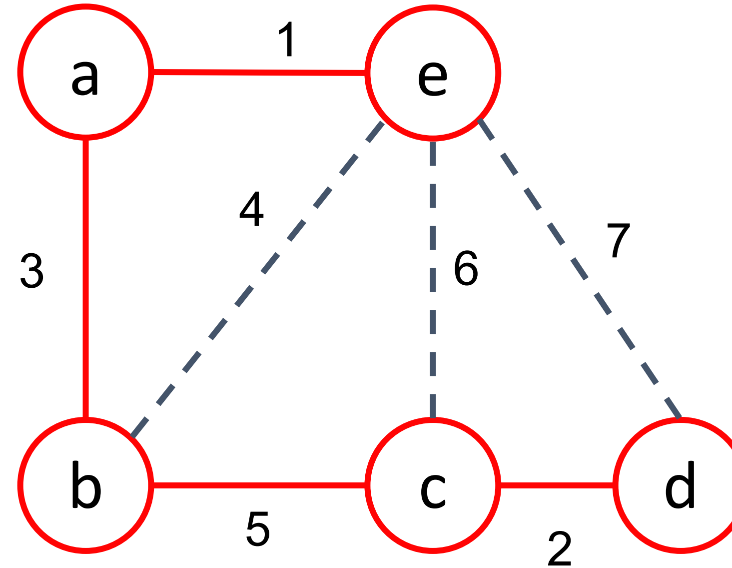
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+	+		+		

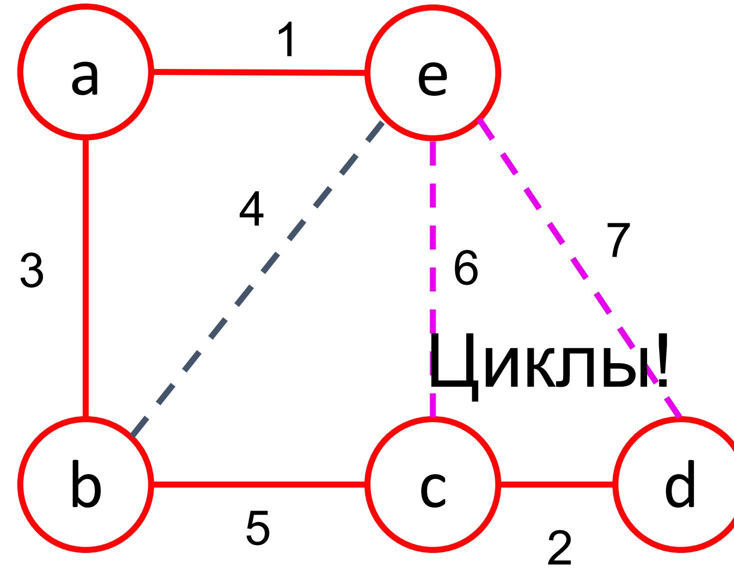
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+	+		+		

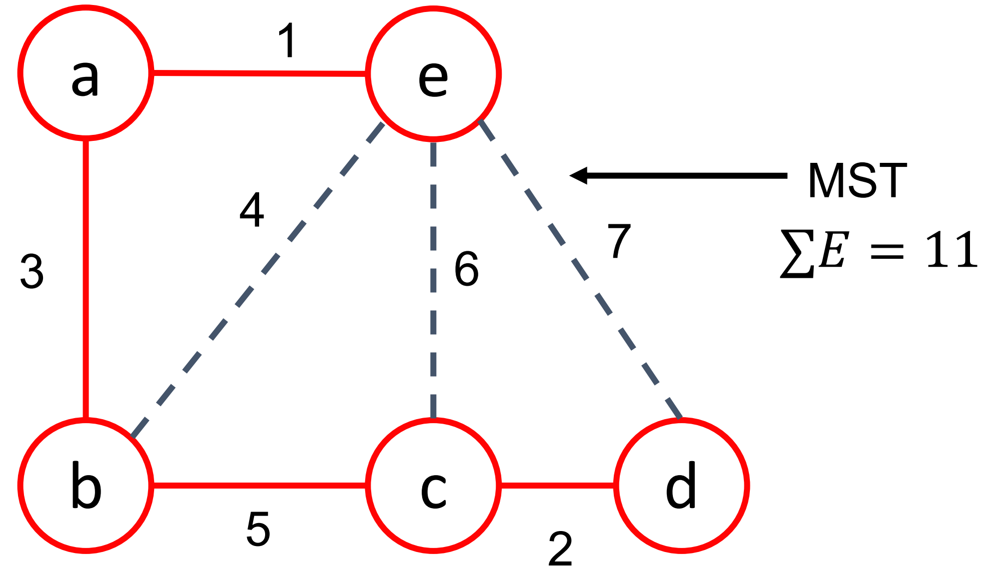
Входит ли это ребро в MST

Алгоритм Краскала Для поиска MST

Идея:

1) Сперва формируем ЛЕС из вершин графа
2) Вычисляем цикломатическое число (сколько ребер должно быть в остовном дереве)
3) Отсортируем все ребра по весу (по возрастанию)
4) Далее добавляем ребра с наименьшим весом, но так чтобы не образовывались циклы

$$Z = m - n + k = 7 - 5 + 1 = 3$$



Ребра (отсортированы)	ae	cd	ab	be	bc	ec	ed
Вес	1	2	3	4	5	6	7
В MST?	+	+	+		+		

Входит ли это ребро в MST

Вопросы

1) Визуально искать цикл быстро, но в реализации это затратная операция
Каждый раз запускать обход в глубину:
Если не считаем z то нужно просмотреть все ребра |E| * O(dfs) => Это очень затратно

2) Дерево строится в процессе из леса, а это значит, что нужно использовать дополнительные структуры:
Использовать массив меток ребер, что входят в MST это + O(|E|) памяти

3) Нужно балансировать между структурами
стартовый граф, по которому ищем
получаемое минимальное остовное дерево:
нужно строить копию графа, чтобы проверить образуется ли цикл при добавлении ребра + операции добавления и удаления, а еще нужно проходить все ребра исходного графа

Вопросы

1) Как представить граф для реализации, чтобы удобно было сортировать ребра по весу?

   удобно в массиве ребер O(|E|) вместо |V|*|V| или |V|*|E| если использовать матрицу инцидентности. |V|*|E| > 3|E|(массив ребер)
   {(u1, v1, weight1),(u2, v2, weight2),...}

2) Как получить результат: итоговое минимальное остовное дерево в виде графа?

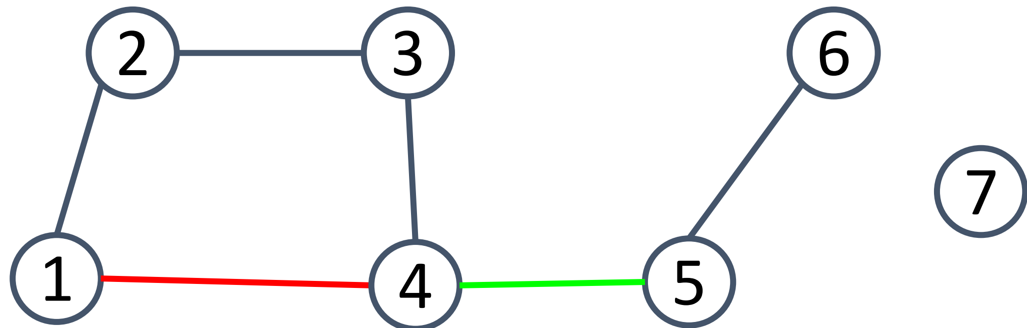
   Эффективно просто в массиве ребер добавить поле(метка) обозначающее принадлежность ребра MST;
   За O(|E|) получить массив ребер MST;
   Зная m-z можно получить список ребер MST менее чем за O(|E|): как только найдены все ребра из MST обход завершить

- Нужно более оптимально искать циклы в дереве
- Не дублировать граф

- dfs не подойдет
- матрица также
- Удобно использовать список ребер + поле для включения в MST

Для оптимизации поиска цикла в дереве нам поможет свойство дерева:
Если в дереве соединить две несмежные вершины появится цикл

Адаптируем св-во под лес

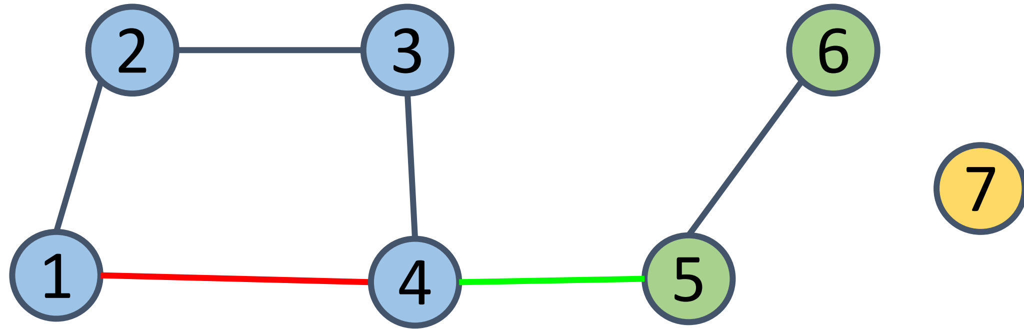


1) цикл 2) U деревьев леса

Вместо поиска цикла можно распознавать случай 1 и 2

Как?

Мы будем раскрашивать(помечать) компоненты



1) Цикл
(ребро соединяет две вершины одного цвета)

2) U деревьев леса
(ребро соединяет две вершины различных цветов)

Реализация

```
//MST – минимальный остов
//G(E, V) – граф

fun Kruskal-MST(G):
    MST = () //храним метки
    for u in G.V:
        createColor(u) //создаем лес
    Sort(G.E) //сортируем ребра
    for (u, v) in G.E:
        if u.color != v.color //не цикл
        then
            MST = MST + {(u, v)}
            unionColor(u, v) //красим в один цвет
    return MST
```

Итог

Мы получили MST без использования dfs, а также дублирования графа

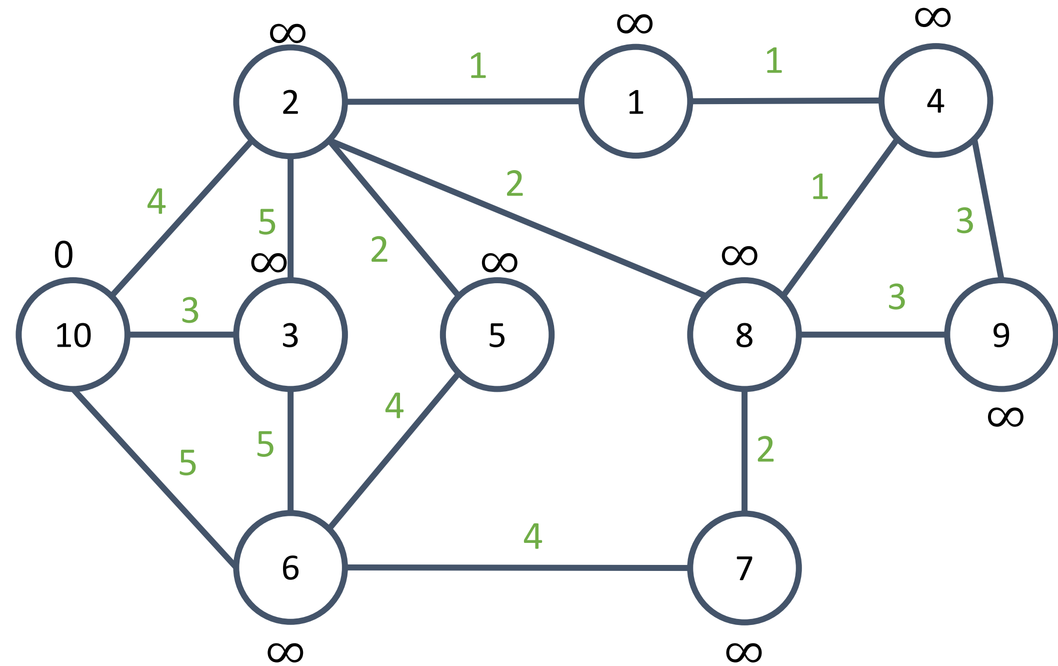
Мы использовали:
- Массив меток ребер
- Массив цветов вершин
- Алгоритм закраски вершин по цвету

Алгоритм Прима

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										



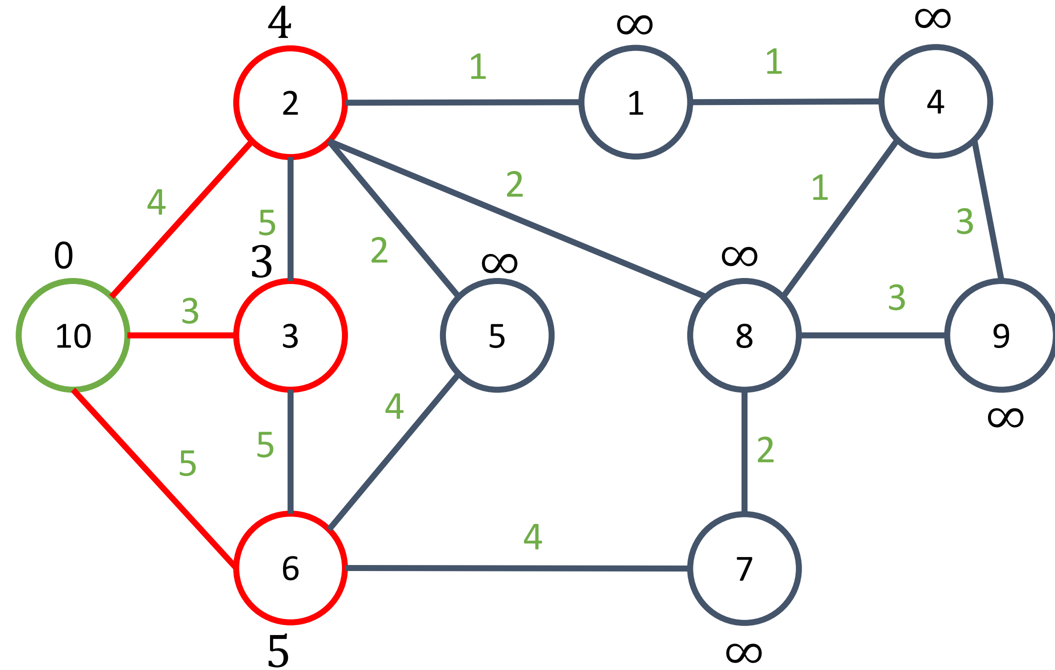
В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима

Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2										
3										
4										
5										
6										
7										
8										
9										
10										



В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Алгоритм Прима

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

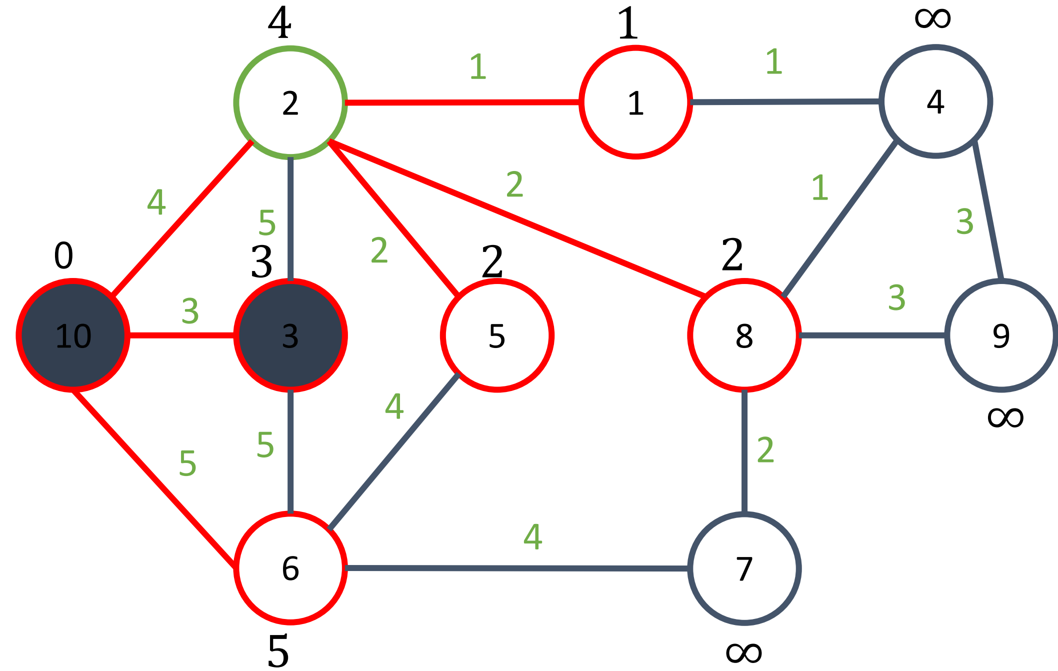
Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3										
4										
5										
6										
7										
8										
9										
10										

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима

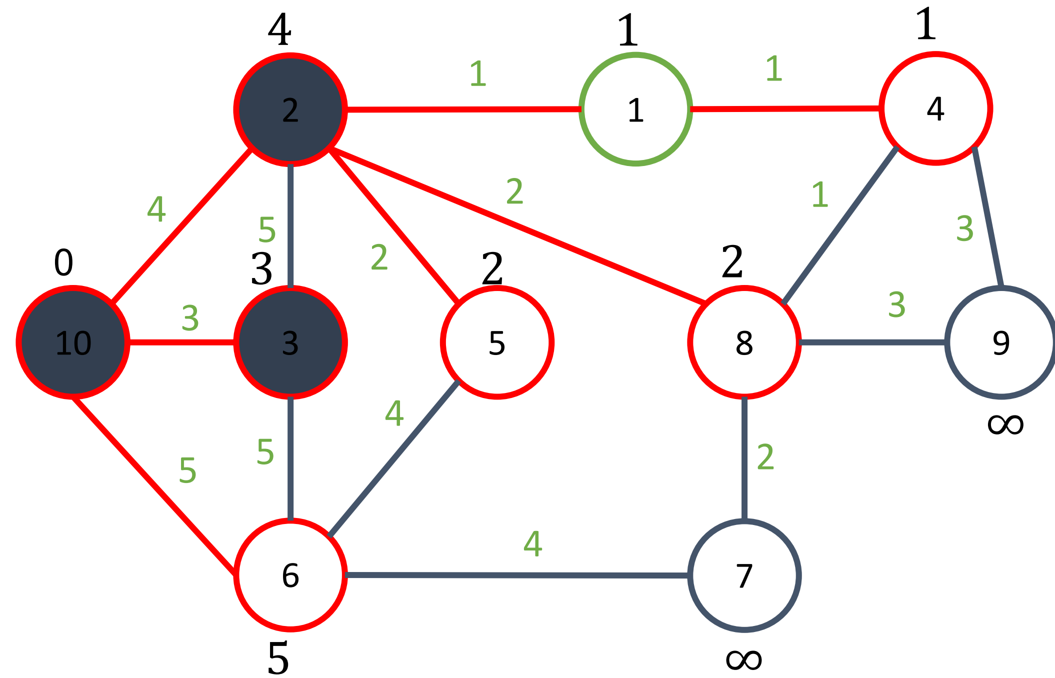


Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4										
5										
6										
7										
8										
9										
10										

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:
1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима



Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5										
6										
7										
8										
9										
10										

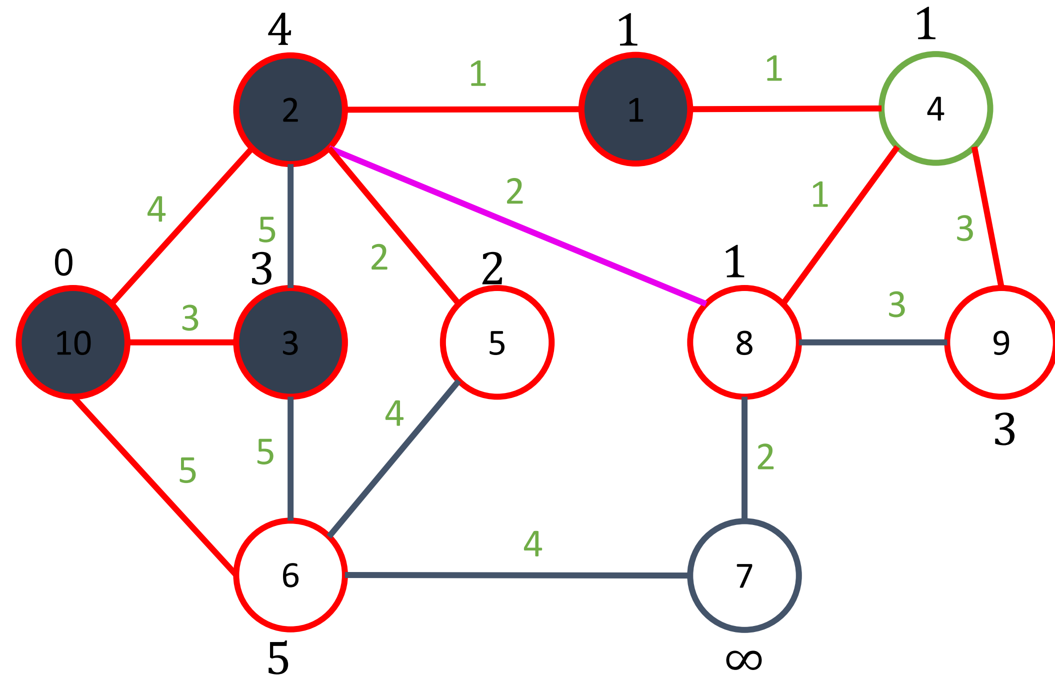
В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима

Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6										
7										
8										
9										
10										



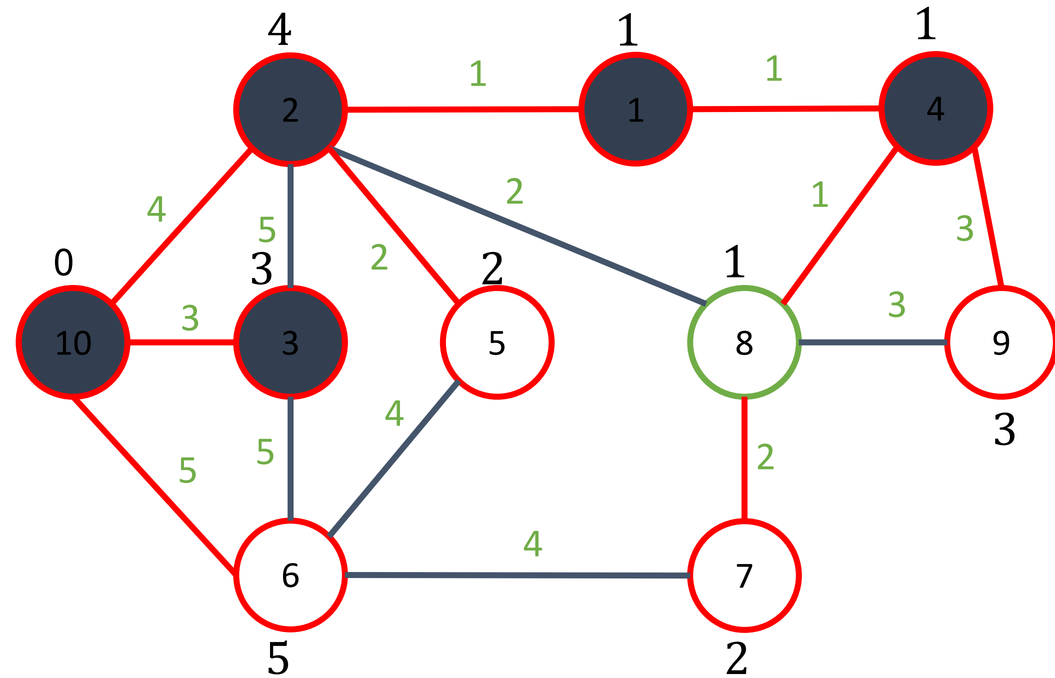
В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7										
8										
9										
10										

Алгоритм Прима

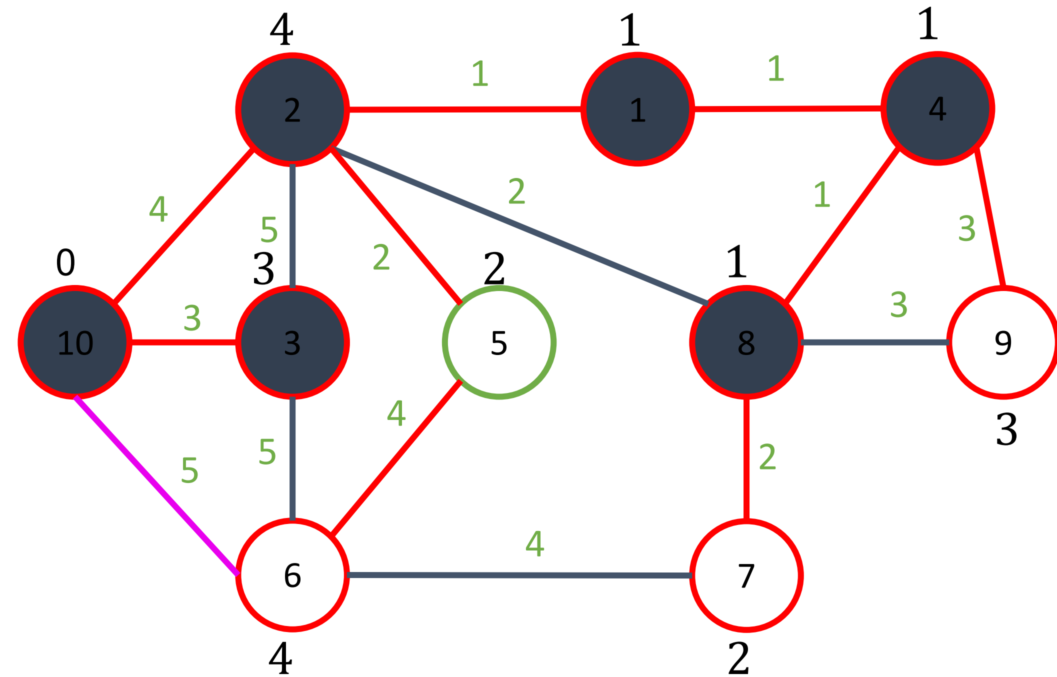


В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
    a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
        i) ребро не добавлено в MST
        ii) Ребро не образует цикл
    b) Добавляем ребро в MST
    c) Помечаем вершину

Алгоритм Прима

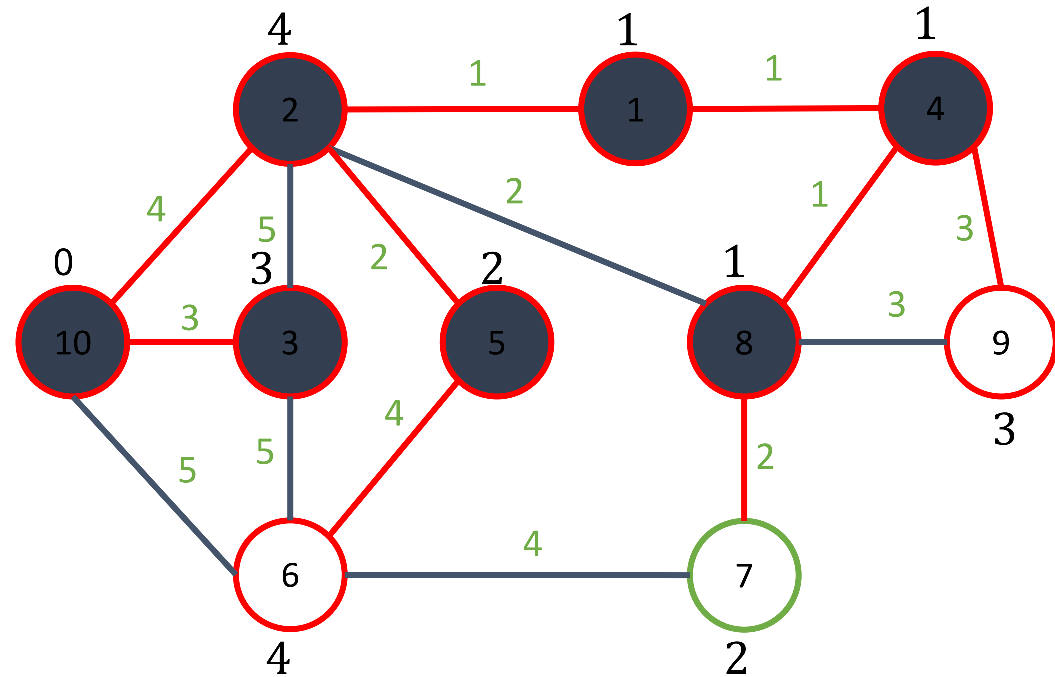


Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7	1	4	3	1	2	4	2	1	3	0
8										
9										
10										

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:
1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима



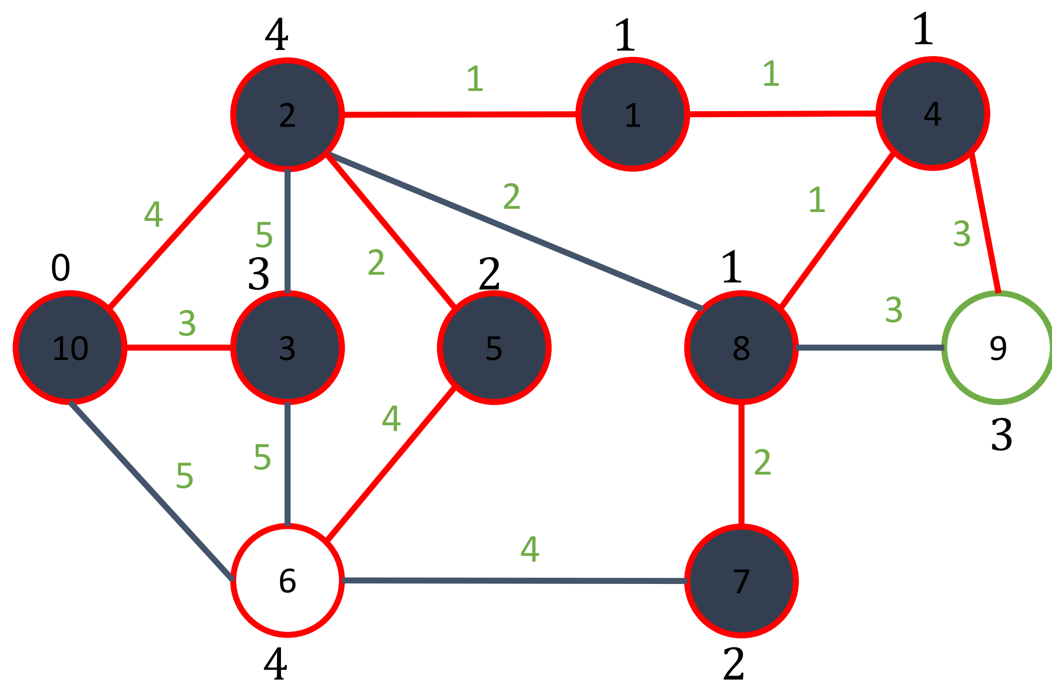
Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7	1	4	3	1	2	4	2	1	3	0
8	1	4	3	1	2	4	2	1	3	0
9										
10										

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:
1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима

Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7	1	4	3	1	2	4	2	1	3	0
8	1	4	3	1	2	4	2	1	3	0
9	1	4	3	1	2	4	2	1	3	0
10										

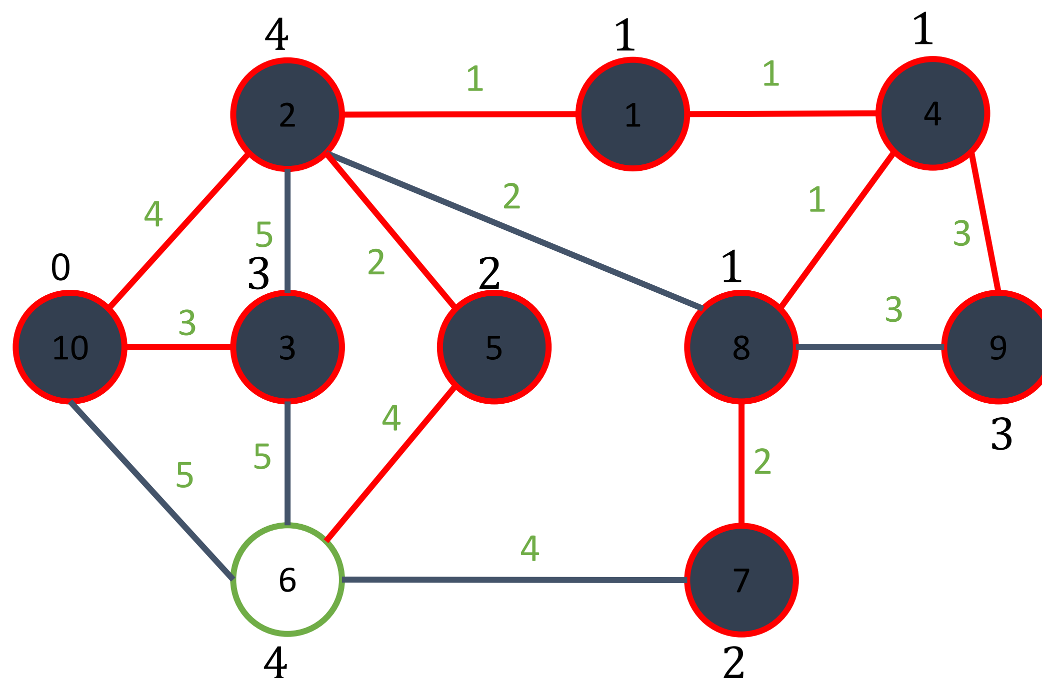


В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:

1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима

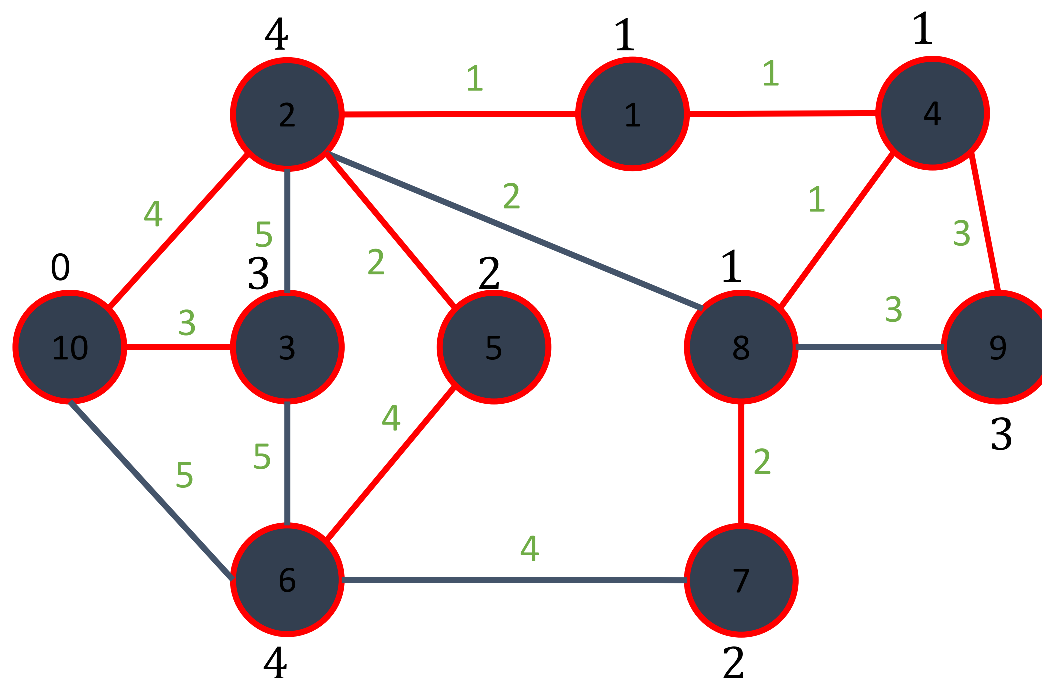


Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7	1	4	3	1	2	4	2	1	3	0
8	1	4	3	1	2	4	2	1	3	0
9	1	4	3	1	2	4	2	1	3	0
10	1	4	3	1	2	4	2	1	3	0

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

Идея:
1) Произвольно выбираем стартовую вершину, помечаем ее
2) Пока не просмотрим все вершины
   a) Выбираем от помеченных вершин ребро с наименьшим весом до непомеченной
      i) ребро не добавлено в MST
      ii) Ребро не образует цикл
   b) Добавляем ребро в MST
   c) Помечаем вершину

Алгоритм Прима



Шаг	1	2	3	4	5	6	7	8	9	10
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
1	∞	4	3	∞	∞	5	∞	∞	∞	0
2	∞	4	3	∞	∞	5	∞	∞	∞	0
3	1	4	3	∞	2	5	∞	2	∞	0
4	1	4	3	1	2	5	∞	2	∞	0
5	1	4	3	1	2	5	∞	1	3	0
6	1	4	3	1	2	5	2	1	3	0
7	1	4	3	1	2	4	2	1	3	0
8	1	4	3	1	2	4	2	1	3	0
9	1	4	3	1	2	4	2	1	3	0
10	1	4	3	1	2	4	2	1	3	0

В таблице будем хранить минимальную длину ребра, которое соединяет эту вершину с остальным деревом

! Важно:
Мы не сохраняем выбранные ребра и никак их не помечаем => нужно будет придумать на этапе реализации как получить MST по ребрам

Как сохранять MST, если мы ничего не помечаем?
На каждом шаге мы пересчитываем min ребро и именно на этом этапе необходимо записать из какой вершины идет min ребро.

Будем использовать массив предков:

1	2	3	4	5	6	7	8	9	10
2	10	10	1	2	5	8	4	4	null

На каждом шаге при пересчете, если меняем значение min ребра, то в массив записываем последнюю помеченную вершину т.к. из нее рассматриваются ребра

Каждый раз искать min среди непомеченных вершин это O(|V|) т.к. придется просмотреть все чистые вершины

Можно оптимизировать поиск

Используем min кучу:
- Поиск min – O(1)
- Перестроение кучи – O(|V|log|V|)
- Построение кучи – O(|V|)

Мы строим кучу вне цикла, а само min значение ищем в цикле

Реализация

Коротко:

1) Создать приоритетную очередь, где стартовая вершина - 0, остальные - бесконечность ∞
2) Пока очередь не пуста
   a) Выбрать минимальную вершину из очереди и убрать ее
   b) Просмотреть все ребра из нее до всех вершин в очереди.
      i) Если рассматриваемое ребро меньше текущего пути
         (1) Запоминаем новый путь
         (2) Запоминаем значение нового пути
         (3) Изменяем положение вершины в очереди

```
//G – исходный граф
//w – весовая функция w(u, v) – вес ребра u v
fun prim-MST(G, s):
   for v in G.V:
      v.key = ∞ //изначально до всех вершин min ребро ∞
      v.p = null //изначально предков нет
   s.key = 0 //стартовая вершина
   Q = createHeap(G.V) //строим min кучу
   while not Q.isEmpty():
      u = Q.extractMin()
      for v in G.E[u]:
         if v.key > w(u, v):
            v.p = u //сохраняем предка
            v.key = w(u, v) //вес нового min ребра
            Q.decreaseKey(v, v.key) //меняем приоритет вершины
```

```
//G – исходный граф
//w – весовая функция w(u, v) – вес ребра u v
fun prim-MST(G, s):
    for v in G.V:
        v.key = ∞ //изначально до всех вершин min ребро ∞
        v.p = null //изначально предков нет
    s.key = 0 //стартовая вершина
    Q = createHeap(G.V) //строим min кучу
    while not Q.isEmpty():
        u = Q.extractMin()
        for v in G.E[u]:
            if v.key > w(u, v):
                v.p = u //сохраняем предка
                v.key = w(u, v) //вес нового min ребра
                Q.decreaseKey(v, v.key) //меняем приоритет вершины
```

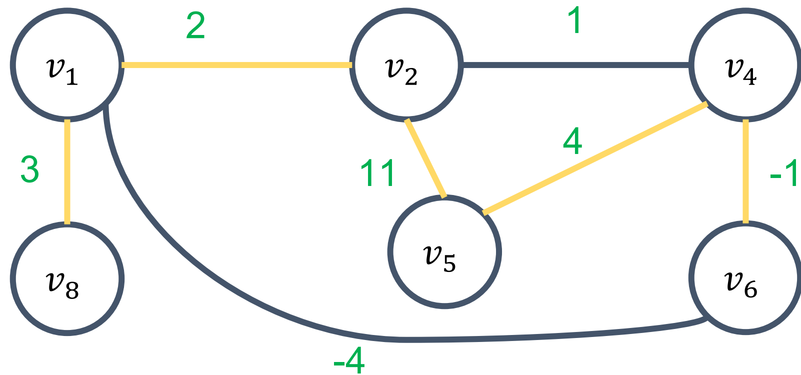
Поиск наикратчайшего пути

Путь (маршрут) - последовательность вершин и ребер вида $v_1 e_1 v_2 e_3 v_4 = v_1 v_2 v_4 = p$

Длина пути количество ребер в нем (НЕ взвешенный граф)

Вес пути - сумма веслом всех ребер пути (взвешенный граф)

Пример



$p = v_8 v_1 v_2 v_5 v_4 v_6 = v_8 \xrightarrow{p} v_6$

Длина пути p= 5

Вес пути p:

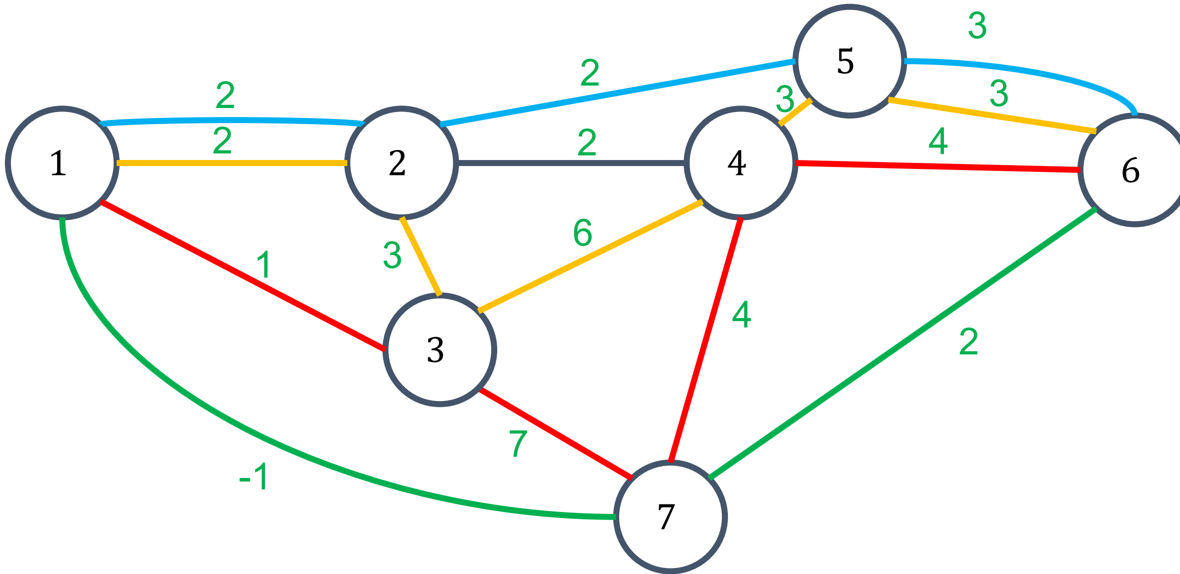
$$\sum_p w = 3 + 2 + 11 + 4 - 1 = 19$$

Поиск наикратчайшего пути

Наикратчайший путь - путь с наименьшим весом (их может быть несколько разных, но с одним весом)

Примечание: вес наикратчайшего пути из u в v будет наименьшим из всевозможных или равен бесконечности, если пути из u в v нет.

Пример



1) $v_1 \overset{p_1}{\rightarrow} v_6 = v_1 v_2 v_5 v_6$

$$\sum_{p_1} w = 2 + 2 + 3 = 7$$

2) $v_1 \overset{p_2}{\rightarrow} v_6 = v_1 v_3 v_7 v_4 v_6$

$$\sum_{p_2} w = 1 + 7 + 4 + 4 = 16$$

3) $v_1 \overset{p_3}{\rightarrow} v_6 = v_1 v_2 v_3 v_4 v_5 v_6$

$$\sum_{p_3} w = 2 + 3 + 6 + 3 + 3 = 17$$

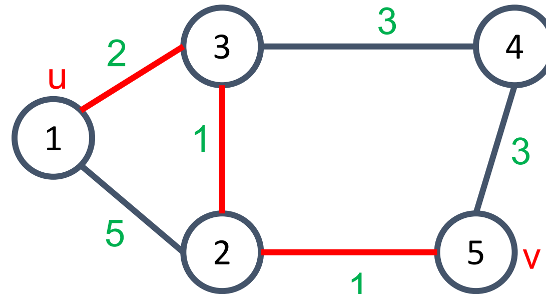
4) $v_1 \overset{p_4}{\rightarrow} v_6 = v_1 v_7 v_6$

$$\sum_{p_4} w = -1+2=1 \text{ (min)}$$

p_4- наименьший путь среди p_1, p_2, p_3, p_4

Варианты задачи поиска наикратчайшего пути

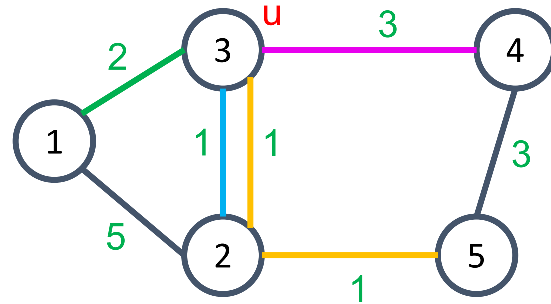
1. Между заданными: из u в v



$p = v_1 v_3 v_2 v_5$

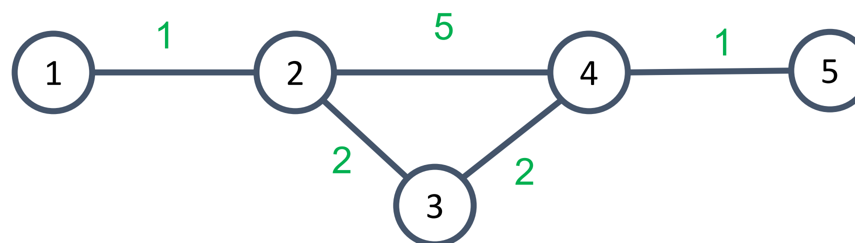
$\sum w = 2 + 1 + 1 = 4$

2. От заданной до остальных



$v_3 \to v_1$	$v_3 \to v_2$	$v_3 \to v_5$	$v_3 \to v_4$
$p = v_3 v_1$	$p = v_3 v_2$	$p = v_3 v_2 v_5$	$p = v_3 v_4$
$\sum w = 2$	$\sum w = 1$	$\sum w = 1 + 1 = 2$	$\sum w = 3$

3. Между вершинами (от всех до всех)



$v_1 \to v_2$, $v_1 \to v_3$, $v_1 \to v_4$, $v_1 \to v_5$, $v_3 \to v_1$, $v_3 \to v_2$, $v_3 \to v_4$, $v_3 \to v_5$

$v_2 \to v_1$, $v_2 \to v_3$, $v_2 \to v_4$, $v_2 \to v_5$, $v_4 \to v_1$, $v_4 \to v_2$, $v_4 \to v_3$, $v_4 \to v_5$

$v_5 \to v_1$, $v_5 \to v_2$, $v_5 \to v_3$, $v_5 \to v_4$

Использовать переборные алгоритмы не эффективно для поиска!

5 раз выполняем поиск кр. путей от вершины до всех

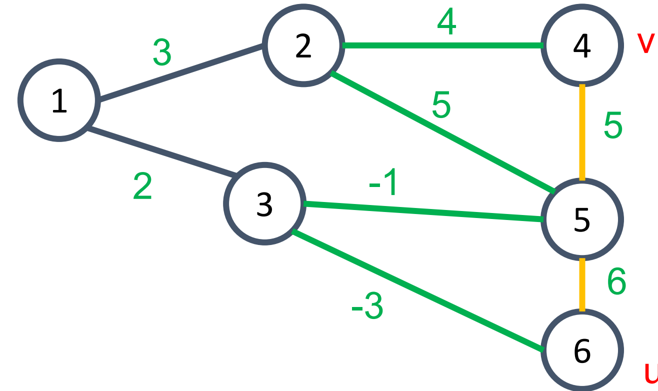
Что может содержать кратчайший путь?

- Ребра отрицательного веса

такие ребра уменьшают вес пути, но
увеличивают число пройденных ребер

не все алгоритмы могут корректно работать с такими ребрами
(в жизни отрицательные ребра встречаются крайне редко)

Например алгоритм Дейкстры



$p_1 = v_6 v_3 v_5 v_2 v_4$

$$\sum_{p_1} w = -3 - 1 + 5 + 4 = 5$$

$p_2 = v_6 v_5 v_4$

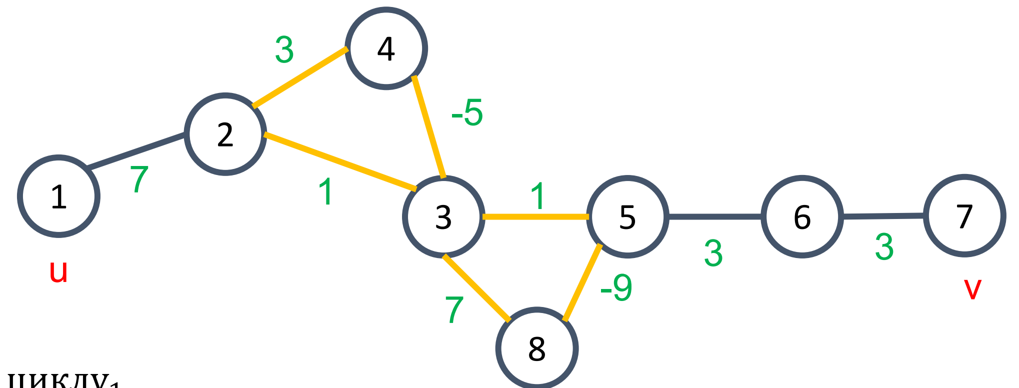
$$\sum_{p_2} w = -3 - 1 + 5 + 4 = 5$$

- Циклы отрицательного веса

$\text{цикл}_1 = v_2 v_3 v_4 v_2 \; w = -1$

$\text{цикл}_2 = v_3 v_5 v_8 v_3 \; w = -1$

$\text{цикл}_3 = v_2 v_4 v_3 v_5 v_8 v_3 v_2 \; w = -2$



$1 \to 7$ может содержать k раз цикл_1

Пройдем 200 раз по циклу_1

$$\sum_{p} w = 7 + 200 \cdot (-1) + 7 - 9 + 3 + 3 = -189$$

При $k \to \infty$;
$\text{w} \to -\infty$

Что может содержать кратчайший путь?

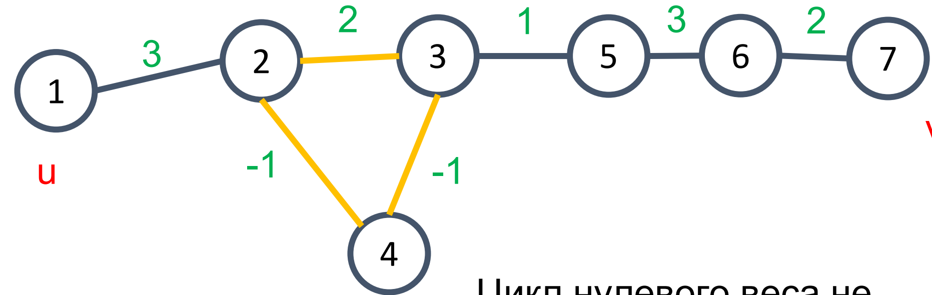
- Циклы нулевого веса

  $\text{цикл} = v_2 v_3 v_4 v_2$

  $w = 0$

$1 \overset{p_1}{\rightarrow} 7; \quad p_1 = v_1 \underbrace{v_2 v_3 v_4 v_2} v_3 v_5 v_6 v_7$

цикл; k раз



Цикл нулевого веса не влияет на вес пути, но он увеличивает число пройденных ребер

Если отбросить такой цикл, то вес пути не изменится

$1 \overset{p_1}{\rightarrow} 7; \quad p_1 = v_1 v_2 v_3 v_5 v_6 v_7$

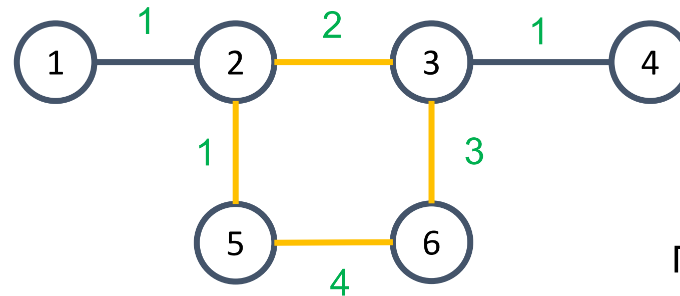
- Циклы положительного веса

  $\text{цикл} = v_2 v_5 v_6 v_3 v_2$

  $w = 10$

$1 \overset{p_1}{\rightarrow} 4; \quad p_1 = v_1 \underbrace{v_2 v_5 v_6 v_3 v_2} v_3 v_4$

цикл; k раз



$$\sum_{p1} w = 1 + 10 \cdot k + 2 + 1$$

При $k \rightarrow \infty$; $\mathrm{w} \rightarrow \infty$ Путь не будет наикратчайшим

Вывод: наикратчайший путь ацикличен

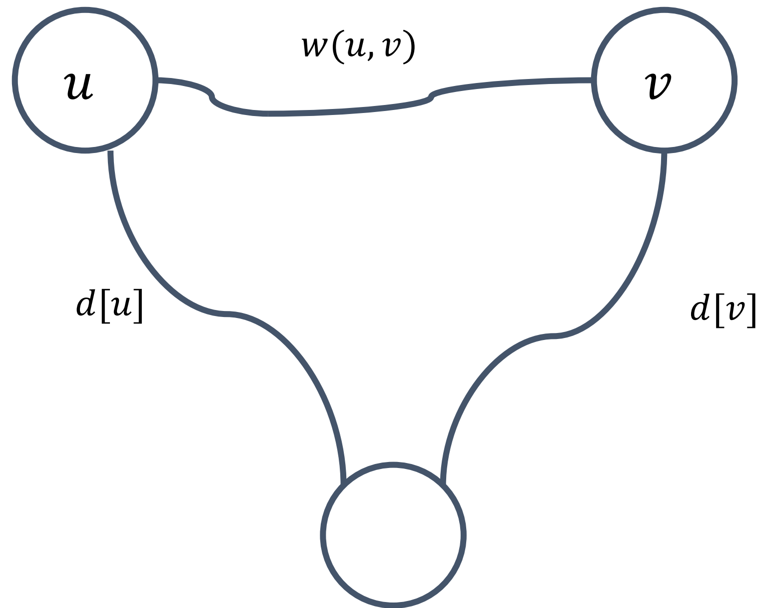
Ацикличный путь содержит не более |V| вершин и |V| - 1 ребер

Ослабление ребра

Ослабление ребра (релаксация: Relax) - попытка улучшить найденный путь до вершины

d – массив расстояний до вершин
$d[u]$- расстояние до вершины u

$w(u, v)$- вес пути от u до v



Ослабление ребра

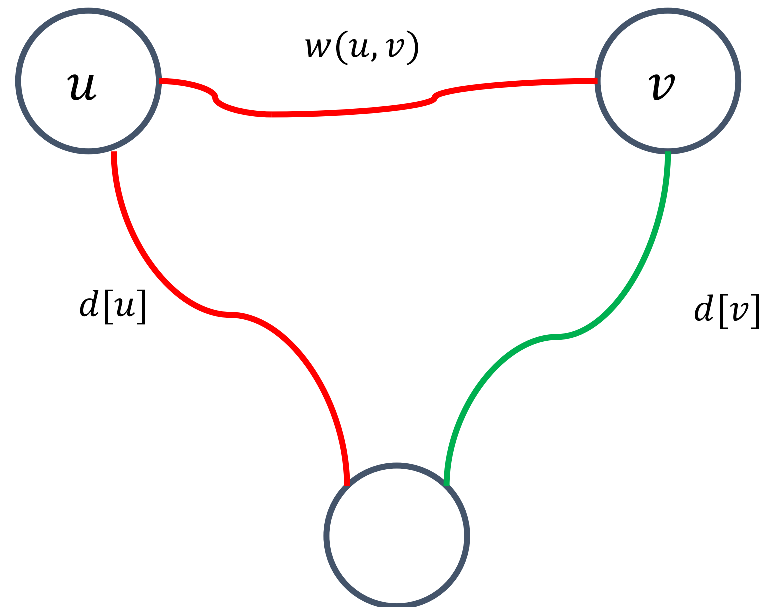
Ослабление ребра (релаксация: Relax) - попытка улучшить найденный путь до вершины

d – массив расстояний до вершин
$d[u]$- расстояние до вершины u

$w(u,v)$- вес пути от u до v

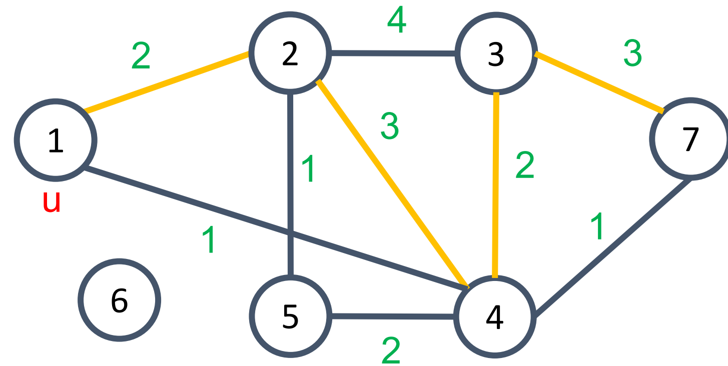
```
Relax(u, v, w[u,v]):
if (d[v] > d[u] + w(u,v))
   d[v] = d[u] + w(u,v)
```

Relax$(u, v, w(u, v))$



Как восстановить наикратчайший путь?

Будем использовать массив предков p.
$p[v]$- Вершина предшествующая вершине v на кратчайшем пути (null если нет)



Например для пути $p_1 = v_1 v_2 v_4 v_3 v_7$
Массив предков будет выглядеть так:

v	1	2	3	4	5	6	7
пред v	null	1	4	2	null	null	3

Заполнять массив предков будем при релаксации ребра.

```
Relax(u, v, w[u,v]):
if (d[v] > d[u] + w(u,v))
    d[v] = d[u] + w(u,v)
    p[v] = u
```

Восстановить путь из массива можно с помощью цикла(начинаем с конечной вершины):

```
i = 7
print(i)
while p[v] is not null:
    print(i)
    i = p[i]
```

Инициализация дополнительных структур

Нужно заполнить d бесконечностями, так как на старте нет найденных кратчайших путей

Массив предков заполнить null, так как предков нет на старте ни для какой вершины

```
//s - стартовая вершина
InitSource(G, s):
    for v in G.V:
        d[v] = ∞
        p[v] = null
    d[s] = 0
```

Такую инициализацию необходимо выполнять при использовании любых алгоритмов поиска наикратчайших путей

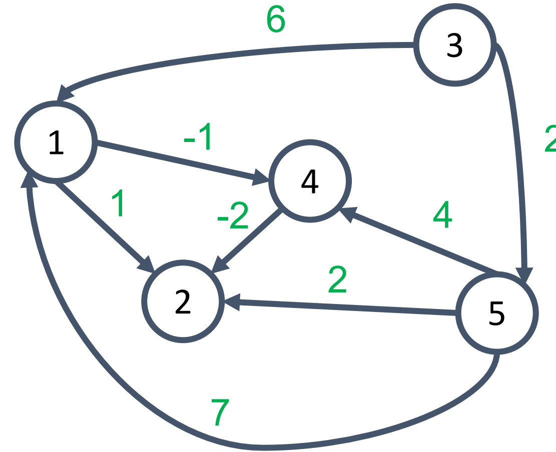
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



Топологически отсортируем данный граф используя алгоритм Демукрона (последовательно удаляем стоки)

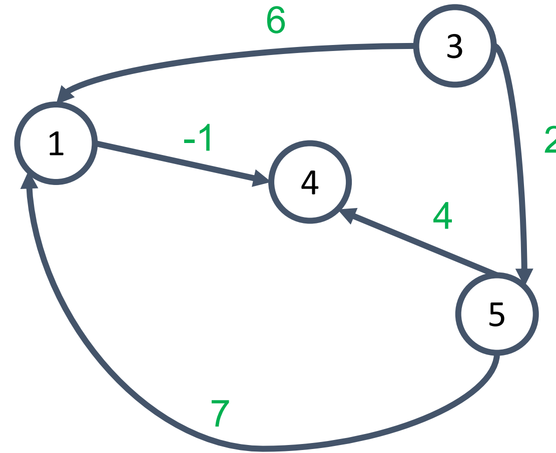
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



2

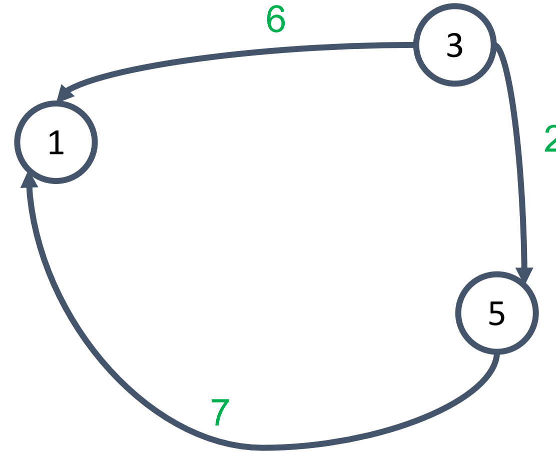
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞

Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞

3

5 1 4 2

Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞

Dag: кратчайшие пути в ациклическом ориентированном графе

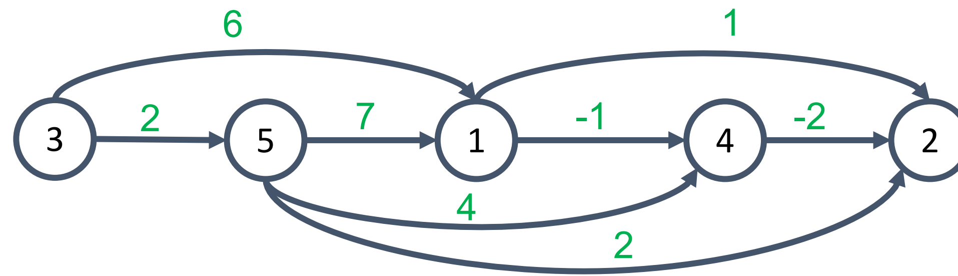
Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞

Исходный граф, после выполнения сортировки



Теперь в порядке топологической сортировки будем производить релаксацию ребер

Таким образом мы найдем кратчайшие пути от s до всех остальных

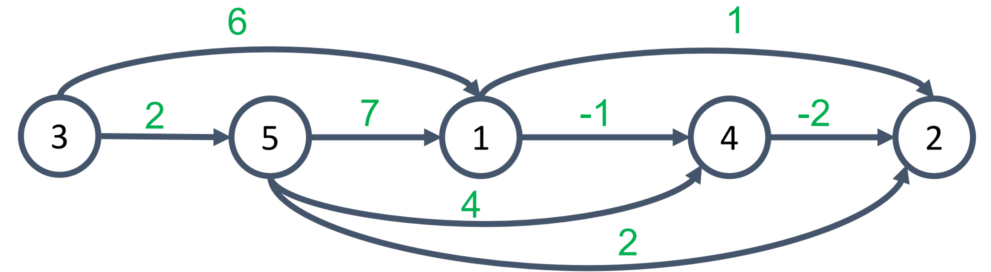
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



p по шагам

шаг	1	2	3	4	5
0	\	\	\	\	\
1					
2					
3					
4					

d по шагам

шаг	1	2	3	4	5
0	∞	∞	0	∞	∞
1					
2					
3					
4					

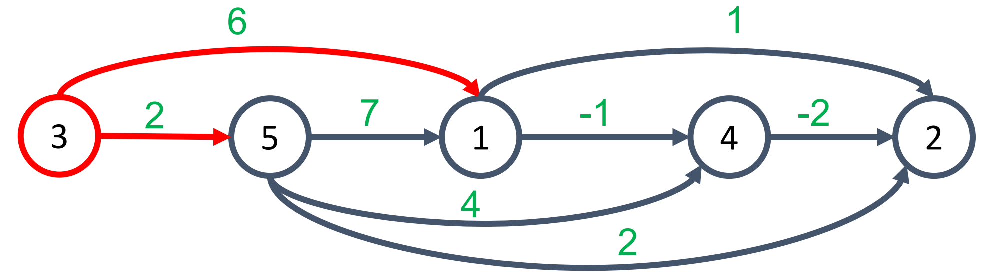
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



p по шагам

шаг	1	2	3	4	5
0	\	\	\	\	\
1	3	\	\	\	3
2					
3					
4					

d по шагам

шаг	1	2	3	4	5
0	∞	∞	0	∞	∞
1	6	∞	0	∞	2
2					
3					
4					

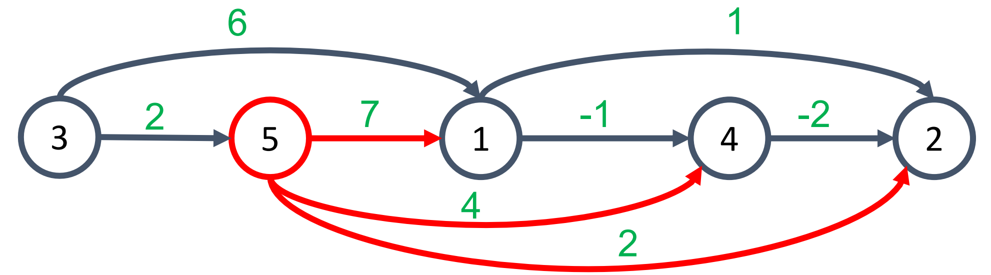
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



p по шагам

шаг	1	2	3	4	5
0	\	\	\	\	\
1	3	\	\	\	3
2	3	5	\	5	3
3					
4					

d по шагам

шаг	1	2	3	4	5
0	∞	∞	0	∞	∞
1	6	∞	0	∞	2
2	6	4	0	6	2
3					
4					

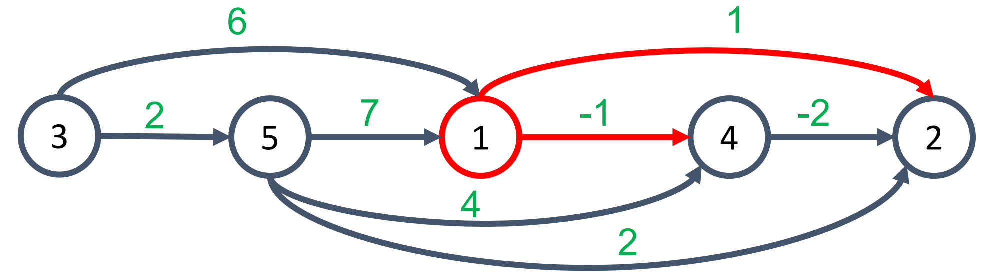
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



p по шагам

шаг	1	2	3	4	5
0	\	\	\	\	\
1	3	\	\	\	3
2	3	5	\	5	3
3	3	5	\	1	3
4					

d по шагам

шаг	1	2	3	4	5
0	∞	∞	0	∞	∞
1	6	∞	0	∞	2
2	6	4	0	6	2
3	6	4	0	5	2
4					

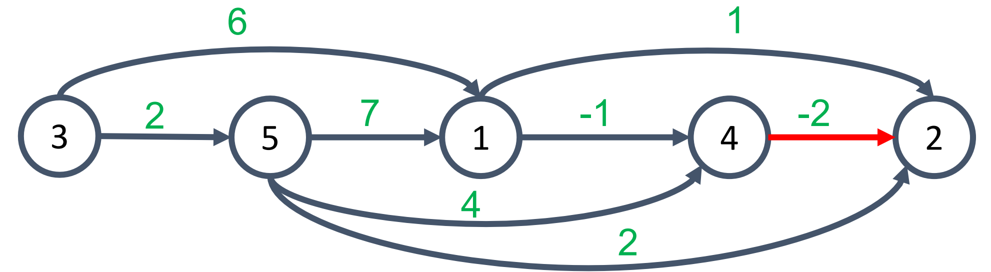
Dag: кратчайшие пути в ациклическом ориентированном графе

Суть:

Ослабление ребер в порядке топологической сортировки

Граф G задан матрицей смежности W, где W[u, w] - вес ребра (u, v)

	1	2	3	4	5
1	∞	1	∞	-1	∞
2	∞	∞	∞	∞	∞
3	6	∞	∞	∞	2
4	∞	-2	∞	∞	∞
5	7	2	∞	4	∞



p по шагам

шаг	1	2	3	4	5
0	\	\	\	\	\
1	3	\	\	\	3
2	3	5	\	5	3
3	3	5	\	1	3
4	3	4	\	1	3

d по шагам

шаг	1	2	3	4	5
0	∞	∞	0	∞	∞
1	6	∞	0	∞	2
2	6	4	0	6	2
3	6	4	0	5	2
4	6	3	0	5	2

Реализация

O(|V|+|E|)

```
        DAG(G, s):
O(|V|+|E|) → TS = TopSort(G)
O(|V|)     → InitSource(G, s)

O(|E|)       for u in TS:
                 for v in G.W[u]:
                     Relax(u, v, w(u, v))
```

Примечание:
TS – стек вершин

Чтобы искать путь от любой вершины до всех остальных мы можем пропускать все вершины, которые выше нашей стартовой

Алгоритм
- Работает с отрицательными ребрами
- Не работает с циклами
- Проходит все ребра лишь один раз

Алгоритм Беллмана-Форда

Поиск кратчайших путей от одной до всех вершин

Суть:

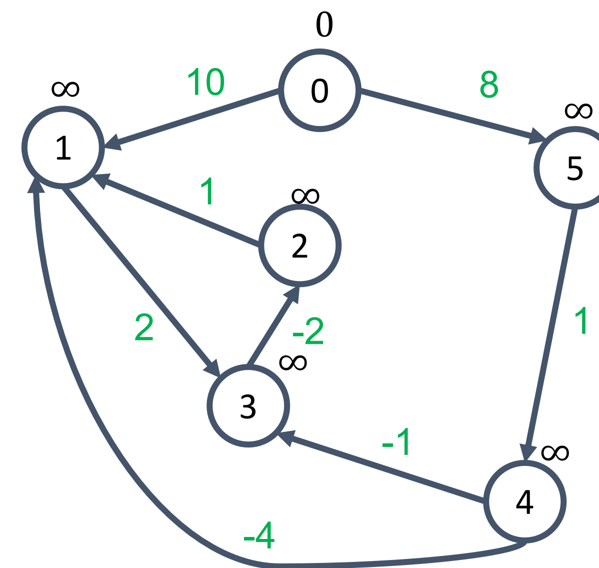
ослаблять все ребра графа |V| - 1 раз

Граф для алгоритма удобно хранить списком ребер

Алгоритм работает с отрицательными ребрами

Алгоритм устойчив к циклам отрицательного веса и позволяет их обнаружить

Цикл отрицательного веса можно восстановить



Список ребер

Вес	u	v
10	0	1
8	0	5
2	1	3
1	2	1
-2	3	2
-4	4	1
-1	4	3
1	5	4

d по шагам

шаг	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1						
2						
3						
4						
5						
6						

Алгоритм Беллмана-Форда

Поиск кратчайших путей от одной до всех вершин

Суть:

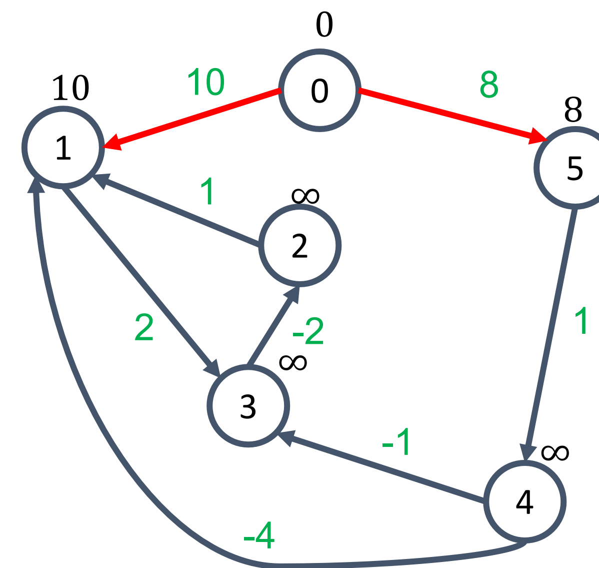
ослаблять все ребра графа |V| - 1 раз

Граф для алгоритма удобно хранить списком ребер

Алгоритм работает с отрицательными ребрами

Алгоритм устойчив к циклам отрицательного веса и позволяет их обнаружить

Цикл отрицательного веса можно восстановить



Список ребер

Вес	u	v
10	0	1
8	0	5
2	1	3
1	2	1
-2	3	2
-4	4	1
-1	4	3
1	5	4

d по шагам

шаг	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1	0	10	∞	∞	∞	8
2						
3						
4						
5						
6						

Алгоритм Беллмана-Форда

Поиск кратчайших путей от одной до всех вершин

Суть:

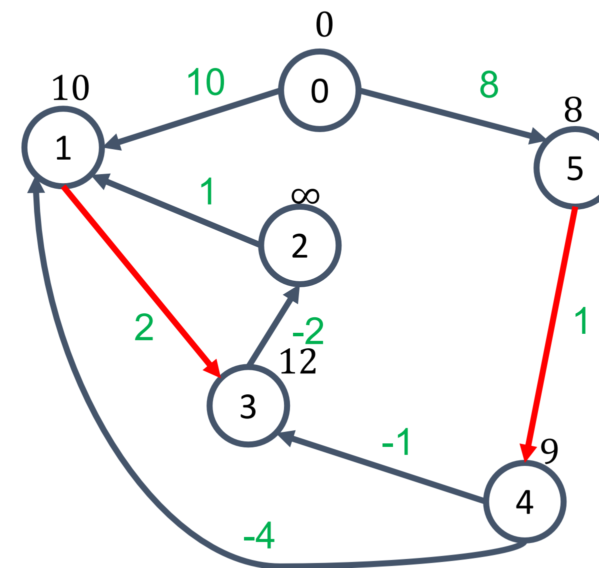
ослаблять все ребра графа |V| - 1 раз

Граф для алгоритма удобно хранить списком ребер

Алгоритм работает с отрицательными ребрами

Алгоритм устойчив к циклам отрицательного веса и позволяет их обнаружить

Цикл отрицательного веса можно восстановить



Список ребер

Вес	u	v
10	0	1
8	0	5
2	1	3
1	2	1
-2	3	2
-4	4	1
-1	4	3
1	5	4

d по шагам

шаг	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1	0	10	∞	∞	∞	8
2	0	10	∞	12	9	8
3						
4						
5						
6						

Алгоритм Беллмана-Форда

Поиск кратчайших путей от одной до всех вершин

Суть:

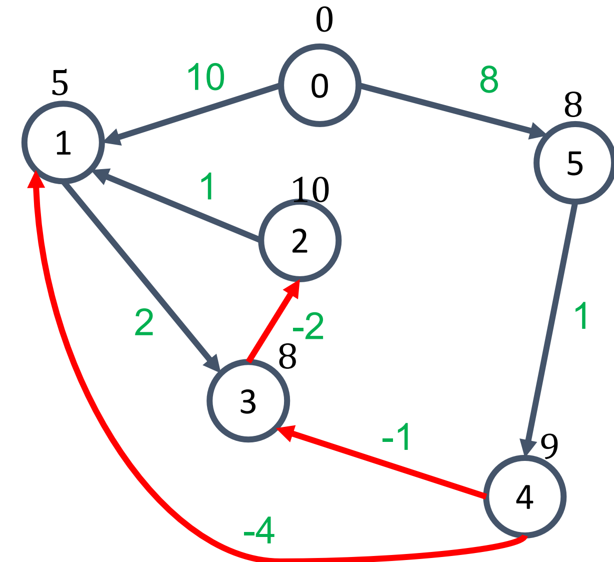
ослаблять все ребра графа |V| - 1 раз

Граф для алгоритма удобно хранить списком ребер

Алгоритм работает с отрицательными ребрами

Алгоритм устойчив к циклам отрицательного веса и позволяет их обнаружить

Цикл отрицательного веса можно восстановить



Список ребер

Вес	u	v
10	0	1
8	0	5
2	1	3
1	2	1
-2	3	2
-4	4	1
-1	4	3
1	5	4

d по шагам

шаг	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1	0	10	∞	∞	∞	8
2	0	10	∞	12	9	8
3	0	10	∞	12	9	8
4	0	5	10	8	9	8
5						
6						

Алгоритм Беллмана-Форда

Поиск кратчайших путей от одной до всех вершин

Суть:

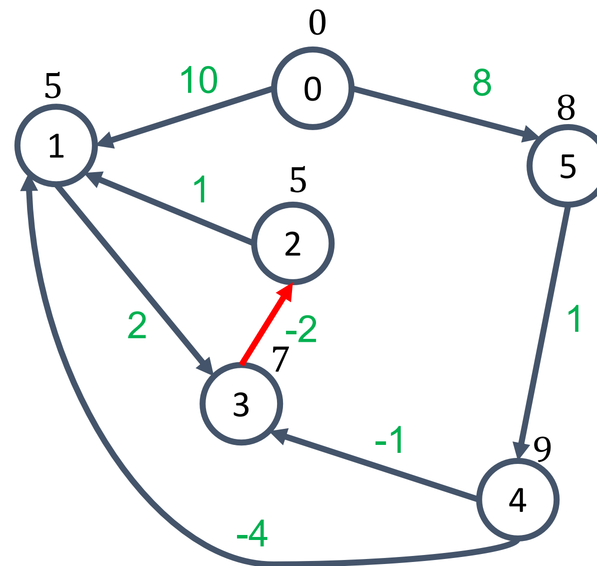
ослаблять все ребра графа |V| - 1 раз

Граф для алгоритма удобно хранить списком ребер

Алгоритм работает с отрицательными ребрами

Алгоритм устойчив к циклам отрицательного веса и позволяет их обнаружить

Цикл отрицательного веса можно восстановить



Список ребер

Вес	u	v
10	0	1
8	0	5
2	1	3
1	2	1
-2	3	2
-4	4	1
-1	4	3
1	5	4

d по шагам

шаг	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1	0	10	∞	∞	∞	8
2	0	10	∞	12	9	8
3	0	10	∞	12	9	8
4	0	5	10	8	9	8
5	0	5	6	7	9	8
6	0	5	5	7	9	8

Реализация

```
Bellman(G, s):
    InitSource(G, s)
    for i = 0 to |V|-1:
        for (u, v) in E:
            Relax(u, v, w(u, v))
    cycle = false
    for (u, v) in E:
        if (d[v] > d[u] + w(u,v)):
            cycle = true
```

$O(|V|)$ — InitSource(G, s)

$O(|V|)$ — for i = 0 to |V|-1

$O(|E|)$ — for (u, v) in E: Relax(u, v, w(u, v))

$O(|V| \cdot |E|)$ — весь цикл

Граф не стоит хранить матрицей смежности.
Потому что обход всех ребер есть $O(|V|^2)$,
если граф неполный обход ребер можно
сделать быстрее, храня граф списком ребер
тогда обход займет всего $O(|E|)$

Как найти цикл отрицательного веса и восстановить его

Если в графе присутствует цикл отрицательного веса, то вес пути, содержащий такой цикл, $\to -\infty$

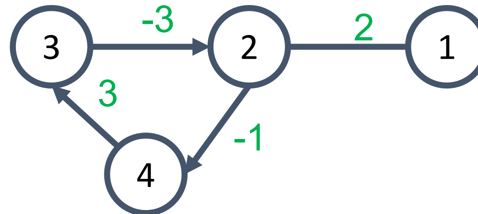
Вес такого пути можно уменьшать на каждой итерации

По алгоритму Беллмана-Форда мы делаем |V|-1 итерацию, почему?

Чтобы рассмотреть все ацикличные пути.
Ацикличный путь содержит
Не более |V| вершин и |V|-1 ребер

Если на |V|-ой итерации происходит релаксация, граф **содержит** цикл отрицательного веса

Чтобы восстановить сам цикл, нам необходимо найти вершину, которая принадлежит циклу.



Если в данном примере начать восстанавливать цикл от вершины 1, то мы не сможем понять, что цикл закончился

Далее от найденной вершины проходим по массиву предков, пока её снова не встретим

P	1	2	3	4
	2	3	4	2

```
cycle = false
for (u, v) in E:
    if (d[v] > d[u] + w(u,v)):
        cycle = true
        vert = v
        break
for i = 0 to |V|: vert = p[vert]
i = p[vert]
While vert != i:
    print(i)
    i = p[i]
```

Чтобы найти вершину принадлежащую циклу, сделаем |V| шагов назад по массиву предков

Такая доработка позволяет восстановить цикл

Алгоритм Дейкстры (жадный)

Суть:

Похож на алгоритм Прима, только при релаксации будет сохранять вес не ребра а всего пути до данной вершины

Жадный - не всегда будут найдены именно кратчайшие пути, но решение будет оптимальным

Не работает с ребрами и циклами отрицательного веса

Почему будет оптимальное решение?

Быстро ищем ответ и гарантируем локальный оптимум, потому что

Лемма

Частичные пути кратчайших путей тоже кратчайшие пути

Док-во:

рассмотрим $p: v_1 \to v_k = < v_1, v_2, \dots, v_k >]\ p$ – кратчайший путь

$1 \le i \le j \le k$

возьмем часть $p \mid p_{ij} =< v_i, v_{i+1}, \dots, v_j > \subseteq p$

$p = v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$

От противного, допустим, что p_{ij} – не наикратчайший путь $\Rightarrow \exists p_{ij}^*: w(p_{ij}^*) < w(p_{ij})$

$\Rightarrow w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$ – не будет минимальным, что противоречит условию

$\Rightarrow p_{ij}$ – кратчайший путь

Алгоритм Дейкстры (жадный)

Жадный алгоритм - алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным

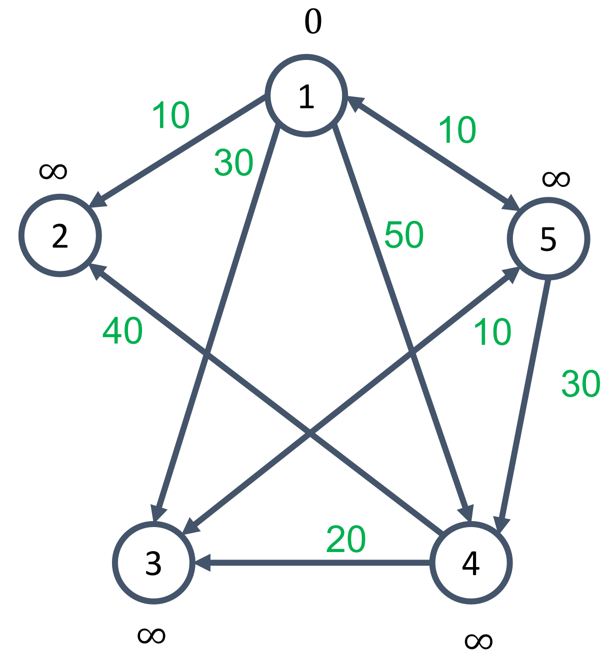
Быстрое и оптимальное решение в жизни выгоднее точного решения, которое нужно долго искать

Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	∞	∞	∞	∞
p	null	null	null	null	null
used					

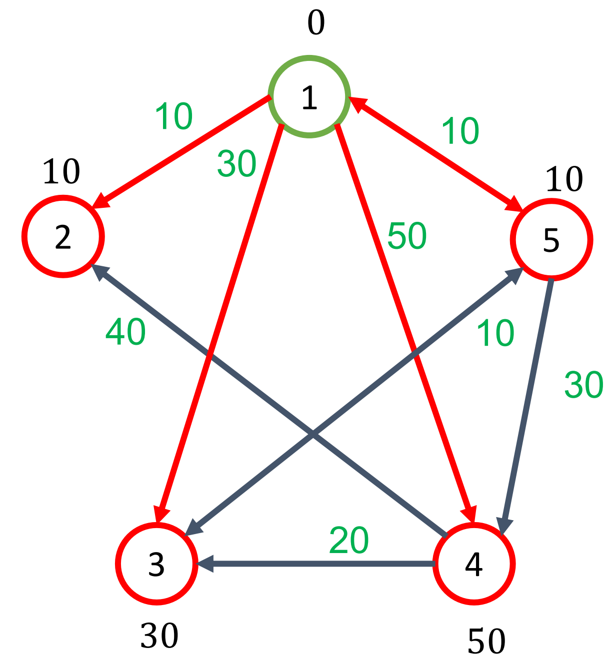


Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	30	50	10
p	null	1	1	1	1
used					

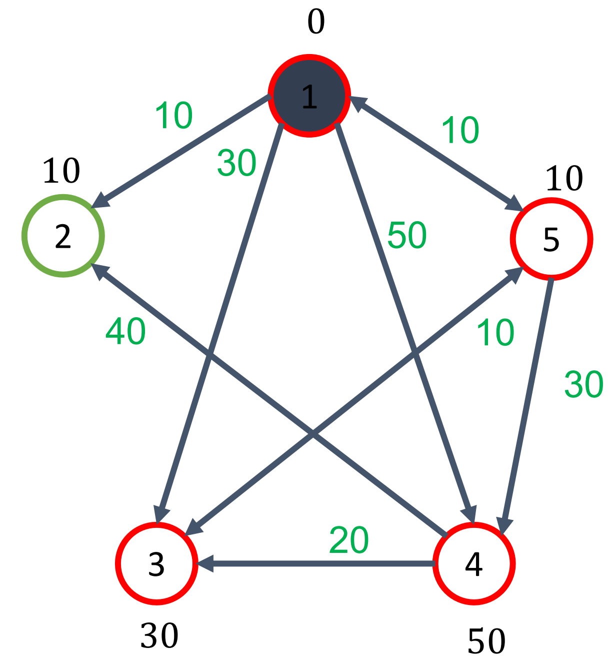


Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	30	50	10
p	null	1	1	1	1
used	+				

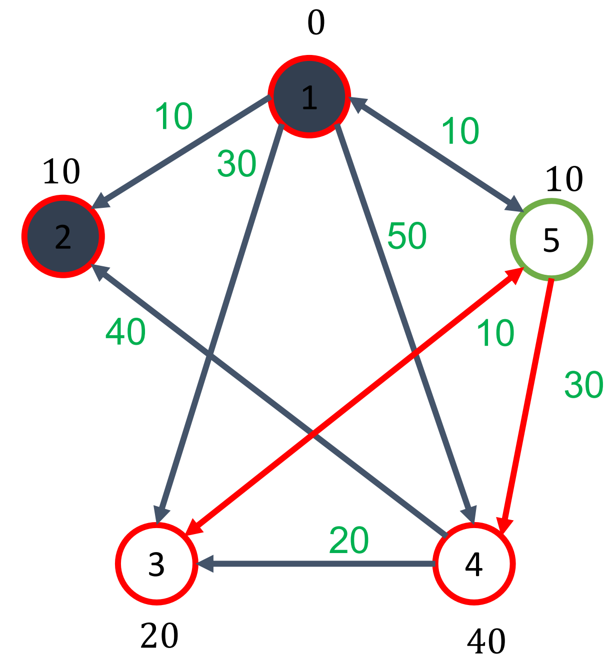


Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	20	40	10
p	null	1	5	5	1
used	+	+			

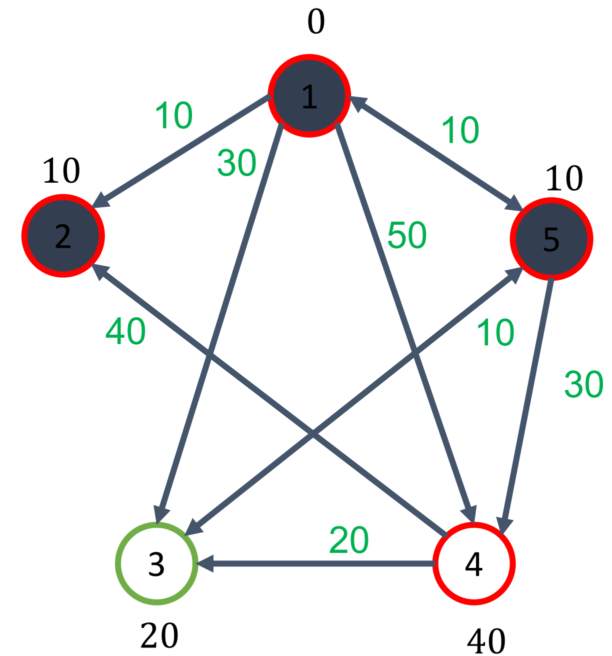


Алгоритм Дейкстры
(жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	20	40	10
p	null	1	5	5	1
used	+	+			+

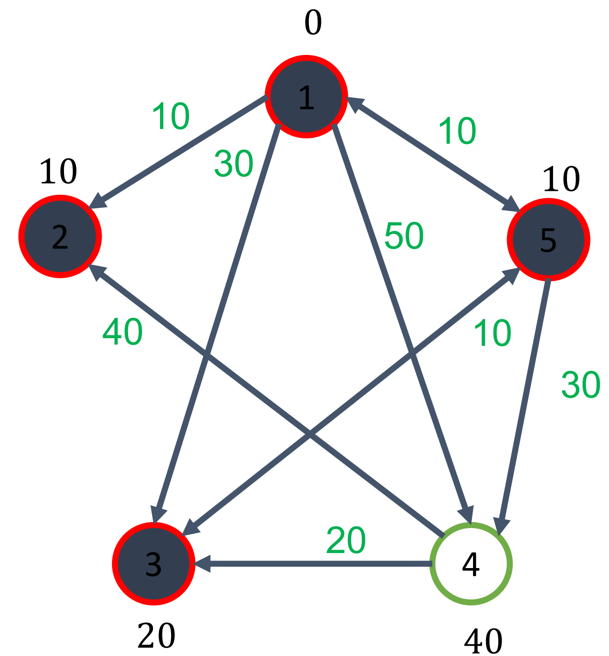


Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	20	40	10
p	null	1	5	5	1
used	+	+	+		+

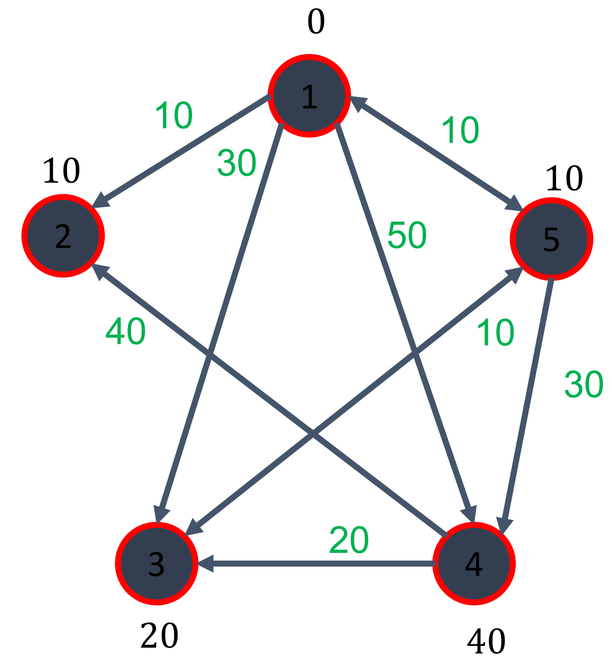


Алгоритм Дейкстры (жадный)

Суть:

- Каждую итерацию выбираем вершину, до которой кратчайший путь
- Помечаем вершину и просматриваем все ребра из нее (пытаемся ослабить)

	1	2	3	4	5
d	0	10	20	40	10
p	null	1	5	5	1
used	+	+	+	+	+



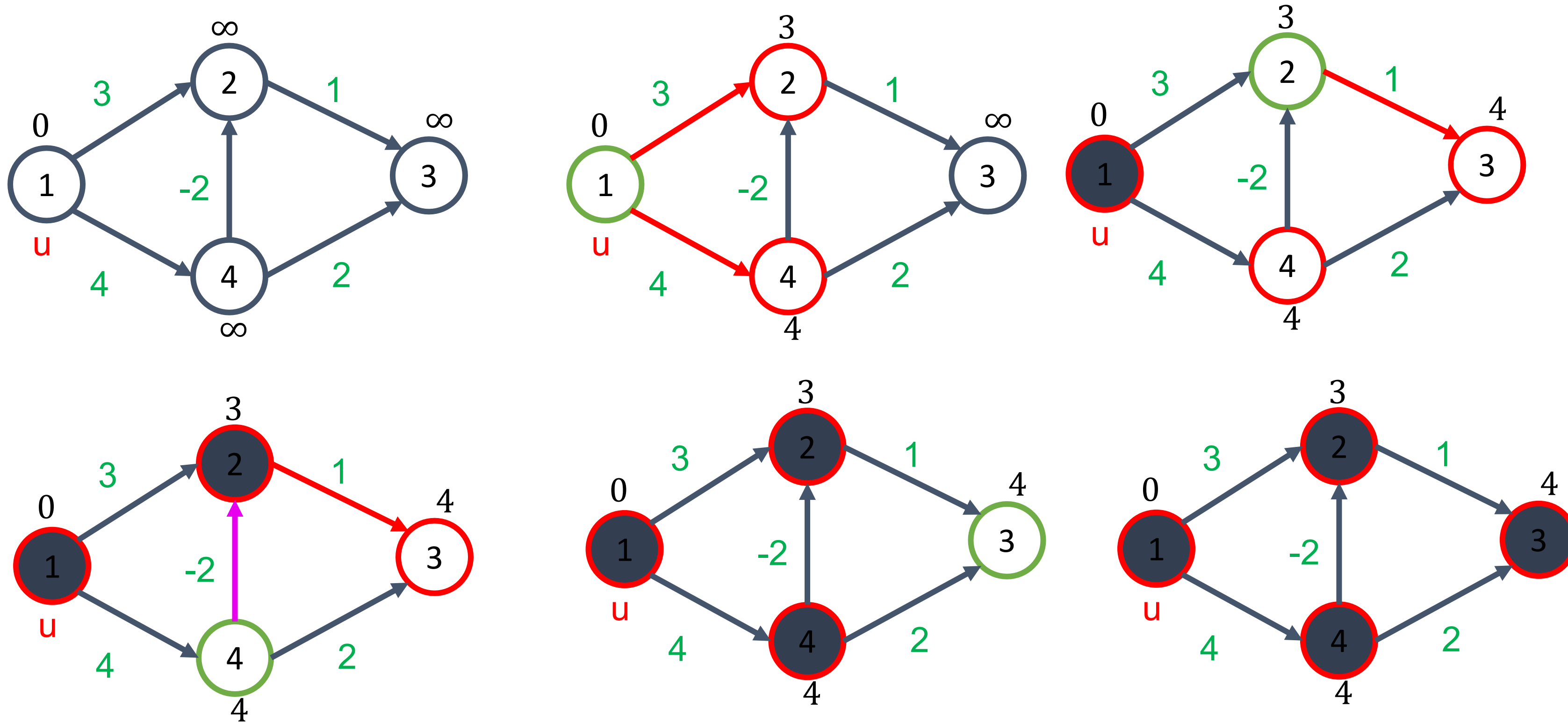
Реализация

```
Deikstra(G, s):
O(|V|)    InitSource(G, s)
O(|V|)    Q=createheap(G.V)
          while not Q.isEmpty():
              u = Q.extractMin()
              used[u] = true
              for v in G.W[u]:
                  Relax(u, v, w(u, v))
                  if (d[v] > d[u] + w(u, v)):
         O(log|V|)Q.decreaseKey(v, d[u]+w(u,v))
```

$O((|V| + |E|)\log|V|)$ = $O(|E| \cdot \log|V|)$ (for the while loop)

Алгоритм Дейкстры
(жадный)

Алгоритм не работает с ребрами отрицательного веса



Мы не релаксируем пути
к помеченной вершине

Алгоритм Флойда-Воршалла

Суть:

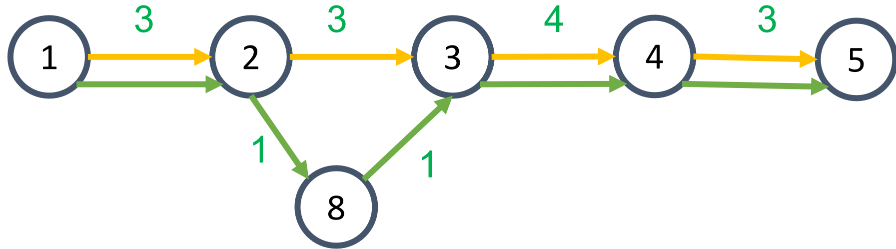
ищем кратчайшие пути от всех до всех, выполняя релаксацию через каждую вершину

Нужны будут дополнительные матрицы:

Матрица длин кратчайших путей

Матрица предков для восстановления кратчайших путей

Ранее мы релаксировали ребро, теперь будем улучшать путь через вершину



Улучшаем $p_1 = v_1 v_2 v_3 v_4 v_5$ через v_8

И получаем $p_2 = v_1 v_2 v_8 v_3 v_4 v_5$

$w_{p_1} = d[1][5] = 13$

$w_{p_2} = d[1][8] + d[8][5] = 12$

Реализация

```
Floyd(G, s):
    for i in V: //i -вершина, через которую релаксируем все пути
        for u in V:  \
                      }//все возможные пути
            for v in V:  /
                d[u][v] = min(d[u][v], d[u][i] + d[i][v])
```

Алгоритм Флойда-Воршалла

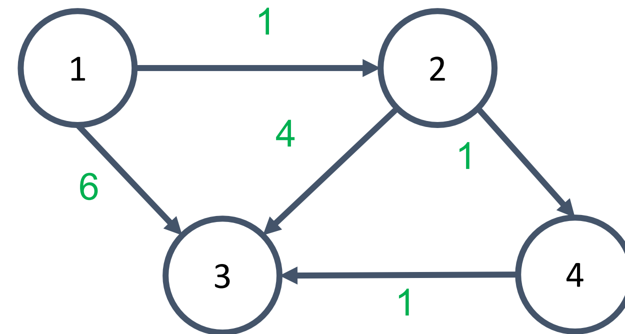
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	6	∞
2	∞	0	4	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

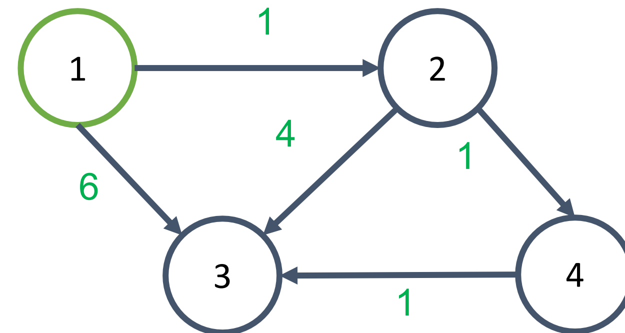
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	6	∞
2	∞	0	4	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

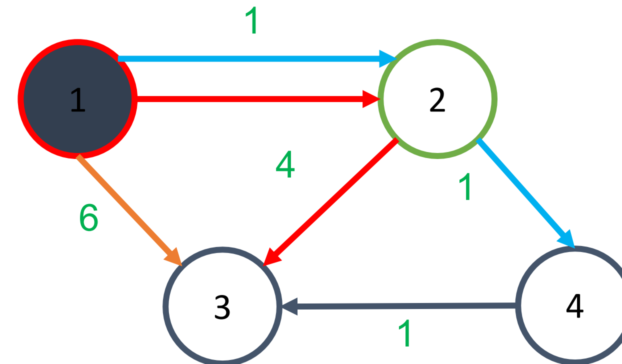
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	5(6)	2(∞)
2	∞	0	4	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

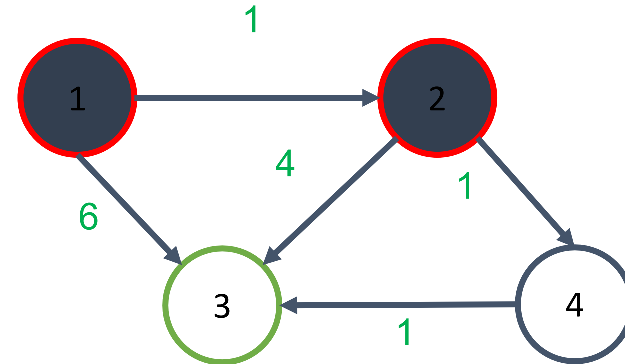
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	5	2
2	∞	0	4	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

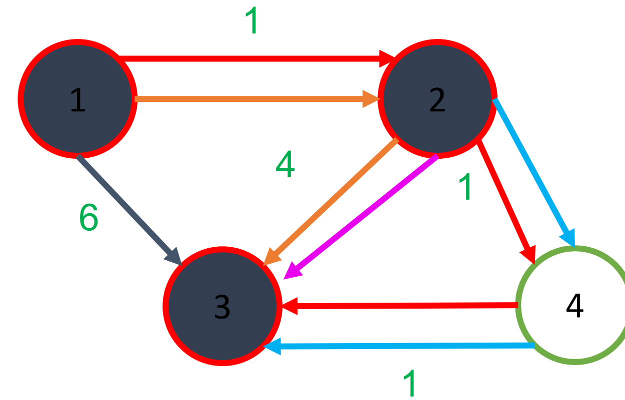
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	3(5)	2
2	∞	0	2(4)	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

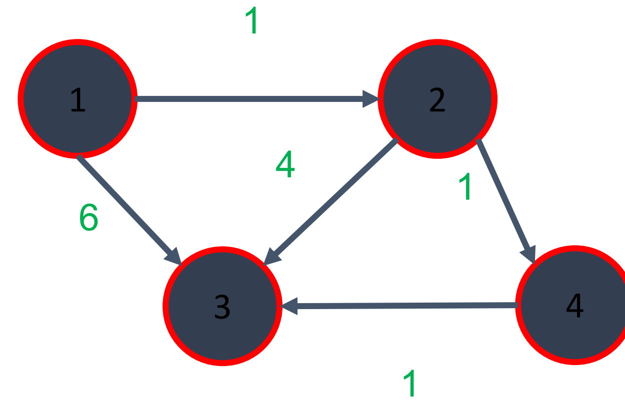
Принцип: релаксируем через i-ю вершину

Выделяем i-й столбец и i-ю строку

Просматриваем все ячейки вне выделенных

Сравниваем значение в ячейке с суммой i-го столбца и i-ой строки соответствующих этой ячейке

w	1	2	3	4
1	0	1	3	2
2	∞	0	2	1
3	∞	∞	0	∞
4	∞	∞	1	0



Алгоритм Флойда-Воршалла

Итог:

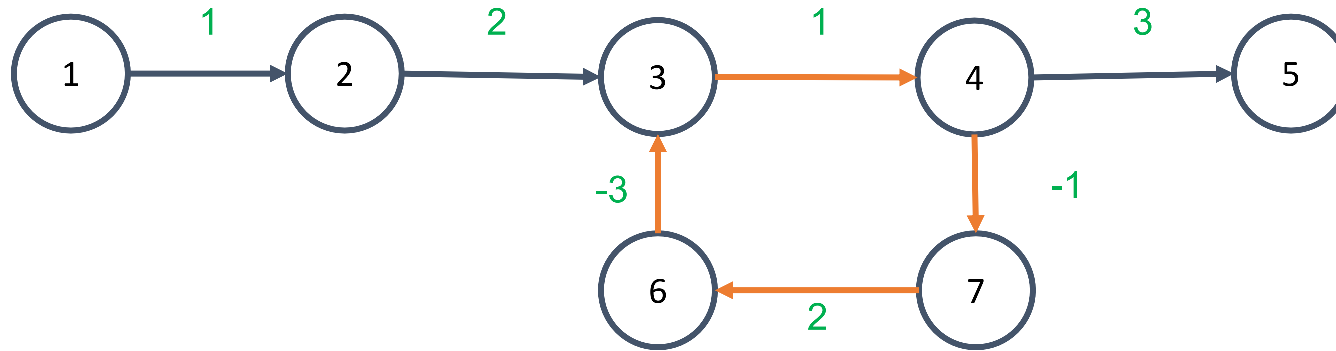
Мы прошли по все вершинам три циклам

1) Шли по промежуточным (релаксируемам) вершинам
2) Вершины начала пути
3) Вершины конца пути

Таким образом, мы проходили всю матрицу размерности |V|x|V| -|V| раз и искали более кратчайшие пути через текущую просматриваемую

Алгоритм Флойда-Воршалла

Циклы отрицательного веса



$p_1 = v_3 v_4 v_7 v_6 v_3$ – цикл отрицательного веса
$w_{p_1} = -1$

На старте $w[3][3] = 0$
При работе алгоритма Флойда мы пытаемся релаксировать $3 \to 3$ через вершины 1, 2, 3, 4, 5, 6, 7
И мы получим, что $w[3][3] < 0$

Если в результирующей матрице на главной диагонали есть отрицательные числа → есть цикл отрицательного веса

Цикл восстанавливается по аналогии с Беллманом-Фордом

Алгоритм эффективно работает с ребрами отрицательного веса