

Лекция 6. Графы 2.

Алгоритмы и структуры данных

Крымов А.Ю.

План лекции 6 «Графы 2»



1. Задачи поиска кратчайших путей.

- 1. Алгоритм Дейкстры.
- 2. Алгоритм А*.
- 3. Алгоритм Беллмана-Форда.
- 2. Минимальные остовные деревья.
 - 1. Алгоритм Прима.
 - 2. Алгоритм Крускала.
 - 3. Системы непересекающихся множеств.
- 3. Задача коммивояжера



Взвешенный граф

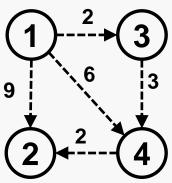


Во взвешенном графе с каждым ребром связано некоторое число (вес), которое обычно интерпретируется как расстояние либо стоимость.

В евклидовом графе узлы соответствуют точкам на плоскости, а вес рёбер равен евклидову расстоянию между вершинами.

Пример: карта дорог.





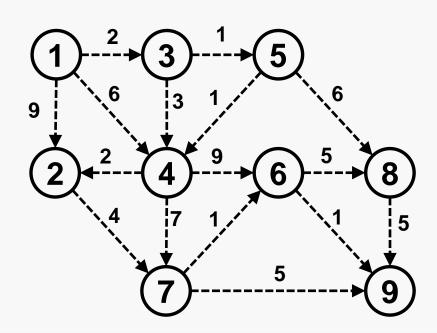
Задачи поиска кратчайших путей



G – ориентированный взвешенный граф

Кратчайшим путём между двумя вершинами u и v в графе G называется такой направленный простой путь из u в v, что никакой другой путь не имеет меньшего веса.

- Веса рёбер не обязательно пропорциональны их длинам.
- Вес это не количество рёбер.
- Может существовать несколько кратчайших путей.
- А может не существовать ни одного пути.



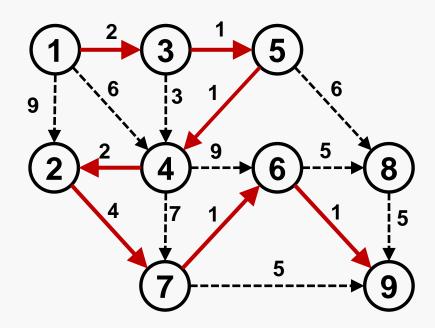
Задачи поиска кратчайших путей



G – ориентированный взвешенный граф

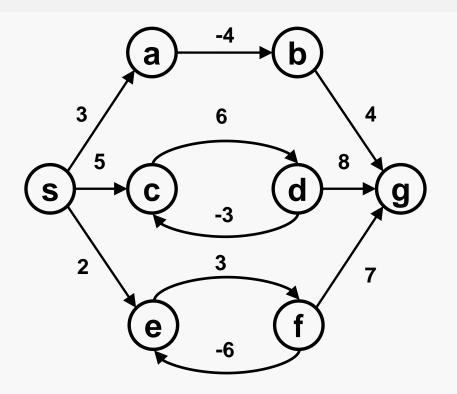
Кратчайшим путём между двумя вершинами u и v в графе G называется такой направленный простой путь из u в v, что никакой другой путь не имеет меньшего веса.

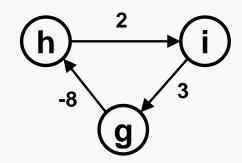
- Веса рёбер не обязательно пропорциональны их длинам.
- Вес это не количество рёбер.
- Может существовать несколько кратчайших путей.
- А может не существовать ни одного пути.



Рёбра с отрицательным весом







Рёбра с отрицательным весом допустимы – (s, a, b) – кратчайший путь из s в b.

Циклы с положительным весом допустимы – w(c, d) + w(d, c) = 3 > 0. Путь (s, c, d) – кратчайший из s в d.

При наличии на пути цикла с отрицательным весом – w(e, f) + w(f, e) = 3 - 6 = -3 < 0кратчайшего пути между вершинами s и f не существует.

Задачи поиска кратчайших путей



- G ориентированный взвешенный граф.
 - 1. SPSP (Single Pair Shortest Path problem) поиск кратчайшего пути между двумя вершинами. Алгоритмы Дейкстры, A*, IdaStar.

- 2. SSSP (Single Source Shortest Paths problem) поиск кратчайших путей из выделенной вершины до всех остальных. Алгоритмы Дейкстры, Беллмана-Форда.
- 3. APSP (All Pairs Shortest Paths problem) поиск кратчайших путей между всеми парами вершин. Алгоритмы Флойда, Джонсона.

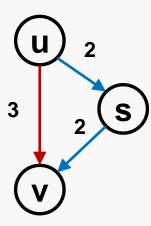
Релаксация



Релаксация ребра – это проверка того, даёт ли продвижение по данному ребру новый кратчайший путь к конечной вершине.

Процедура релаксации ребра

```
bool Relax( u, v ) {
    if( d[v] > d[u] + w( u, v ) ) {
        d[v] = d[u] + w( u, v );
        pi[v] = u;
        return true;
    }
    return false;
}
```



d[v] = 4 При релаксации (u, v) d[v] получит новое значение 3.



Алгоритм Дейкстры похож на BFS, но вместо обычной очереди используется очередь с приоритетом.

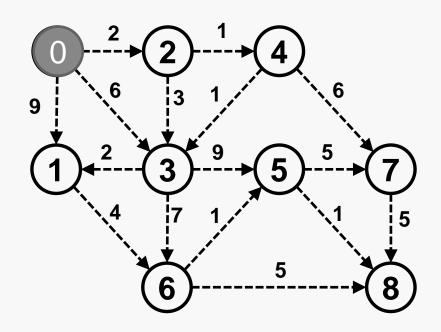
Веса рёбер больше 0.

Встретили вершину в первый раз

Если удалось релаксировать ребро, то надо изменить ее позицию в очереди с приоритетом.

```
void Dijkstra( G, s ) {
    pi[V] = -1;
    d[V] = INT MAX;
    d[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : ребра из u ) {
            if( d[v] == INT MAX ) {
                d[v] = d[u] + w(u, v);
                pi[v] = u;
                q.push( v );
            } else if(Relax( u, v )) {
               q.DecreaseKey( v, d[v] );
```



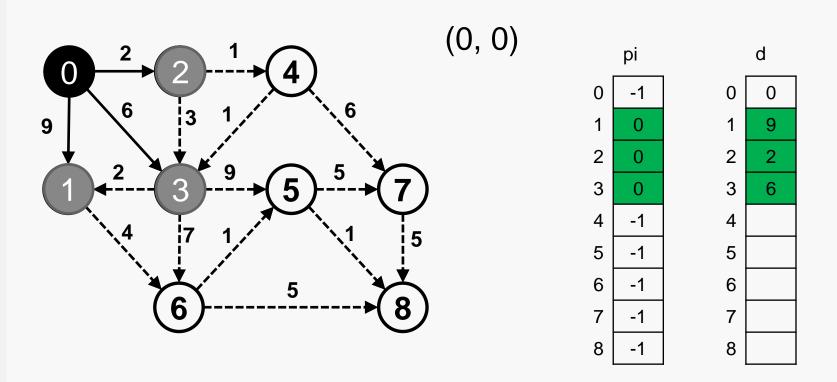


pi		
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	
8	-1	

d		
0	0	
1		
2		
2 3 4		
4		
5		
6		
7		
8		

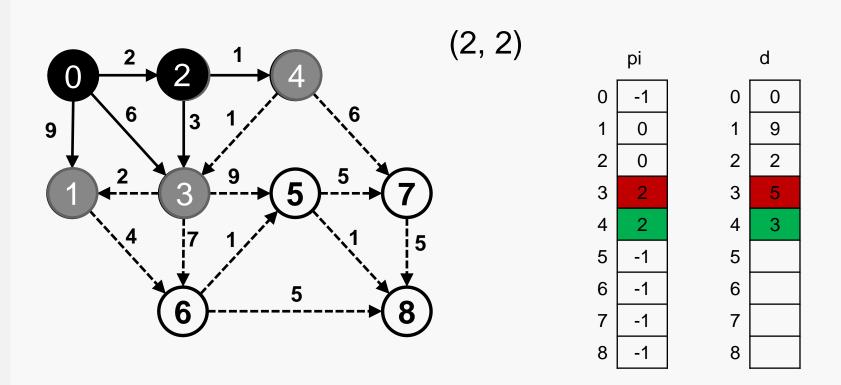
Q: (0, 0)





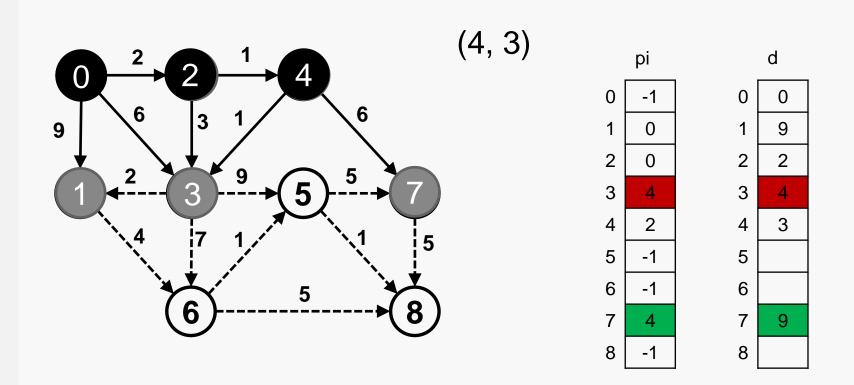
Q: (2, 2) (3, 6) (1, 9)





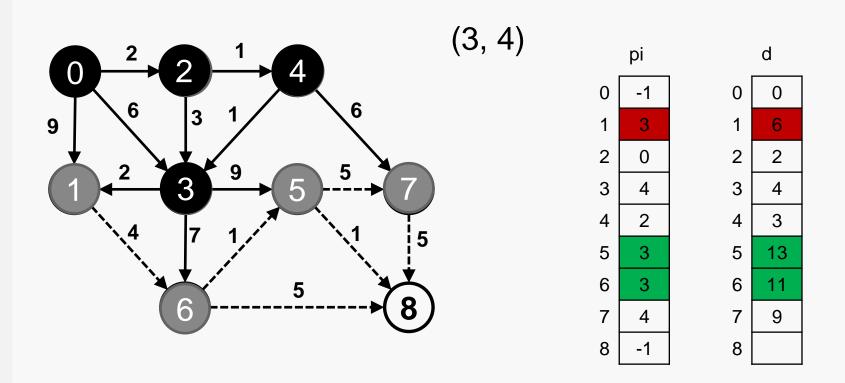
Q: (4, 3) (3, 5) (1, 9)





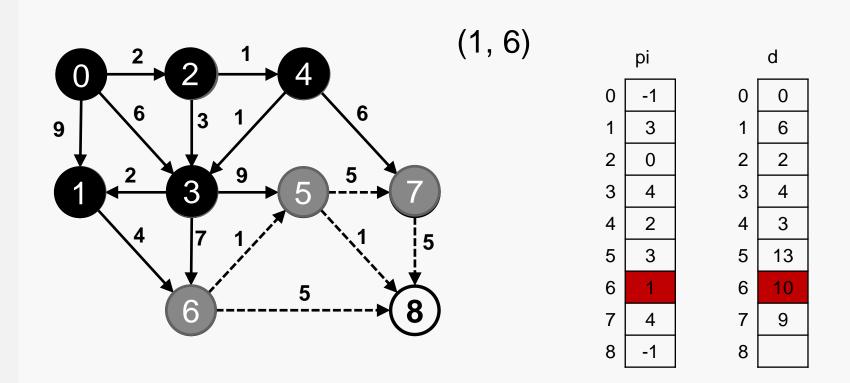
Q: (3, 4) (1, 9) (7, 9)





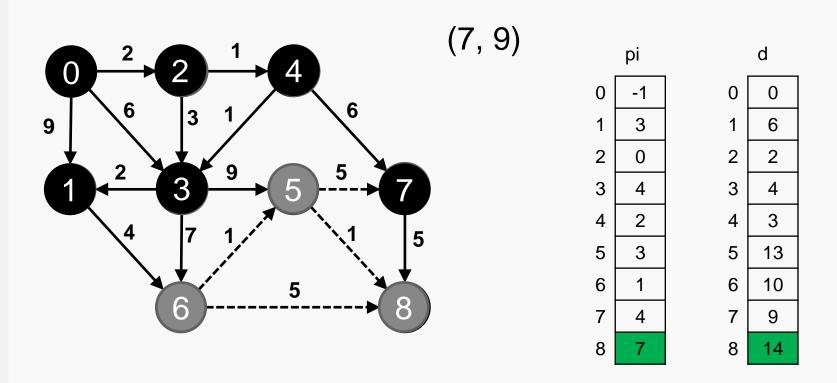
Q: (1, 6) (7, 9) (5, 13) (6, 11)





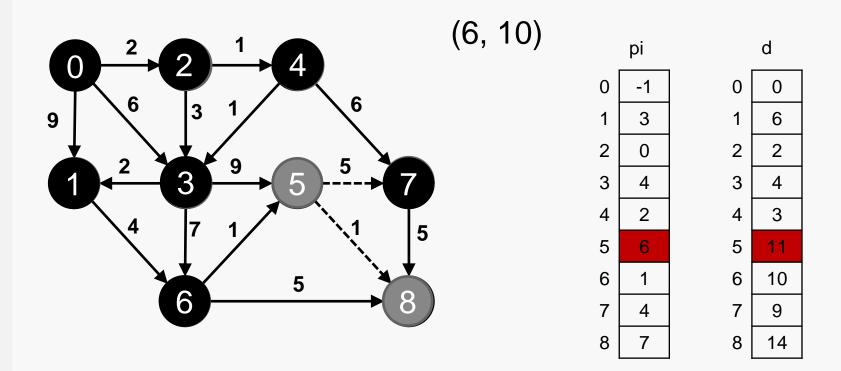
Q: (7, 9) (5, 13) (6, 10)





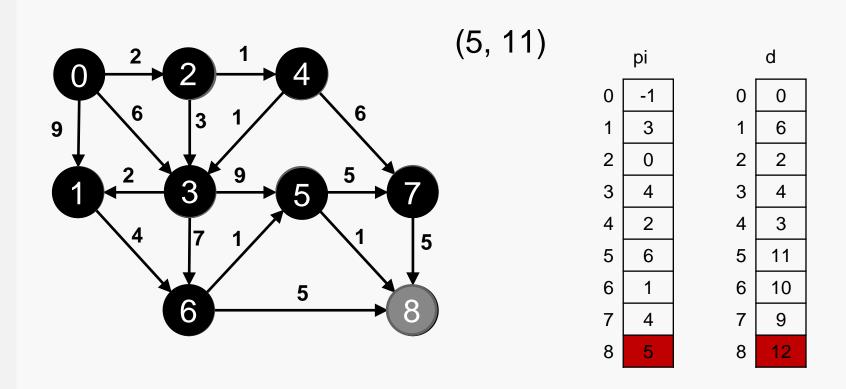
Q: (6, 10) (5, 13) (8, 14)





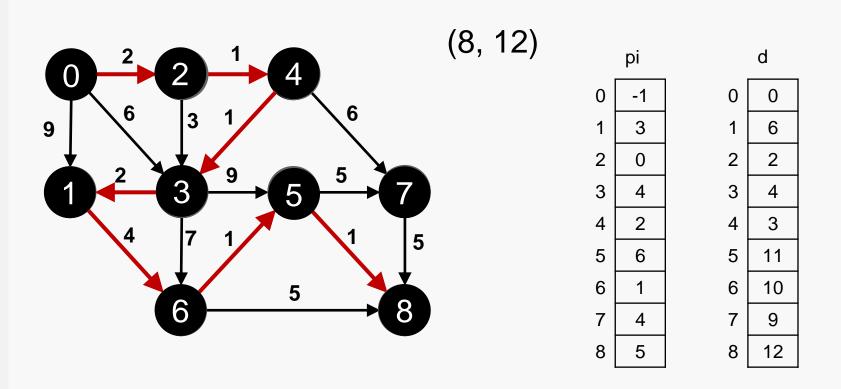
Q: (5, 11) (8, 14)





Q: (8, 12)





Кратчайший путь из 0 в 8 восстанавливаем по массиву рі: 0, 2, 4, 3, 1,6, 5, 8



Оценка времени работы. Будем использовать в качестве реализации очереди с приоритетом двоичную кучу.

- Добавление узла: $O(\log V)$. Не более V операций.
- Уменьшение значения ключа: $O(\log V)$. Не более E операций.

Всего
$$T = O((E + V) \cdot log V)$$
.

Используемая память M = O(V).



Важная тонкость.

Как найти узел v в двоичной куче, чтобы вызывать q. Decrease Key(v, d[v])?

<u>Решение 1.</u> Хранить позицию і в куче для каждого узла v. В момент изменения позиции в куче обновлять эту позицию, храняющуюся в описании узла.

<u>Решение 2.</u> Использовать вместо бинарной кучи множество set < pair < d, v >>.

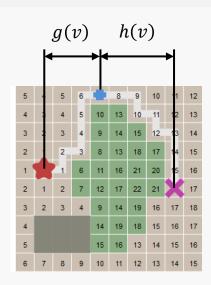
Во время релаксации удалять старую пару <d[v], v>, добавлять новую.

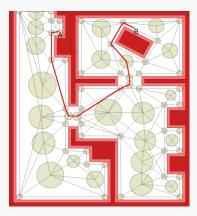


Алгоритм A* (A-star) - это алгоритм поиска кратчайшего пути между *двумя* вершинами графа.

В процессе работы алгоритма рассчитывается функция

f(v) = g(v) + h(v), где g(v) - наименьшая стоимость пути из стартовой вершины в v h(v) - эврестическое приближение стоимости пути от v к конечной цели







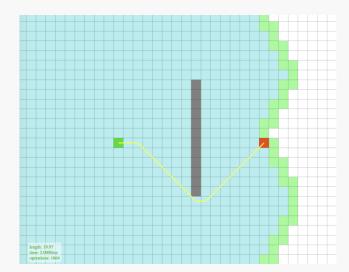
Сравнение как и в алгоритме Дейкстры

Значение эвристики используем только для для изменения порядка в очереди с приоритетом

```
void aStar( G, s ) {
   pi[V] = -1;
   d[V] = INT MAX;
   d[s] = 0;
   priority_queue<int> q; q.push( s );
   while( !q.empty() ) {
       u = q.top(); q.pop();
        for((u, v): E){
            if(d[v] == INT_MAX) {
                q.push(v);
            if( d[v] > d[u] + w(u, v) ) {
               d[v] = d[u] + w(u, v);
               pi[v] = u;
               q.DecreaseKey( v, d[v] → h(v) );
```

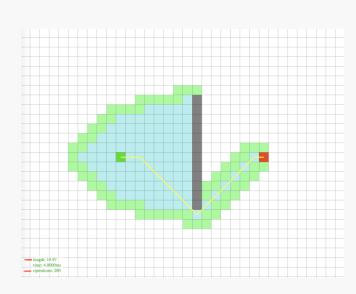


Дейкстра



Длина пути – 19.97 Кол-во итераций – 1664

A*

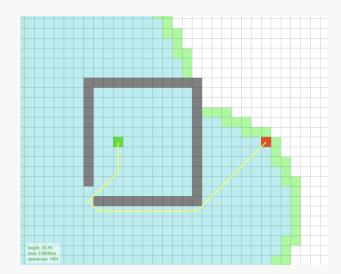


Длина пути – 19.97 Кол-во итераций – 280

http://qiao.github.io/PathFinding.js/visual/

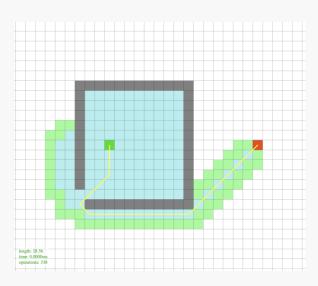


Дейкстра



Длина пути – 28.56 Кол-во итераций – 1604

A*



Длина пути – 28.56 Кол-во итераций – 338

http://qiao.github.io/PathFinding.js/visual/



Эвристика h(v) должна быть:

- Допустимой для любой вершины v значение h(v) меньше или равно весу кратчайшего пути от v до цели.
- Монотонной с ней должно выполняться неравенство треугольника. Т.е. если есть путь из v в u, то $g(v) + h(v) \le g(u) + h(u)$

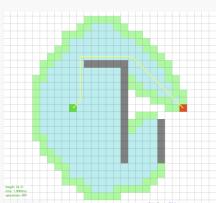
Примеры эвристик:

- Манхеттенское расстояние h(v) = |v.x u.x| + |v.y u.y|
- Расстояние Чебышёва $h(v) = \max(|v.x u.x|, |v.y u.y|)$
- Евклидово расстояние $h(v) = \sqrt[2]{(v.x u.x)^2 + (v.y u.y)^2}$

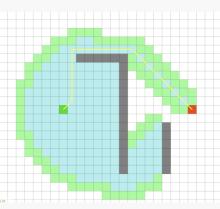


Манхеттенское расстояние

Расстояние Чебышёва



Евклидово расстояние



Длина пути во всех примерах одинаковая – 24.31

http://qiao.github.io/PathFinding.js/visual/

Алгоритм Беллмана-Форда



Поиск кратчайших путей из выделенной вершины до всех остальных (SSSP)

Если в графе нет циклов отрицательного веса, достижимых из s, то алгоритм Беллмана-Форда находит все кратчайшие пути из s до остальных вершин.

Если в графе есть достижимый из s цикл отрицательного веса, то алгоритм Беллмана-Форда сообщит о его наличии. (и может выдать его).

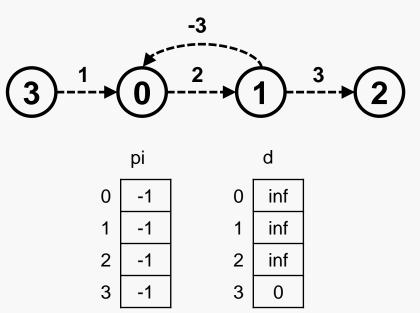
```
bool BellmanFord( G, s ) {
    d[s] = 0;
    for( int i = 0; i < V - 1; ++i ) {
        for( (u, v) : E ) Relax( u, v );
    }
    // Детектирование цикла.
    for( (u, v) : E ) {
        if( Relax( u, v ) )
            return false;
    }
    return true;
}</pre>
```

Сложность: $O(|V| \cdot |E|)$



Релаксируем все рёбра V-1 раз

Стартовая вершина 3, инициализируем d[3] = 0



Процедура релаксации ребра

```
bool Relax( u, v ) {
    if( d[v] > d[u] + w( u, v ) )
    {
        d[v] = d[u] + w( u, v );
        pi[v] = u;
        return true;
    }
    return false;
}
```



d

inf

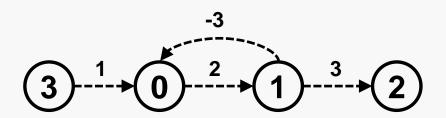
inf

inf

0

Релаксируем все рёбра V-1 раз

Стартовая вершина 3, инициализируем d[3] = 0

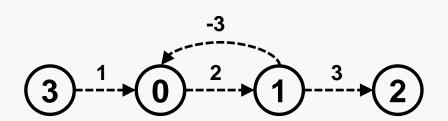


	pi	
)	-1	0
1	-1	1
2	-1	2
3	-1	3



Релаксируем все рёбра V-1 раз

Итерация 1



pi			
0	3		
1	-1		
2	-1		
3	-1		

~		
1		
inf		
inf		
0		



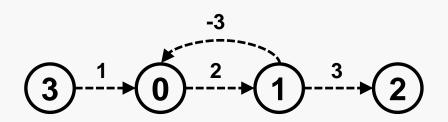
d

0

3

Релаксируем все рёбра V-1 раз

Итерация 2



	pi	
С	1	
1	0	
2	1	
3	-1	

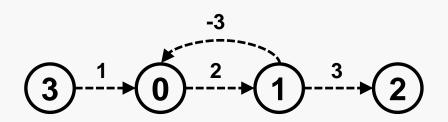


d

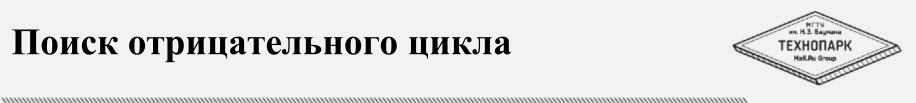
0

Релаксируем все рёбра V-1 раз

Итерация 3

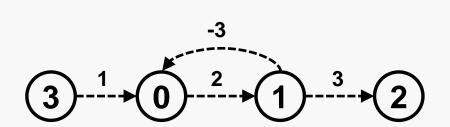


	pi	
)	1	
1	0	
2	1	
3	-1	



Если отрицательный цикл есть, то на V- ой итерации должна произойти релаксация

Итерация 4

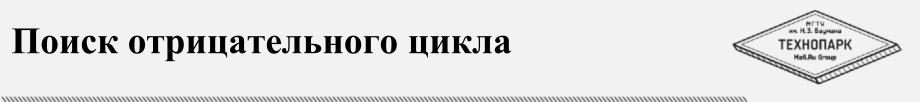


pi			
0	1		
1	0		
2	1		
3	-1		

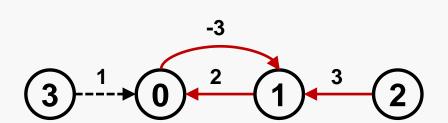
d

Нужно запомнить номер вершины, на которой произошла релаксация.

В данном примере это вершина 2.



Для поиска вершины, лежащей на цикле, проходим по предкам V раз от запомненной на предыдущем шаге вершины.

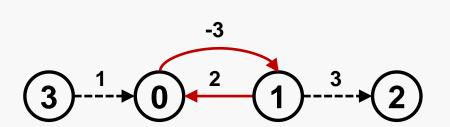


pi			d
С	1	0	-6
1	0	1	-1
2	1	2	2
3	-1	3	0

После этого мы точно окажемся внутри цикла.



Для поиска цикла от найденной вершины проходим по предкам, пока не встретим эту же вершину.



pi			d
0	1	0	-6
1	0	1	-1
2	1	2	2
3	-1	3	0

А эту же вершину встретим обязательно, потому что находимся внутри цикла.

Остовные деревья



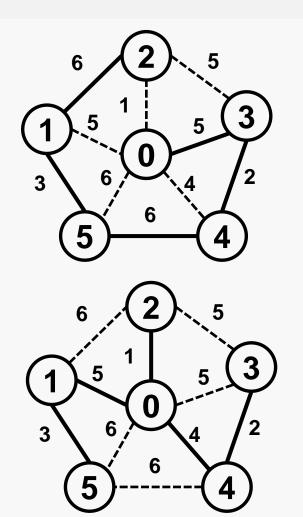
Пусть G – связный неориентированный граф.

Остовное дерево графа G состоит из минимального подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам.

Остовное дерево графа G — ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины.

Минимальное остовное дерево (MST)

графа G — это его ациклический связный подграф, в который входят все его вершины, обладающий минимальным суммарным весом ребер.





Алгоритм Прима похож на алгоритм Дейкстры, но в качестве весовой функции используем длину кратчайшего ребра от текущей вершины до дерева.

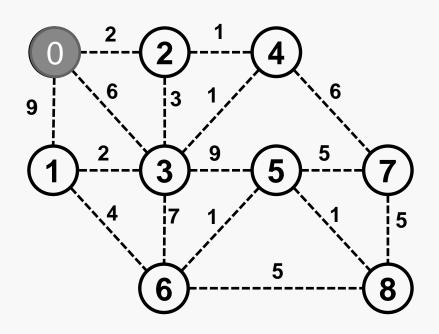
Веса рёбер больше 0.

Встретили вершину в первый раз

Если удалось релаксировать вершину, то надо изменить ее позицию в очереди с приоритетом.

```
void Prima( G, s ) {
    pi[V] = -1;
    min e[V] = INT MAX;
    min e[s] = 0;
    priority_queue<int> q; q.push( s );
    while( !q.empty() ) {
        u = q.top(); q.pop();
        for( ( u, v ) : ребра из u ) {
            if( min e[v] == INT MAX ) {
              min_e[v] = w(u, v);
                pi[v] = u;
                q.push( v );
            } else if(Relax( u, v )) {
                q.DecreaseKey( v, min e[v] );
```

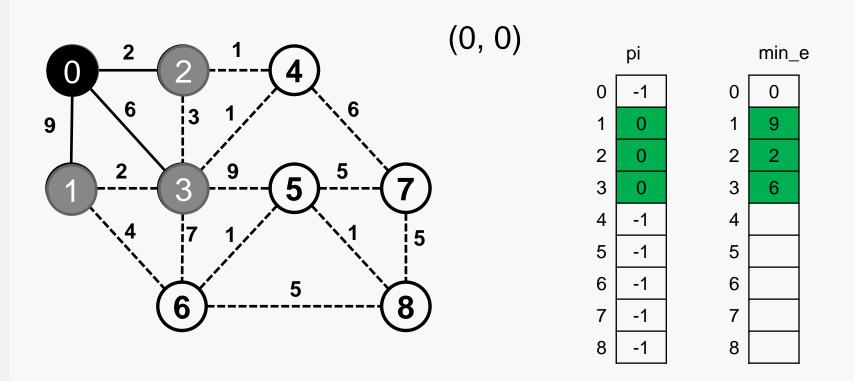




pi			min_	е
О	-1	0	0	
1	-1	1		
2	-1	2		
3	-1	3		
4	-1	2 3 4 5		
5	-1	5		
6	-1	6		
7	-1	7		
3	-1	8		

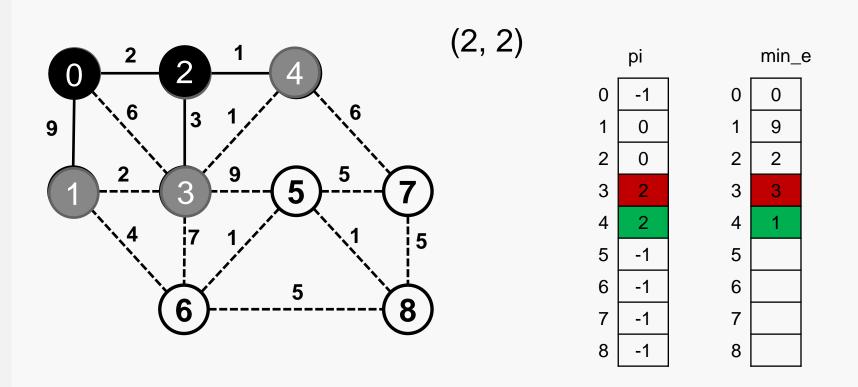
Q: (0, 0)





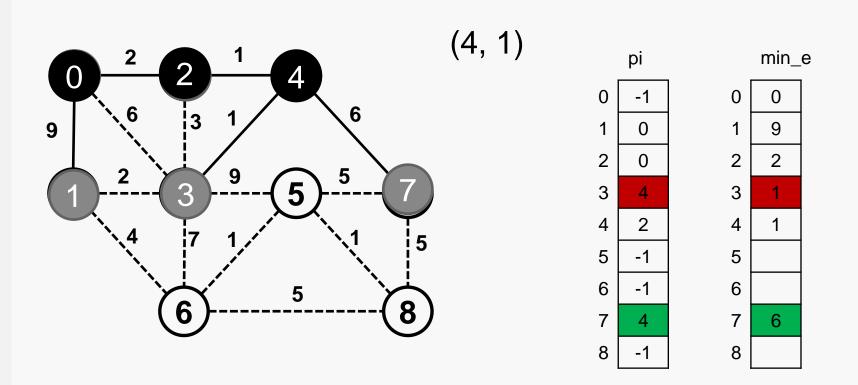
Q: (2, 2) (3, 6) (1, 9)





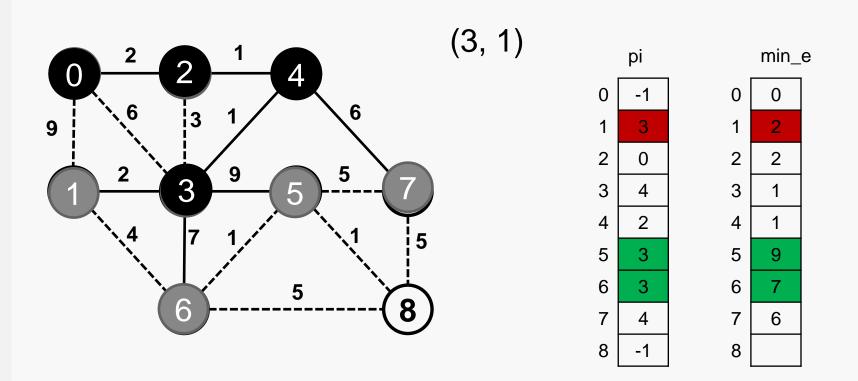
Q: (4, 1) (3, 3) (1, 9)





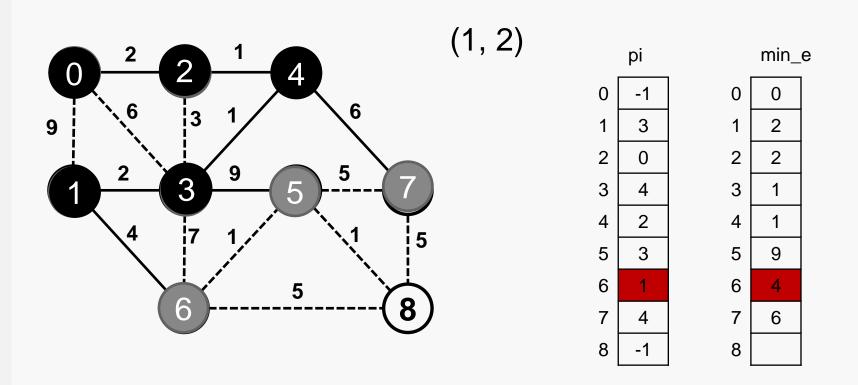
Q: (3, 1) (1, 9) (7, 6)





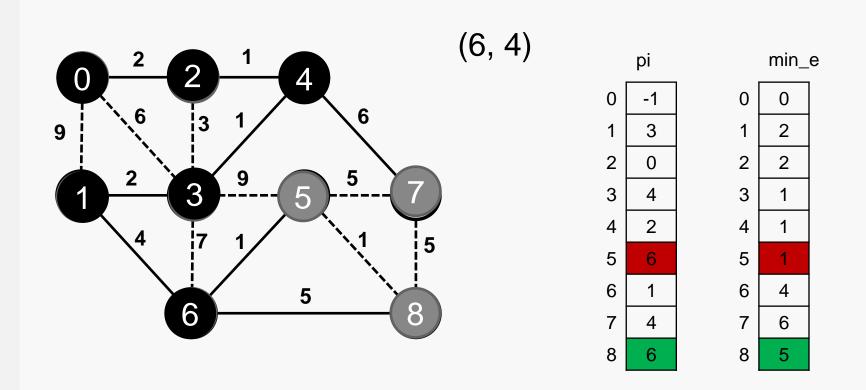
Q: (1, 2) (7, 6) (6, 7) (5, 9)





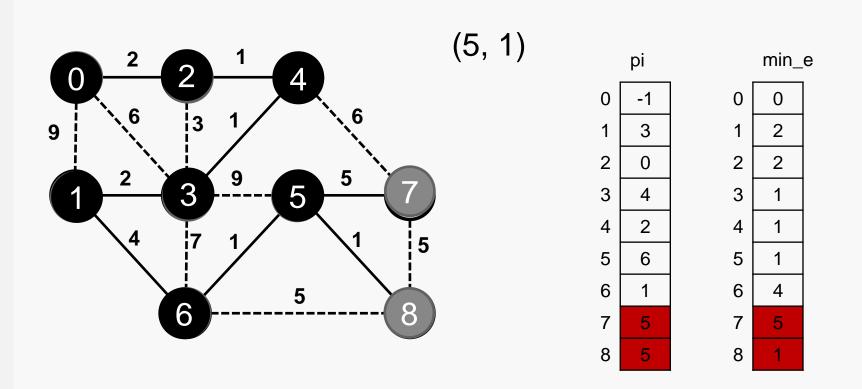
Q: (6, 4) (7, 6) (5, 9)





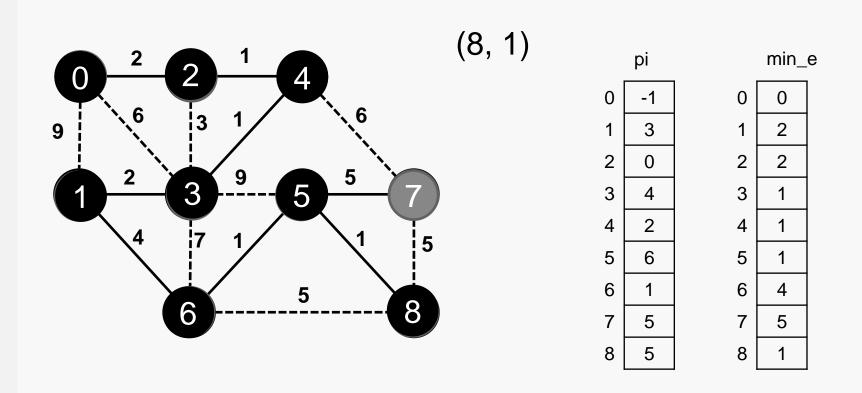
Q: (5, 1) (8, 5) (7, 6)





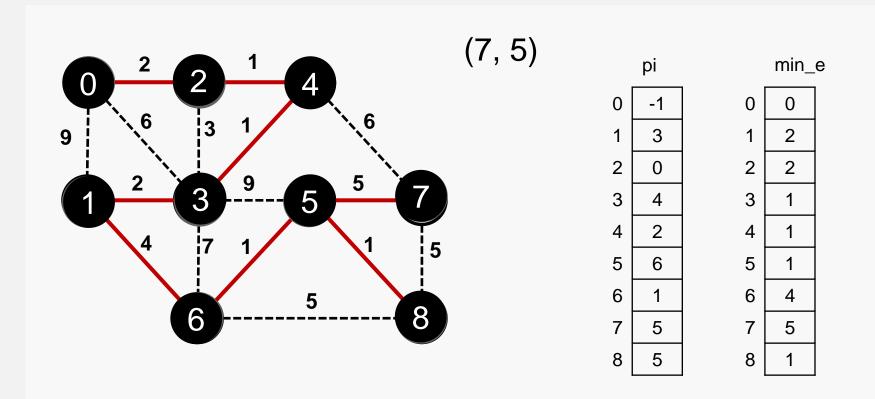
Q: (8, 1) (7, 5)





Q: (7, 5)





Q:

Алгоритм Прима. Оценка



В качестве очереди с приоритетом для вершин можно использовать кучу или сбалансированное дерево поиска, как в алгоритме Дейкстры.

В этом случае одна итерация состоит из

- 1) Извлечение вершины из очереди с приоритетом $O(\log V)$
- 2) Обработка ребер, инцидентных извлеченной вершине $O(\log V)$ на каждое ребро.

Пункт 1) будет выполнять V раз.

Пункт 2) будет выполняться Е раз.

Общая оценка времени работы: $T = O((E + V) \cdot logV)$

Оценка доп. памяти: M = O(V).

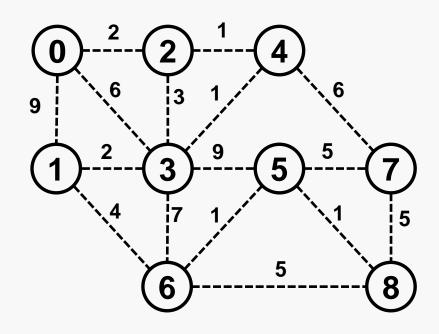


Алгоритм Крускала – еще один алгоритм поиска минимального остовного дерева.

- 1) Сортируем все ребра графа по весу.
- 2) Инициализируем лес деревьев. Изначально каждая вершина маленькое дерево (пень).
- 3) Последовательно рассматриваем ребра графа в порядке возрастания веса. Если очередное ребро соединяет два разных дерева из леса, то объединяем эти два дерева этим ребром в одно дерево. Если очередное ребро соединяет две вершины одного дерева из леса, то пропускаем такое ребро. Повторяем 3), пока в лесу не останется одно дерево.

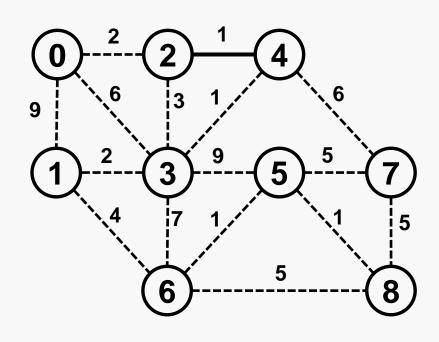


w(2, 4) = 1
w(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
w(1, 3) = 2
w(2, 3) = 3
w(1, 6) = 4
w(5, 7) = 5
w(6, 8) = 5
w(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
w(0, 1) = 9
w(3, 5) = 9



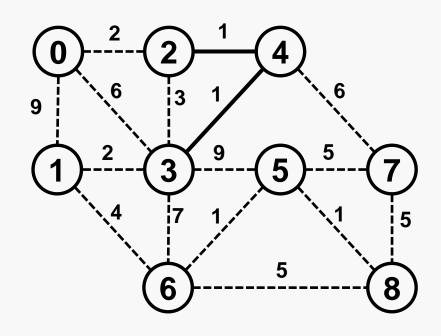


W(2, 4) = 1
w(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
w(1, 3) = 2
w(2, 3) = 3
w(1, 6) = 4
w(5, 7) = 5
w(6, 8) = 5
w(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
w(0, 1) = 9
w(3, 5) = 9



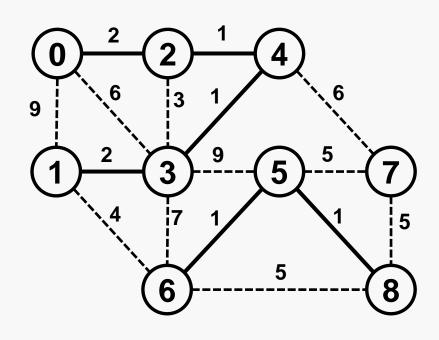


W(2, 4) = 1
w(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
w(1, 3) = 2
w(2, 3) = 3
w(1, 6) = 4
w(5, 7) = 5
w(6, 8) = 5
w(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
w(0, 1) = 9
w(3, 5) = 9





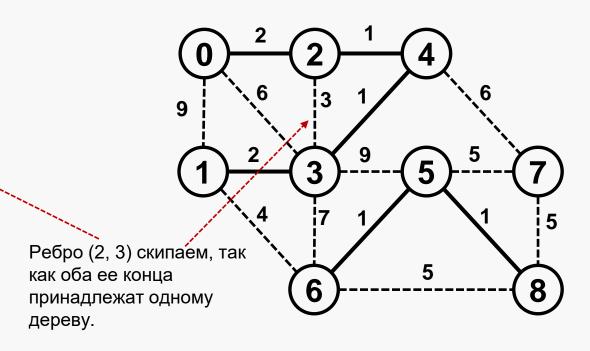
w(2, 4) = 1
w(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
VV(O, Z) = Z
W(1, 3) = 2
w(2, 3) = 3
w(1, 6) = 4
w(F 7) - F
w(5, 7) = 5
w(6, 8) = 5
•
w(7, 8) = 5
w(0, 3) = 6
w(0, 3) = 6
w(4, 7) = 6
· ·
w(3, 6) = 7
w(0, 1) = 9
` ' '
w(3, 5) = 9





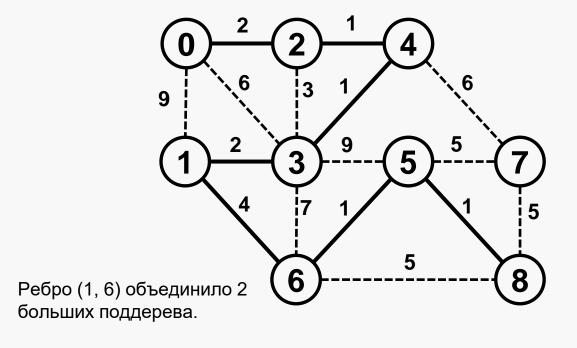
₩(2,	4)	=	4	
₩(3,	4)	=	1	
₩ (5,	6)	=	4	
₩ (5,	8)	=	4	
₩ (0,	2)	=	2	
₩(1,	3)	=	2	
₩(2,	3)	=	3	▼.
w(1,	6)	=	4	
w(5,	7)	=	5	
w(6,	8)	=	5	
w(7,	8)	=	5	
w(0,	3)	=	6	
w(4,	7)	=	6	
w(3,	6)	=	7	
w(0,	1)	=	9	
	_ \		_	

w(3, 5) = 9



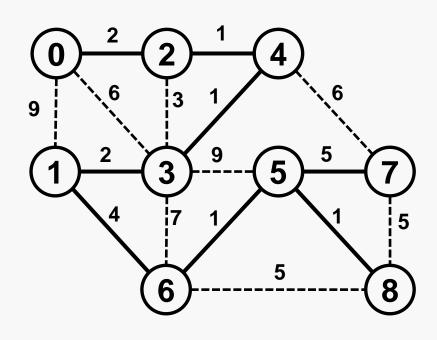


W(2, 4) = 1
w(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
w(1, 3) = 2
w(2, 3) = 3
W(1, 6) = 4
w(5, 7) = 5
w(6, 8) = 5
w(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
w(0, 1) = 9
w(3, 5) = 9



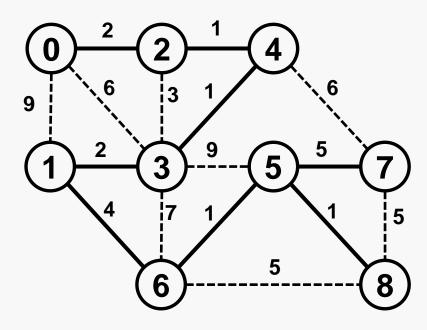


W(2, 4) = 1
W(3, 4) = 1
w(5, 6) = 1
w(5, 8) = 1
w(0, 2) = 2
w(1, 3) = 2
W(2, 3) = 3
w(1, 6) = 4
w(5, 7) = 5
w(6, 8) = 5
w(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
w(0, 1) = 9
w(3, 5) = 9
\





W(2, 4) = 1
W(3, 4) = 1
w(5, 6) = 1
W(5, 8) = 1
W(0, 2) = 2
W(1, 3) = 2
W(2, 3) = 3
W(1, 6) = 4
W(5, 7) = 5
w(6, 8) = 5
W(7, 8) = 5
w(0, 3) = 6
w(4, 7) = 6
w(3, 6) = 7
W(0, 1) = 9
(0 =)



Начиная с ребра (6, 8), у оставшихся рёбер концы принадлежат одному поддереву. Скипаем их.

Алгоритм Крускала. Оценка



Время работы алгоритма складывается из:

- 1. Сортировка рёбер = $O(E \log E)$
- 2. Объединение поддеревьев $O(V^2)$

Общее время работы = $O(E \log E) + O(V^2)$

Алгоритм Крускала. СНМ



Требуется СД «Система Непересекающихся Множеств» (CHM=DSU) с операциями

- 1) Make_set(u) создать множество с одним элементом u.
- 2) Find_set(u) найти множество по элементу u, чтобы можно было сравнить их (Find_set(u) == Find_set(v)).
- 3) Union_set(u, v) объединить два множества, одно из которых содержит элемент u, а другое элемент v.

Алгоритм Крускала. СНМ



Особенности реализации:

 Храним элементы каждого множества в виде дерева. Корень дерева называется представителем или лидером. Он возвращается методом Find_set.

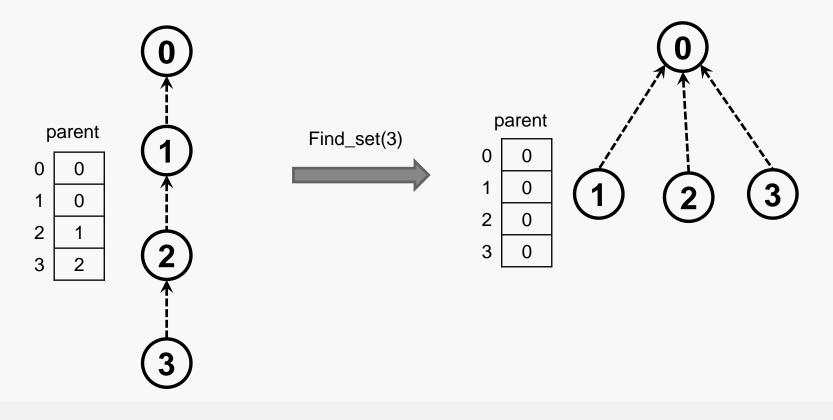
- Сравнение двух деревьев сравнение представителей.
- Объединение двух деревьев одному представителю в качестве родителя устанавливается другой представитель.
- Родитель корня (представителя) сам представитель.



СНМ. Эвристика сжатия пути.



При каждом новом поиске все элементы, находящиеся на пути от корня до искомого элемента, вешаются под корень дерева.

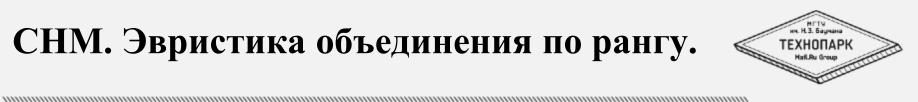


СНМ. Эвристика сжатия пути.

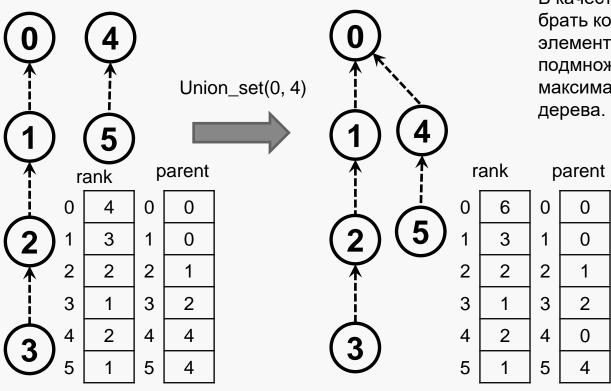


```
// Сжатие пути
int find_set (int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set (parent[v]);
}
```

СНМ. Эвристика объединения по рангу.



Идея в том, что мы присоединяем дерево с меньшим рангом к дереву с большим рангом.



В качестве ранга можно брать количество элементов в подмножестве или максимальную глубину

Алгоритм Крускала. СНМ. Оценка



Время работы алгоритма складывается из:

- 1. Сортировка рёбер = $O(E \log E)$
- 2. Объединение поддеревьев с помощью СНМ и двух эвристик $O(A^{-1}(V)) \approx O(1)$

Общее время работы = $O(E \log E)$

 $A^{-1}(V)$ - обратная функция Аккермана.

Это настолько медленно растущая функция, что для любых мыслимых применений ее значение не превышает 4 (примерно для $n \le 10^{600}$).

NP - полнота



Классы сложности алгоритмов:

• Класс Р – задачу можно решить за полиномиальное время от количества входных данных $(0(n^k)$, где n – кол-во входных данных, k - константа)

- Класс NP задача поддается проверке за полиномиальное время
- Класс NP полных задач подмножество задача класса NP, к которой можно свести любую задачу из этого класса за полиномиальное время.

Точно понятно, что $P \subseteq NP$

Проблема равенства классов P = NP — математическая проблема тысячелетия.

Большинство исследователей склонятся к тому, что $P \neq NP$

NP - полнота. Примеры задач.



Примеры NP-полных задач:

- Задача коммивояжёра
- Задача выполнимости булевских формул
- Кратчайшее решение пятнашек размером $n \times n$

- Сапёр
- Тетрис



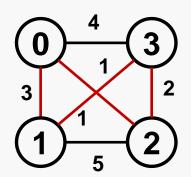
Пусть G – полный неориентированный взвешенный граф.

Коммивояжеру нужно совершить *кратичайший* тур, или гамильтонов цикл, посетив каждый город ровно по одному разу и завершив путешествие в том же городе, из которого он выехал.

Метрическая задача коммивояжера - на матрице стоимостей выполняется неравенство треугольника.

Задача коммивояжера - NP-полная, то есть существование быстрого алгоритма маловероятно.





Приближенные алгоритмы.



Многие задачи, представляющие практический интерес, являются NP-попными.

Варианты решения:

- 1. Если объем входных данных небольшой, то может подойти алгоритм, время работы которого выражается показательной функцией.
- 2. Иногда удается выделить важные частные случаи, разрешимые в течение полиномиального времени.
- 3. Иногда есть возможность найти за полиномиальное время решение, близкое к оптимальному.

Алгоритм, возвращающий решения, близкие к оптимальным, называется приближенным алгоритмом.

Оценка качества приближенных алгоритмов.



Говорят, что алгоритм обладает коэффициентом аппроксимации $\rho(n)$, если

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n)$$

n — размер входных данных

С – стоимость решения

 C^* – стоимость оптимального решения

ho(n)-приближенный алгоритм — алгоритм, в котором достигается коэффициент аппроксимации ho(n)



2-приближенный алгоритм

• G = (V, E) – полный неориентированный граф

- Каждому ребру $(u,v) \in E$ ставим в соответствие целочисленную стоимость $c(u,v) \ge 0$
- c(u,v) удовлетворяют неравенству треугольника, то есть для всех $u,v,w\in V$

$$c(u, w) \le c(u, v) + c(v, w)$$

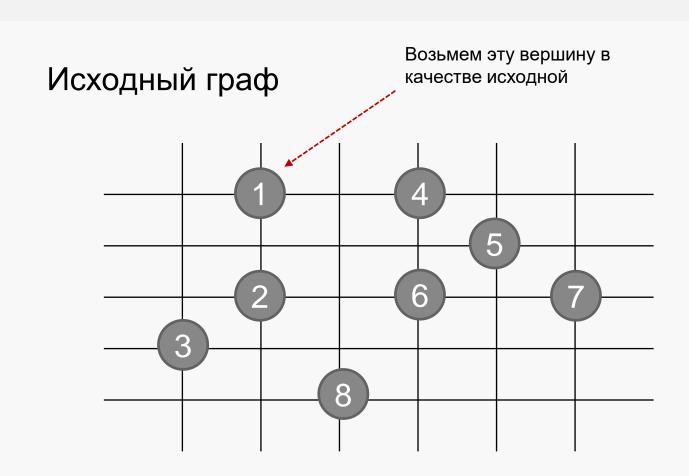
Если неравенство треугольника не выполняется, то приближенного алгоритма с полиномиальным временем работы не существует, если только не справедливо соотношение P = NP.



2-приближенный алгоритм Approx_TSP_Tour

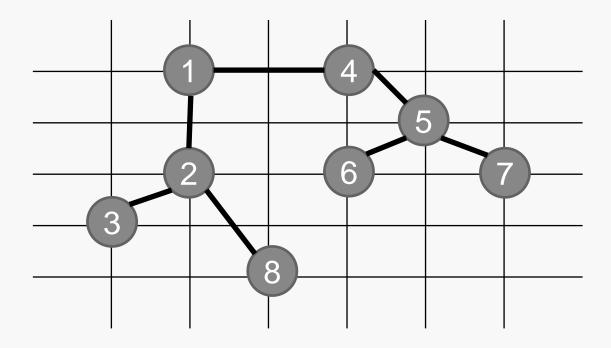
- 1. Выбирается произвольная вершина, которая будет корневой
- 2. Из этой вершины строится минимальное остовное дерево T для графа G
- 3. Обходим дерево Т в прямом порядке, получаем список вершин L в порядке прямого обхода дерева Т.
- 4. Получаем гамильтонов цикл Н, который посещает вершины в порядке их перечисления в списке L.





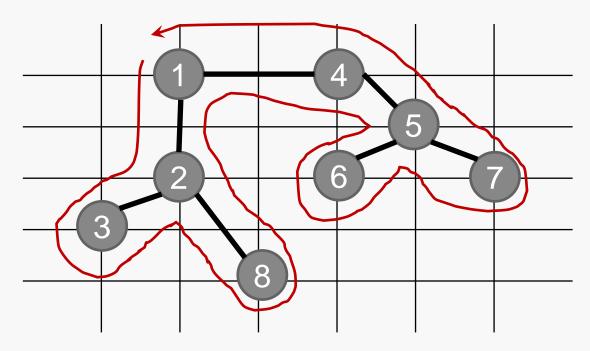


Строим минимальное остовное дерево





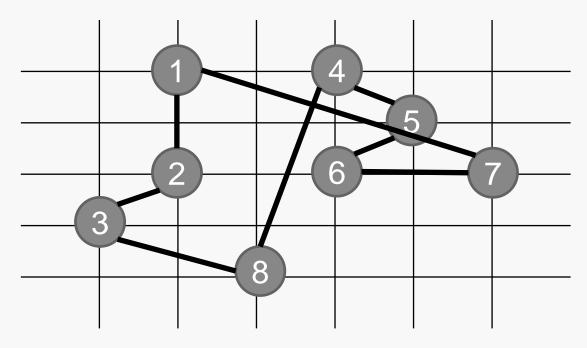
Обходим минимальное минимальное остовное дерево в прямом порядке, получаем список вершин L



$$L = \{1, 2, 3, 8, 4, 5, 6, 7\}$$



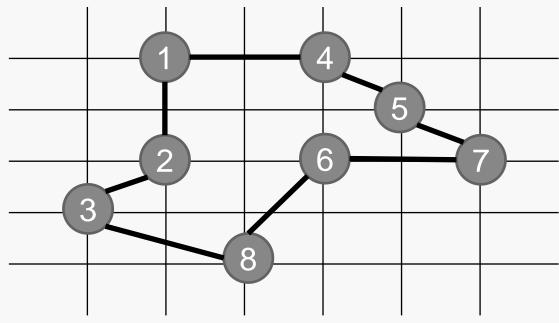
Получаем гамильтонов цикл H, который посещает вершины в порядке их перечисления в списке L.



$$L = \{1, 2, 3, 8, 4, 5, 6, 7\}$$



Оптимальный тур H^*



Длина тура H ~ 19.074 Длина оптимального тура H^* ~ 14.715



<u>Теорема.</u> Алгоритм Approx_TSP_Tour является 2-приближенным алгоритмом с полиномиальным временем работы, позволяющим решить задачу коммивояжера, в которой удовлетворяется неравенство треугольника.

Док-во.

 H^* - оптимальный тур. Удалив из него одно ребро получим остовное дерево, причем вес минимального остовного дерева Т является нижней границей стоимости оптимального тура, то есть $c(T) \leq c(H^*)$.

При полном обходе минимального остовного дерева все ребра посещаются 2 раза, обозначим этот обход через W. Из этого следует, что $c(W) = 2c(T) \Rightarrow c(W) \leq 2c(H^*)$, то есть стоимость обхода W не более чем в 2 раза больше стоимости оптимального тура.



Док-во (продолжение).

Но W – не тур, он посещает некоторые вершины более 1 раза. Но исходя из неравенства треугольника можно исключить из обхода все посещения вершины, кроме первой, и при этом стоимость пути не возрастет. То есть мы получим цикл H, он является гамильтоновым, так как вершины посещаются по одному разу. Так как H получается путем удаления вершин из W, то

$$c(H) \le c(W) \Rightarrow c(H) \le 2c(H^*)$$



Вопросы?

Спасибо за внимание!