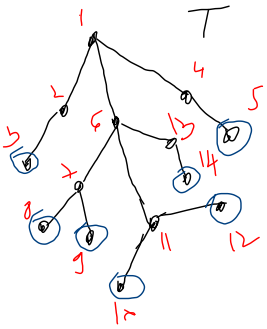
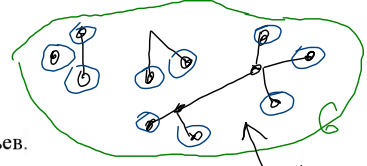




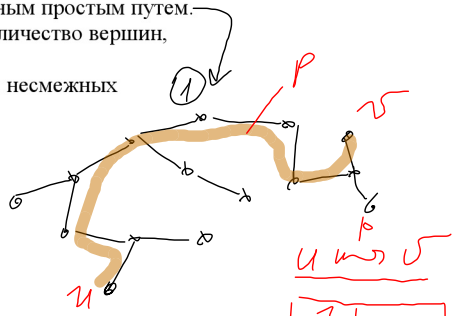
Деревья

(1) **Дерево** – связный ациклический граф

(6) **Лес** — граф, являющийся набором непересекающихся деревьев.

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.



Примечание:

- Любое ребро дерева – это мост (4)
- Дерево - это двудольный граф (5)

(2) p=14 вершин

q=13=p-1

и v, u

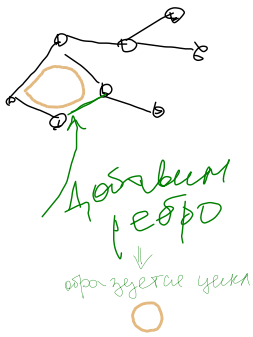
∃!p

по-другому никак

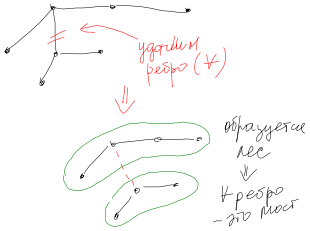
(0) **Лист** – висячая (концевая) вершина дерева (которая имеет степень равную единице).

есть и в лесу, и на дереве

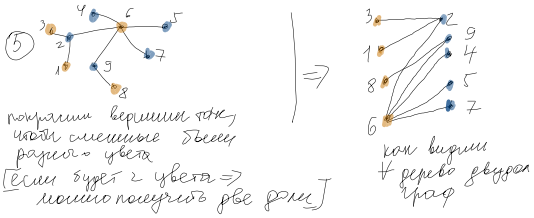
(3)



(4)



(5)



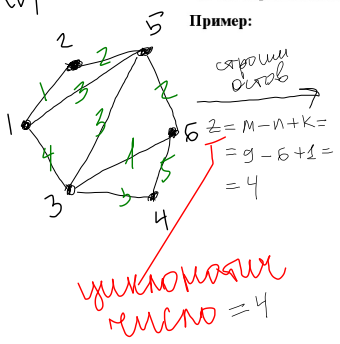
m = 9 (ребер)
n = 6 (вершин)
k = 1 (комп. связн.)
G(V,E)

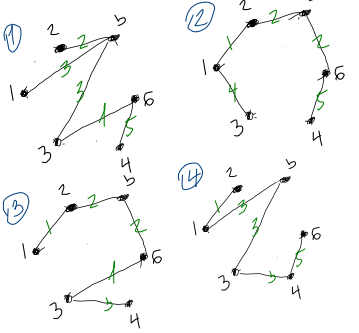
Минимальное остовное дерево

Остов графа (*англ. spanning tree*) – связный и ациклический частичный граф исходного графа

(частичный граф - тот же самый граф, только меньше ребер, а если ациклический - то это дерево, таким образом, остов графа – это его скелет в виде дерева)

Пример:





Σ ребер остова дерева

Σ① = 14
Σ② = 14
Σ③ = 15
Σ④ = 8

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) выше

MST – это наименьшее по Σ веса ребер остовное дерево

для G посчитано 4 остова из них MST это ④ дерево, но для точного ответа необходимо построить все возможные остовы графа

⇓

это долго !

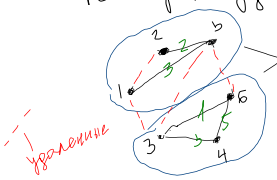
мы знаем z = 4 (цикломатическое число) сколько нужно удалить ребер, чтобы получить остов

⇓

будем удалять ребра с наиб весом?

⇓

нельзя удалять любые ребра



нужно сохранять связность графа и как итог получить остовное дерево, а не лес, как на примере

⇓

нужно удалять ребра с наиб весом, но при этом следить за связностью графа

⇒ использовать dfs на каждом шаге

⇒ не эффективно

⇓ ?

Как построить MST быстро и не используя много памяти (например хранить все остовы)

Примечание: для ∀ взвешенного графа м.б. несколько разных MST

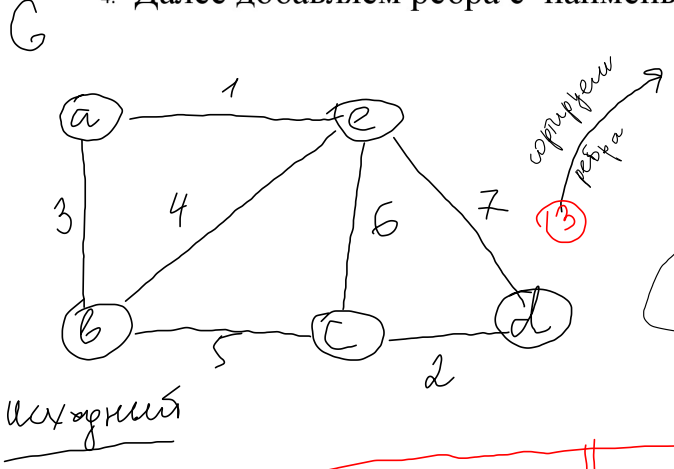
↳ А как получить все возможные MST для G? (ответ на практике)

1.Алгоритм Краскала

для поиска MST

Идея:

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

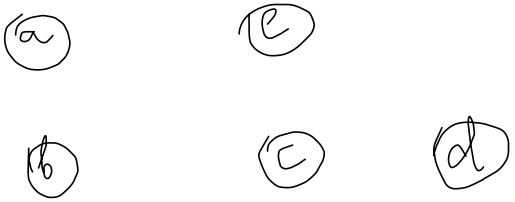


Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST							

массив пометок входит ли ребро в MST (вначале все не входят)

1 вычислим $z = m - n + k = 7 - 5 + 1 = 3 \Rightarrow$ удалить нужно 3 ребра или в MST $7 - 3 = 4$ ребра

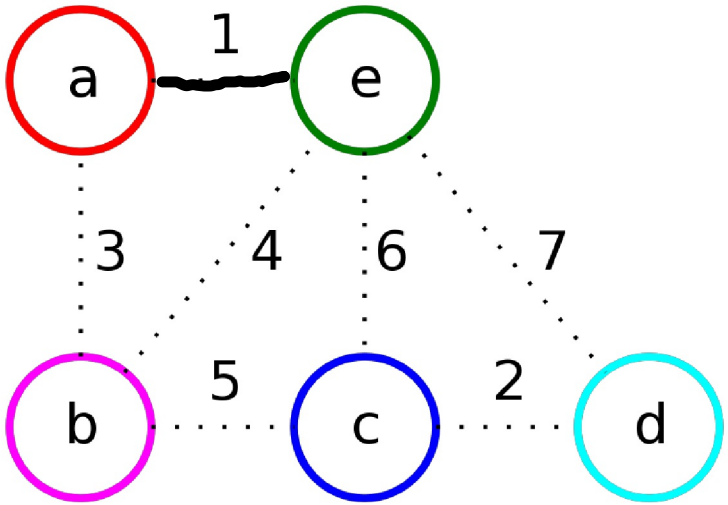
2, построим лес из вершин



3, выше в таблице уже показаны отсортир. ребра по возрастанию

4) выполняем по шагам добавление ук ребер

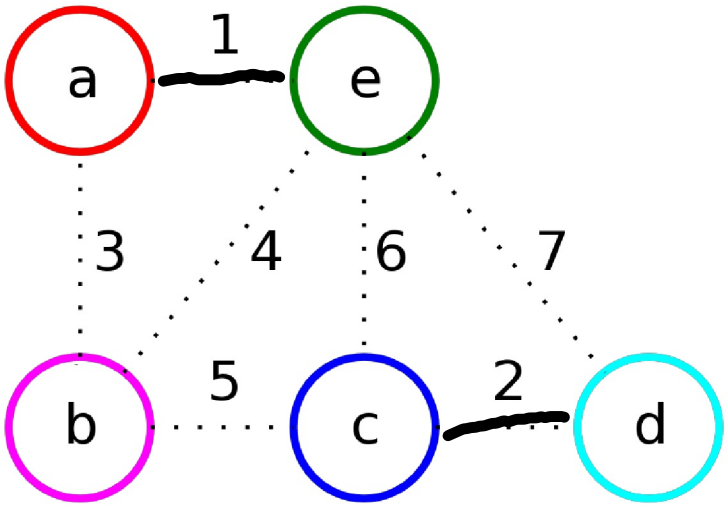
ШАГ 1: просматриваем первое ребро **ae**



Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+						

добавляем в MST так ценка не ограничивает вес min

ШАГ 2: просматриваем первое ребро **cd**

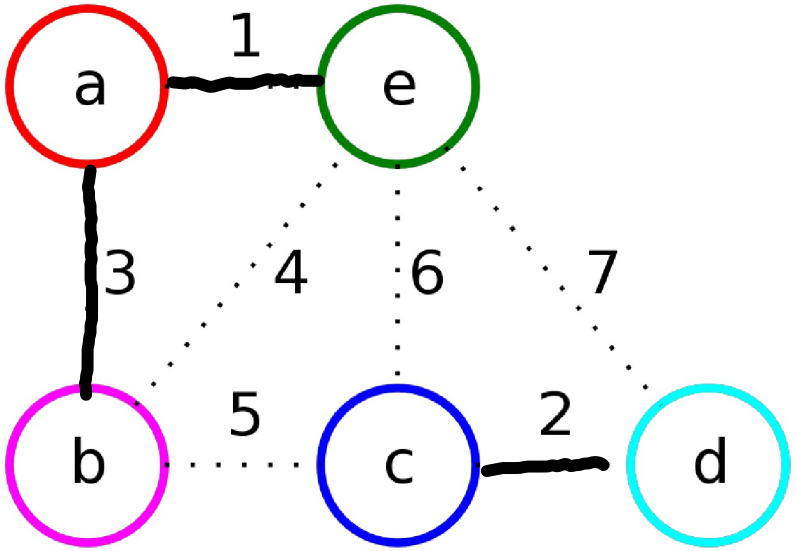


Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+	+					

добавляем нет ценки вес min

уже видим лес из трех деревьев

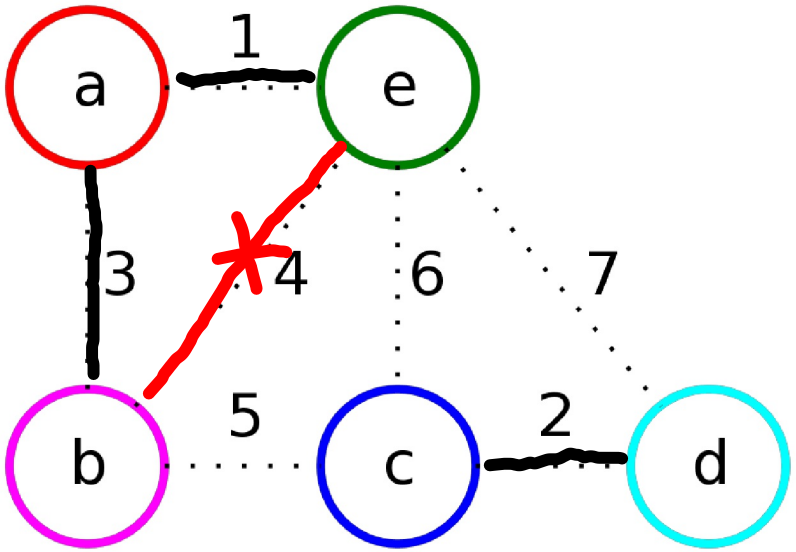
ШАГ 3: просматриваем первое ребро **ab**



Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+	+	+				

добавляем — нет цикла или вес

ШАГ 4: просматриваем первое ребро **be**

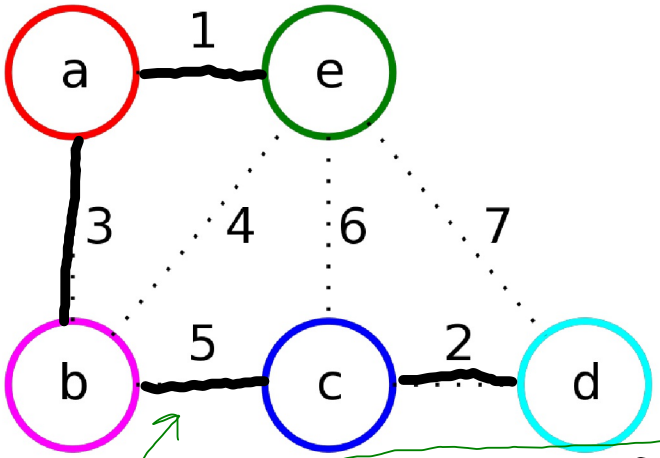


Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+	+	+	X			

нет так образуется цикл a-e-b-a

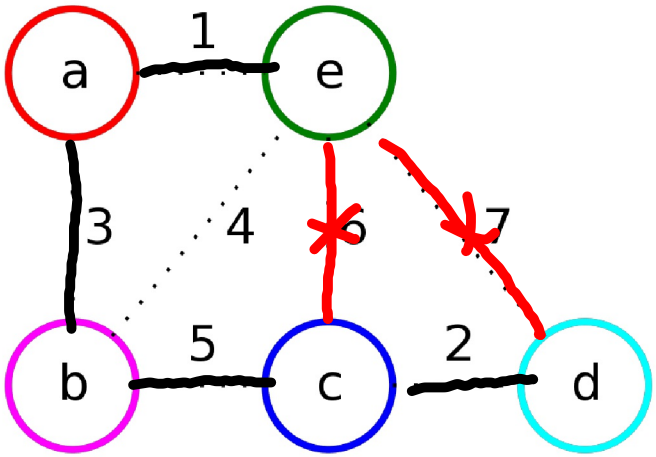
=> ответ будет не дерево

ШАГ 5: просматриваем первое ребро **bc**



Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+	+	+	X	+		

ШАГ 6: просматриваем первое ребро **ed**



Рёбра (отсортированы)	**ae**	**cd**	**ab**	**be**	**bc**	**ec**	**ed**
Веса рёбер	1	2	3	4	5	6	7
Ребро в MST	+	+	+	X	+	X	X

ВОПРОСЫ?

1. **Визуально искать цикл быстро, но в реализации это затратная операция: Каждый раз запускать обход в глубину ?**
   $|E| \cdot O(|E|+|V|)$ → очень долго
   → [+] если не считаем (E) то нужно просмотреть все ребра $|E| \cdot O(dfs)$ ⇒ это очень затратно
2. **Дерево строится в процессе из леса, а это значит использовать дополнительные структуры ?**
   + $O(|E|)$ по памяти
   → использ массив меток ребер, что входит в MST это + $O(|E|)$ памяти
3. **Нужно балансировать между структурами:**
   a. **стартовый граф, по которому ищем**
   b. **получаемое минимальное остовное дерево**
   $+O(|V| \times |V|)$ для граф по памяти
   → нужно строить копию графа, чтобы проверять образуется ли цикл при добавл ребра + операции добавления и удаления, а еще нужно проходить все ребра исходного графа
4. **Как представлять граф для реализации, чтобы удобно было сортировать ребра по весу ?**
   удобно в массиве ребер $O(E)$ вместо $|V| \times |V|$
   → если исп матрицу инцид-ти, то будет $|V| \times |E|$ памяти ⇒ это больше чем $3 \times |E|$
   массив ребер → { (1 2 22) — ребро ( ) ( ) } (вер вер вес)
5. **Как получить результат: итоговое минимальное остовное дерево в виде графа ?**
   эфф-но просто в массиве ребер добавить поле [метка] обозначающее входит ли ребро в MST
   ⇒ за $O(|E|)$ получить по массиву список ребер MST
   ⇒ зная m-z можно получить список ребер MST менее чем за $O(|E|)$ т.к. как только найдены все ребра из MST обход завершить!

Нужно более оптимально искать циклы в дереве

+ не дублировать граф

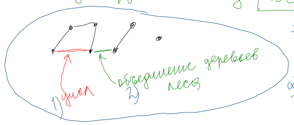
(X) dfs → не использ (X)

(X) + матрица копии (X)

на списке (массиве) ребер + поле для включение в MST

для оптимизация поиска цикла в дереве нам поможет св-во дерева
+ребро для несмежных вершин дерева ⇒ будет цикл

адаптируем св-во под [лес]



1 ситуация — образуется цикл если внутри комп

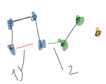
2 ситуация — не образуется цикл т.к между компонентами

⇓

Используем вместо поиска цикла распознавание ситуаций типа 1) и 2)

Как ?

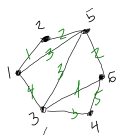
мы будем раскрашивать/помечать компоненты

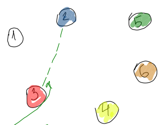


1) из вершины в вершину одного цвета
2) соединяем верш разного цвета

⇓ адаптируем под алгоритм Крускала

1) строим лес из вершин графа





это лес из 6 деревьев из 6 комп связности и все они разного цвета

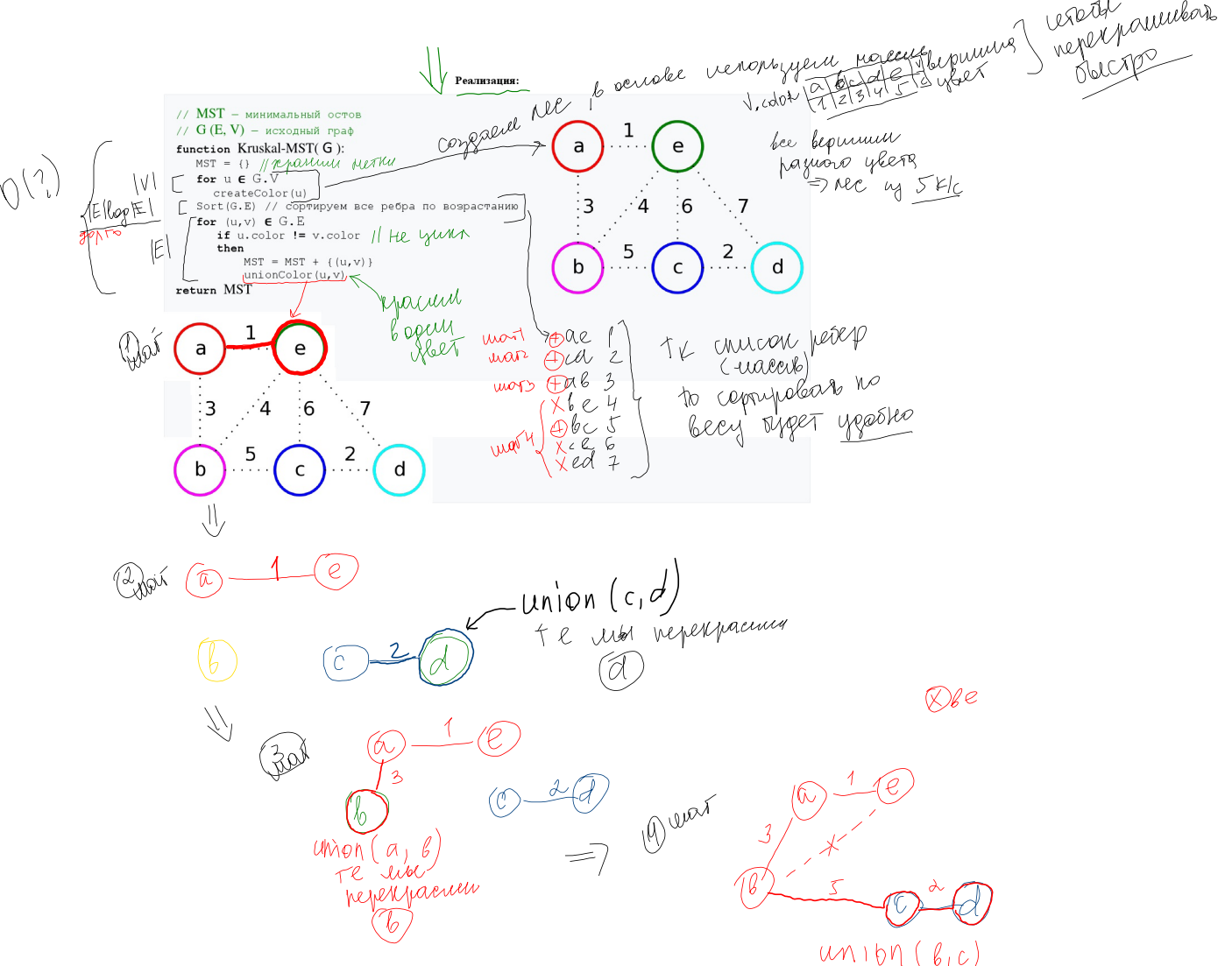
сортируем ребра
12 1
36 1
...
46 5

при добавлении перекрашиваем в один цвет обе компоненты

⇓

Реализация:

```
// MST – минимальный остов
// G (E, V) – исходный граф
function Kruskal-MST( G ):
   MST = {} // храним метки
   for u ∈ G.V
      createColor(u)
   Sort(G.E) // сортируем все ребра по возрастанию
   for (u,v) ∈ G.E
      if u.color != v.color // не цвета
      then
         MST = MST + {(u,v)}
         unionColor(u,v)
return MST
```



Мы получили MST
и не использ dfs
а так же не дублировали граф

+ массив меток ребер
+ массив цветов вершин
+ алгоритм закраски вершин по цвету

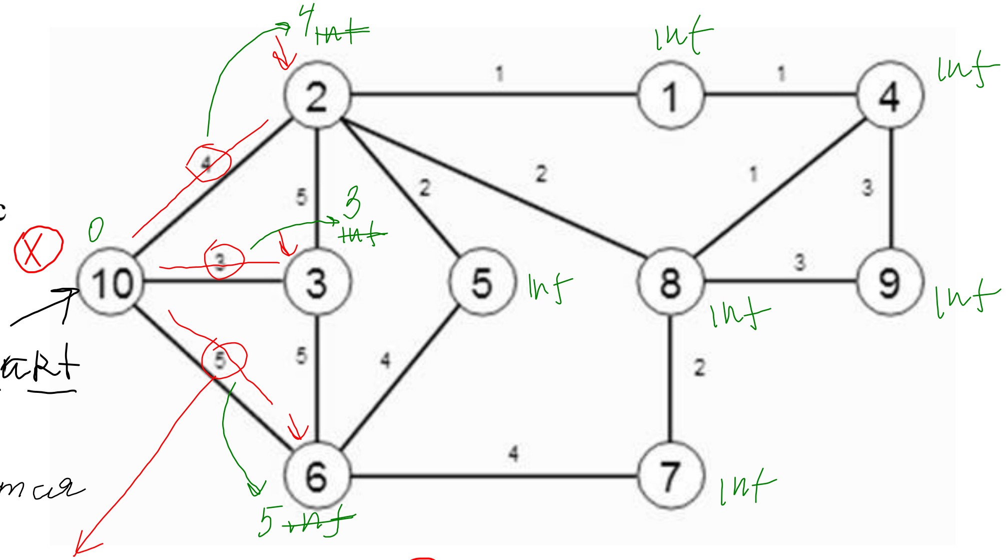
! эффективно !

2.Алгоритм Прима

поиск MST

Идея:

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



Для реализации цикла понадобится вспомогательный массив

индексы это вершина

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

значения: длина min по весу ребра до вершины (индекс) от всех помеченных

на каждом шаге пересчитываем длины т.к. новая вершина помечается

⇓

на старте у всех вершин inf = ∞

(N) пересчитали min ребра от помеченной (10) верш

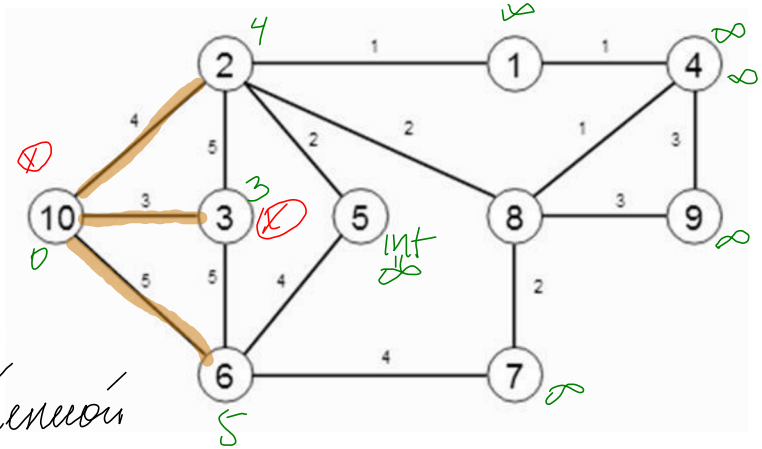
показано на графе

(X) – помеченная вершина

1. Начнём алго с вершины (10) ⇒ помечаем её

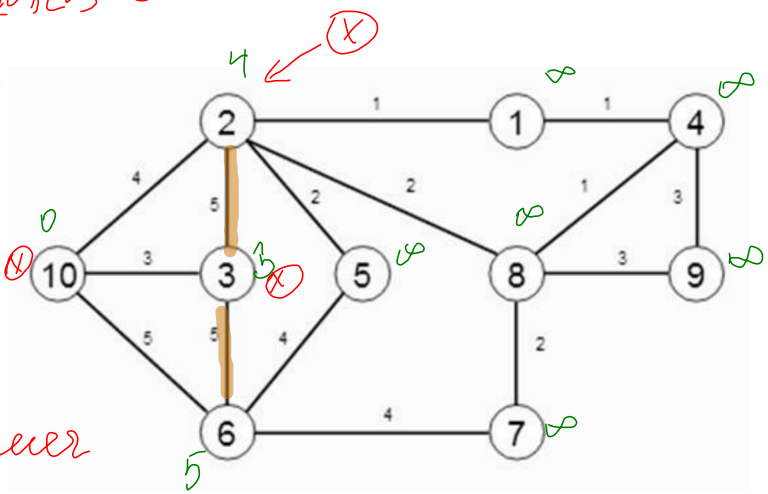
2. Выполним 1ю итерацию! от помеченной вершины (10) просмотрим все ребра и обновим min в массиве до непомеченных вершин т.к. после пометки с вершиной больше не работаем

ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	**3**	inf	inf	5	inf	inf	inf	0
			3							0
			3							0
			3							0
			3							0
			3							0
			3							0
			3							0
			3							0



ищем среди непомеч вершин ту до кот min ребро от помеченной
⇒ это 5=3 с min ребр w[10][3]=3

ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	3	inf	inf	5	inf	inf	inf	0
2	inf	**4**	3	inf	inf	5	inf	inf	inf	0
		4	3							0
		4	3							0
		4	3							0
		4	3							0
		4	3							0
		4	3							0
		4	3							0
		4	3							0



1) перечитываем от новой помеч вершины (3) min ребра (ничего не изменилось)
2) выбираем по пересеч массиву новую вершину до кт ведет min ребро от помеченных (10) и (3) и это (2) с w[3,2]=4

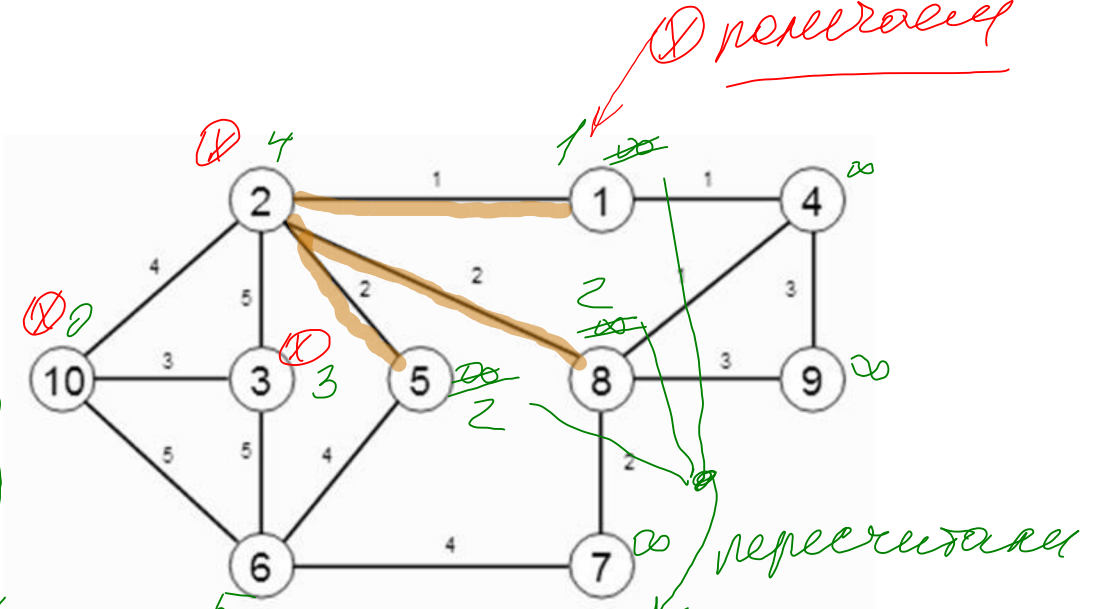
! важно: мы не сохраняем выбранные ребра и никак их не помечаем ⇒ нужно будет придумать на этапе реализации как получить MST по ребрам

ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	3	inf	inf	5	inf	inf	inf	0
2	inf	4	3	inf	inf	5	inf	inf	inf	0
3	**1**	4	3	inf	2	5	inf	2	inf	0
	1	4	3							0
	1	4	3							0
	1	4	3							0
	1	4	3							0
	1	4	3							0
	1	4	3							0

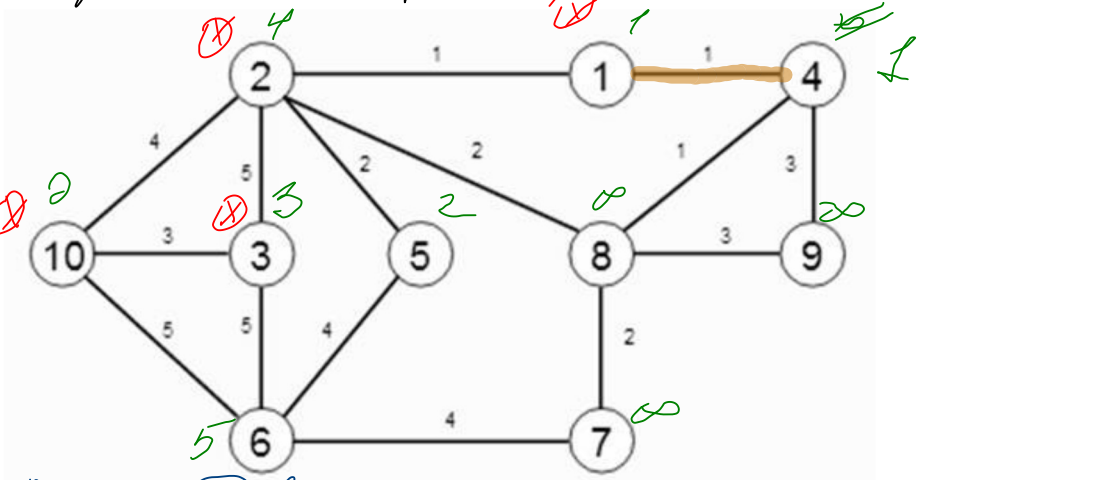
2) выбираем вершину с min ребром до неё
это (1) w[2][1] = 1

ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	3	inf	inf	5	inf	inf	inf	0
2	inf	4	3	inf	inf	5	inf	inf	inf	0
3	1	4	3	inf	2	5	inf	2	inf	0
4	1	4	3	**1**	2	5	inf	2	inf	0
	1	4	3	**1**						0
	1	4	3	**1**						0
	1	4	3	**1**						0
	1	4	3	**1**						0

2) помечаем новую вер, до кот min ребро
это (4) с w[3][4] = 1



1) пересчитали min ребра до вершин от ранее помеченной (2)

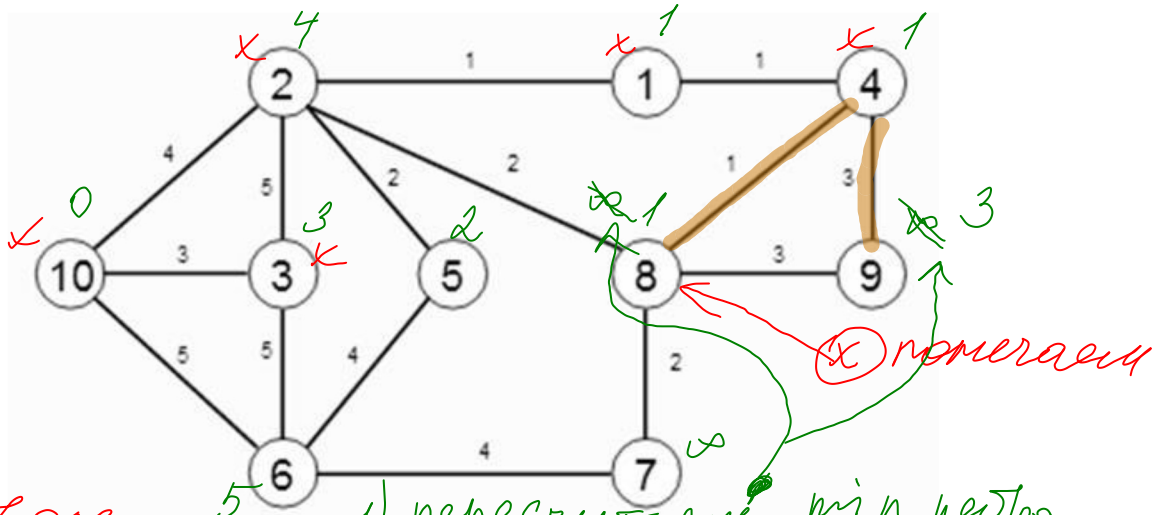


1) от помеч (1) вер, просмотрим ребра и пересчитаем min ребра доне помеч (шаг 4)

ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	3	inf	inf	5	inf	inf	inf	0
2	inf	4	3	inf	inf	5	inf	inf	inf	0
3	1	4	3	inf	2	5	inf	2	inf	0
4	1	4	3	1	2	5	inf	2	inf	0
5	1	4	3	1	2	5	inf	**1**	3	0
	1	4	3	1				**1**		0
	1	4	3	1				**1**		0
	1	4	3	1				**1**		0

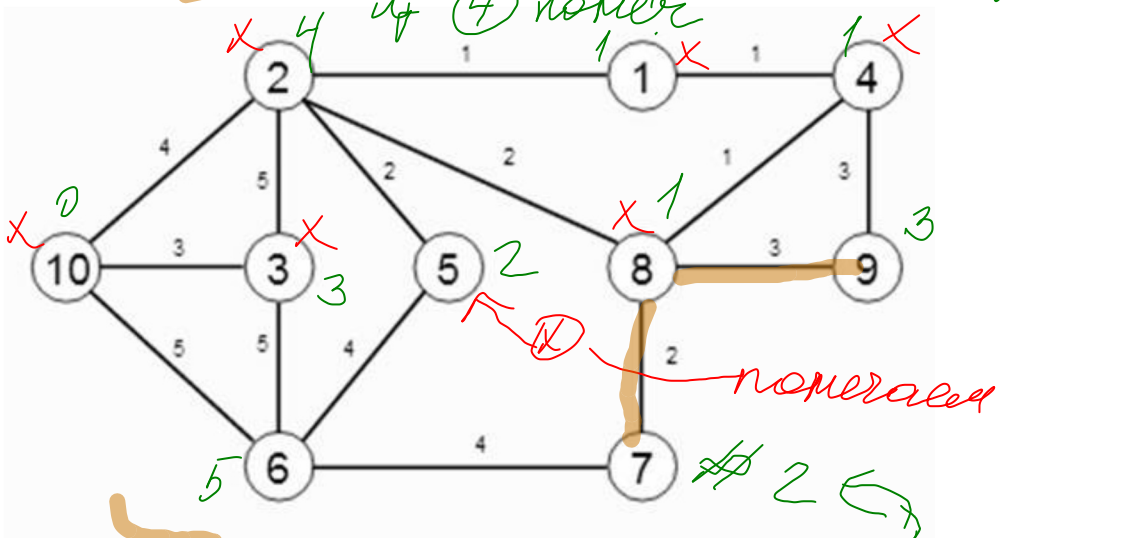
2) помечаем вершину ⑧ с min ребром

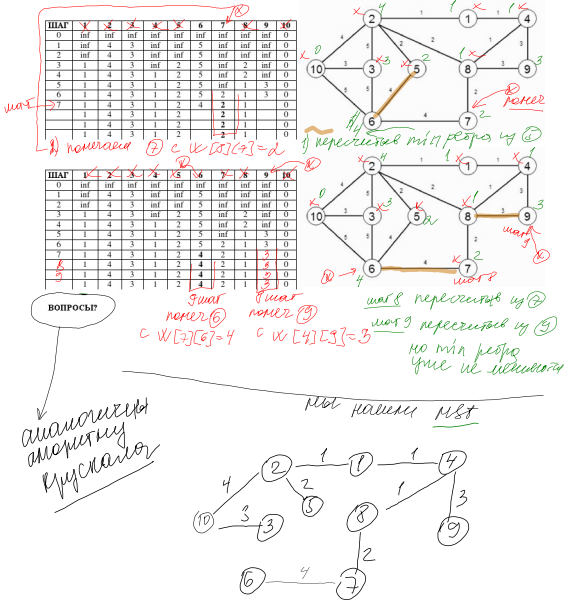
W[4][8] = 1



ШАГ	1	2	3	4	5	6	7	8	9	10
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	0
1	inf	4	3	inf	inf	5	inf	inf	inf	0
2	inf	4	3	inf	inf	5	inf	inf	inf	0
3	1	4	3	inf	2	5	inf	2	inf	0
4	1	4	3	1	2	5	inf	2	inf	0
5	1	4	3	1	2	5	inf	1	3	0
6	1	4	3	1	**2**	5	2	1	3	0
	1	4	3	1	**2**			1		0
	1	4	3	1	**2**			1		0
	1	4	3	1	**2**			1		0
	1	4	3	1	**2**			1		0

2) помечаем ⑤ с W[3][5] = 2
min ребром





аналогичный алгоритму Крускала

мы нашли MST

длина MST = 4+1+1+1+3+2+2+4 = 18

?1 Как мы восстановим само MST по ребрам, если не запоминали, не помечали ничего

⇒ интуитивно

нужно адаптировать под реализацию

как?

⇓

на каждом шаге мы пересчитывали min ребра и имели на этом этапе необходимо запомнить из какой вершины идет ребро с min весом W[i][j] ⇒

⇒ для σ_j · предок = σ_i

⇓

сама массив предков

σ_j	1	2	3	4	5	6	7	8	9	10
вершина предок σ_i	2	10	10	1	2	7	8	4	4	(пусто)

т.к. стартовая

на каждом шаге при пересчете, если менялось значение min ребра, то в предка запоминаем последнюю помеч вершину т.к. из нее рассматр ребра

?2 каждый раз искать min среди непомеч вершин это O(|V|)

т.к. нужно будет все просмотреть

⇓

оптимизировать можно?

⇓

используем для поиска min кучу с min ⇒ поиск min O(1)

а перестроение кучи за O(|V|log|V|)

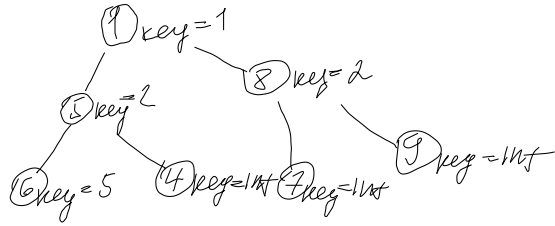
построение кучи O(|V|)

почему не будет быстрее

↓

т.к. мы строим кучу еще внутри поиска MST, а min ищем в самом цикле

пример кучи с min на шаг 3 после пересчета ребер



Реализация

Коротко:

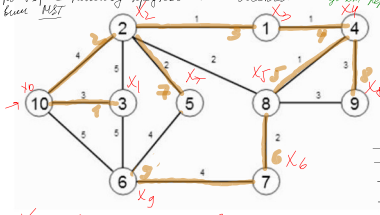
1. Создать приоритетную очередь, где стартовая - 0, остальные - бесконечность (inf)
2. Пока очередь не пуста
   a. Выбрать минимальную вершину из очереди и убрать ее = u
   b. Просмотреть все рёбра из неё до всех вершин в очереди (u, v)
      i. Если просматриваемое ребро меньше текущего пути
         1. Запоминаем новый путь V.p = u
         2. Запоминаем значение нового пути V.key = W[u][v]
         3. Изменяем положение вершины в очереди Q.decrease(v, W[u][v])

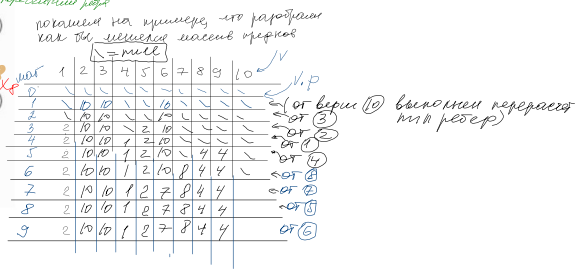
W[i][j] – все ребра
W – взвешенная матрица смежности = это G
Q – куча с min (приор очередь)
V – вершина в ней запомнит-но хранит
V.p = предок т.е. вершина из кт в V идет min ребро
V.key = значение min ребра до вершины V
по значению key строим кучу с min

O(E·log|V|)
O(E log V + 2V)
O(2V)
O(V)
O(E log V)
O(V) все for O(E)
O(log V)

```
// G – исходный граф
// w – весовая функция
function prim-MST(G, s):
    for v ∈ G.V
        v.key = ∞ // изначально до всех вершин min ребро (∞)
        v.p = null // пустых, нет предков у вершин по min ребру
    s.key = 0 // стартовая вершина ( у нас была 1)
    creatHeap( G.V ) // строим кучу с min
    while not Q.isEmpty()
        u = Q.extractMin()
        for v ∈ E.G[u]
            if v ∈ Q and v.key > w(u,v)
                v.p  = u // сохраняем предка
                v.key = w(u,v) // пересчитав min ребро
                Q.decreaseKey(v, v.key)// sift-UP (v) просеивание
```

по V.P = массиву предков мы восстановим MST





шаг	1	2	3	4	5	6	7	8	9	10	
0	\	\	\	\	\	\	\	\	\	\	
1	\	10	10	\	\	10	\	\	\	\	от (3)
2	\	10	10	\	2	10	\	\	\	\	от (2)
3	2	10	10	1	2	10	\	\	\	\	от (1)
4	2	10	10	1	2	10	\	\	\	\	от (4)
5	2	10	10	1	2	10	\	4	4	\	от (5)
6	2	10	10	1	2	10	8	4	4	\	от (8)
7	2	10	10	1	2	7	8	4	4		от (7)
8	2	10	10	1	2	7	8	4	4		от (6)
9	2	10	10	1	2	7	8	4	4		от (6)

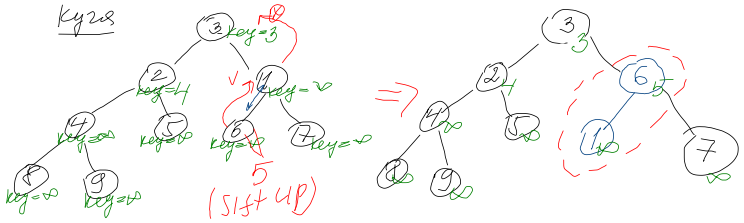
X_i – это пометка вершины на шаге i

i – добавление ребра к MST

Разберем пример перестроения кучи с min при изменении min у вершин при пересчете min ребер

Шаг 1

пересчитали для (3) и (2) и пересчитываем для (6)



когда у (6) изменился пересчет с 8 до 5 необходимо провести просеивание вверх sift UP (V, W[3][6])

всего 2 вспомогат-х массива [min ребра, предки]
+ очередь для меток вершин и ускорения поиска min
что позволяет эфф искать MST

⇓

Для восстановления MST необходимо пройтись по массиву предков V.P и восстановить ребра по всему W[V.p][V]

V	1	2	3	4
V.P	2	10	10	

для V=1 W[2][1] ребро (2,1)

Поиск наикратчайшего пути

Путь (маршрут) – последовательность вершин и ребер вида $p = v_1 e_1 v_2 e_2 v_3 \dots v_k$

- длина пути количество ребер в нем (НЕ взвешенный граф)
- вес пути - сумма весов всех ребер пути (взвешенный граф)

Путь: **p = <V0, V1, V2, ... Vk>** – обозначение без ребер

Вес пути **w(p)** = $\sum_{i=1}^{k} w(e_i) = \sum_{i=1}^{k} W[v_i][v_{i+1}]$

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 возможных или равен бесконечности, если пути из **u в v** нет

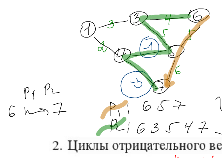
Варианты задачи поиска наикратчайшего пути

1. между заданными: **из u в v**
2. от заданной до всех остальных
3. между всеми вершинами (от всех до всех)

Использовать переборные алгоритмы не эфф-но для поиска

Важные ограничения: может ли наикратчайший путь содержать ?

1. Ребра отрицательного веса



ребро (4,5) $w(4,5) = -1$
ребро (4,7) $w(4,7) = -3$

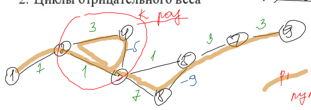
такие ребра уменьшают вес пути, но увеличивают число пройденных ребер для уменьшения

$p_1 \; p_2$
$6 \rightsquigarrow 7$

p_1: 6 5 7
p_2: 6 3 5 4 7 ⇒ $w(p_1) = 11$ ← пример
$w(p_2) = 5$

⇒ не все алгоритмы могут корректно работать с такими ребрами (в жизни так тоже ребра берет, редко) → например жадные (Дейкстра) (Прима)

2. Циклы отрицательного веса



как влияют такие циклы на наикр пути?

$1 \overset{p_1}{\rightsquigarrow} 9$
путь p_1 может содержать к раз цикл 2 3 4 2 ⇒ ∃ пройдем по циклу 200 раз тогда

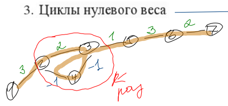
$w(p_1) = 7 + 200 \cdot (-1) + 7 + (-9) + 3 + 3 = -189$
200 раз по циклу

накручивая по циклу отриц веса можно длину пути $\rightarrow -\infty$ т.к. $k \rightarrow \infty$

• наличие такого цикла в пути снижает вес пути

цикл 1: 2 3 4 2 $w(1) = -1$
цикл 2: 3 5 8 3 $w(2) = -1$
цикл 3: 2 4 3 5 8 3 2 $w(3) = -2$
→ это циклы отрицат веса, т.е. $\sum$ весов ребер цикла < 0 (отрицательная)

3. Циклы нулевого веса



цикл 2 4 3 2 имеет вес = 0
⇒ он цикл нулевого веса

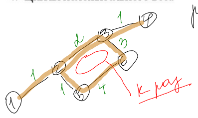
$1 \overset{p_1}{\rightsquigarrow} 7$
p_1: 1 2 3 4 2 3 5 6 7 (цикл нулевого веса) к раз

$w(p_1) = 3 + 0 \cdot k + 2 + 1 + 3 + 2 = 11$ (к и 0 ≠ числом $w(p_1)$ не изменится)

т.е. цикл нулевого веса не влияет на вес пути, но увеличивает число пройденных ребер ⇒ если отбросить такой цикл из пути вес не изменится

преобразуем p_1 → $p_1' = 1\,2\,3\,5\,6\,7$
$w(p_1') = w(p_1) = 11$

4. Циклы положительного веса



цикл 1: 2 5 6 3 2
$w(1) = 2 + 1 + 4 + 3 = 10$
→ это цикл положит-го веса (обычный)

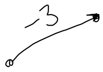
рассмотрим $1 \overset{p_1}{\rightsquigarrow} 4$
p_1: 1 2 3 6 5 2 3 4 (цикл положит. к раз веса)

$w(p_1) = 1 + 10 \cdot k + 2 + 1 = 14$

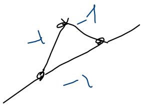
∃ k = 20
$w(p_1, k=20) = 1 + 20 \cdot 10 + 2 + 1 = 204$

$k \uparrow \Rightarrow w \uparrow$
чем больше циклов в пути ⇒ тем больше w (вес) пути ⇒ наикр путь не будет наикратч

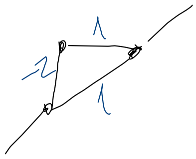
Выводы



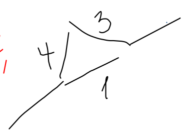
1. ребра отрицат веса наикр путь может содержать тк уменьшается $w(p) \downarrow$



2. циклы отрицат веса
наикрт путь не может содержать тк такой вес такого пути может устремить к $-\infty$ $w(p) \rightarrow -\infty$ ⇒ ∄ наикратч в этом случае



3. цикл нулевого веса
наикратч путь не может содержать тк не влияет на вес, но значение может увеличить число таких путей и время поиска такого пути



4. цикл положит веса
не содержит наикратч пути т.к. $w(p) \uparrow$ при наличии цикла

Итог

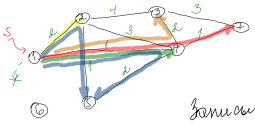
Наикратчайший путь (p) — д.б. ацикличным
∃ G(V,E) ⇒ в ∀p не более |V| вершин и |V|−1 ребер

Ослабление ребра (релаксация)

Как мы будем искать вес наикратчайшего ациклического пути ?

distance [v] или d[v] - вес кратчайшего пути от S до V

- на каждом шаге это оценка веса кратчайшего пути сверху (на старте считаем бесконечностями или что пути такого нет)



$S = 1$

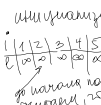
$V = \{1,2,3,4,5,6,7\}$

будем искать от (1) до всех остальных вершин наикратч пути

* наикрат путь p — ациклический и содержит не более $|V|$ вершин и $|V|-1$ ребер

Записывая длину наикрат будем в массив

distance [i] / d [i]



инициализация

i	1	2	3	4	5	6	7
e	0	∞	∞	∞	∞	∞	∞

до начала поиска считаем, что от S до всех вершин ребра не имеют, т.е. $l = \infty$ длина пути

после поиска наикрат путей

i	1	2	3	4	5	6	7	← индекс вершин
e	0	2	3	1	3/3	∞	2	← длина наикрат пути от S до i

т.к. $S=1 \Rightarrow distance[s] = 0 = distance[1] = 0$

расстояние для вершины, от кот ищем пути до остальных 0 т.к наикр путь от самой себя до себя $= 0$

Как искать наикрат-й путь ?

суть: будем просматривая граф и искать возможные пути, если они будут лучше (более найденных → дополнять новыми

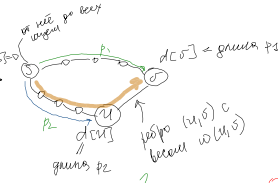
O(1)

Ослабление ребра - попытка улучшить найденный путь до вершины

релаксация ребра

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

смысл



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

тыкаемся ослабить ребро (u,v), а именно:

улучшим возможный наикрат путь до вершины (v), кт пойдем по ребру (u,v) и будем иметь длину равную $d[u] + w(u,v)$ где $d[u]$ — ранее найденный наикрат путь до (u) от S + + вес ребра $w(u,v)$

как короче добраться до (v)

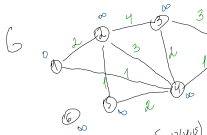
то путь p1	или	p2 + (u,v) новый путь

нужно сравнить

$d[v]$? $d[u] + w(u,v)$

если новый путь $p_2 + (u,v)$ будет короче ранее найденного p_1, то перезаписываем значение $d[v] = d[u] + w(u,v)$ теперь мы нашли новый более короткий путь

проверяем будет ли короче до (v) от (s) добраться по ребру (u,v) с учетом ранее найденного наикрат пути до (u)



i	1	2	3	4	5	6	7
e	0	∞	∞	∞	∞	∞	∞

(i=1) искать будем от S=1 вершины до всех остальных

v & $w(u,v)$ попробуем последовательно ослабить все ребра, начиная с ребер инцидентных (1)

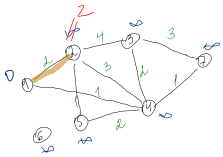
1) Relax (1,2,2);

$d[1]=0 \quad d[2]=\infty$

? $d[2] > d[1] + w(1,2) = 2$

$\infty > 0 + 2$

$d[2] = d[1] + w(1,2) = 0+2 = 2$



i	1	2	3	4	5	6	7
e	0	2	∞	∞	∞	∞	∞

перезаписали более короткий путь до (2)

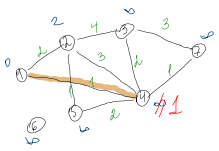
2) Relax (1,4,1)

$d[1]=0 \quad d[4]=\infty \quad w(1,4)=1$

? $d[4] > d[1] + w(1,4)$

$\infty > 0 + 1$

$d[4] = d[1] + w(1,2) = 0+1 = 1$



i	1	2	3	4	5	6	7
e	0	2	∞	1	∞	∞	∞

перезаписали более короткий путь до (4)

теперь возьмем вершину (2) и ослабим все инцидентные ей ребра !

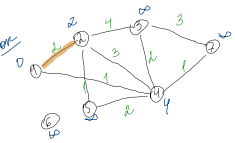
3) Relax (2,1,1)

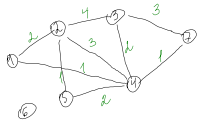
$d[2]=2 \quad d[1]=0 \quad w(1,2)=\infty$

? $d[1] > d[2] + w(1,2)$

$0 > 2 + \infty$

$d[1]$ остается не изменным





4) Relax (2,5,1)

$d[2]=2 \quad d[5]=\infty \quad w(2,5)=1$

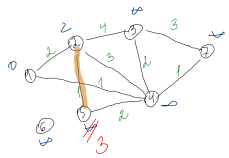
? $d[5] > d[2] + w(2,5)$

$\infty > 2 + 1$

$d[5] = d[2] + w(2,5) = 2+1 = 3$

i	1	2	3	4	5	6	7
e	0	2	∞	1	~~∞~~ 3	∞	∞

перезаписали более короткий путь до (5)



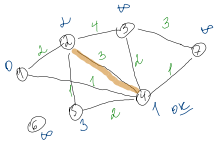
5) Relax (2,4,3)

$d[2]=2 \quad d[4]=1 \quad w(2,4)=3$

? $d[4] > d[2] + w(2,4)$

$1 > 2 + 3$

$d[4]$ содержит более оптимальный путь $\Rightarrow$ не перезаписываем



6) Relax (2,3,4)

$d[2]=2 \quad d[3]=\infty \quad w(2,3)=4$

? $d[3] > d[2] + w(2,4)$

$\infty > 2 + 4$

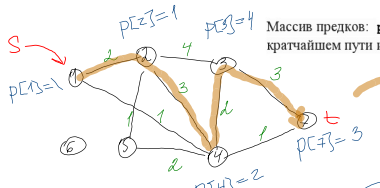
$d[3] = d[2] + w(2,4) = 2+4 = 6$

i	1	2	3	4	5	6	7
e	0	2	~~∞~~ 6	1	3	∞	∞

перезаписываем на более оптимальный путь до (3)

Как восстановить наикратчайший путь ?

Массив предков: **predki[v] или p[v]** – предшествующая вершина на кратчайшем пути или null



p1

p1 : S ⇝ t Как восстановить путь p1 ?

Используем уже знакомый массив предков

predki = p

вершина	1	2	3	4	5	6	7
и предок	\	\	\	\	\	\	\

\ = null

состояние массива после того как нашли p1 : S ⇝ t

состояние массива до начала выполнения поиска наикратчайших путей

predki = p

v	1	2	3	4	5	6	7
u	\	1	4	2	\	1	3

если нам нужно восстановить путь p1 от 1 ⇝ 7, то

```
i = 7   print(i)
while p[i] <> null
    print(' ')
    i = p[i]
```

результат

7 3 4 2 1 наш искомый путь

(используя стек можно реверсивно вывести путь 1 → 2 → 4 → 3 → 7)

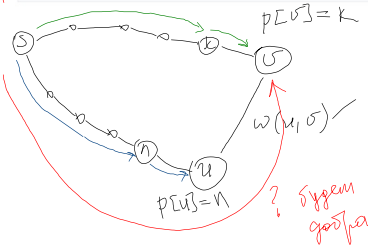
⇒ КАК НАМ ПОЛУЧИТЬ ЭТОТ ВОЛШЕБНЫЙ МАССИВ ПРЕДКОВ ?

ОТВЕТ

Доработаем реализацию ослабления для восстановления кратчайшего пути

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

O(1)



? будет ли короче через u и ребро (u,v) добраться до v

⇓

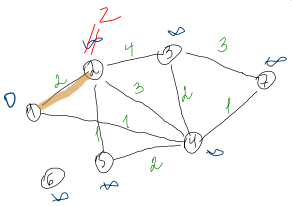
если короче до v через ребро (u,v)

⇓

p[v] = u

т.к. теперь для v предок u вершина ослабленного ребра (u,v)

рассмотрим несколько ослаблений ребер и формирование массива предков



1) Relax (1,2,2);

d[1]=0 d[2]=∞

? d[2] > d[1] + w(1,2)=2

∞ > 0+2

⇓

d[2] = d[1] + w(1,2) = 0+2 = 2

p[2] = 1

мы нашли более короткий путь до ② через ребро (1,2) ⇒ предок для ② это ①

distance = d

i	1	2	3	4	5	6	7
e	0	2	∞	∞	∞	∞	∞

2

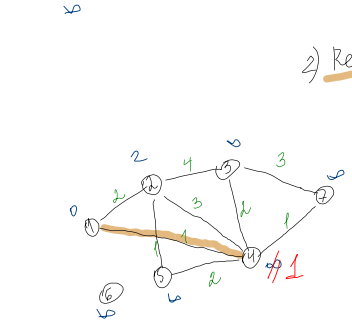
перезаписали более короткий путь до ②

predki = p

v	1	2	3	4	5	6	7
u	\	1	\	\	\	\	\

1

изменим предка



2) Relax (1,4,1)

d[1]=0 d[4]=∞ w(1,4)=1

? d[4] > d[1] + w(1,4)

∞ > 0 + 1

⇓

d[4] = d[1] + w(1,2) = 0+1 = 1

p[4] = 1

найден более короткий путь до ④ через ребро (1,4) ⇒ предок для ④ это ①

d

i	1	2	3	4	5	6	7
e	0	2	∞	1	∞	∞	∞

1

перезаписали более короткий путь до ④

p

v	1	2	3	4	5	6	7
u	\	1	\	1	\	\	\

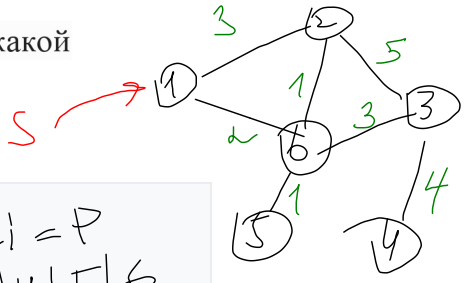
1

изменим предка так как нашли более короткий путь

Инициализация дополнительных структур

Нужно заполнить distance [v] бесконечностями, так как на старте нет найденных кратчайших путей

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




```
InitSource(G, s):
    for v ∈ V
        d[v] = infinity
        predki[v] = null
    d[s] = 0
```

O(|V|)

до стартовой длина пути = 0

predki = P

u	1	2	3	4	5	6
v	\	\	\	\	\	\

distance = d

0	1	2	3	4	5	6
0	∞	∞	∞	∞	∞	∞

d[s] = 0

такого рода инициализация структур
- массив предков
- длина кратчайших путей до вершин

необходимо будет выполнять для всех алгоритмов поиска кратчайших путей

также как и используемое relax (u, v, w(u, v))

в ∀ алгоритме необходимая операция ослабление ребра

и именно во время ослабления мы корректируем d[v] и p[v] !

1. DAG: кратчайшие пути в ациклическом ориентированном графе

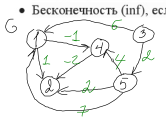
Суть: ослабление ребер в порядке топологической сортировки

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

G

	1	2	3	4	5
1	inf	inf (1)	inf	-1	inf
2	inf	inf	inf	inf	inf
3	6	inf	inf	inf	2
4	inf	-2	inf	inf	inf
5	7	inf (2)	inf	inf (4)	inf

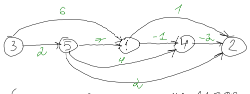
- Вес дуги (число), если дуга есть
- Бесконечность (inf), если дуги нет



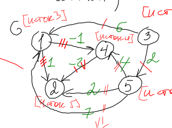
топологически отсорт, используем алг. Демукрона (будем удалять истоки)

ТОПСОРТ: 3 5 1 4 2

вершины по топ сорту слева направо



перерисуем граф по его топологич сортир



\ первое удаление исх дуг
\\ второе удаление
\\\ третье удал-ие
\\\\ четвертое

ист1 ист2 ист3 ист4 ист5
3 5 1 4 2
это топологич сортировка

благодаря топсорту ацикл графа мы видим что все пути идут по направлению топсорта ⇒ вариантов исследований становится намного меньше
т.к. после нахождения для вершины оптим. пути и перехода к поиску для следующ вершины согласно порядку в топологич. сортир для предыдущих вершин пути не будут пересчитываться т.к. все пути направлены по тополог. сорт.

нашли наикр пути для ① и ②

при поиске наикр пути для ③ и ④ пути для ① и ② не будут пересчитаны тк к ним нет дуг

⇒

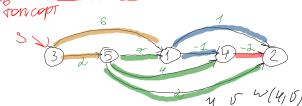
Суть АЛГ: ищем наикрат от S до всех остальных
1) топологич сортируем ациклич. ор. граф
2) согласно топ. сортировке от S и до последней вершине топ сорта просматрив. вершины: выполняем ослабление всех инцидентных ребер просматриваемой вершине
→ записываем длину пути в [d]
→ предков в массив [p]

⇓ коротко
1) топ сорт
2) по порядку топ сорт ослабляем все ребра } ⇒ в d → длины; в p → сами пути

смотри на примере

топологически отсортир граф G:

→ просмотрим вершины в порядке топсорт ⇒ [3, 5, 1, 4, 2]



u v w(u,v)

от вершины ③ шаг 1 Relax (3, 5, 2)
Relax (3, 1, 6)

от ⑤ шаг 2 Relax (5, 1, 7)
Relax (5, 4, 4)
Relax (5, 2, 2)

от ① шаг 3 Relax (1, 4, -1)
Relax (1, 2, 1)

от ④ шаг 4 Relax (4, 2, -2)

от ② нет исх. дуг

⇒ в итоге мы получили все наикрат пути от ③ до всех

predki = p

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

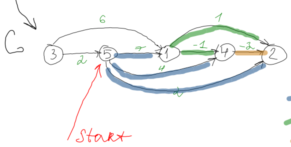
distance = d (S → 3)

1	2	3	4	5
inf	inf	0	inf	inf
6	inf	0	inf	2
6	4	0	6	2
6	4	0	5	2
6	3	0	5	2

Инициализация массивов перед выполнением алгоритма

? А если бы нужно было найти все возможные пути от вершины ⑤ ?

рассмотрим G и топологич отсортир
3 5 1 4 2



начинаем просматрив вершины и ослаблять инцидентные ей ребра согласно топсорту от заданной вершины ⑤

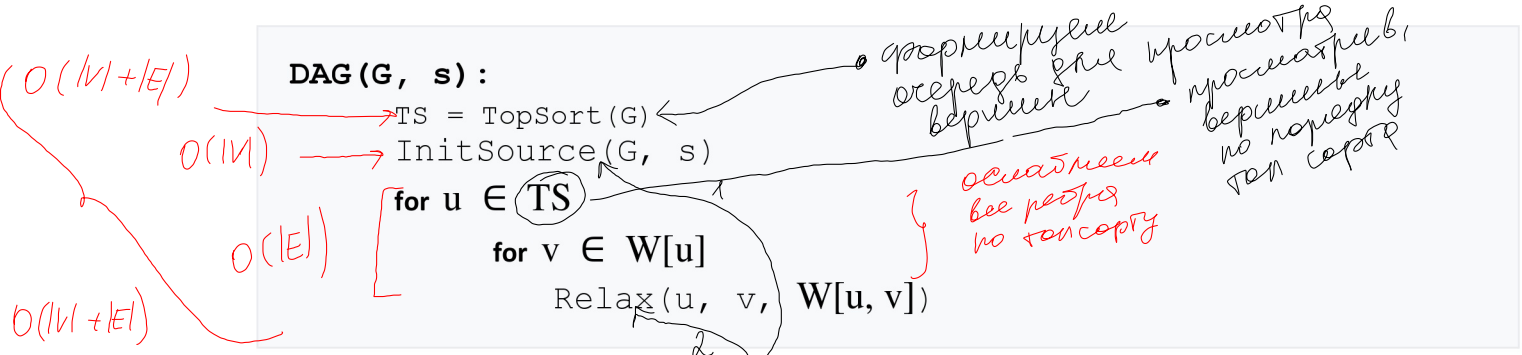
от ⑤ шаг 1
от ① шаг 2
от ④ шаг 3
от ② нет дуг

(ребра разобраны слева на примере)

distance / predki

i	1	2	3	4	5	1	2	3	4	5
	7	2	∞	4	0	5	5	\	5	⅄

результаты совсем другие, но сам алгоритм тот же
и как мы видим до предшествующей вершины ③ в топ сорте пути нет



работа с массивами d и p происходит внутри

1) → заполнили изначал данными
2) → при каждом ослаблении корректируем p и d

Примечание: как модернизировать реализацию, чтобы можно было найти кратчайшие пути от ∀ вершины в топ сорте, а не только от первой?

!! ответ

TS – очередь просмотра вершин, а значит при инициализации проверяем ? s = первой вершине TS.first в очереди

1. если да → без изменений
2. если нет → удаляем вершины из очереди пока не найдем s

!!

мы просмотрим только необходимые дуги и найдем корр кратчайшие пути

Алгоритм

1) Работает с отрицат-ми ребрами (−3)
2) Не работает с циклами при ∃ ∀
3) проходит только один раз все ребра

от одной (5) до всех остальных в G

2. Алгоритм Беллмана - Форда (поиск наикратчайших путей)

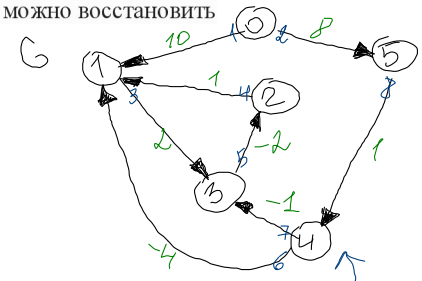
Суть: ослаблять все ребра графа |V|−1 раз

- 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	1	2
1	10	**0**	1
2	8	0	5
3	2	1	3
4	1	2	1
5	-2	3	2
6	-4	4	1
7	-1	4	3
8	1	5	4

граф G



10 — вес ребра w
№ — № ребра в списке ребер

Рассмотрим алг на примере: выполним |V| − 1 = 5 раз всех дуг из списка (каждый раз будем идти по порядку ребер в списке)

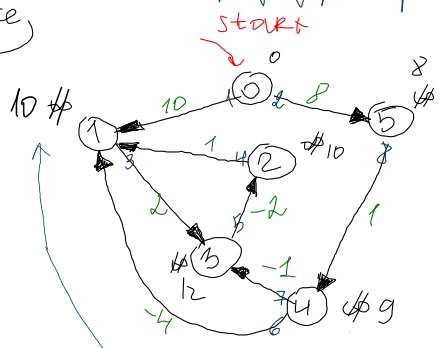
start

инициализация массива по нач. алгор.

distance

Первая итерация:

ШАГ	0	1	2	3	4	5
0	0	inf	inf	inf	inf	inf
1	0	**10**	inf	inf	inf	inf
2	0	10	inf	inf	inf	**8**
3	0	10	inf	**12**	inf	8
4	0	10	inf	12	inf	8
5	0	10	**10**	12	inf	8
6	0	10	10	12	inf	8
7	0	10	10	12	inf	8
8	0	10	10	12	**9**	8



№ у дуг на графе совпадают с номерами в списке ребер

∞ — значение в массиве distance[v_i]

10 — вес дуги из матрицы смежности W[u][v]

u v w(u,v)

шаг 1) Relax (0, 1, 10)

$d[0]=0 \quad d[1]=\infty \quad w(0,1)=10$

$d[1] > d[0] + w(0,1)$

$\infty > 0 + 10$

$d[1] = d[0] + w(0,1) = 0+10=10$

шаг 2) Relax (0, 5, 8)

$d[0]=0 \quad d[5]=\infty \quad w(0,5)=8$

$d[5] > d[0] + w(0,5)$

$\infty > 0 + 8$

$d[5] = d[0] + w(0,5) = 0+8=8$

и т.д. выполним Relax всех |E|=8 дуг

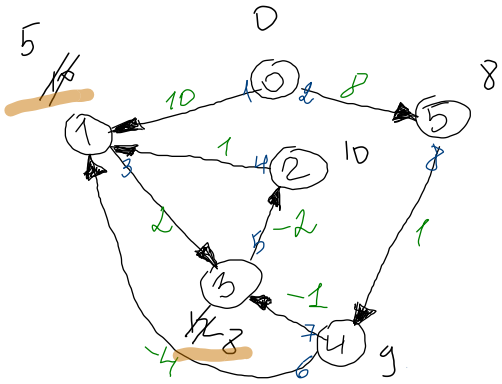
в таблице и на графе справа покажем результат ослабления все |E| ребер на первой итерации

из |V|−1 итераций

⇒ рассмотрим d^{10} из |V| − 1

Вторая итерация: distance

ШАГ	0	1	2	3	4	5
0	0	10	10	12	9	8
1	0	10	10	12	9	8
2	0	10	10	12	9	8
3	0	10	10	12	9	8
4	0	10	10	12	9	8
5	0	10	10	12	9	8
6	0	**5**	10	12	9	8
7	0	5	10	**8**	9	8
8	0	5	10	8	9	8



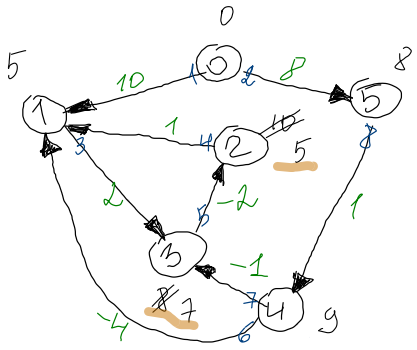
№ у дуг на графе синим показаны номера в списке ребер

∞ – значение в массиве distance[v_i]

10 – вес дуги из вершины с номером 5

Третья итерация: distance

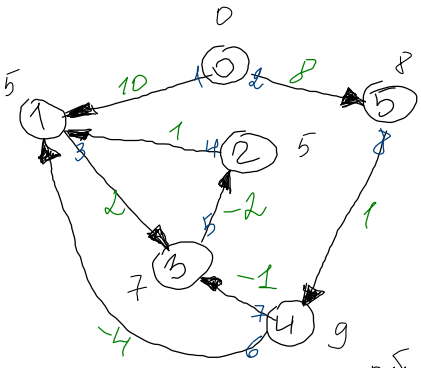
ШАГ	0	1	2	3	4	5
0	0	5	10	8	9	8
1	0	5	10	8	9	8
2	0	5	10	8	9	8
3	0	5	10	**7**	9	8
4	0	5	10	7	9	8
5	0	5	**5**	7	9	8
6	0	5	5	7	9	8
7	0	5	5	7	9	8
8	0	5	5	7	9	8



выделенные изменения

Четвертая итерация:

ШАГ	0	1	2	3	4	5
0	0	5	5	7	9	8
1	0	5	5	7	9	8
2	0	5	5	7	9	8
3	0	5	5	5	9	8
4	0	5	5	7	9	8
5	0	5	5	7	9	8
6	0	5	5	7	9	8
7	0	5	5	7	9	8
8	0	5	5	7	9	8



(!) изменения после 4-ой итерации никаких не произошло

⇓

можно остановить выполнение алгоритма т.к. на $|V|-1$ последней итерации так же не произойдет изменений, потому что будут выполнены точно такие же действия как и на 4-ой итер-и

пятая итерация

distance

↑ не изменяется

0	1	2	3	4	5
0	5	5	7	9	8

ответ

Пятая итерация:

Шестая итерация: зачем нужна |V| = 6 итерация?

Реализация

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

```
Bellman(G, s):
    InitSource(G, s)        // O(V)  и массив d - distance и массив p - predki
    for i=1 to |V|-1        // |V|
        for (u,v) ∈ E       // |E|
            Relax(u, v, w[u,v])   } ослабляем все ребра по списку
//--------для цикла отрицательного веса------
    cycle = false   // считаем, что цикла нет
    for (u,v) ∈ E   ← итерация |V|
        if (d[v] > d[u] + w[u,v])
            cycle = true   ← цикл обнаружен
```

?1 Не матрица смежности — почему?

если хранить граф в ней и использ в Беллмане-Форде, то обход всех ребер по матрице смежности займет $|V| \cdot |V| = |V|^2$, что точно больше чем ребер в практически ∀ графе кроме полного

⇓ список ребер

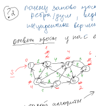
↙ храним меньше по памяти

↘ вместо $|V|^2$ в цикле получаем |E| ⇒ быстрее по такой структуре

?2 почему заново просматриваем ребра/дуги, ведь в DAG просматриваем только инцидентные вершине и шли так же по вершинам?

ответ прост: у нас есть циклы $\Rightarrow_1$ нет топсорта

$\Rightarrow_2$ мы находя для вершины наикрат для просмотренных вершин путь может быть пересчитан



если идти по вершинам с ① по ⑦ и рассмотр только 1 раз инцидентн дуги?

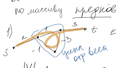
1) такой алгоритм для ⑤ вообще не нашел пути от ① до всех путей

2) когда просматривали вершину ②, то до было d[2] = ∞ ⇒ ни одна дуга из ② не ослаблена пути

?3 Как найти цикл отрицат веса и восстановить его?

Восстановить цикл отр. веса можно как и ∀ путь по массиву предков, но как понять, что такой цикл есть?

1)



← если в G есть такой цикл, то наикратч путь его содержащий → −∞ по весу

⇓

для s ⇝ t можно бесконечно улучшать вес пути p

⇓

после |V| − 1 итерации пути содержащие отр. цикл могут быть улучшены

⇓

если мы выполним ещё одну итерацию ослабления всех ребер |V|ю итерацию, то при наличии цикла отр. веса хотя бы 1 путь будет улучшен

2) Почему именно |V| − 1 итерация в алг-ме Ф-Беллмана?

ранее мы выяснили, что ∀ наикратч путь имеет не более |V| вершин и |V| − 1 ребер и путь д/б ацикличен

⇓

для поиска ∀ p необходимо найти не более |V| − 1 ребер

⇓

все ребра ослабляем |V| − 1, что нам гарантирует нахождение наикратч путей, но не обязательно понадобится все |V| − 1 итерации

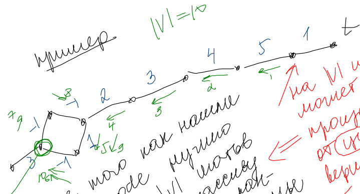
⇓

→ если цикла отр. веса нет, то за ≤ |V| − 1 итераций ослабления ребер мы гарантировано найдем наикратч. пути

→ если цикл отр. веса есть, то наикратч пути можно улучшать до −∞

, следовательно, выполнив $|V|^{ю}$ итерацию мы однозначно понимаем есть цикл отр. веса в графе или нет, а именно, если после $|V|^{ой}$ итерацией для ∀ v изменился d[v] ⇒ цикл есть

пример |V| = 10



после |V| шагов по p[ ] мы окажемся в цикле

после того как нашли save_node нужно сделать |V| шагов назад по массиву предков, что гарантирует, что после мы будем в цикле смотри

на |V| итерации может изменение произойти далеко от цикла, но на вершину путь до кт от s содержит цикл

проверка [НО]

покажем как восстановить цикл отр. веса

выполним $|V|^{ю}$ итерацию алгоритма

```
save_node = null
for (u,v) ∈ E
    if ( d[v] > d[u] + w(u,v) )
        "цикл есть"
        save_node = v
        break
```

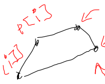
// если условие выполниться ⇒ путь улучшился ⇒ цикл есть!

← сохраняем вершину в кот улучшился путь

break ← досрочно заканчиваем for

Теперь от сохраненной save_node надо восстановить по массиву предков цикл

```
i = p[save_node]
while save_node ≠ i
{
    print(i)
    i = p[i]
}
```



// ⇒ цикл восстановлен

[НО] точно ли save_node принадлежит циклу отрицат веса?

поиск кратчайших путей

3. Алгоритм Дейкстры (жадный) (от 1 до всех)

Суть: похож на алгоритм Прима, только пройдем по всему графу от заданной 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 (P_ij) (P)

Частичные пути кратчайших путей тоже кратчайшие пути

Док-во: рассмотрим $p: v_1 \rightsquigarrow v_k = \langle v_1, v_2, \dots, v_k \rangle$ ∃ p – кратч путь $v_1 \rightsquigarrow v_k$

$1 \le i \le j \le k$ →

возьмем часть p | $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle \subseteq p$

имеем $p = v_1 \rightsquigarrow v_i \rightsquigarrow v_j \rightsquigarrow v_k$ (p_{1i}, p_{ij}, p_{jk})

↳ от противного допустим p_{ij} – не кратчайш ⇒ $\exists p^*_{ij}$: $w(p^*_{ij}) < w(p_{ij})$ ⇒

$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$ не будет min

min $w(p^*_{ij}) > w(p^*_{ij}) \Rightarrow \exists p^*$: $w(p^*) < w(p)$

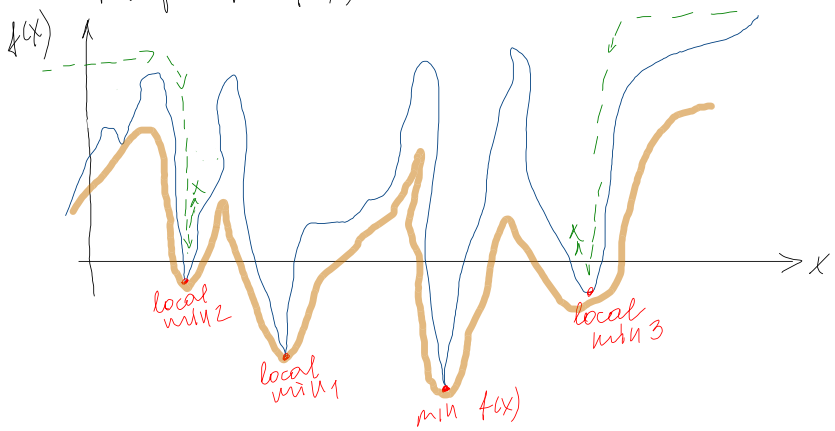
что противоречит условию недели т.к. p – путь кратчайший ⇒ (↯) p_{ij} – кратчайший

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

быстрое и оптимальное решение в целом выгоднее точного решения, но кт нужно долго искать

иллюстрирующий пример

на примере f(x) и её min и локальных min



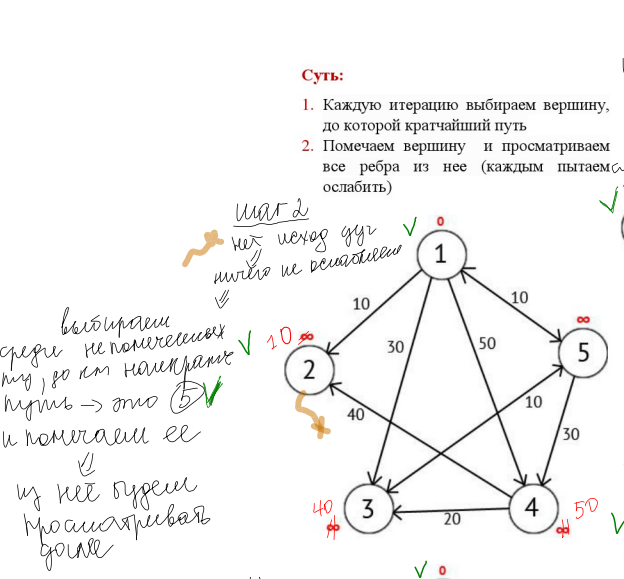
– время выполнения алг находящ min f(x) программер

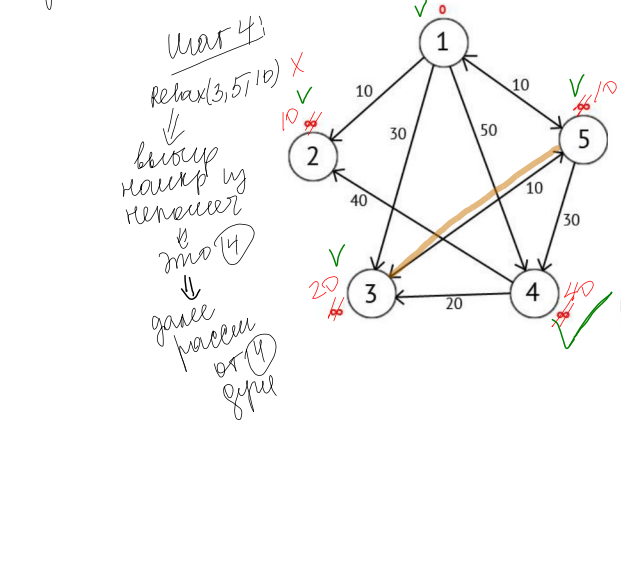
– время жадного алгоритма, кт нашел min но локальный, но кт отрабатывает перебор средними что позволяет прийти верное решение

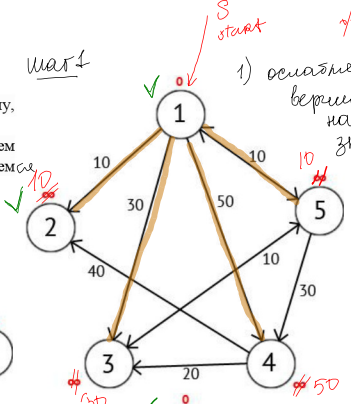
всего f(x) ⇒ долго

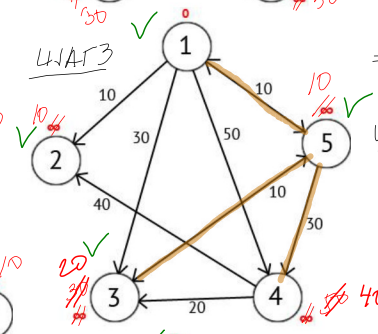
Суть:

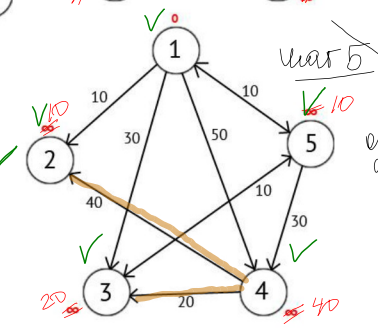
1. Каждую итерацию выбираем вершину, до которой кратчайший путь
2. Помечаем вершину и просматриваем все ребра из нее (каждым пытаемся ослабить)











Реализация

```
// G – исходный граф
// W – взвешенная матрица смежности
// Q - куча с минимумом (приоритетная очередь)
function Deikstra(G, s):
    InitSource(G, s)
    ALL V.used = 0 // vi.used = 0
    s.d = 0
    Q=creatHeap( G.V )// строим кучу с min по V.d согласно условиям
    while not Q.isEmpty()
        u = Q.extractMin() // находим кратчайшие пути для непомеченных вершин
        u.used = 1 // помечаем u
        for v ∈ W[u]
            Relax(u, v, w[u, v])
            If  (d[v] > d[u] + w(u,v)
                Q.decreaseKey(v, d[u] + w[u, v])
```

O(|V| + |E| log|V|)
O(|E| · log|V|)

|V|
|V|
|V|
|E|
logV
E logV

s.d = 0 → distance
s.p = null → predki

пока не ясно какой вершин не найден кратчайший => все не помечены

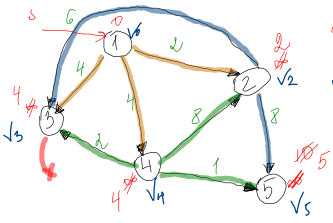
min s.d = 0

=> { s.d = 0, б. d = ∞

ослабляем все ребра от (u)

пересортировываем кучу с min

рассмотрим на примере



k - все ребра
d - значение в массиве distance
√n шаг пометки вершин как тех что нашли кратчайшие пути

шаг	1	2	3	4	5
0	**0**	∞	∞	∞	∞
1	0	**2**	4	4	∞
2	0	2	**4**	4	10
3	0	2	4	**4**	10
4	0	2	4	4	**5**

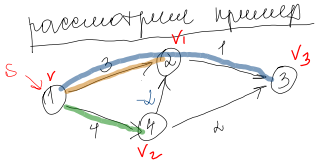
distance

инициализируем массивы (1) – стартовая вершина до всех вершин

1) Как работает куча с min мы рассмотрели ранее, но никто не запрещает использ массив и искать min в нем за O(|V|)

2) При реализации с очередью (куча с min) нужны ли метки у вершин u.used = 1 ?) хотя мы удалили из кучи, что равнозначно u.used = 1
? нужно подумать?
(для реализации на массиве точно понадобятся)

(1) (3) Алгоритм не работает с ребрами / дугами с отриц весом!
Почему?



на графе покажем как найдем кратч путь и не найдем алг Дейкстры

(НО) когда (4) будет помечена как вершина с кратч путем от нее будут ослабляться дуги, но только до непомеченных вершин

до ранее помеченной (2) путь по дуге (4,2) с весом -2 не улучшит d[2] т.к. (2) помечена

Дейкстра даст не верный ответ до вершины (2) и всех остальных, что содержат (2) в своем пути

=> и до (3) будет ответ верный

тут подсказка нужны ли метки u.used = 1

алгоритм не эфф-ен при наличии ребер с отриц весом

с циклами отриц веса не работает тем более!

? ответ на вопрос какое представление графа использовать весьма прост => ответ найдете сами или изучаете на практике

4. Алгоритм Флойда - Воршалла

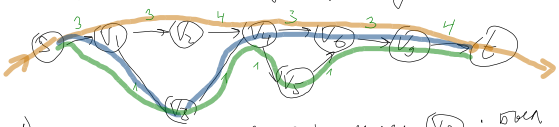
Суть: ищем кратчайшие пути от всех до всех, выполняя релаксацию через каждую вершину

от $\forall v_i$ до $\forall v_j$, где $i \neq j$

Нужны будут дополнительные матрицы:

- d[][] — матрица длин кратчайших путей (копируем матрицу расстояний)
- Pred[][] — матрица предков для восстановления кратчайших путей (на старте все ячейки = null)

→ ранее мы релаксировали ребро, теперь будем улучшать путь через вершину



1) улучшаем путь $s \rightsquigarrow t$ через v_8; d[s][t] = 20 ⇓ d[s][t] = 15 — было $s\ v_1 v_2 v_4 v_8 v_9\ t$, стало $s\ v_1 v_6 v_4 v_8 v_9\ t$

2) теперь релаксируем $s \rightsquigarrow t$ через v_5: d[s][t] = 14 — стало $s\ v_1 v_8 v_4 v_5 v_6 v_9\ t$

теперь мы видим как через 5 вершину можно улучшить путь

было d[s][t], но через v_k d[s][k] + d[k][t] намного короче

⇓ нужно сравнивать

d[s][t] > d[s][k] + d[k][t] и если это так ⇒ d[s][t] = d[s][k] + d[k][t] — по сути это поиск min из двух чисел

(!) d[s][t] = min(d[s][t], d[s][k] + d[k][t])

короткая схема $s \xrightarrow{p} t$ (p — путь)

если $v \in p$ ⇒ если вершина 5, через кт мы релаксируем ($\notin p$) не принадлежит пути м/у вершинами, то улучшить и не получается

Алгоритм

```
for v_k ∈ G.V                    // идем по всем вершинам и пытаемся через них релаксировать
    for s ∈ G.V                  // перебираем все возможные пути для релаксации через v_k
        for t ∈ G.V
            d[s][t] = min(d[s,t], d[s][v_k] + d[v_k][t])
```

s — вершина начала пути

t — вершина конца пути

ищем min ⇓ улучшаем

расмотрим циклы
(1) – берем вершины по порядку, через них будем релаксировать
(2) – u-вершина начала пути, что пытаемся улучшить
(3) – v-вершина конца этого пути
→ проходим всю матрицу графа

Реализация

d – матрица кратч путей

```
Floyd(W):
    d = W
    for i ∈ V
        for u ∈ V
            for v ∈ V
                d[u][v] = min ( d[u][v],  d[u][i] + d[i][v] )
```

⇒ V×V×V ⇒ ⇒ O(V³)

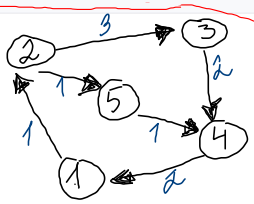
нарисовали все V итерации

матр смежн

W(G)

веса

	1	2	3	4	5
1	0	1	inf	inf	inf
2	inf	0	3	inf	1
3	inf	inf	0	2	inf
4	2	inf	inf	0	inf
5	inf	inf	inf	1	0



i=0

inf – нет дуги м/д вер

если завести матрицу предков, то можно сохранить пути → после завери по матрице предков восст напр. пути

В указанном месте необходимо записывать предков после успешной релаксации

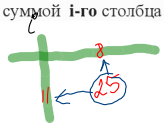
if (d[u][v] < d[u][i] + d[i][v])

PREDKI[][] =

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

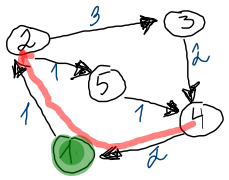
Начинаем выполнение внутр циклов согласно принципу



25 > 8+11 ⇒ ~~25~~ ← 17

i=1

	1	2	3	4	5
1	0	1	inf	inf	inf
2	inf	0	3	inf	1
3	inf	inf	0	2	inf
4	2	3	inf	0	inf
5	inf	inf	inf	1	0

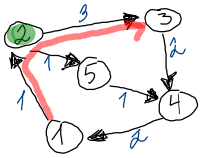


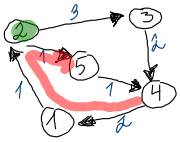
путь из 4 → 2:

прорелаксир-ли через вершину 1

i=2

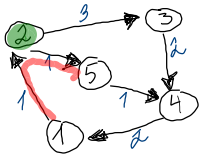
	1	2	3	4	5
1	0	1	4	inf	2
2	inf	0	3	inf	1
3	inf	inf	0	2	inf
4	2	3	6	0	4
5	inf	inf	inf	1	0

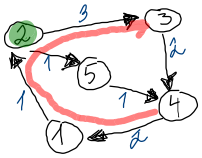




пути: 1 → 5
1 → 3
4 → 5
4 → 3
прорелаксированы через 2

6) примечание:
Inf + ∀ число = Inf



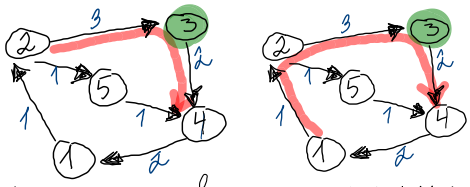


при выполн. итерации в i-ом столбце или i-ой строке Inf, то соответств ячейки можно не просматривать т/к через (i) релаксацию не пройдет

пути: 2 ⇝ 4 проходит через (3)
1 ⇝ 4

L = 3

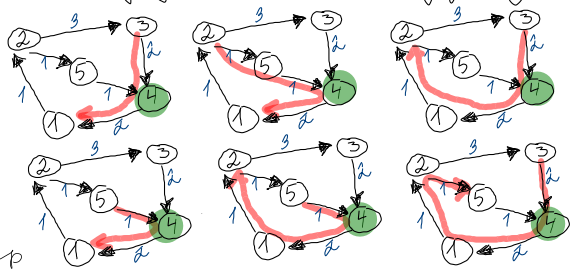
	1	2	3	4	5
1	0	1	4	6	2
2	inf	0	3	5	1
3	inf	inf	0	2	inf
4	2	3	6	0	4
5	inf	inf	inf	1	0



(!) как мы видим улучшение на предыдущей итерации, вместо улучшения в следующей

L = 4

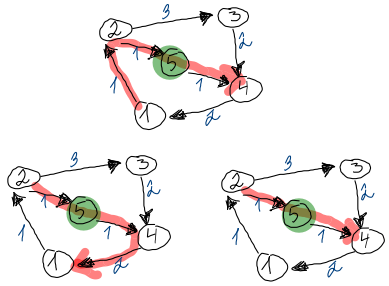
	1	2	3	4	5
1	0	1	4	6	2
2	7	0	3	5	1
3	4	5	0	2	6
4	2	3	6	0	4
5	3	4	7	1	0



пути: 3 ⇝ 1, 3 ⇝ 2, 5 ⇝ 1, 5 ⇝ 2 проходит через (4)

L = 5 последняя

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



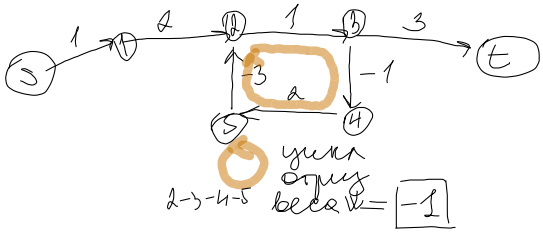
пути: 1 ⇝ 4, 2 ⇝ 1, 2 ⇝ 4 проходят через (5)

ИТОГ:
Мы прошли по всем вершинам три цикла
1) шли по промежуточным (релаксируемым) 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.

(!) Циклы отрицательного веса!

Именно для поиска циклов отрицательного веса мы в матрице расстоянии на диагонали записали нули:



1) в W[2,2] = 0 на старте
2) выполнение |V| итераций по всей матрице мы улучшим путь 2⇝2 через (3),(4),(5) тем самым получим в W[2,2] = δ < 0 (отрицательное)
3) если в результирующей матрице на диагонали есть числа δ < 0, меньше нуля (отрицат.) ⇒ есть цикл отриц веса
4)* восстановить цикл можно по аналогии с алгоритмом форда – беллмана*

(+) алгоритм эфф-но работает с отрицат ребрами и циклами