



Кратчайший маршрут

<https://github.com/larandaA/alg-ds-snippets>

Определение 1

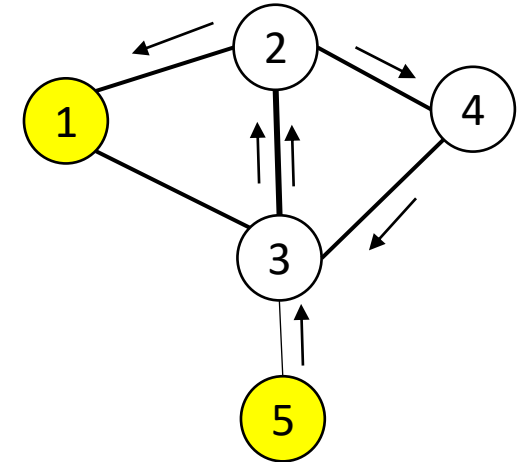
Маршрутом в графе между заданной парой вершин v_1 и v_{k+1} называется чередующаяся последовательность вершин и рёбер вида:

$$v_1 \ e_1 \ v_2 \ e_2 \ \dots \ v_k \ e_k \ v_{k+1}$$

где $e_i = \{v_i \ v_{i+1}\}$, для всех i от 1 до k .

Если нет кратных рёбер, то маршрут часто задают простым перечислением его вершин:

$$v_1 \ v_2 \ \dots \ v_k \ v_{k+1}$$

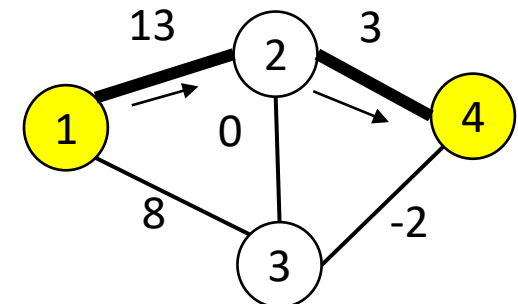


маршрут: 5, 3, 2, 4, 3, 2, 1
длина: 6

Маршрут может проходить по некоторым рёбрам несколько раз. Длина маршрута – число рёбер в нём.

Определение 2

Для взвешенного графа **вес маршрута** между заданной парой вершин v_1 и v_{k+1} определяется как сумма весов рёбер, входящих в этот маршрут.



маршрут: 1, 2, 4
вес: $13+3=16$

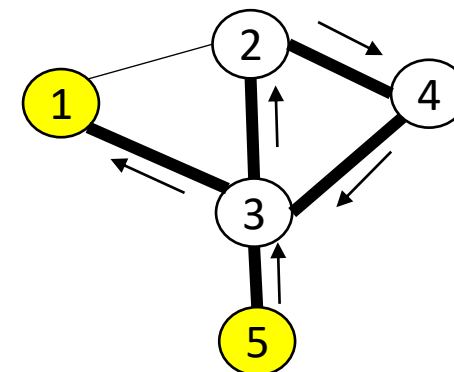
Определение 3

Цепь -

маршрут, в котором каждое ребро встречается не более одного раза (цепь может проходить через некоторые вершины несколько раз).

Замкнутая цепь называется **циклом**.

цепь: 5, 3, 2, 4, 3, 1



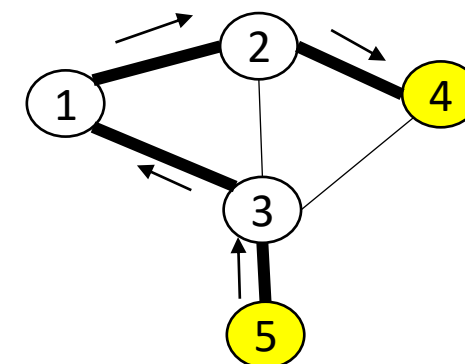
Определение 4

Простая цепь –

цепь, в которой каждая вершина встречается не более одного раза.

Замкнутая простая цепь называется **простым циклом**.

простая цепь: 5, 3, 1, 2, 4



Для ориентированного графа

- ✓ **ориентированный маршрут**
- ✓ **ориентированная цепь**
- ✓ **ориентированный цикл**

вводятся идентично тому, как это было сделано для графа.

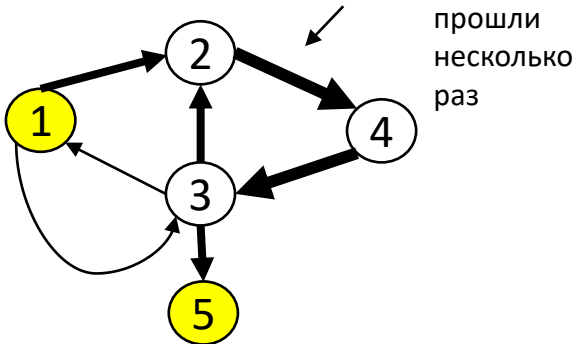
Определение 5

Путь в орграфе - ориентированный маршрут, в котором каждая вершина встречается не более одного раза.

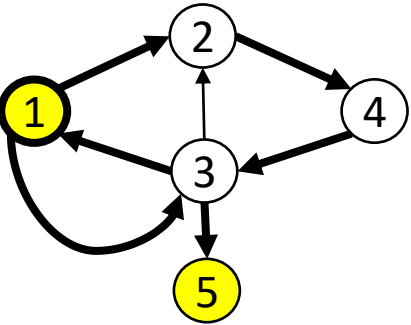
Замкнутый путь называется **контуром**.

граф	орграф
маршрут	ориентированный маршрут
цепь	ориентированная цепь
цикл	ориентированный цикл
простая цепь	путь
простой цикл	контур

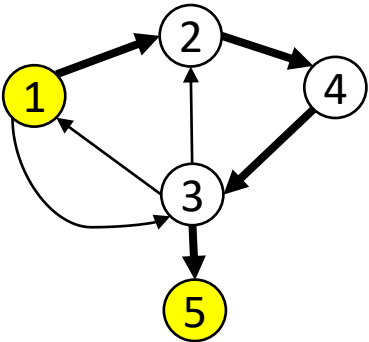
ориентированный маршрут
1, 2, 4, 3, 2, 4, 3, 5



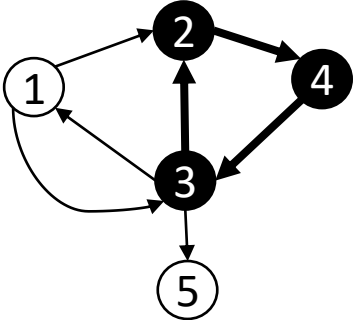
ориентированная цепь
1, 2, 4, 3, 1, 3, 5



путь
1, 2, 4, 3, 5

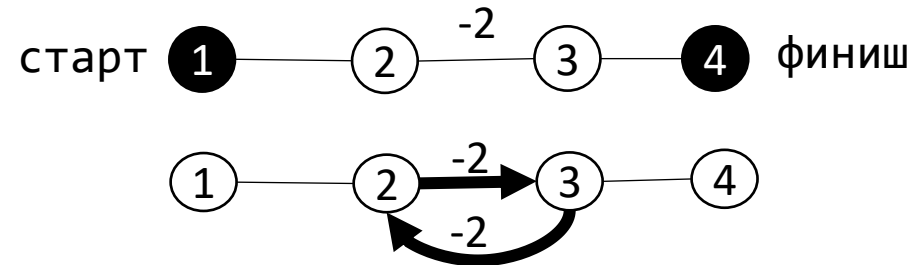


контур
2, 4, 3, 2

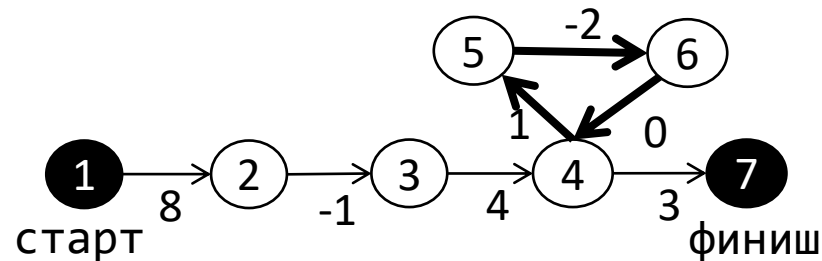


Кратчайший маршрут. Отрицательные веса

Если в графе есть **рёбра отрицательного веса**, то, заменяя ребро на две противоположно направленные дуги такого же веса, получаем контур отрицательно веса, задача построения кратчайшего маршрута не имеет решения.



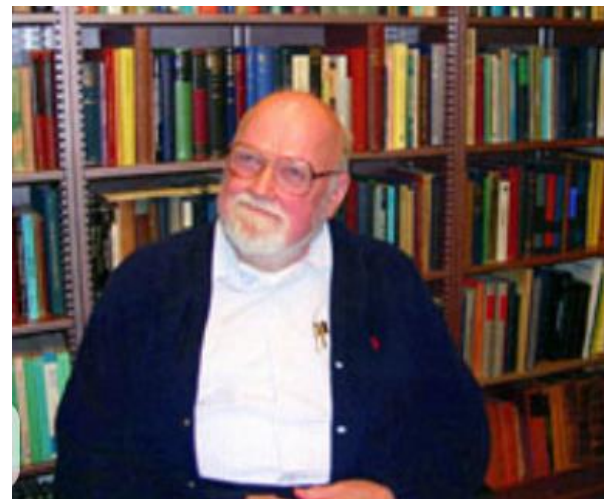
Если в орграфе есть **контур отрицательного веса, достижимый из стартовой вершины (и из которого достижима точка финиша)**, то задача построения кратчайшего маршрута также не имеет решения.



В общем случае, когда допускаются циклы (контуры) отрицательного веса, задача нахождения кратчайшей простой цепи (кратчайшего пути) между заданной парой вершин остаётся корректной, но становится **NP-трудной** (она не менее трудна, чем NP-полная задача о гамильтоновой цепи).

Кратчайший маршрут

1956 год (сделал набросок алгоритма, решая другую задачу,
а эта задача возникла, как подзадача)



**Лестер Рэндольф Форд
младший**

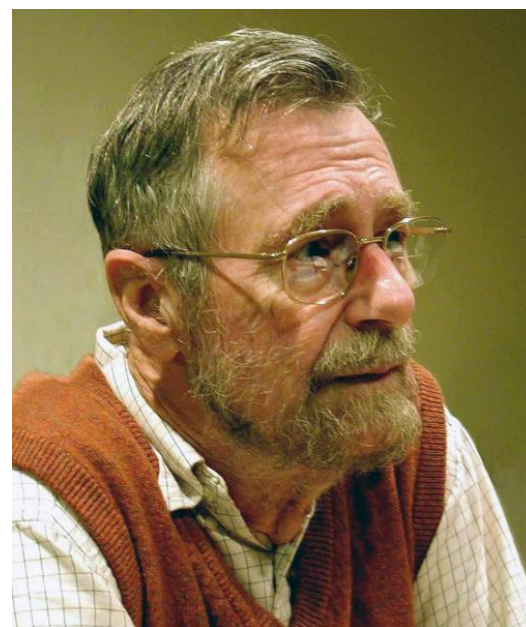
англ. *Lester Randolph Ford, Jr.*

1927 – 2017

США

Научная сфера - математик

1959 год



**Эдсгер Вйбе
Дейкстра**

Edsger Wybe Dijkstra

1930 – 2002

Нидерланды

Научная сфера -
информатик

1958 год

(опубликовал алгоритм решения данной задачи)

Ричард Эрнест Бэллман

(англ. *Richard Ernest*

Bellman

1920 – 1984

США

Научная сфера –
математика, теория
управления



Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда

- ✓ Находит кратчайшие маршруты между заданной вершиной и всеми вершинами, достижимыми из неё.
- ✓ Алгоритм допускает наличие в орграфе дуг отрицательного веса.

Время работы - $O(n \cdot m)$.

Если в орграфе были контуры отрицательного веса, достижимые из стартовой вершины, то после выполнения n ($n = |V|$) итераций алгоритм Беллмана-Форда найдёт один из таких контуров отрицательного веса, а задача построения кратчайшего маршрута не имеет решения.

Если в орграфе нет контуров отрицательного веса, достижимых из точки старта, то, если вершина v достижима из вершины u , то в качестве кратчайшего (u, v) -маршрута алгоритм **найдёт кратчайший путь**.

Алгоритм Беллмана – Форда

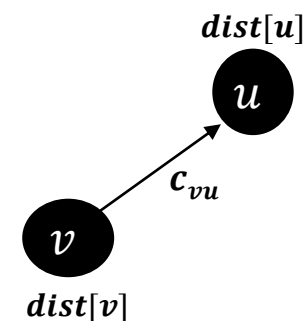
1. Присваиваем стартовой вершине метку $dist[start] = 0$, а для остальных вершин $dist[v] = INF$.
Величина $dist[v]$ – оценка сверху на длину кратчайшего пути от стартовой вершины до вершины v .
2. Выполняем $(n - 1)$ итерацию алгоритма.

На каждой итерации просматриваем все дуги орграфа в произвольном порядке.

Пусть (v, u) текущая дуга, пробуем уменьшить метку вершины u по следующей формуле:

релаксация по дуге (v, u)
от лат. *relaxatio* «ослабление»

$$dist[u] = \min\{ dist[u], dist[v] + c_{vu} \}$$



Если на некоторой итерации ни одна вершина не изменила свою метку, то алгоритм можно досрочно завершить, задача нахождения кратчайших маршрутов от стартовой вершины во все достижимые вершины – **решена**.

Предположим, что на $(n - 1)$ -й итерации алгоритма были релаксации:

- ✓ необходимо выполнить ещё одну n -ю итерацию алгоритма и, если не было релаксаций, то задача решена;
- ✓ если на n -ой итерации были релаксации, то в орграфе существует контур отрицательного веса, который достижим из точки старта и задача нахождения кратчайших маршрутов из стартовой вершины во все достижимые из неё вершины не имеет решения (алгоритм Беллмана-Форда позволяет восстановить один из контуров отрицательного веса, который достижим из точки старта).

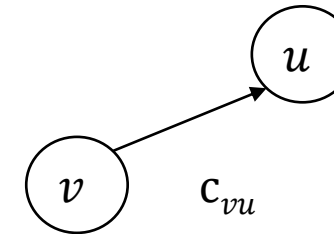
Алгоритм Беллмана – Форда. Псевдокод

```
def distances(start):  
    dist[start] = 0  
  
    for i in range(n - 1):  
        for v, c, u in edges:  
            dist[u] = min(dist[u], dist[v] + c)
```

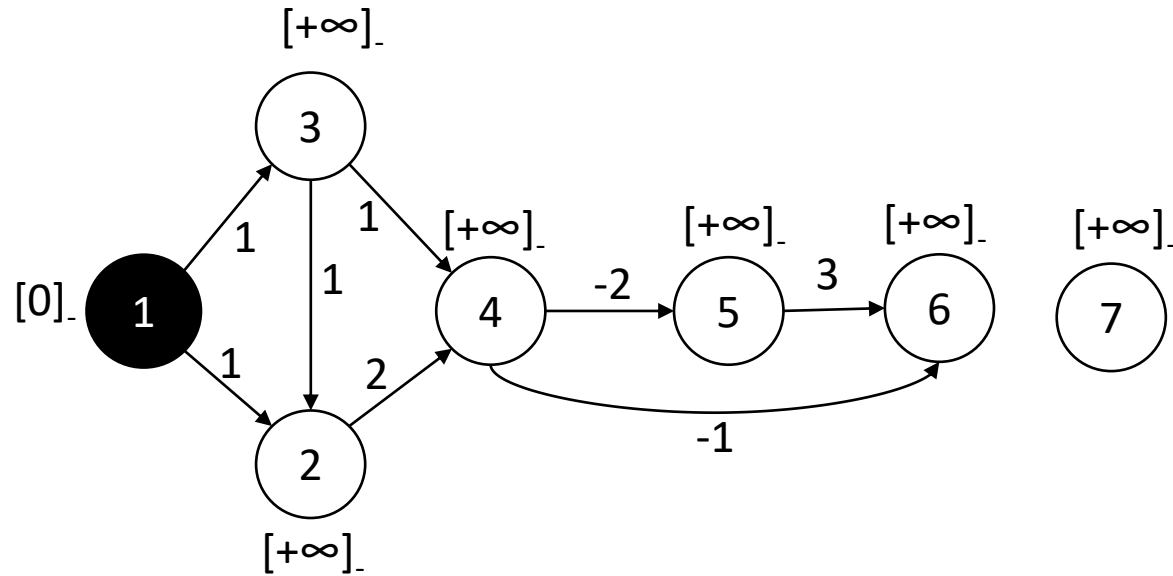
```
def distances(start):  
    dist[start] = 0  
  
    for i in range(n - 1):  
        for v, c, u in edges:  
            if dist[u] > dist[v] + c: # <-- раскрыли  
                dist[u] = dist[v] + c # <-- код в  
                pred[u] = v           # <-- блок `if`
```

Орграф задан **списками дуг**:

для каждой дуги (v, u) задаётся начальная вершина дуги v , конечная вершина дуги u и стоимость c_{vu} .



Алгоритм Беллмана – Форда.

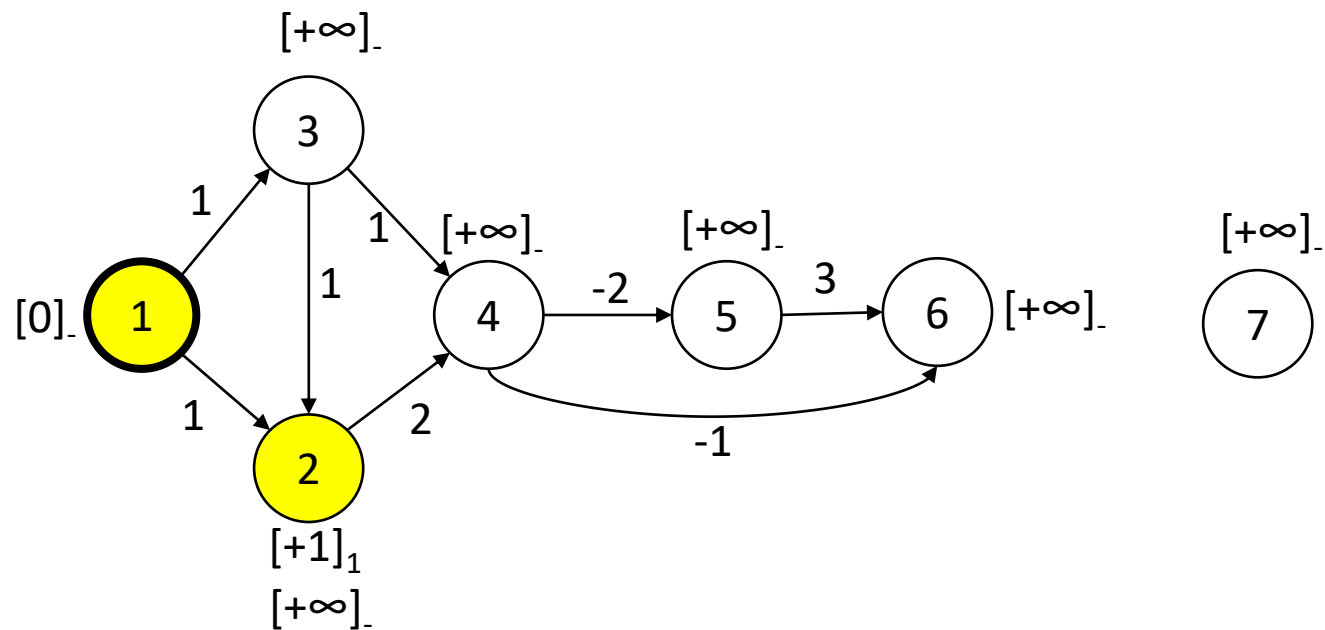


dist	1	2	3	4	5	6	7
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

порядок, в котором будем
просматривать дуги:

(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

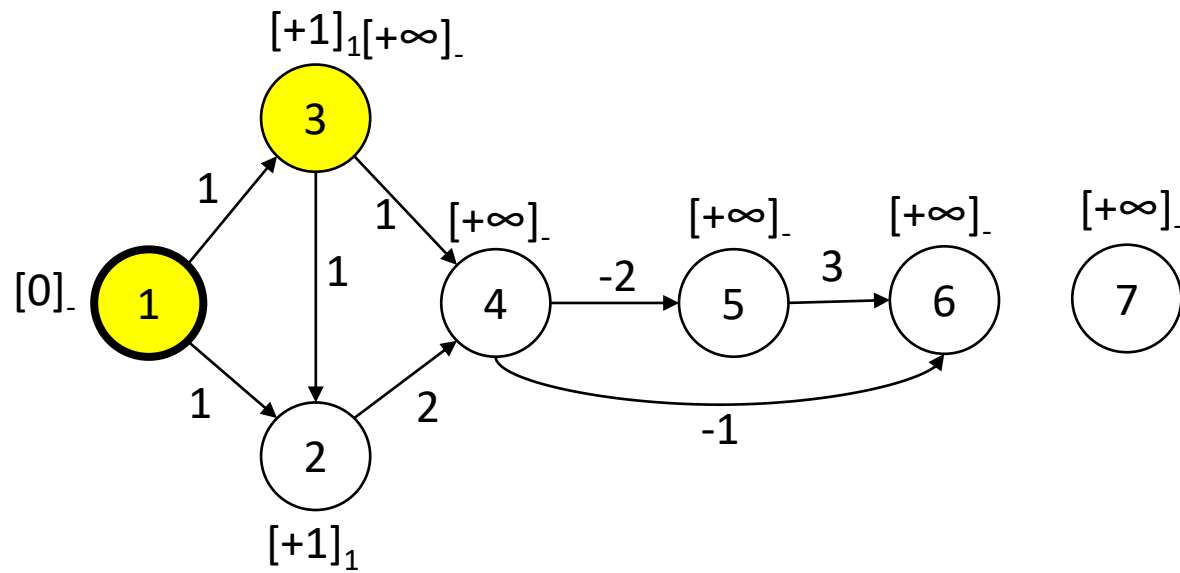
```
def distances(start):  
    dist[start] = 0  
  
    for i in range(n - 1):  
        for v, c, u in edges:  
            dist[u] = min(dist[u], dist[v] + c)
```



(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

```
def distances(start):
    dist[start] = 0

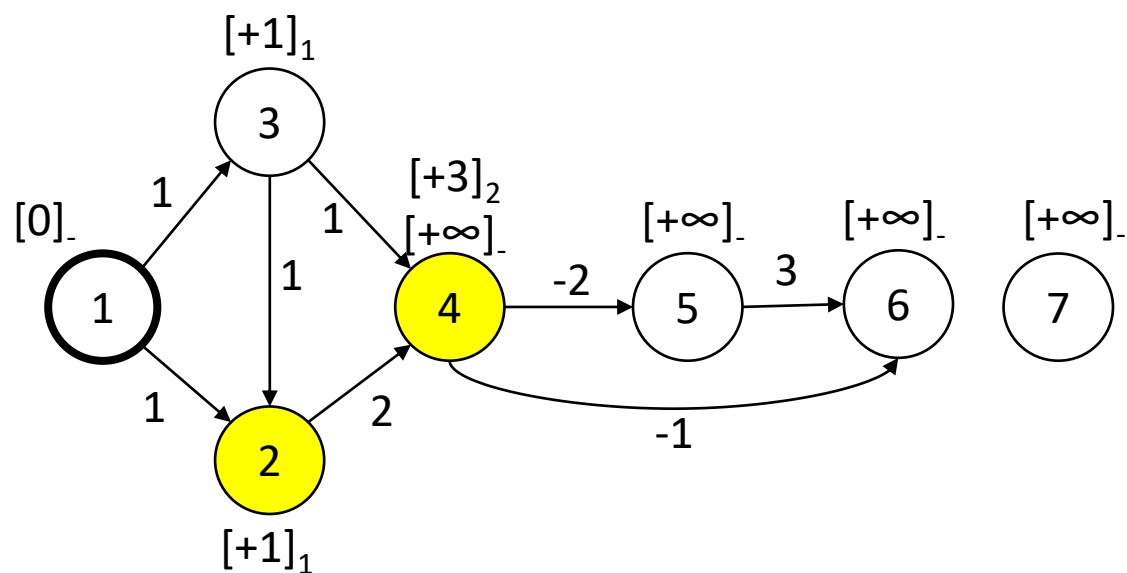
    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```



(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```



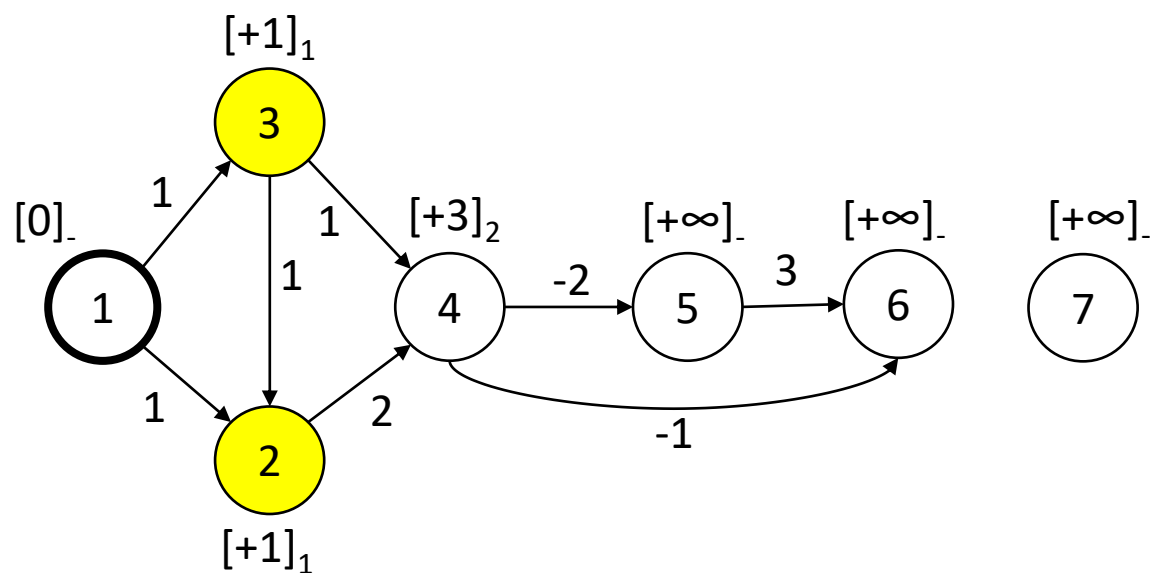
(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

```

def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)

```



(1,2)

(1,3)

(2,4)

(3,2)

(3,4)

(4,5)

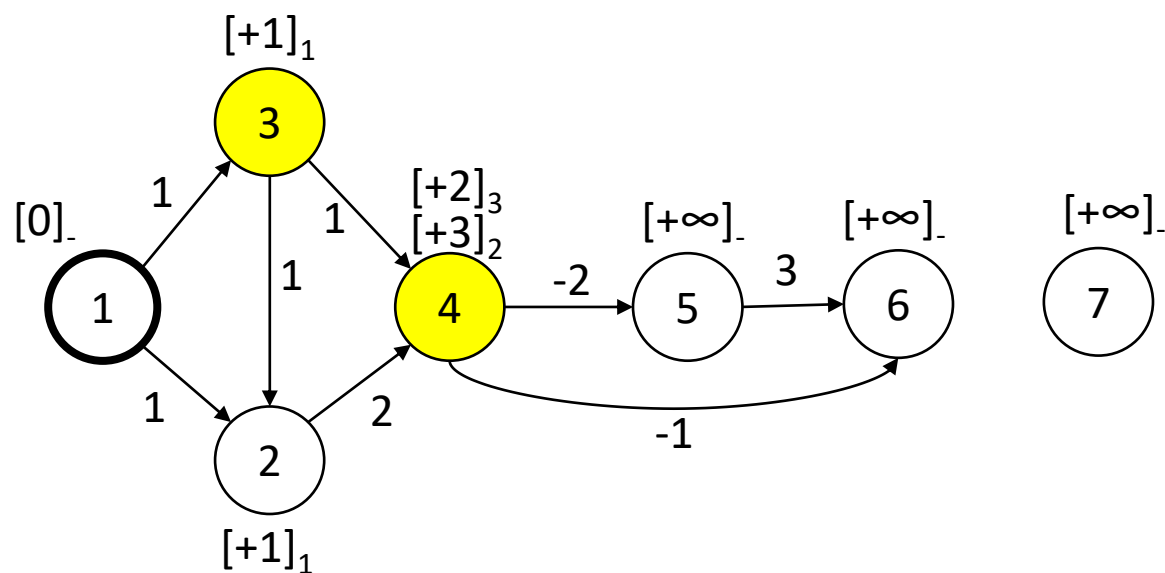
(5,6)

(4,6)

```

def distances(start):
    dist[start] = 0

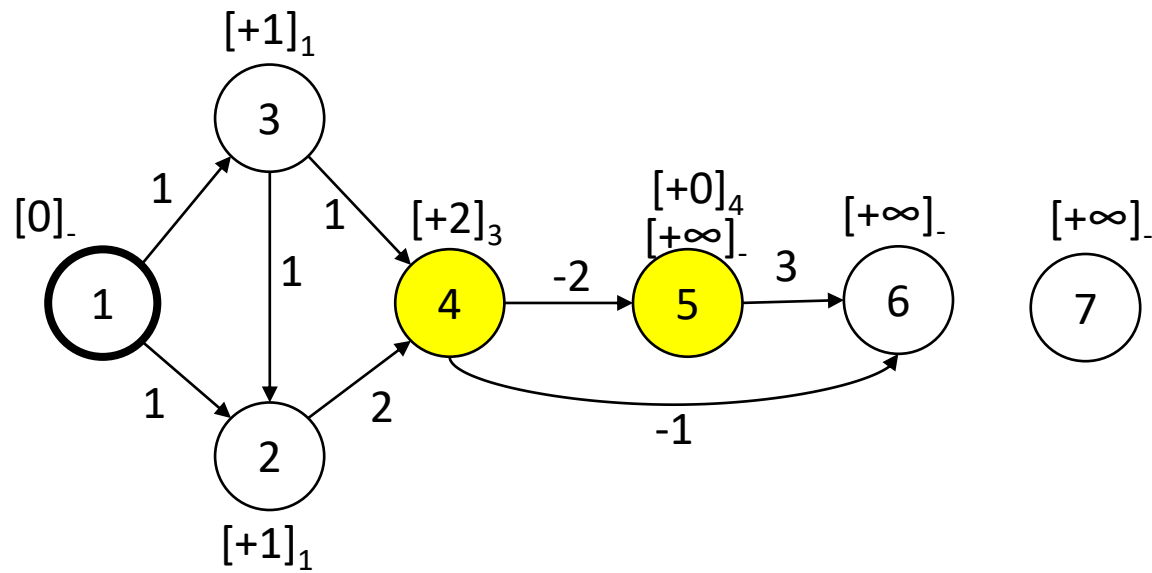
    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
  
```



(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```

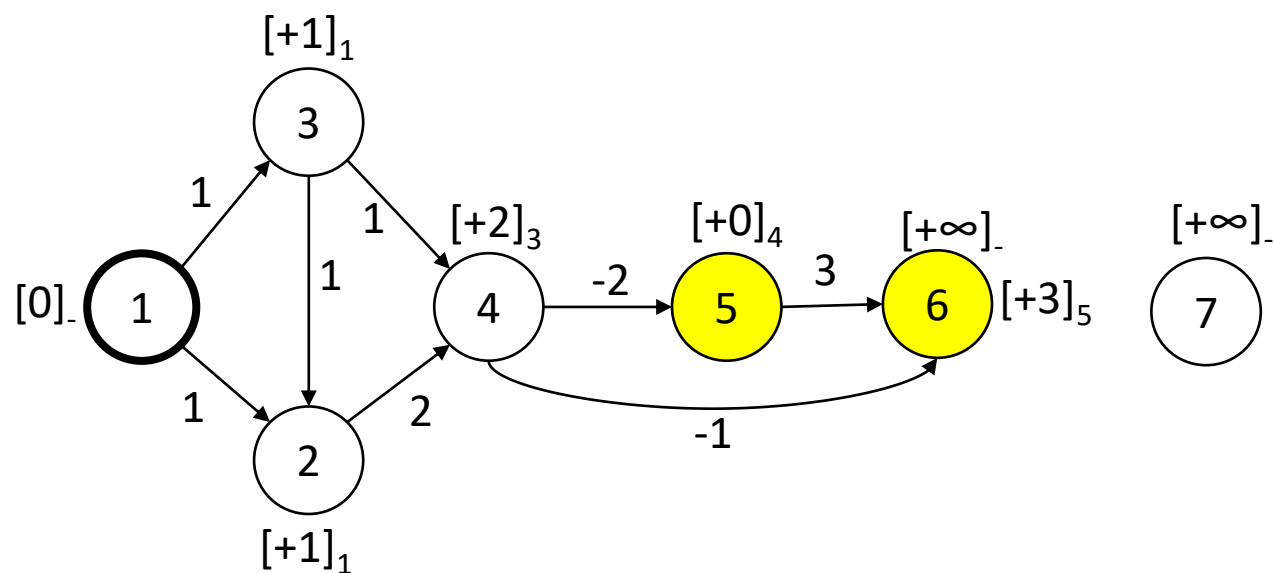



(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

```

def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
  
```



(1,2)

(1,3)

(2,4)

(3,2)

(3,4)

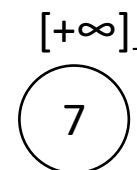
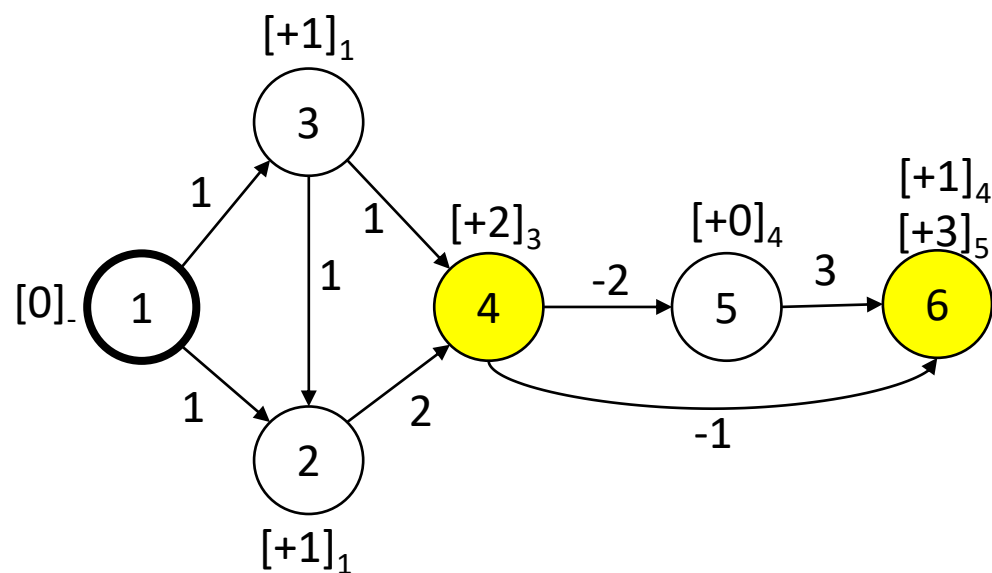
(4,5)

(5,6)

(4,6)

```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```



(1,2)

(1,3)

(2,4)

(3,2)

(3,4)

(4,5)

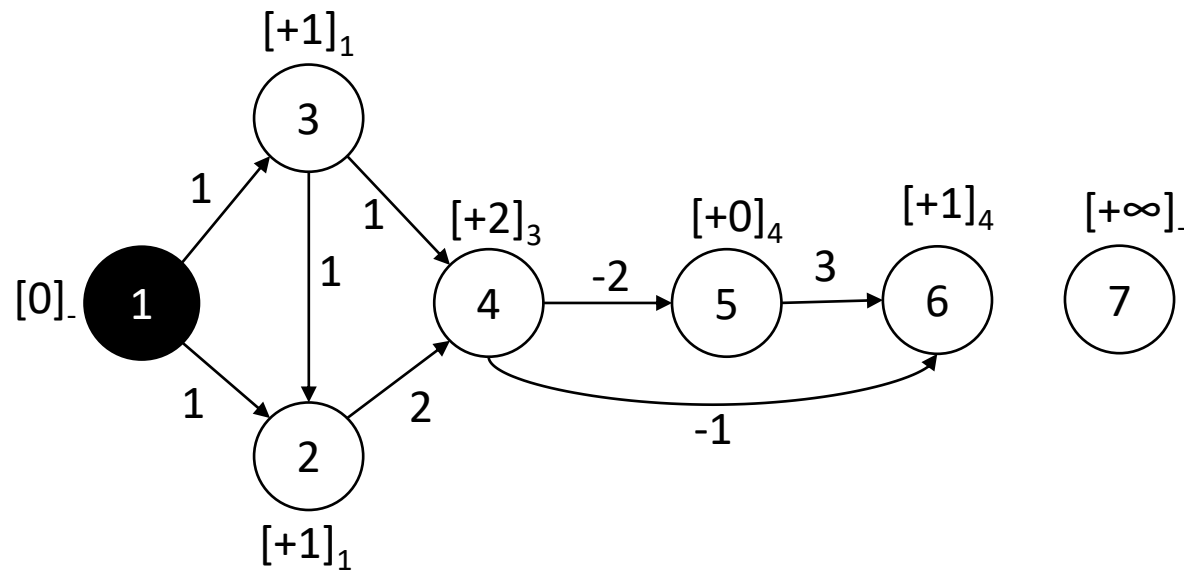
(5,6)

(4,6)

```

def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
  
```

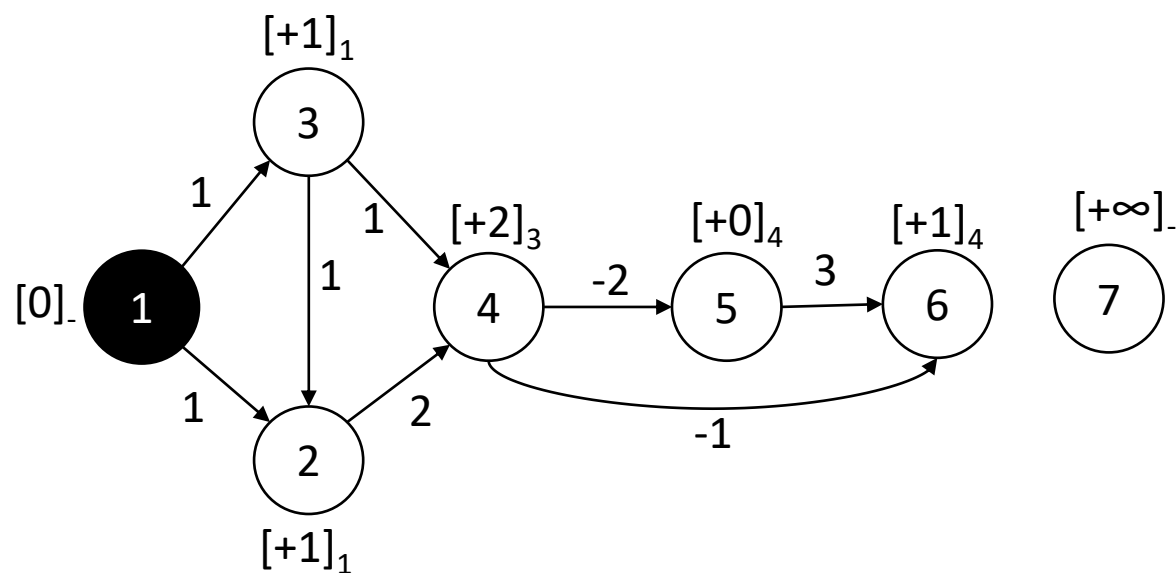


(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

dist	1	2	3	4	5	6	7
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1:	0	1	1	2	0	1	$+\infty$

```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```



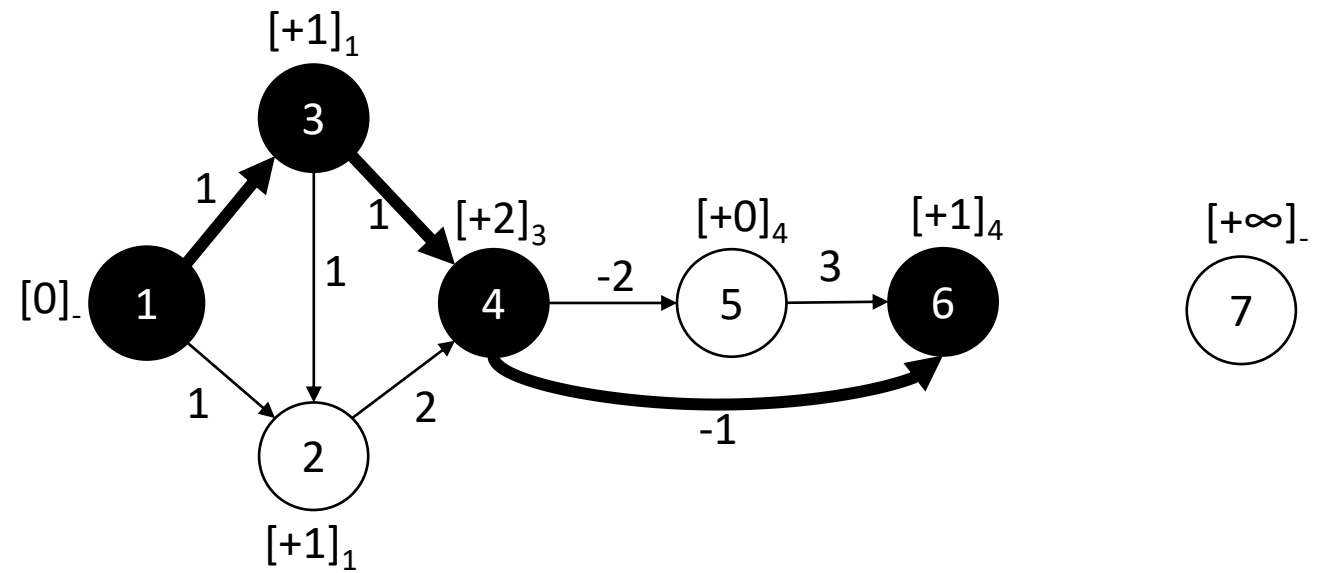
(1,2)
(1,3)
(2,4)
(3,2)
(3,4)
(4,5)
(5,6)
(4,6)

dist	1	2	3	4	5	6	7
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1:	0	1	1	2	0	1	$+\infty$
2:	0	1	1	2	0	1	$+\infty$

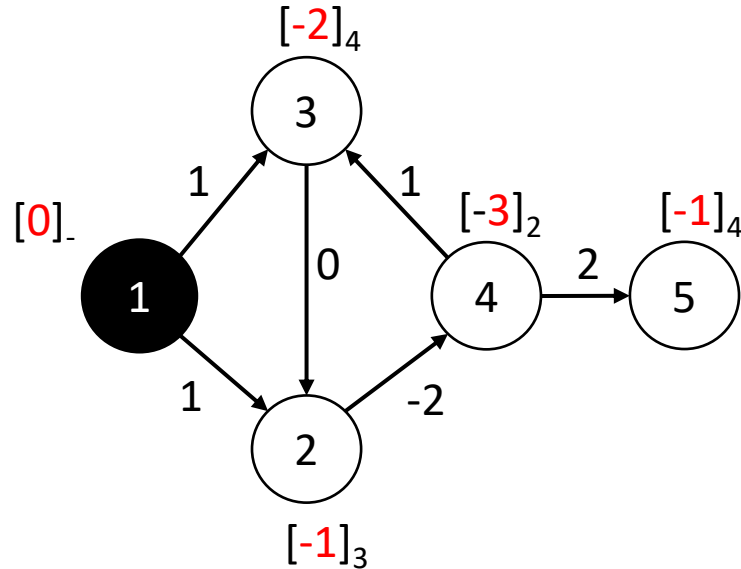
```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            dist[u] = min(dist[u], dist[v] + c)
```

Восстановление кратчайшего пути из 1 в 6 (используем сформированный массив предшественников):



Алгоритм Беллмана – Форда. Контур отрицательного веса



(1,2)
(1,3)
(2,4)
(3,2)
(4,3)
(4,5)

	1	2	3	4	5
Dist	0	Inf	Inf	Inf	Inf
1:	0	1_1	$1_1, 0_4$	-1_4	1_4
2:	0	0_3	0_3	-1_2	1_4
3:	0	0_3	-1_4	-2_2	0_4
4:	0	-1_3	-1_4	-2_2	0_4
5:	0	-1_3	-2_4	-3_2	-1_4

Для восстановления контура отрицательного веса:

берём любую вершину, которая изменила свою метку (эта вершина либо лежит на контуре, либо достижима из него, так как путь в неё содержит как минимум n дуг, что говорит о наличии в нём контура) и двигаемся по массиву предшественников **pred**, пока некоторая вершина не встретится при движении дважды.

Например: $5 \leftarrow 4 \leftarrow 2 \leftarrow 3 \leftarrow 4$

```
def distances(start):
    dist[start] = 0

    for i in range(n - 1):
        for v, c, u in edges:
            if dist[u] > dist[v] + c: # <-- раскрыли
                dist[u] = dist[v] + c # <-- код в
                pred[u] = v           # <-- блок `if`
```

Алгоритм Беллмана – Форда. Псевдокод
(граф задан списками смежности, в графе могут быть контуры отрицательного веса).

```
def distances(start):  
    dist[start] = 0  
  
    for i in range(n - 1):  
  
        for v, gv in enumerate(g): # <-- вот  
            for u, c in gv:         # <-- здесь!  
  
                if dist[u] > dist[v] + c:  
                    dist[u] = dist[v] + c  
                    pred[u] = v  
  
    for v, gv in enumerate(g): # <-- и вот  
        for u, c in gv:         # <-- здесь!  
  
            if dist[u] > dist[v] + c:  
                panic!
```


Обоснование корректности алгоритм Беллмана – Форда

Предположим, что в орграфе отсутствуют контуры отрицательного веса, достижимые из точки старта.

Обоснование корректности алгоритма следует непосредственно следующих двух утверждений.

Утверждение 1

Так как кратчайший маршрут не содержит повторяющихся вершин, то длина любого кратчайшего маршрута не превосходит $(n - 1)$.

Утверждение 2

После выполнения *i*-ой итерации для всех вершин v , для которых кратчайшие $(start, v)$ -маршруты содержат не более *i* дуг, текущая верхняя оценка $dist[v]$ равна длине кратчайшего $(start, v)$ -пути.

Утверждение доказывается методом математической индукции.

Поэтому после выполнения $n - 1$ итерации алгоритма кратчайшие маршруты для всех вершин, достижимых из точки старта, будут построены.

Время работы алгоритма Беллмана – Форда: $O(n \cdot m)$.

Алгоритм Э. Дейкстры (1959)

Находит кратчайшие маршруты между вершиной ***u*** и всеми вершинами, достижимыми из неё.

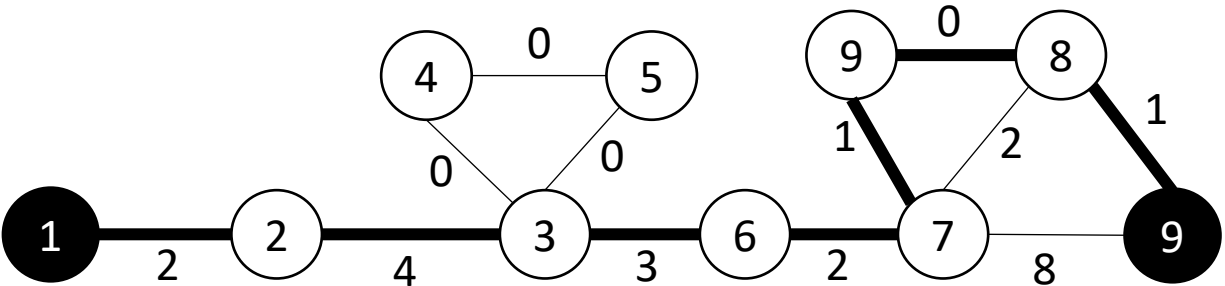
Алгоритм корректно работает только для графов (орграфов) **без рёбер (дуг) отрицательного веса.**

Время работы алгоритма Дейкстры зависит от выбранной структуры данных:

- $O(n^2 + m)$
- массив;
- $O(m \cdot \log m)$
- бинарная куча (в кучу добавляются рёба/дуги);
- $O(n \cdot \log n) + O(m \cdot \log n)$
- бинарная куча (в кучу добавляются вершины + операция модиф. ключа);
- $O(m) + O(n \cdot \log n)$
- куча Фибоначчи (в кучу добавляются вершины + операция модиф. ключа);

↖
оценка усреднённая;

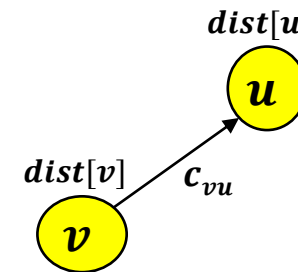
Если вершина v достижима из вершины u , то в качестве кратчайшего (u, v) -маршрута алгоритм найдёт кратчайшую простую цепь (для графа)/ кратчайший путь (для орграфа).



Алгоритм Дейкстры

1. Присваиваем стартовой вершине *start* метку $dist[start] = 0$, а для остальных вершин $dist[v] = INF$. Для произвольной вершины *v* значение $dist[v]$ – оценка сверху на длину кратчайшего маршрута от стартовой вершины до неё.
2. Считаем все вершины не обработанными $processed[v] = FALSE$
3. Пока все вершины, достижимые из точки старта не будут обработаны, выполняем следующие действия.
 - 3.1 Находим среди необработанных вершин вершину с самой маленькой меткой *dist*.
Предположим, что это вершина *v*.
 - 3.2 Полагаем для вершины *v* значение $dist[v]$ – длиной кратчайшего (*start*, *v*)-маршрута.
 - 3.3 Полагаем вершину *v* обработанной: $processed[v] = TRUE$.
 - 3.4 Просматриваем все смежные с *v* необработанные вершины *u* и выполняем релаксацию по дуге (*v*, *u*).

$$dist[u] = \min\{ dist[u], dist[v] + c_{vu} \}$$



Подробное доказательство корректности алгоритма Дейкстры:

[Теория алгоритмов : учеб. пособие / П.А. Иржавский \[и др.\].- Минск : БГУ, 2013.-С.108-113.](#)

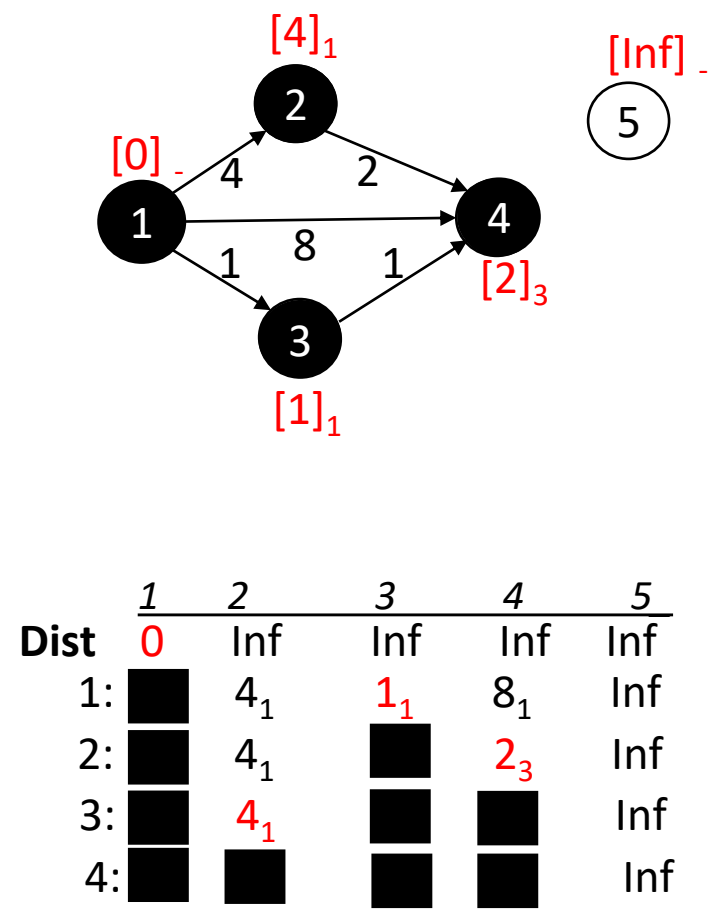
Пример.

Алгоритм Дейкстры

1. Присваиваем стартовой вершине *start* метку *dist[start] = 0*, а для остальных вершин *dist[v] = INF*. Для произвольной вершины *v* значение *dist[v]* – оценка сверху на длину кратчайшего маршрута от стартовой вершины до неё.
2. Считаем все вершины не обработанными
processed[v] = FALSE
- 3 . Пока все вершины, достижимые из точки старта не будут обработаны, выполняем следующие действия.

- 3.1 Находим среди необработанных вершин вершину с самой маленькой меткой *dist*.
Предположим, что это вершина *v*.
- 3.2 Полагаем для вершины *v* значение *dist[v]* – длиной кратчайшего (start,v)-маршрута.
- 3.3 Полагаем вершину *v* обработанной:*processed[v] = TRUE*.
- 3.4 Просматриваем все смежные с *v* необработанные вершины *u* и выполняем релаксацию по дуге (*v, u*)

$$dist[u] = min\{ dist[u], dist[v] + c_{vu} \}$$



Программная реализация алгоритма Дейкстры на массиве

```
def distances(start):
    dist[start] = 0

    while True: # <-- здесь!

        min_v = None
        for v, dv in enumerate(dist):
            if not processed[v] and (min_v is None or dv < dist[min_v]):
                min_v = v

        if min_v is None or dist[min_v] == INF:
            break

        processed[min_v] = True
        for u, cu in g[min_v]:
            dist[u] = min(dist[u], dist[min_v] + cu)
```

поиск всех минимумов в массиве $O(n^2)$ + **все релаксации** $O(m)$ + **просмотр списков смежности** $O(n + m) = O(n^2 + m)$

Реализация алгоритма Дейкстры на бинарной куче

1. Считаем все вершины не обработанными

processed[*v*] = ***FALSE***.

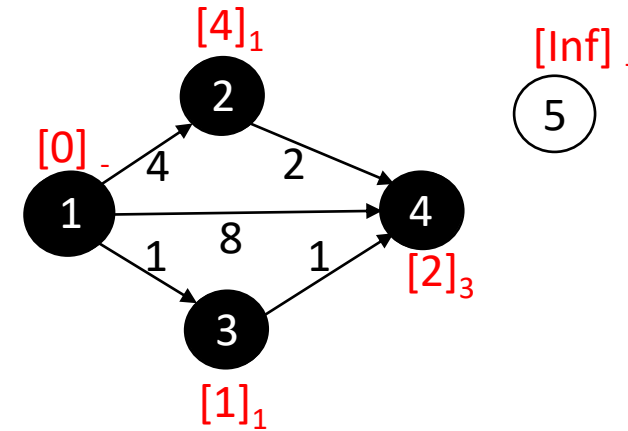
Все элементы массива ***dist*** полагаем равными *INF*, так как кратчайшие пути для вершин ещё не определены.

2. Присваиваем стартовой вершине ***start*** приоритет 0, а остальным вершинам ***d*** [*v*] = *INF*.

3. Добавляем в приоритетную очередь (***min_heap***) пару (***start***, 0).

4. Пока приоритетная очередь не станет пустой, выполняем следующие действия.

- ✓ удаляем элемент из приоритетной очереди (предположим, что это (*v*, ***d***[*v*]));
- ✓ если вершина *v* уже была обработана, то возвращаемся к шагу 4 алгоритма;
- ✓ полагаем вершину *v* обработанной: ***processed***[*v*] = ***TRUE***;
- ✓ просматриваем все смежные с *v* необработанные вершины *u* и добавляем в приоритетную очередь: (*u*, ***d***[*v*] + *c_{vu}*).



(*v* = 1, ***d***[1] = 0) →

(*v* = 2, ***d***[2] = 4)
(*v* = 4, ***d***[4] = 8)
(*v* = 3, ***d***[3] = 1) →

(*v* = 2, ***d***[2] = 4)
(*v* = 4, ***d***[4] = 8)
(*v* = 4, ***d***[4] = 2) →

(*v* = 2, ***d***[4] = 4) →
(*v* = 4, ***d***[4] = 8)

(*v* = 4, ***d***[4] = 8) →

Программная реализация алгоритма Дейкстры на бинарной куче

```
def distances(start):
    q = PriorityQueue()
    q.enqueue((0, start))

    while not q.empty():
        (dv, v) = q.dequeue()

        if processed[v]:
            continue

        processed[v] = True
        dist[v] = dv

        for (u, cu) in g[v]:
            if not processed[u]:
                q.enqueue((dv + cu, u))
```

```
def distances(start):
    q = PriorityQueue()
    q.enqueue((0, start))

    while not q.empty():
        (dv, v) = q.dequeue()

        if processed[v]:
            continue

        processed[v] = True
        dist[v] = dv

        for (u, cu) in g[v]:
            if not processed[u] and dv + cu < dist[u]:
                q.enqueue((dv + cu, u))
```

$O(m \log m)$ +
бинарная куча

$O(n + m)$
просмотр списков смежности

Алгоритм	Время работы	Структура данных	Условия применимости
Форда-Беллмана	$O(n \cdot m)$	массив	нет контуров (циклов) отрицательного веса
Дейкстры	$O(n^2 + m)$	массив	неотрицательные веса
	$O(m \cdot \log m)$	бинарная куча, в куче – рёбра (дуги)	неотрицательные веса
	$O(n \cdot \log n) + O(m \cdot \log n)$	бинарная куча (вершины графа, модификации ключа при релаксации);	неотрицательные веса
	$O(n \cdot \log n) + O(m)$ – усреднённо	куча Фибоначчи (вершины графа, модификации ключа при релаксации);	неотрицательные веса

Специальный класс графов

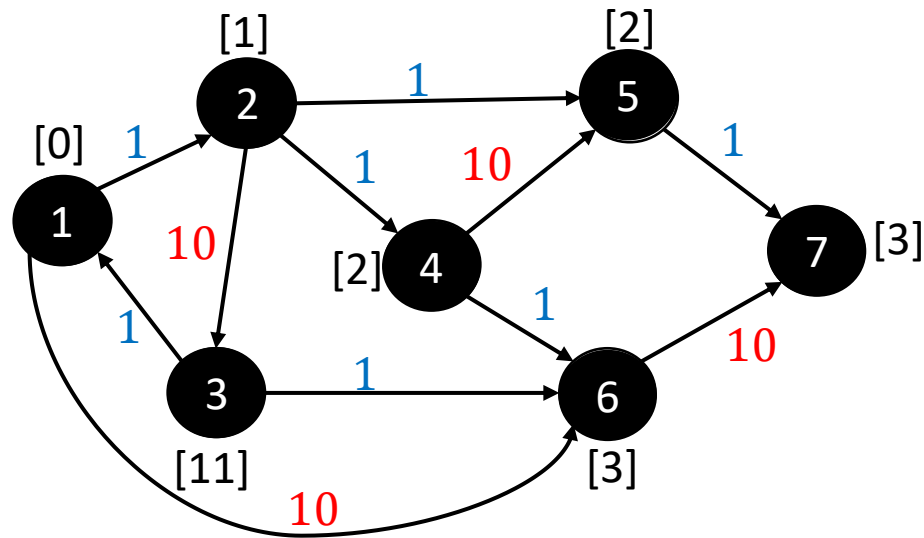
Предположим, что веса дуг орграфа (графа) могут принимать только два значения: ***a*** или ***b***.

Тогда интерфейс «приоритетной очереди» в алгоритме Дейкстры можно реализовать на двух очередях (**queue**):

(1) если $c_{vu} = \mathbf{a}$, то $(d[v] + \mathbf{a}, u) \Rightarrow queue_{\mathbf{a}}$

(2) если $c_{vu} = \mathbf{b}$, то $(d[v] + \mathbf{b}, u) \Rightarrow queue_{\mathbf{b}}$

Несложно показать, что наименьшее значение ***d*** в каждой из очередей имеет тот элемент, который в него был добавлен раньше.



$queue_{\mathbf{a}=10}$ (10, v=6) (11, v=3)

$queue_{\mathbf{b}=1}$ (1, v=2) (2, v=5) (2, v=4) (3, v=7) (3, v=6)

Оценка времени работы алгоритма Дейкстры для специального класса графов

Поиск элемента с минимальным значением d и его удаление — $O(1)$.

Добавление элемента в очередь (*queue*) — $O(1)$.

Общее число добавления элементов в *queue* ограничено числом дуг m .

Время реализации алгоритма Дейкстры (для специального класса графов) — $O(n + m)$, если орграф задан списками смежности.

Если веса дуг принимают только два значения 0 или 1 , то можно реализовать интерфейс приоритетной очереди на двухсторонней очереди (*deque*).

При переходе по дуге веса 0 добавляем пару $(d[v] + 0, u)$ в начало очереди, а при переходе по дуге веса 1 добавляем пару $(d[v] + 1, u)$ — в конец.

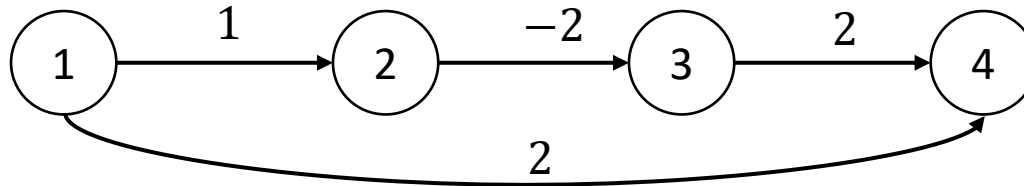
Наименьшее значение d в *deque* — у первого элемента.

Время реализации алгоритма Дейкстры — $O(n + m)$, если орграф задан списками смежности.

Кратчайшие маршруты
между всеми парами вершин

Найти кратчайшие пути между всеми парами вершин:

в орграфе могут быть отрицательные веса дуг, но нет контуров отрицательного веса.



Подходы

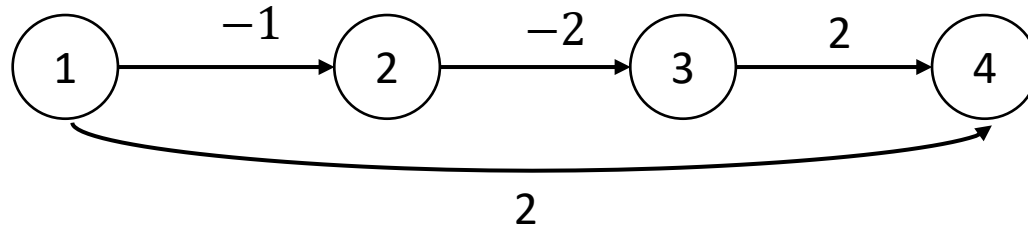
(1) Запуск из каждой вершины алгоритма Форда-Беллмана
 $O(n^2 \cdot m)$.

(2) Свести задачу к неотрицательным весам, а затем из каждой вершины запустить алгоритм Дейкстры, реализованный, например, на бинарной куче
 $O(?) + O(n \cdot m \cdot \log m)$.

(3) Алгоритм Флойда- Уоршелла (Варшалла) – $O(n^3)$ (память – $O(n^2)$).

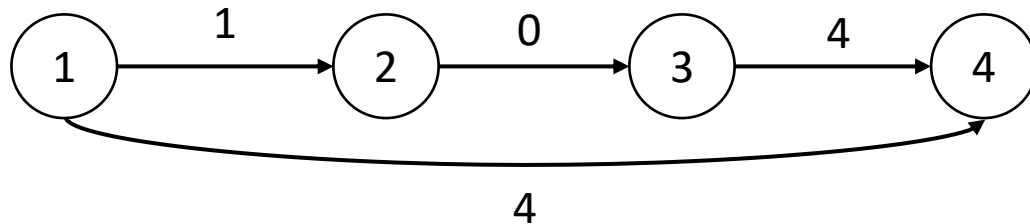
Вопрос

Как свести задачу к неотрицательным весам, чтобы затем из каждой вершины запускать алгоритм Дейкстры?



???

Пусть c_{min} — минимальный отрицательный вес дуги. Преобразуем веса дуг орграфа, увеличив вес каждой дуги на величину $|c_{min}|$.



ОШИБКА!

Необходимо так изменить веса дуг орграфа, чтобы они стали неотрицательными, но при этом сохранялись кратчайшие пути.

Такое преобразование носит название **метод потенциалов**.

Метод потенциалов

1977 год



Джонсон, Дональд Брюс

Donald B. Johnson

1933 -1994

США

ученый-компьютерщик,

исследователь в области проектирования и анализа алгоритмов;

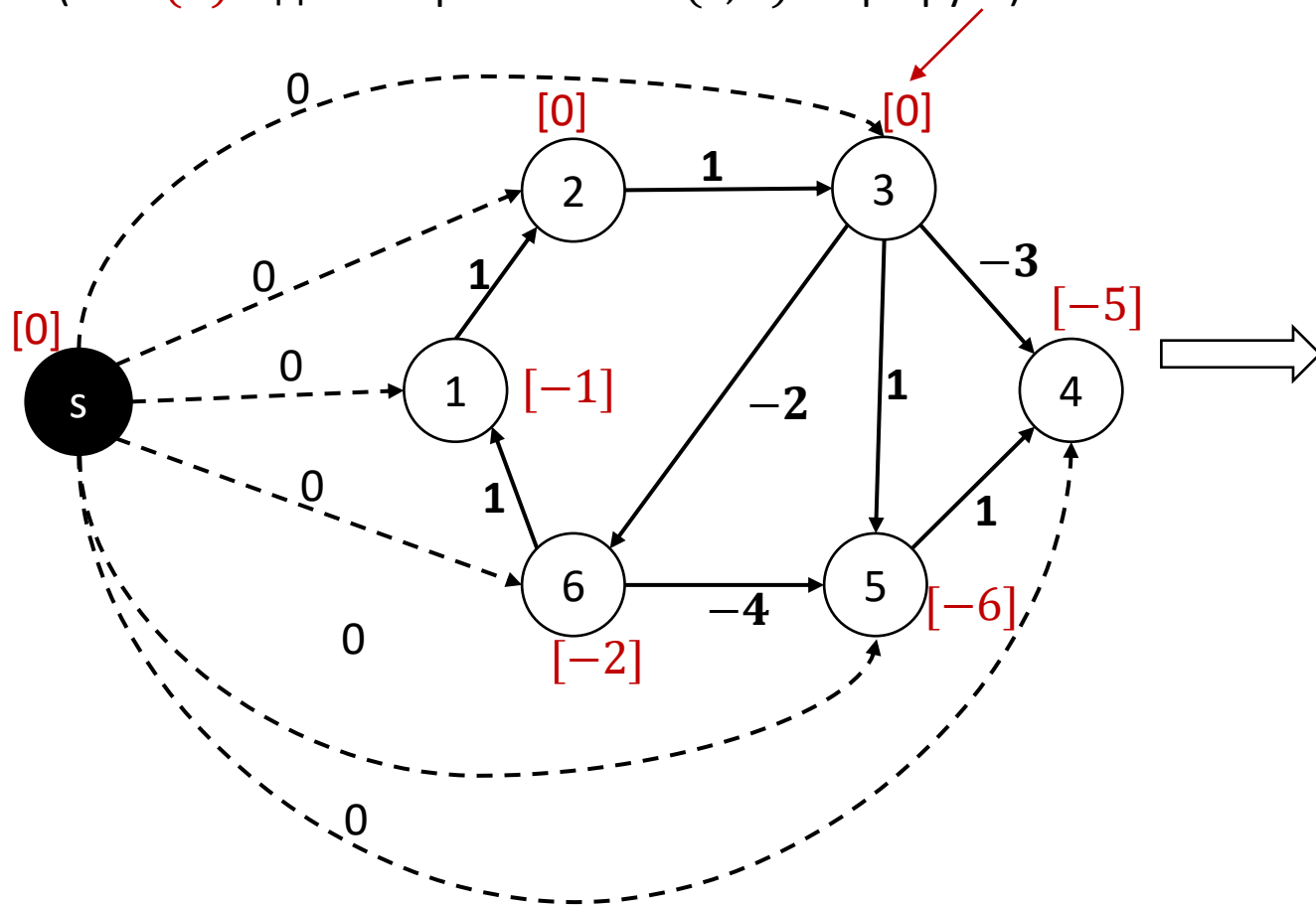
описал структуру данных d-куча;

последняя работа - в области параллельных вычислений

доктор наук, профессор

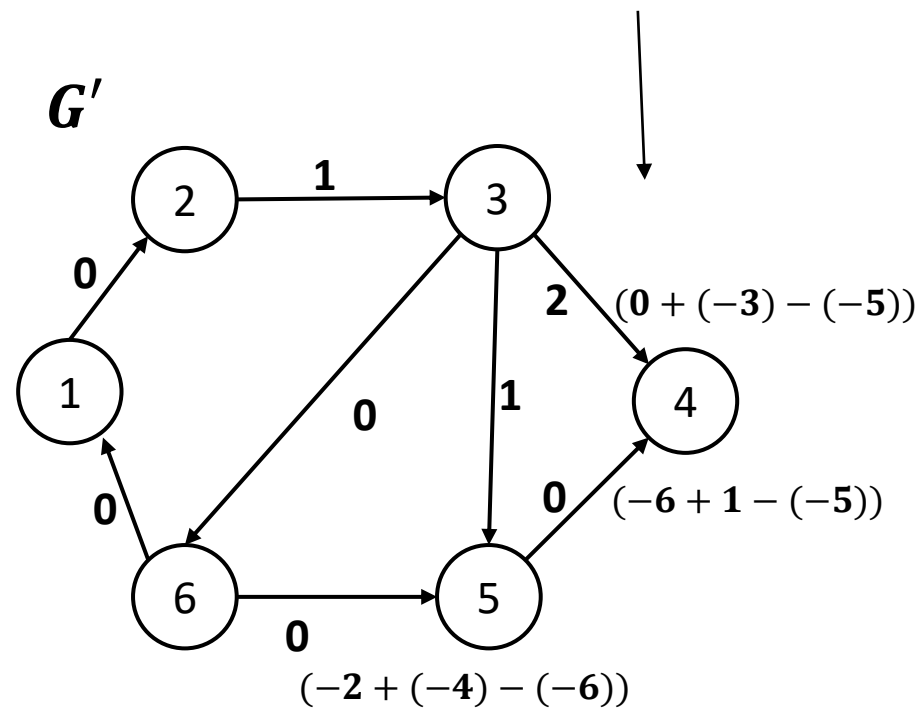
Алгоритм Джонсона (метод потенциалов)

1. Вводим новую вершину s , которую соединяем дугами со всеми вершинами орграфа.
2. Вес фиктивных дуг полагаем равным 0.
3. Один раз запускаем алгоритм Форда-Беллмана из вершины s , находим кратчайшие маршруты из s во все вершины ($dist(v)$ – длина кратчайшего (s, v) -маршрута).



4. Изменяем веса дуг исходного орграфа по следующему правилу:

$$c'(v, u) = dist(v) + c(v, u) - dist(u)$$



Обоснование корректности алгоритма Джонсона

$$1) c'(v, u) = \text{dist}(v) + c(v, u) - \text{dist}(u) \geq 0$$

Верно, так как $(\text{dist}(v) + c(v, u))$ — длина некоторого (s, u) -пути, а $\text{dist}(u)$ — длина кратчайшего (s, u) -пути.

2) Стоимость любого (v, u) -пути P в модифицированном орграфе равна:

$$c'(P) = \text{dist}(v) + c(P) - \text{dist}(u).$$

Поэтому, если взять любой другой (v, u) -путь L , то его стоимость равна:

$$c'(L) = \text{dist}(v) + c(L) - \text{dist}(u).$$

Следовательно, преобразование Джонсона сохраняет кратчайшие пути, т.е.

$$c(P) \leq c(L) \Leftrightarrow c'(P) \leq c'(L)$$

Время работы алгоритма Джонсона

$$O(m \cdot n)$$

преобразование к неотрицательным весам с сохранением кратчайших путей);

+

$$O(n \cdot m \cdot \log m)$$

запуск для каждой вершины в орграфе G' алгоритма Дейкстры

=

$$O(n \cdot m \cdot \log m).$$

Алгоритм Флойда-Уоршелла (Варшалла)

1959 год

Бернар Рой (Bernard Roy)



1934-2017 (83 года)

Почетный профессор Парижского университета-Дофин.

В 1992 году награжден Золотой медалью ЕВРО, высшей наградой в области исследований операций в Европе.

(опубликовал в 1959 году практически такой же алгоритм, но результат остался незамеченным)

1962 год

Stephen Warshall

Стивен Уоршелл



Дата рождения 15 ноября 1935
Место рождения Нью-Йорк, Нью-Йорк, США
Дата смерти 11 декабря 2006 (71 год)
Место смерти Глостер, Эссекс, Массачусетс, США
Страна США
Альма-матер Гарвардский университет

Роберт В Флойд

Robert W Floyd



Флойд в 1976 году

Дата рождения 8 июня 1936
Место рождения Нью-Йорк
Дата смерти 25 сентября 2001 (65 лет)
Место смерти Станфорд
Страна США
Научная сфера Информатика
Место работы Университет Карнеги — Меллон
Стэнфордский университет
Альма-матер Чикагский университет

(опубликован одновременно)

Алгоритм Флойда-Уоршелла (Варшалла)

Предположим, что вершины графа занумерованы целыми числами от **1** до $|V|$.

Веса дуг орграфа могут быть отрицательными, но предполагается, что нет контуров отрицательного веса.

матрица весов дуг орграфа

$$c_{ij} = \begin{cases} 0, & \text{если } i = j, \\ +\infty, & \text{если } i \neq j \text{ и } (i, j) \notin E, \\ \text{вес дуги } (i, j), & \text{если } i \neq j \text{ и } (i, j) \in E. \end{cases}$$

В основе алгоритма Флойда-Уоршелла лежит принцип динамического программирования.

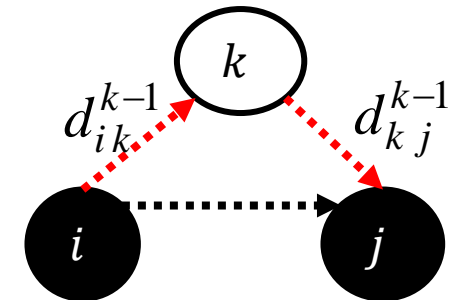
Пусть d_{ij}^k — длина кратчайшего пути, соединяющего вершины i и j и проходящего возможно только через промежуточные вершины с номерами $\{1, 2, \dots, k\}$, при этом $i, j \notin \{1, \dots, k\}$.

Тогда справедливо следующее рекуррентное соотношение:

$$d_{ij}^k = \begin{cases} c(i, j), & \text{если } k = 0, \\ \min \left(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1} \right), & \text{если } k \geq 1. \end{cases}$$

↑
путь НЕ проходит
через вершину k

путь проходит
через вершину k



Алгоритм Флойда-Уоршелла (Варшалла)

```
for i in range(n):
```

```
    for j in range(n):
```

```
        d[i,j]=cij
```

```
for k in range(n):
```

```
    for i in range(n):
```

```
        for j in range(n):
```

```
            if d[i,k]+d[k,j]<d[i,j]
```

```
                d[i,j]=d[i,k]+d[k,j]
```

матрица весов дуг орграфа

$$c_{ij} = \begin{cases} 0, & \text{если } i = j, \\ +\infty, & \text{если } i \neq j \text{ и } (i,j) \notin E, \\ \text{вес дуги } (i,j), & \text{если } i \neq j \text{ и } (i,j) \in E. \end{cases}$$

Время работы: $O(n^3)$

Кратчайший маршрут между всеми парами вершин

допускаются дуги отрицательного веса, но нет контуров отрицательного веса

Запуск из каждой вершины алгоритма Форда-Беллмана	$O(n^2 \cdot m)$
Метод потенциалов сведения задачи к задаче с неотрицательными весами и последующее применение из каждой вершины алгоритма Дейкстры (интерфейс приоритетной очереди реализуется, например, на бинарной куче)	$O(n \cdot m \cdot \log m)$
Алгоритм Флойда- Уоршелла	$O(n^3)$ дополнительная память $-O(n^2)$

Общие задачи в iRunner для закрепления навыков

0.10 Кратчайший путь. Алгоритм Дейкстры



Спасибо за внимание!