Сириус
IT-Колледж

Лекция 8. Кратчайшие пути в ациклических ориентированных графах

Нахождение кратчайших путей из одной вершины в графах с рёбрами отрицательного веса, алгоритм Беллмана-Форда, проверка наличия цикла отрицательного веса. Кратчайшие пути между всеми парами вершин: алгоритм Флойда-Уоршелла.

Кратчайшие пути в ациклических ориентированных графах

Пусть дан ациклический ориентированный взвешенный граф. Требуется найти вес кратчайшего пути из u в v.

Пусть d — функция, где $d(i)$ — вес кратчайшего пути из u в i. Ясно, что $d(u)$ равен 0. Пусть $w(i,j)$ — вес ребра из i в j. Будем обходить граф в порядке топологической сортировки. Получаем следующие соотношения:

$$d(i) = \min_{j:j \rightsquigarrow i} \left(d(j) + w(j,i)\right)$$

Так как мы обходим граф в порядке топологической сортировки, то на i-ом шаге всем $d(j)$ (j такие, что существует ребро из j в i) уже присвоены оптимальные ответы, и, следовательно, $d(i)$ также будет присвоен оптимальный ответ.

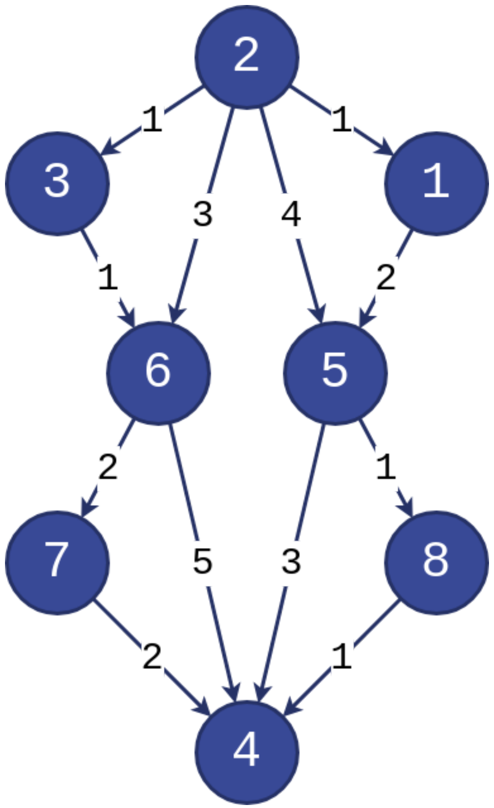
Кратчайшие пути в ациклических ориентированных графах. Пример

Требуется найти вес кратчайшего пути из 2 в 4 графе:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1					2			
2	1		1		4	3		
3						1		
4								
5				3				1
6				5			2	
7				2				
8				1				

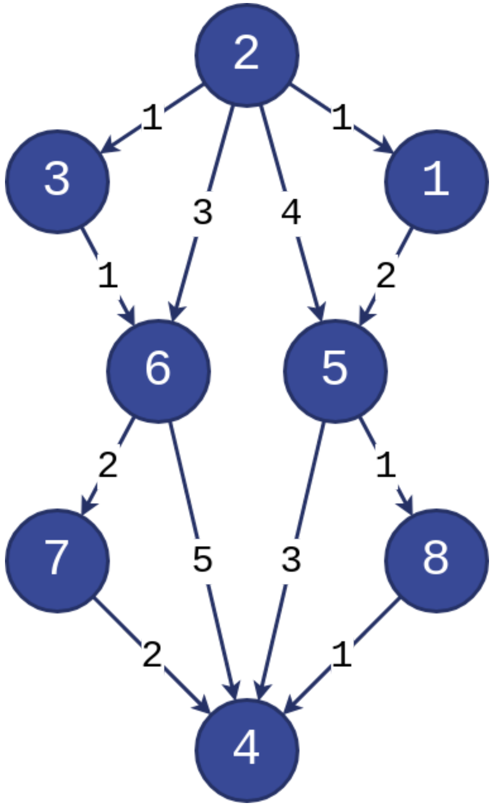
Кратчайшие пути в ациклических ориентированных графах. Пример

Требуется найти вес кратчайшего пути из 2 в 4 графе:



Кратчайшие пути в ациклических ориентированных графах. Пример

Требуется найти вес кратчайшего пути из 2 в 4 графе:

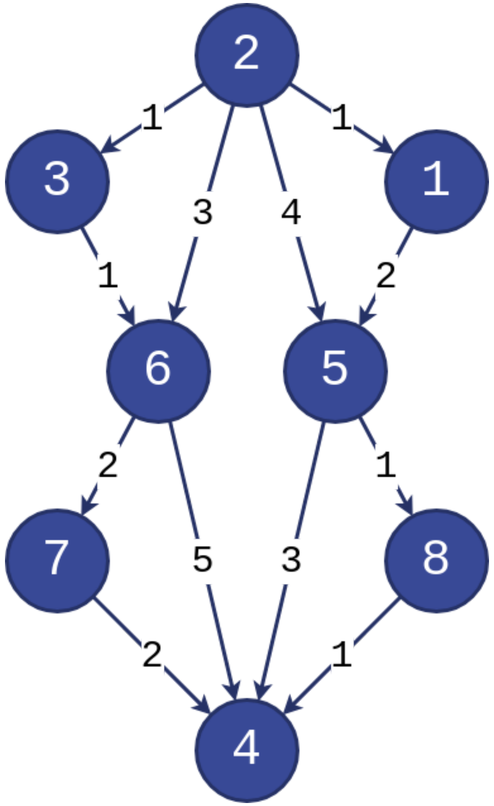


Массив p (топологическая сортировка) будет выглядеть следующим образом:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$p[i]$	2	3	6	7	1	5	8	4

Кратчайшие пути в ациклических ориентированных графах. Пример

Требуется найти вес кратчайшего пути из 2 в 4 графе:



Массив p (топологическая сортировка) будет выглядеть следующим образом:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$p[i]$	2	3	6	7	1	5	8	4

Массив расстояний d будет выглядеть следующим образом:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$d[i]$	1	0	1	5	3	2	4	4

Сириус
IT-Колледж

Кратчайшие пути в ациклических ориентированных графах

```
def minDist(n, w, u):
    dist = [float('inf')] * n
    dist[u] = 0

    p = topSort(w)

    for i in range(n)
        for j in range(n):
            if w[i][j] > 0:
                dist[j] = min(d[j], dist[p[i]] + w[p[i]][j])
```

Кратчайшие пути в графах, с рёбрами отрицательного веса

Для заданного взвешенного графа $G = (V, E)$ найти кратчайшие пути из заданной вершины s до всех остальных вершин. В случае, когда в графе G содержатся циклы с отрицательным суммарным весом, достижимые из s, сообщить, что кратчайших путей не существует.

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда предназначен для решения задачи поиска кратчайшего пути на графе. Для заданного ориентированного взвешенного графа алгоритм находит кратчайшие расстояния от выделенной вершины-источника до всех остальных вершин графа.

Алгоритм Беллмана-Форда масштабируется хуже других алгоритмов решения указанной задачи (сложность $O(|V||E|)$ против $O(|E|+|V|ln(|V|))$ у алгоритма Дейкстры), однако его отличительной особенностью является применимость к графам с произвольными, в том числе отрицательными, весами.

Алгоритм Беллмана-Форда

1. Инициализация: всем вершинам присваивается предполагаемое расстояние $dist[v] = \infty$, кроме вершины-источника, для которой $dist(u) = 0$.
2. Релаксация множества рёбер E
   i. Для каждого ребра $e = (v, z) \in E$ вычисляется новое предполагаемое расстояние $new\_dist(z) = dist(v) + w(e)$.
   ii. Если $new\_dist(z) < dist(z)$, то происходит присваивание $dist(z) = new\_dist(z)$ (релаксация ребра e).
3. Алгоритм производит релаксацию всех рёбер графа до тех пор, пока на очередной итерации происходит релаксация хотя бы одного ребра.

Сириус
IT-Колледж

Алгоритм Беллмана-Форда

```
class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def printArr(self, dist):
        print("Расстояние до стартовой")
        for i in range(self.V):
            print(f"{i}\t\t{dist[i]}")
```

```
    def BellmanFord(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0

        for _ in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u] != float("Inf") \
                and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w

        self.printArr(dist)
```

Проверка наличия цикла отрицательного веса

Алгоритм Форда-Беллмана сможет бесконечно делать релаксации среди всех вершин этого цикла и вершин, достижимых из него. Следовательно, если не ограничивать число фаз числом $n - 1$, то алгоритм будет работать бесконечно, постоянно улучшая расстояния до этих вершин.

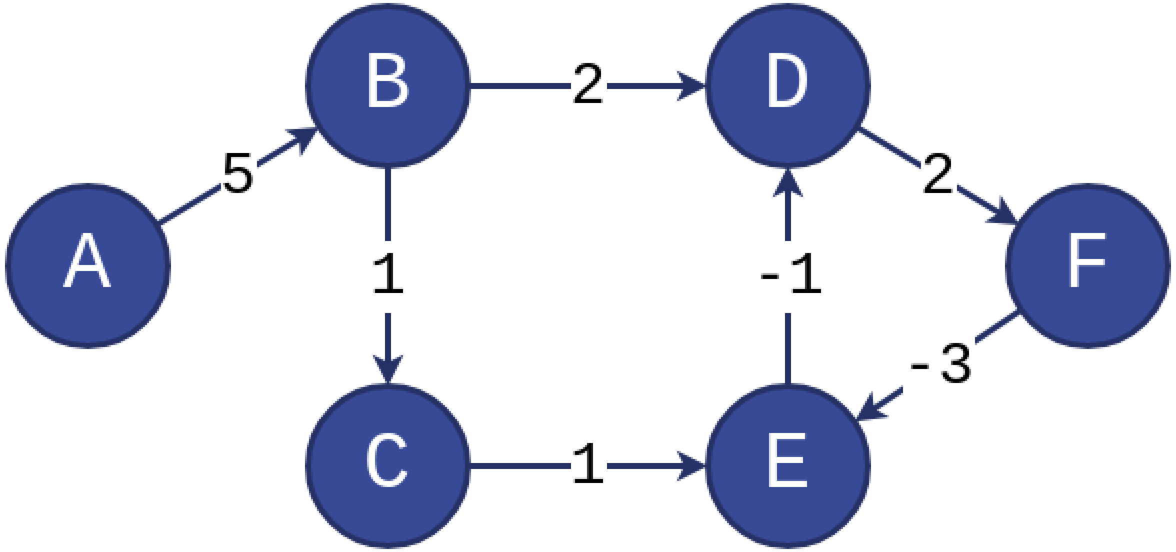
Отсюда мы получаем **критерий наличия достижимого цикла отрицательного веса**: если после $n - 1$ фазы мы выполним ещё одну фазу, и на ней произойдёт хотя бы одна релаксация, то граф содержит цикл отрицательного веса, достижимый из v; в противном случае, такого цикла нет.

Сириус
IT-Колледж

Проверка наличия цикла отрицательного веса

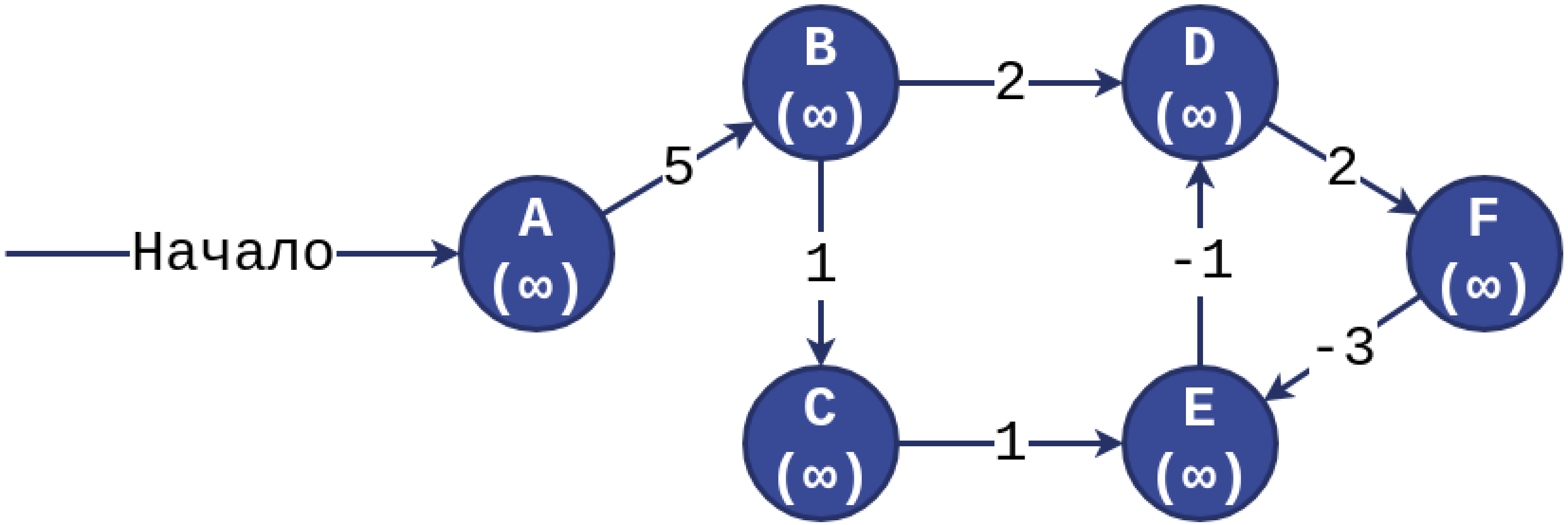
```
def BellmanFord(self, src):
    dist = [float("Inf")] * self.V
    dist[src] = 0

    for _ in range(self.V - 1):
        for u, v, w in self.graph:
            if dist[u] != float("Inf") \
            and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    for u, v, w in self.graph:
        if dist[u] != float("Inf") \
        and dist[u] + w < dist[v]:
            print("Граф содержит отрицательный цикл")
            return

    self.printArr(dist)
```

Алгоритм Беллмана-Форда. Пример

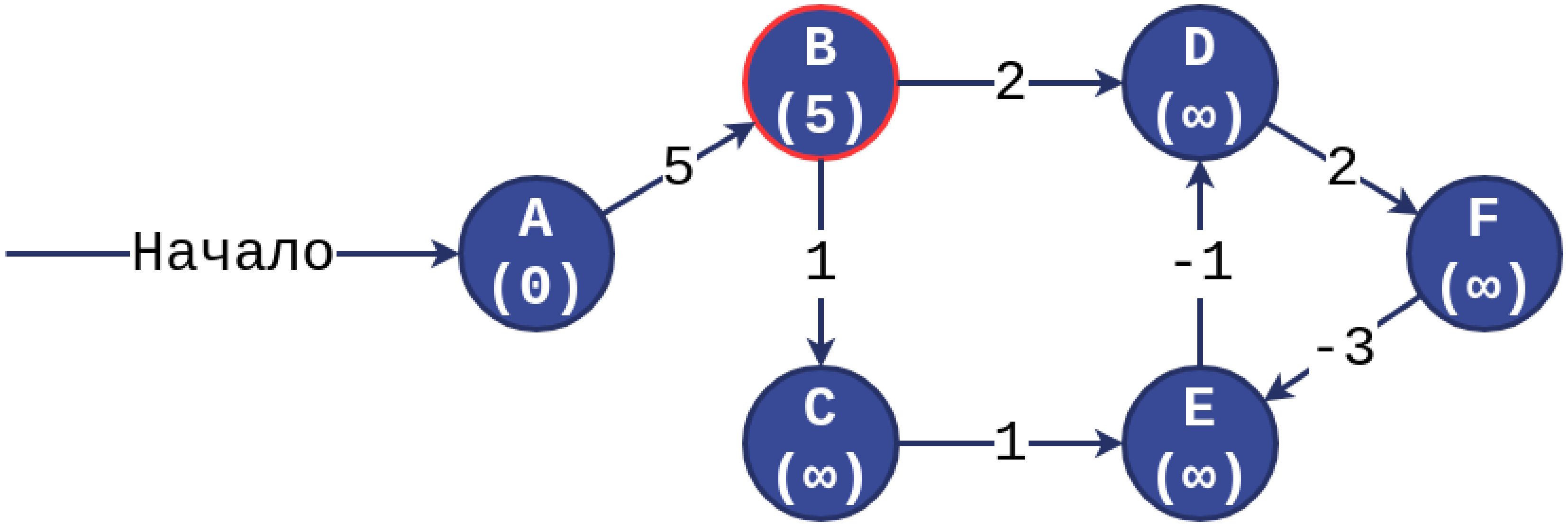


Сириус IT-Колледж

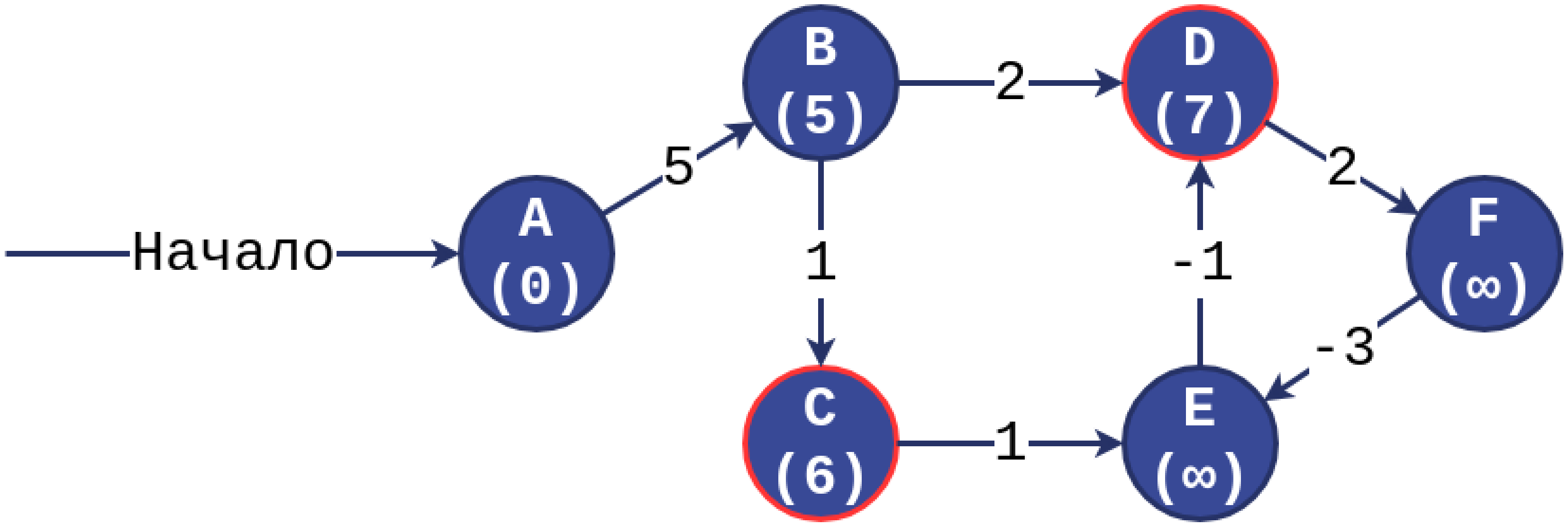
Алгоритм Беллмана-Форда. Пример



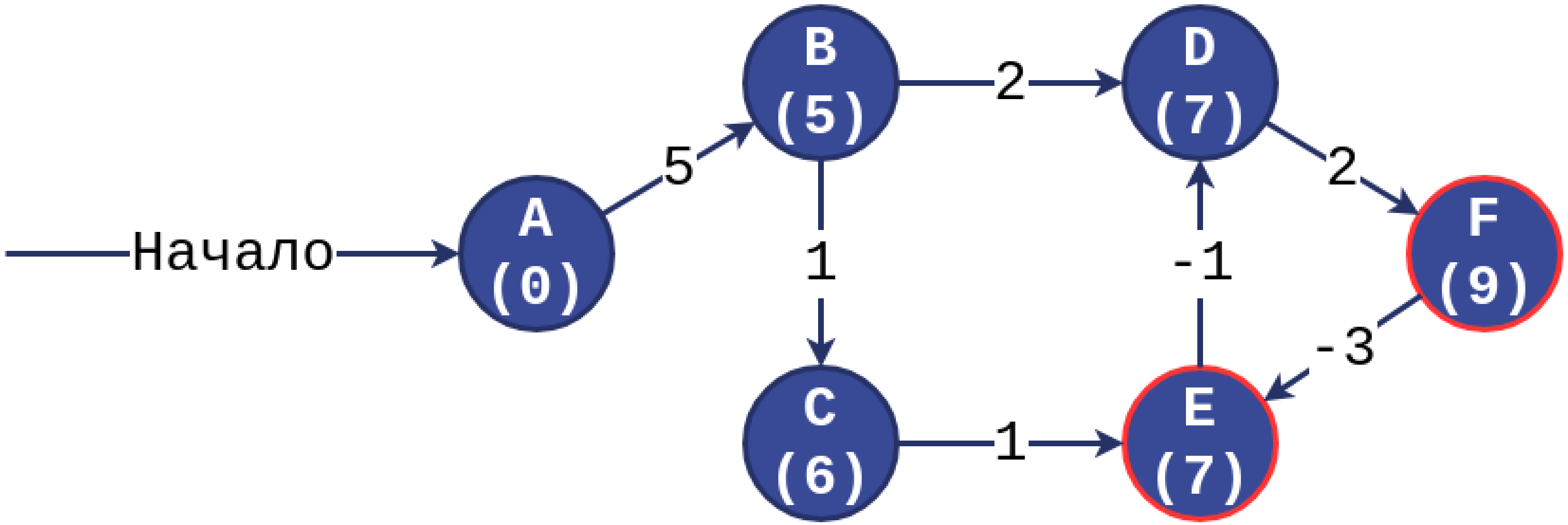
Алгоритм Беллмана-Форда. Пример



Алгоритм Беллмана-Форда. Пример

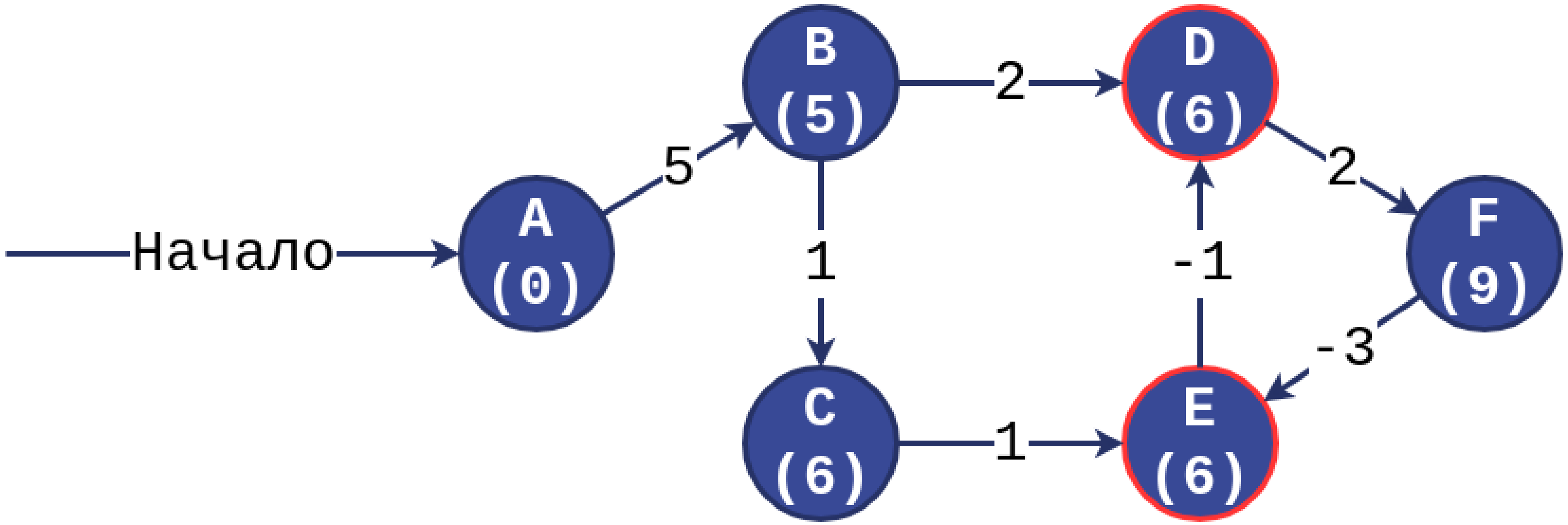


Алгоритм Беллмана-Форда. Пример

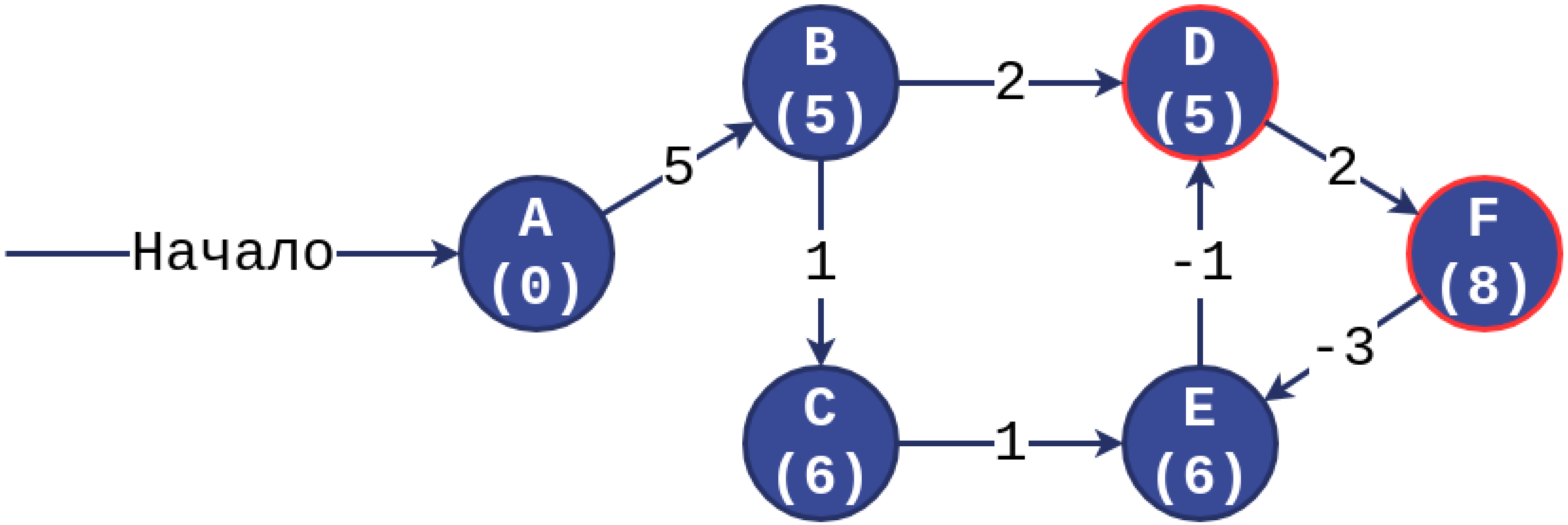


Сириус
IT-Колледж

Алгоритм Беллмана-Форда. Пример

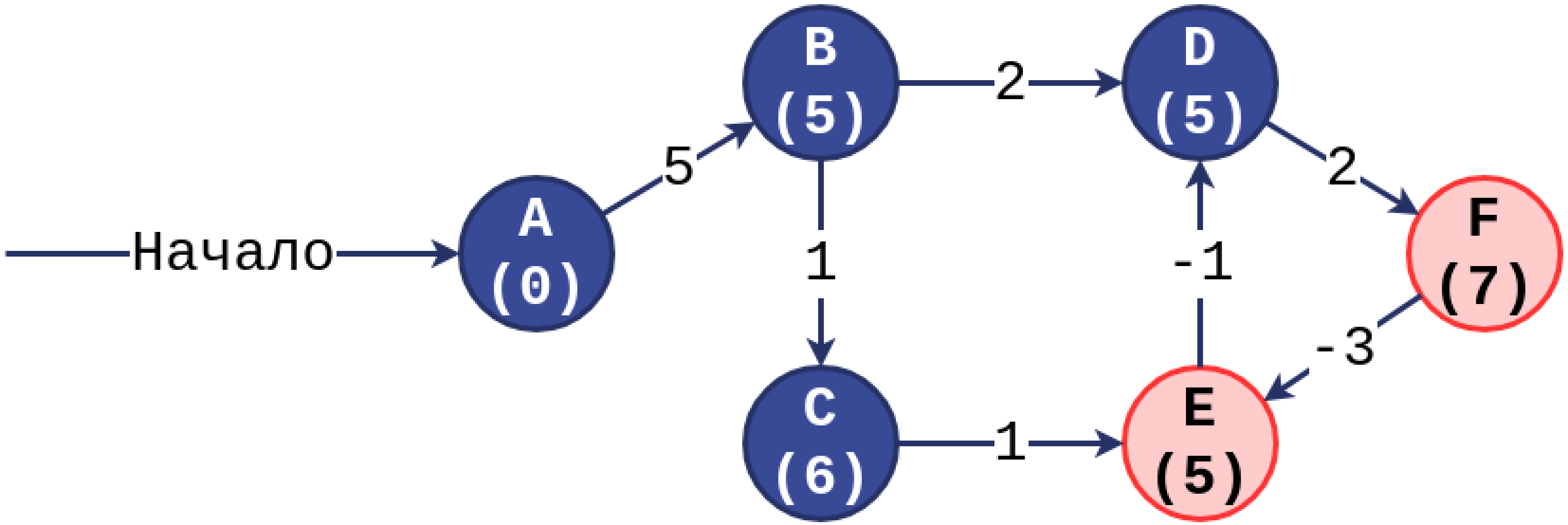


Алгоритм Беллмана-Форда. Пример



Сириус IT-Колледж

Алгоритм Беллмана-Форда. Пример



Кратчайшие пути между всеми парами вершин. Алгоритм Флойда-Уоршелла

Алгоритм Флойда (алгоритм Флойда–Уоршелла) — алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Работает корректно, если в графе нет циклов отрицательной величины, а в случае, когда такой цикл есть, позволяет найти хотя бы один такой цикл. Алгоритм работает за $O(n^3)$ времени и использует $O(n^2)$ памяти. Разработан в 1962 году.

Постановка задачи

Дан взвешенный ориентированный граф $G(V, E)$, в котором вершины пронумерованы от 1 до n.

$$\omega_{uv} = \begin{cases} \text{weight of } uv & \text{if } uv \in E \\ +\infty, & \text{if } uv \notin E \end{cases}$$

Требуется найти матрицу кратчайших расстояний d, в которой элемент d_{ij} либо равен длине кратчайшего пути из i в j, либо равен $+\infty$, если вершина j не достижима из i.

Кратчайшие пути между всеми парами вершин. Алгоритм Флойда-Уоршелла

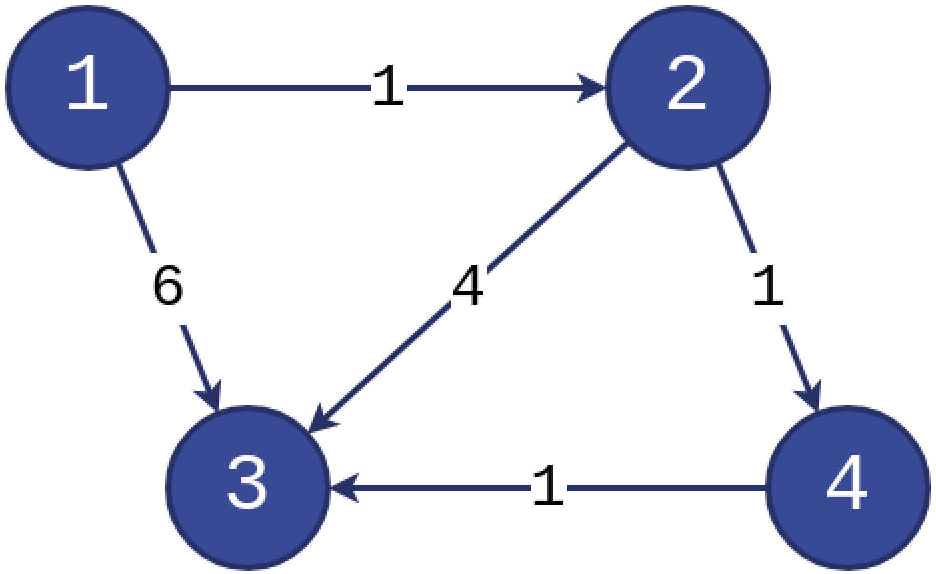
Обозначим длину кратчайшего пути между вершинами u и v, содержащего, помимо u и v, только вершины из множества $\{1..i\}$ как $d_{uv}^{(i)}$, $d_{uv}^{(0)} = \omega_{uv}$.

На каждом шаге алгоритма, мы будем брать очередную вершину (пусть её номер — i) и для всех пар вершин u и v вычислять $d_{uv}^{(i)} = \min\left(d_{uv}^{(i-1)}, d_{ui}^{(i-1)} + d_{iv}^{(i-1)}\right)$. То есть, если кратчайший путь из u в v, содержащий только вершины из множества $\{1..i\}$, проходит через вершину i, то кратчайшим путем из u в v является кратчайший путь из u в i, объединенный с кратчайшим путем из i в v. В противном случае, когда этот путь не содержит вершины i, кратчайший путь из u в v, содержащий только вершины из множества $\{1..i\}$ является кратчайшим путем из u в v, содержащим только вершины из множества $\{1..i-1\}$.

Кратчайшие пути между всеми парами вершин. Алгоритм Флойда-Уоршелла

```python
def floydWarshall(graph):
    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))
    for k in range(V):
        for i in range(V):
            for j in range(V):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
```

Алгоритм Флойда-Уоршелла. Пример

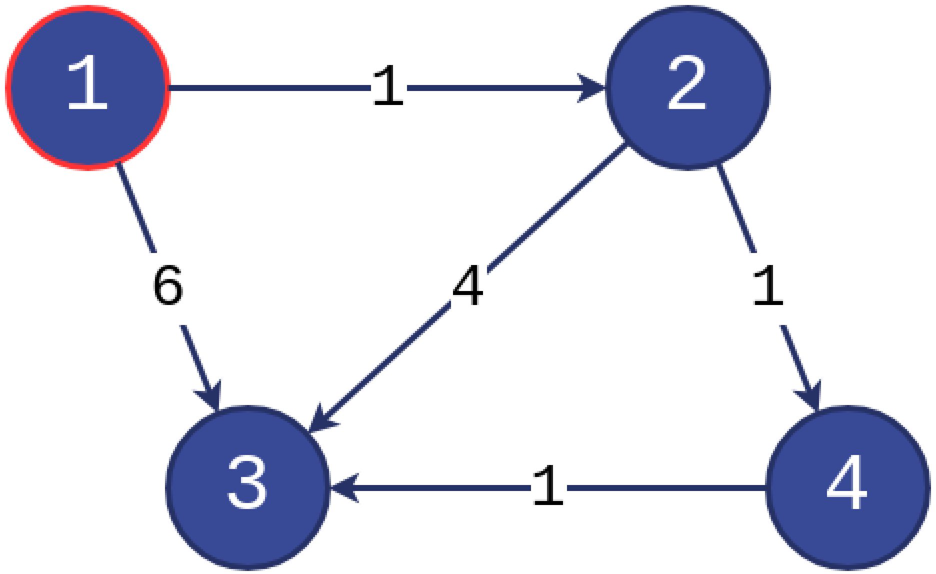


$$i = 0$$

$$\begin{pmatrix} \times & 1 & 6 & \infty \\ \infty & \times & 4 & 1 \\ \infty & \infty & \times & \infty \\ \infty & \infty & 1 & \times \end{pmatrix}$$

Сириус IT-Колледж

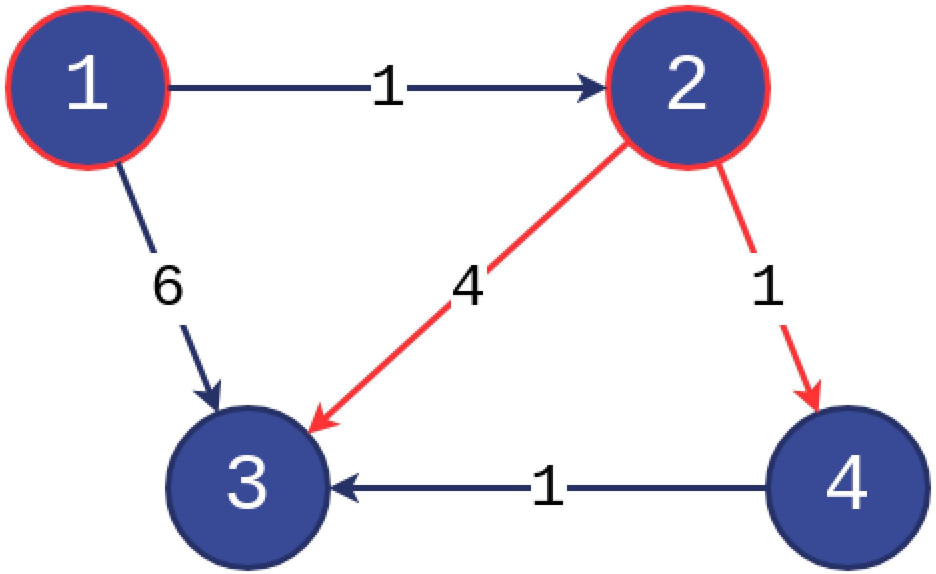
Алгоритм Флойда-Уоршелла. Пример



$$i = 1$$

$$\begin{pmatrix} \times & 1 & 6 & \infty \\ \infty & \times & 4 & 1 \\ \infty & \infty & \times & \infty \\ \infty & \infty & 1 & \times \end{pmatrix}$$

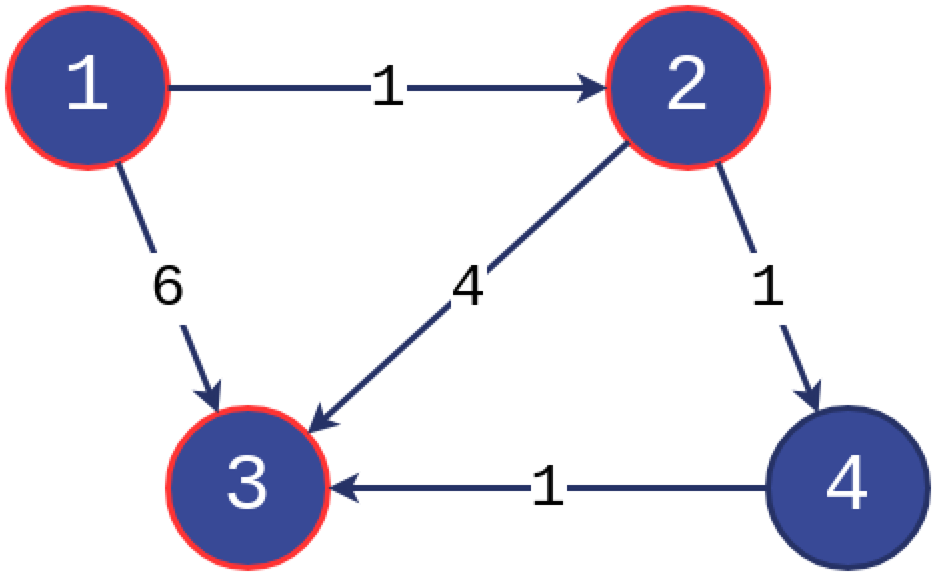
Алгоритм Флойда-Уоршелла. Пример



$$i = 2$$

$$\begin{pmatrix} \times & 1 & \color{red}{5} & \color{red}{2} \\ \infty & \times & 4 & 1 \\ \infty & \infty & \times & \infty \\ \infty & \infty & 1 & \times \end{pmatrix}$$

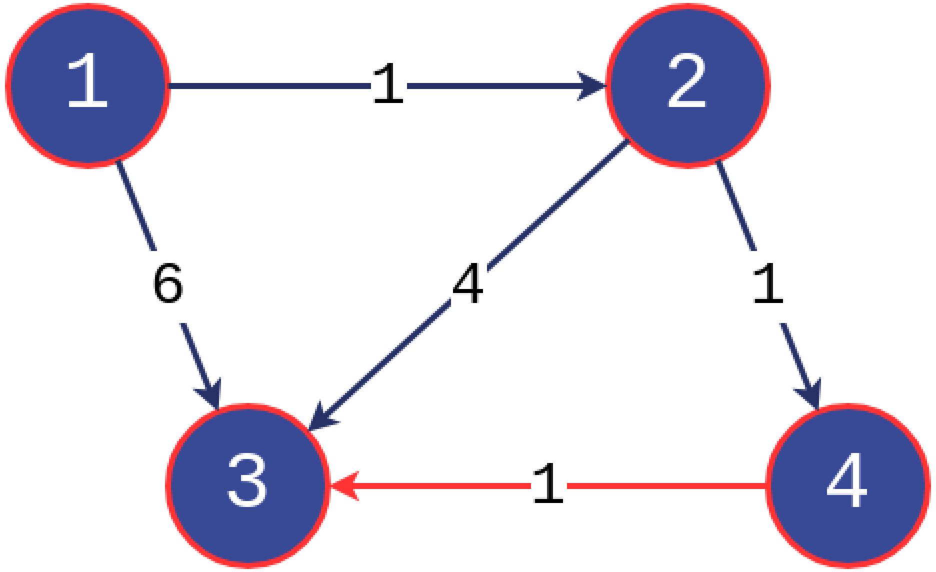
Алгоритм Флойда-Уоршелла. Пример



$$i = 3$$

$$\begin{pmatrix} \times & 1 & 5 & 2 \\ \infty & \times & 4 & 1 \\ \infty & \infty & \times & \infty \\ \infty & \infty & 1 & \times \end{pmatrix}$$

Алгоритм Флойда-Уоршелла. Пример



$$i = 4$$

$$\begin{pmatrix} \times & 1 & \color{red}{3} & 2 \\ \infty & \times & \color{red}{4} & 1 \\ \infty & \infty & \times & \infty \\ \infty & \infty & 1 & \times \end{pmatrix}$$