Взвешенные графы

Борис Золотов (Б09)
Михаил Иванов (Б05, Б06)
Иван Казменко (Б01, Б02)
Владислав Макаров (Б03)
Арина Филимонова (Б10)

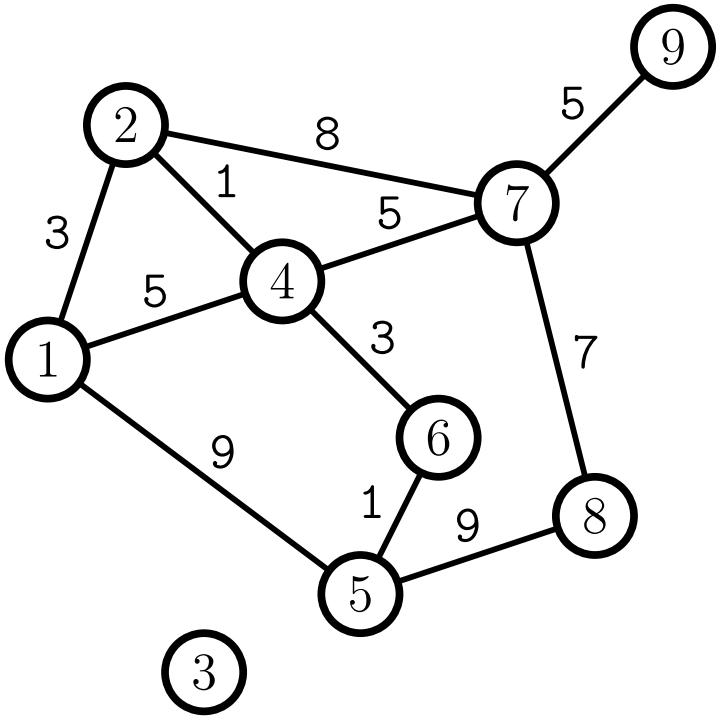
Санкт-Петербургский Государственный Университет

Вторник, 10 декабря 2024 года

Содержание

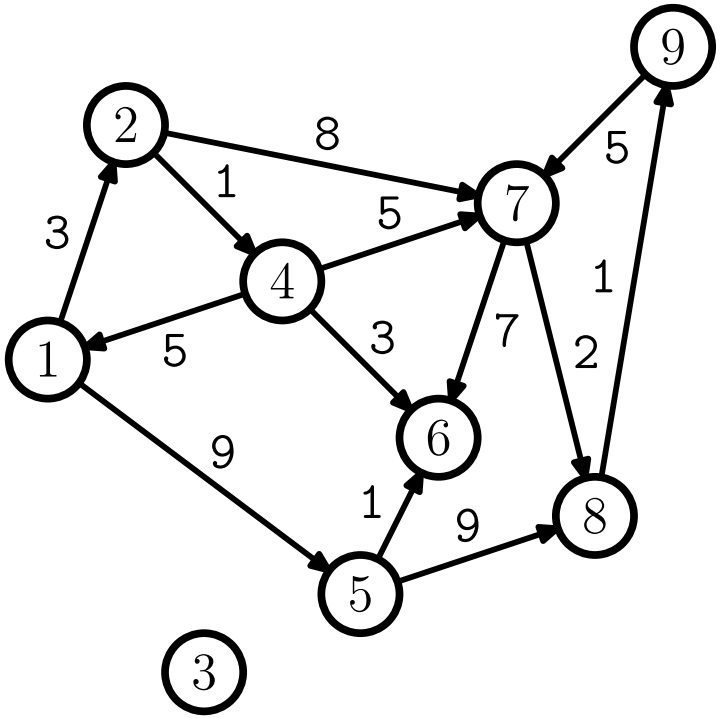
Содержание

Определения



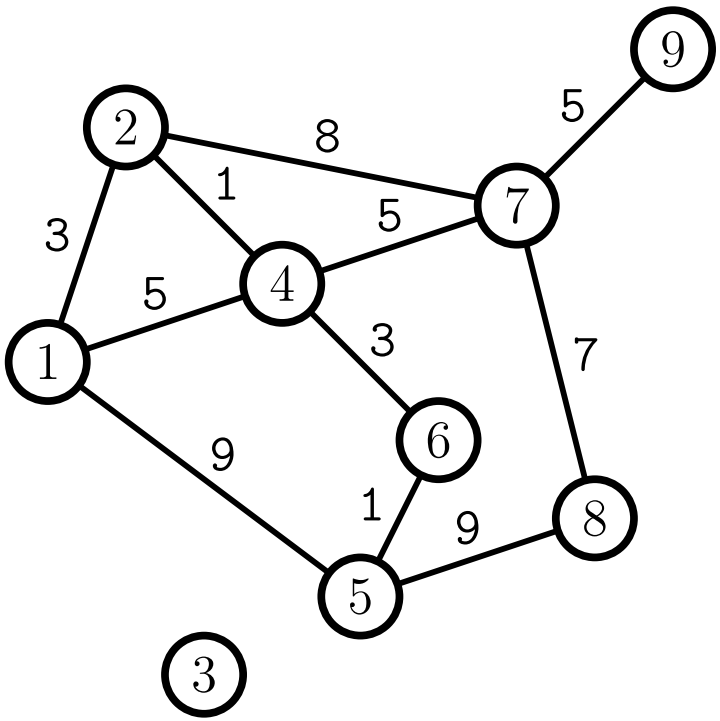
Определение: взвешенный граф (неориентированный) — это пара (V, E), где V — множество вершин, а E — множество (неориентированных) рёбер. Каждое ребро — это неупорядоченная пара вершин, которой приписано число — его вес.

Определения



Определение: взвешенный ориентированный граф (орграф) — это пара (V, E), где V — множество вершин, а E — множество дуг (ориентированных рёбер). Каждое ребро — это упорядоченная пара вершин, которой приписано число — его вес.

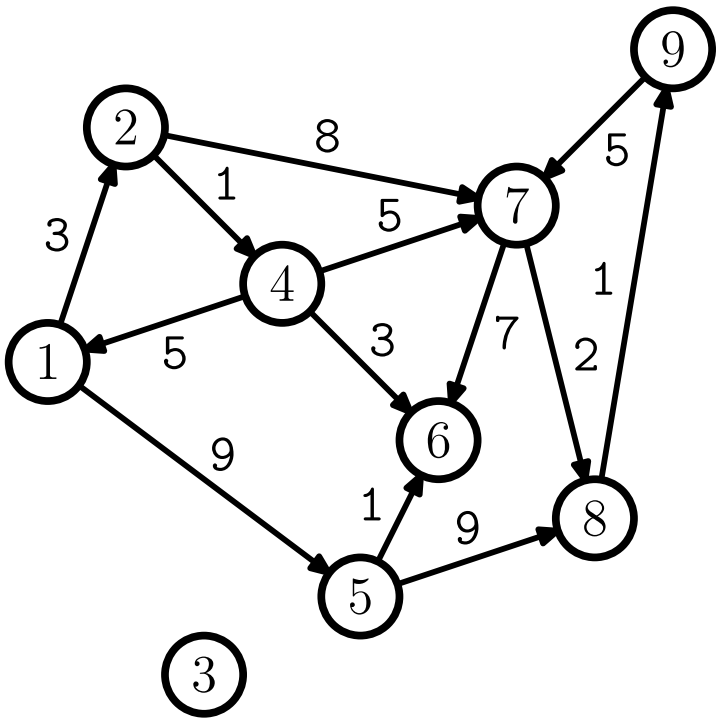
Задание графа



Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

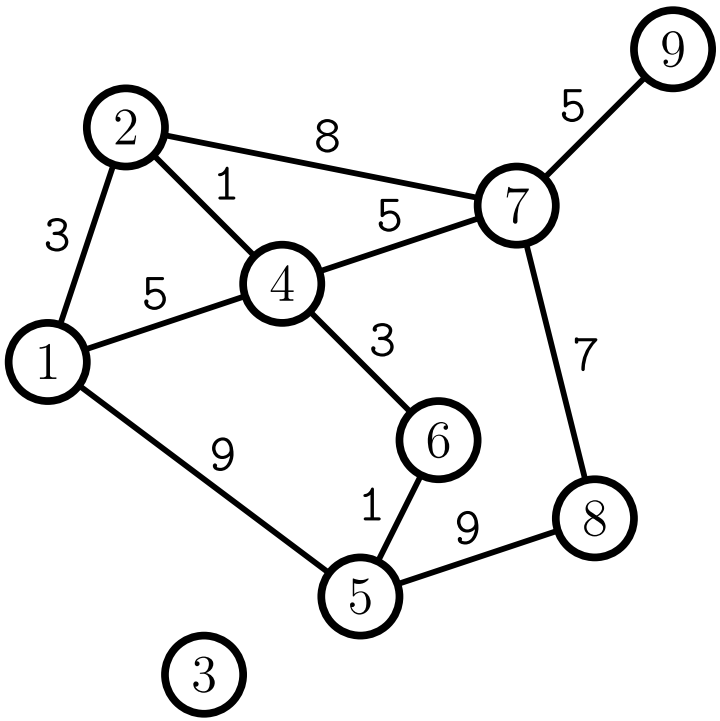
Задание графа



Ввод:

```
9 13
1 2 3
2 7 8
2 4 1
4 1 5
7 6 7
4 6 3
4 7 5
5 8 9
5 6 1
7 8 2
9 7 5
1 5 9
8 9 1
```

Матрица смежности

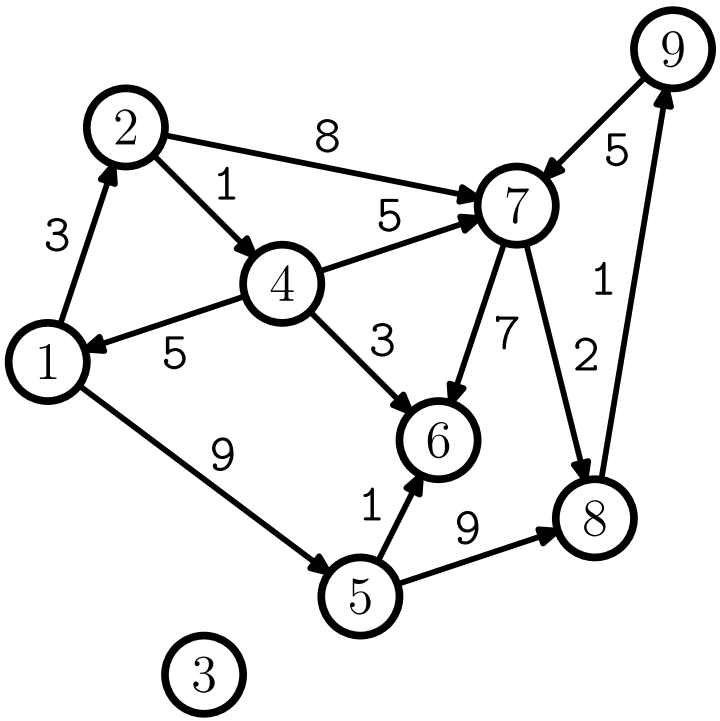


Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

	1	2	3	4	5	6	7	8	9
1	∞	3	∞	5	9	∞	∞	∞	∞
2	3	∞	∞	1	∞	∞	8	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	5	1	∞	∞	∞	3	5	∞	∞
5	9	∞	∞	∞	∞	1	∞	9	∞
6	∞	∞	∞	3	1	∞	∞	∞	∞
7	∞	8	∞	5	∞	∞	∞	7	5
8	∞	∞	∞	∞	9	∞	7	∞	∞
9	∞	∞	∞	∞	∞	∞	5	∞	∞

Матрица смежности

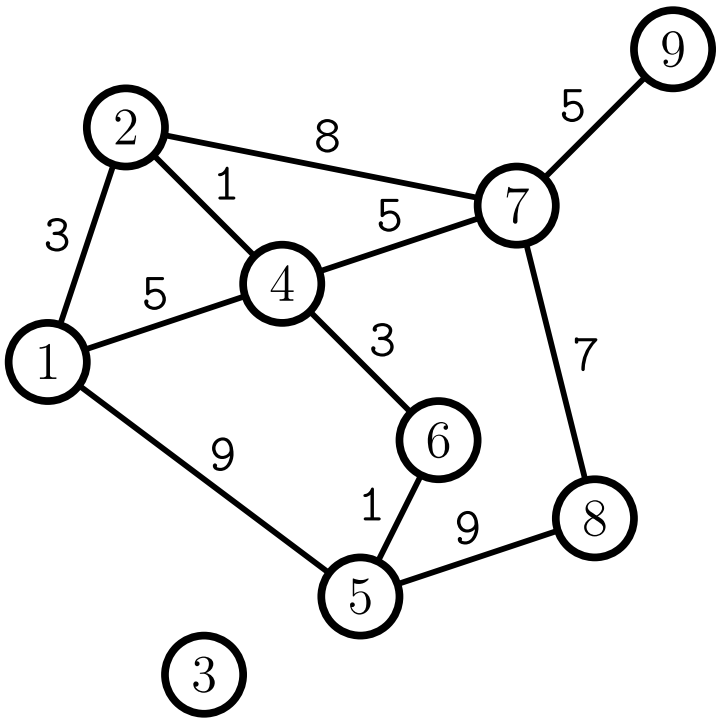


Ввод:

```
9 13
1 2 3
2 7 8
2 4 1
4 1 5
7 6 7
4 6 3
4 7 5
5 8 9
5 6 1
7 8 2
9 7 5
1 5 9
8 9 1
```

	1	2	3	4	5	6	7	8	9
1	∞	3	∞	∞	9	∞	∞	∞	∞
2	∞	∞	∞	1	∞	∞	8	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	5	∞	∞	∞	∞	3	5	∞	∞
5	∞	∞	∞	∞	∞	1	∞	9	∞
6	∞	∞	∞	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	7	∞	2	∞
8	∞	∞	∞	∞	∞	∞	∞	∞	1
9	∞	∞	∞	∞	∞	∞	5	∞	∞

Списки смежных рёбер

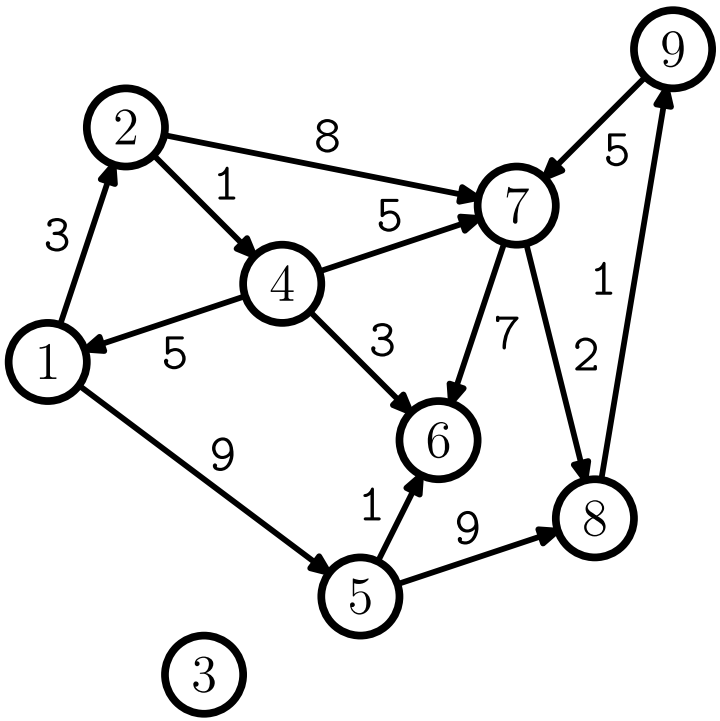


Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

1	2,3	4,5	5,9	
2	1,3	7,8	4,1	
3				
4	1,5	2,1	6,3	7,5
5	8,9	6,1	1,9	
6	4,3	5,1		
7	2,8	4,5	9,5	8,7
8	5,9	7,7		
9	7,5			

Списки смежных рёбер



Ввод:

```
9 13
1 2 3
2 7 8
2 4 1
4 1 5
7 6 7
4 6 3
4 7 5
5 8 9
5 6 1
7 8 2
9 7 5
1 5 9
8 9 1
```

1	2,3	5,9	
2	7,8	4,1	
3			
4	1,5	6,3	7,5
5	8,9	6,1	
6			
7	6,7	8,2	
8	9,1		
9	7,5		

Матрица смежности

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  int const infinity = 0x3F3F3F3F;
 6
 7  int main () {
 8      int n, m;
 9      cin >> n >> m;
10      vector <vector <int> > a (n,
11          vector <int> (n, infinity));
12      for (int j = 0; j < m; j++) {
13          int u, v, w;
14          cin >> u >> v >> w;
15          u -= 1;
16          v -= 1;
17          a[u][v] = w;
18          a[v][u] = w;
19      }
20      return 0;
21  }
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

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

Матрица смежности

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  int const infinity = 0x3F3F3F3F;
 6
 7  int main () {
 8      int n, m;
 9      cin >> n >> m;
10      vector <vector <int> > a (n,
11          vector <int> (n, infinity));
12      for (int j = 0; j < m; j++) {
13          int u, v, w;
14          cin >> u >> v >> w;
15          u -= 1;
16          v -= 1;
17          a[u][v] = w;
18
19      }
20      return 0;
21  }
```

Ввод:

```
9 13
1 2 3
2 7 8
2 4 1
4 1 5
7 6 7
4 6 3
4 7 5
5 8 9
5 6 1
7 8 2
9 7 5
1 5 9
8 9 1
```

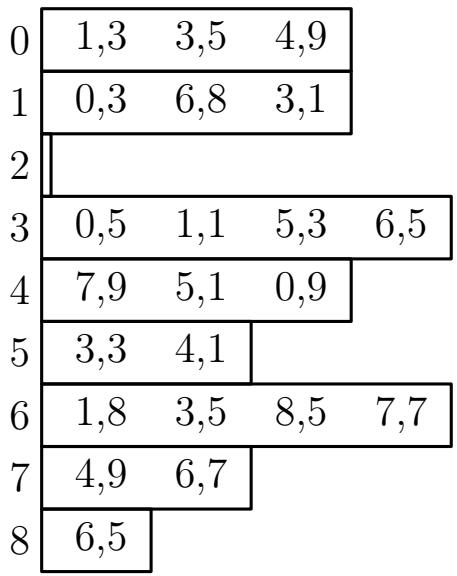
	0	1	2	3	4	5	6	7	8
0	∞	3	∞	∞	9	∞	∞	∞	∞
1	∞	∞	∞	1	∞	∞	8	∞	∞
2	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	5	∞	∞	∞	∞	3	5	∞	∞
4	∞	∞	∞	∞	∞	1	∞	9	∞
5	∞	∞	∞	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	∞	∞	7	∞	2	∞
7	∞	∞	∞	∞	∞	∞	∞	∞	1
8	∞	∞	∞	∞	∞	∞	5	∞	∞

Списки смежных рёбер

```cpp
#include <iostream>
#include <vector>
using namespace std;

struct Edge {int v; int w;};

int main () {
    int n, m;
    cin >> n >> m;
    vector <vector <Edge> > adj (n);

    for (int j = 0; j < m; j++) {
        int u, v, w;
        cin >> u >> v >> w;
        u -= 1;
        v -= 1;
        adj[u].push_back ({v, w});
        adj[v].push_back ({u, w});
    }
    return 0;
}
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

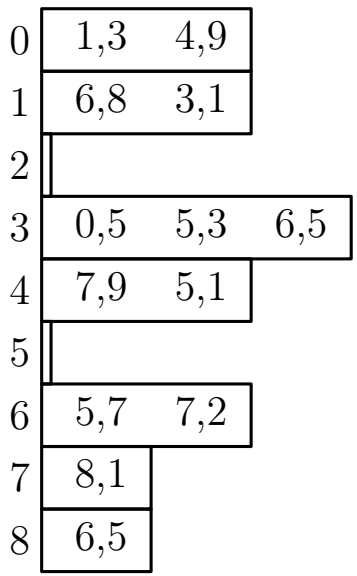


Списки смежных рёбер

```cpp
#include <iostream>
#include <vector>
using namespace std;

struct Edge {int v; int w;};

int main () {
    int n, m;
    cin >> n >> m;
    vector <vector <Edge> > adj (n);

    for (int j = 0; j < m; j++) {
        int u, v, w;
        cin >> u >> v >> w;
        u -= 1;
        v -= 1;
        adj[u].push_back ({v, w});

    }
    return 0;
}
```

Ввод:

```
9 13
1 2 3
2 7 8
2 4 1
4 1 5
7 6 7
4 6 3
4 7 5
5 8 9
5 6 1
7 8 2
9 7 5
1 5 9
8 9 1
```



Содержание

Задача

- Дан взвешенный граф (или орграф).
- Длина пути — сумма длин его рёбер (или дуг).
- Задача 1: найдите кратчайший путь от вершины u до вершины v.

Задача

- Дан взвешенный граф (или орграф).
- Длина пути — сумма длин его рёбер (или дуг).
- Задача 1: найдите кратчайший путь от вершины u до вершины v.
- Задача 2: найдите кратчайшие пути от вершины u до всех остальных вершин.

Задача

- Дан взвешенный граф (или орграф).
- Длина пути — сумма длин его рёбер (или дуг).
- Задача 1: найдите кратчайший путь от вершины u до вершины v.
- Задача 2: найдите кратчайшие пути от вершины u до всех остальных вершин.
- Задача 3: найдите кратчайшие пути между всеми парами вершин.

Содержание

Алгоритм Флойда–Уоршолла: идея

- Рассмотрим матрицу A, в которой клетка (u, v) содержит длину кратчайшего пути из u в v.
- Начнём с матрицы смежности и будем улучшать её.
- Будем рассматривать вершины по порядку.

Алгоритм Флойда–Уоршолла: идея

- Рассмотрим матрицу A, в которой клетка (u, v) содержит длину кратчайшего пути из u в v.
- Начнём с матрицы смежности и будем улучшать её.
- Будем рассматривать вершины по порядку.
- Для каждой вершины $k = 1, 2, \ldots, n$:
  - Рассмотрим каждую пару вершин (i, j).
  - Длина пути из i в j не больше $A_{i,j}$.

Алгоритм Флойда–Уоршолла: идея

- Рассмотрим матрицу A, в которой клетка (u, v) содержит длину кратчайшего пути из u в v.
- Начнём с матрицы смежности и будем улучшать её.
- Будем рассматривать вершины по порядку.
- Для каждой вершины $k = 1, 2, \ldots, n$:
  - Рассмотрим каждую пару вершин (i, j).
  - Длина пути из i в j не больше $A_{i,j}$.
  - Попробуем пройти из i в j через k: $A_{i,k} + A_{k,j}$.
  - Если этот путь короче, запишем $A_{i,j} := A_{i,k} + A_{k,j}$.

Алгоритм Флойда–Уоршолла: идея

- Рассмотрим матрицу A, в которой клетка (u, v) содержит длину кратчайшего пути из u в v.
- Начнём с матрицы смежности и будем улучшать её.
- Будем рассматривать вершины по порядку.
- Для каждой вершины $k = 1, 2, \ldots, n$:
  - Рассмотрим каждую пару вершин (i, j).
  - Длина пути из i в j не больше $A_{i,j}$.
  - Попробуем пройти из i в j через k: $A_{i,k} + A_{k,j}$.
  - Если этот путь короче, запишем $A_{i,j} := A_{i,k} + A_{k,j}$.
- После этого матрица A построена.

Алгоритм Флойда–Уоршолла: код

```
1       for (int i = 0; i < n; i++)
2           a[i][i] = 0;
3
4
5
6
7
8       for (int k = 0; k < n; k++)
9           for (int i = 0; i < n; i++)
10              for (int j = 0; j < n; j++)
11                  if (a[i][j] > a[i][k] + a[k][j])
12                      a[i][j] = a[i][k] + a[k][j];
13
14
15
16      int x = 0, y = n - 1;
17      cout << a[x][y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Алгоритм Флойда–Уоршолла: код

```
 1       vector <vector <int> > p (n, vector <int> (n));
 2       for (int i = 0; i < n; i++) {
 3           a[i][i] = 0;
 4           for (int j = 0; j < n; j++)
 5               p[i][j] = j;
 6       }
 7
 8       for (int k = 0; k < n; k++)
 9           for (int i = 0; i < n; i++)
10               for (int j = 0; j < n; j++)
11                   if (a[i][j] > a[i][k] + a[k][j]) {
12                       a[i][j] = a[i][k] + a[k][j];
13                       p[i][j] = p[i][k];
14                   }
15
16       int x = 0, y = n - 1;
17       cout << a[x][y] << endl;
18       while (x != y) {
19           cout << x + 1 << " ";
20           x = p[x][y];
21       }
22       cout << x + 1 << endl;
```

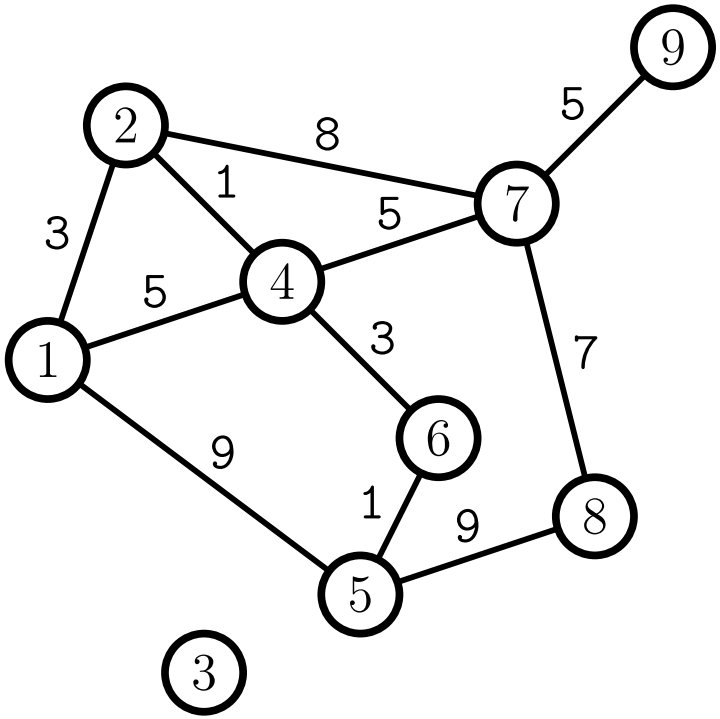
Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
1 2 4 7 9
```

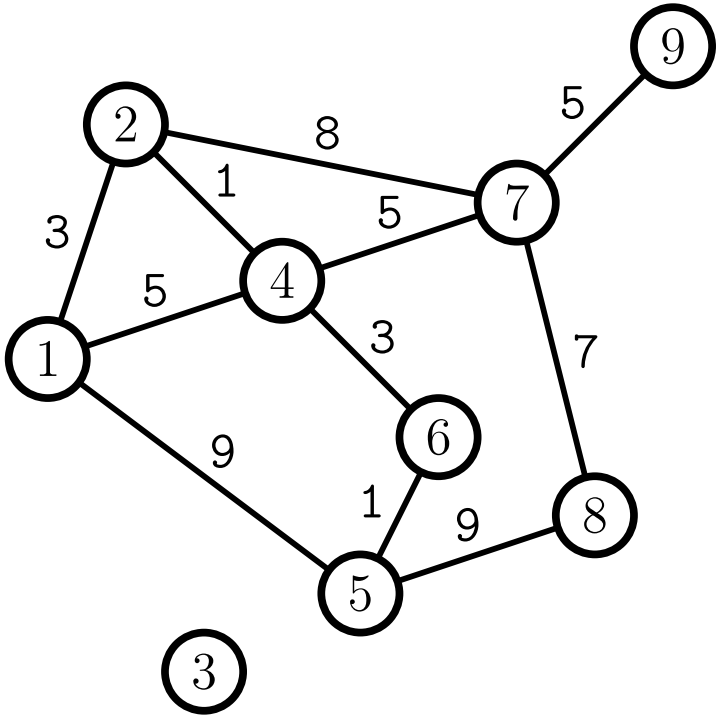
Алгоритм Флойда–Уоршолла: пример



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

$k = 0$

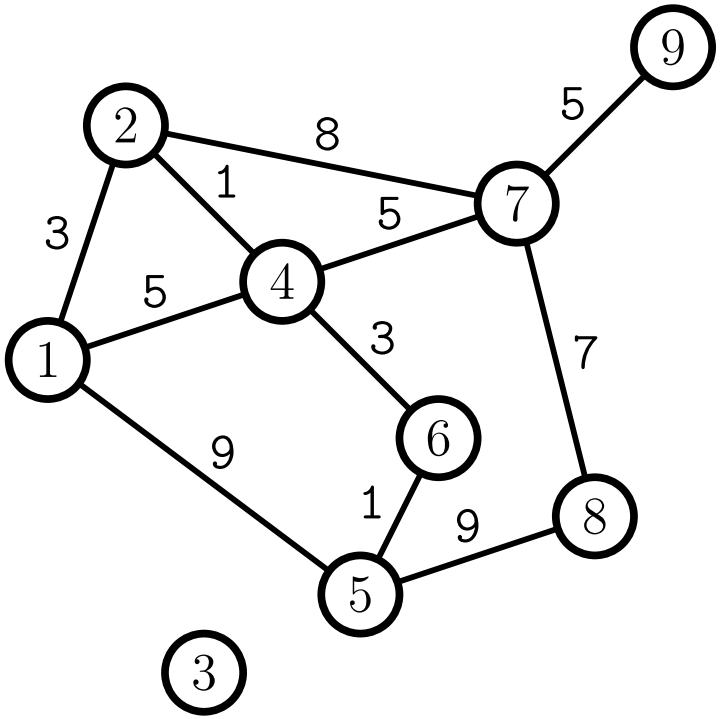
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	5	9	∞	∞	∞	∞
2	3	0	∞	1	12	∞	8	∞	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	5	1	∞	0	14	3	5	∞	∞
5	9	12	∞	14	0	1	∞	9	∞
6	∞	∞	∞	3	1	0	∞	∞	∞
7	∞	8	∞	5	∞	∞	0	7	5
8	∞	∞	∞	∞	9	∞	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$k = 1$

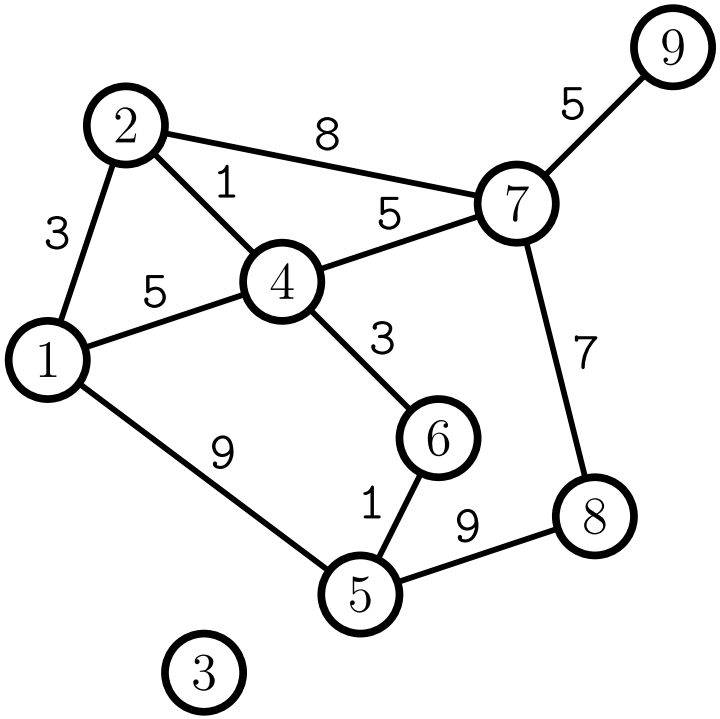
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	9	∞	11	∞	∞
2	3	0	∞	1	12	∞	8	∞	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	13	3	5	∞	∞
5	9	12	∞	13	0	1	20	9	∞
6	∞	∞	∞	3	1	0	∞	∞	∞
7	11	8	∞	5	20	∞	0	7	5
8	∞	∞	∞	∞	9	∞	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$$k = 2$$

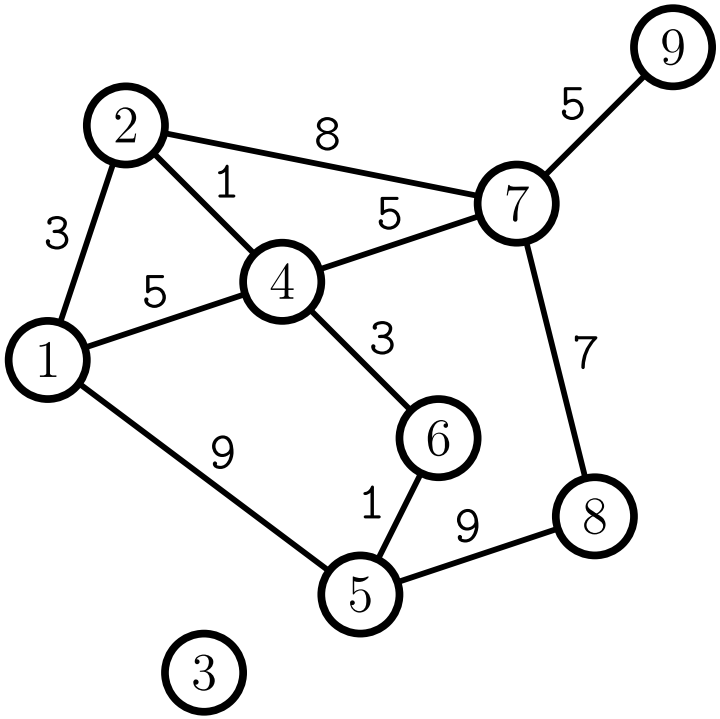
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	9	∞	11	∞	∞
2	3	0	∞	1	12	∞	8	∞	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	13	3	5	∞	∞
5	9	12	∞	13	0	1	20	9	∞
6	∞	∞	∞	3	1	0	∞	∞	∞
7	11	8	∞	5	20	∞	0	7	5
8	∞	∞	∞	∞	9	∞	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$k = 3$

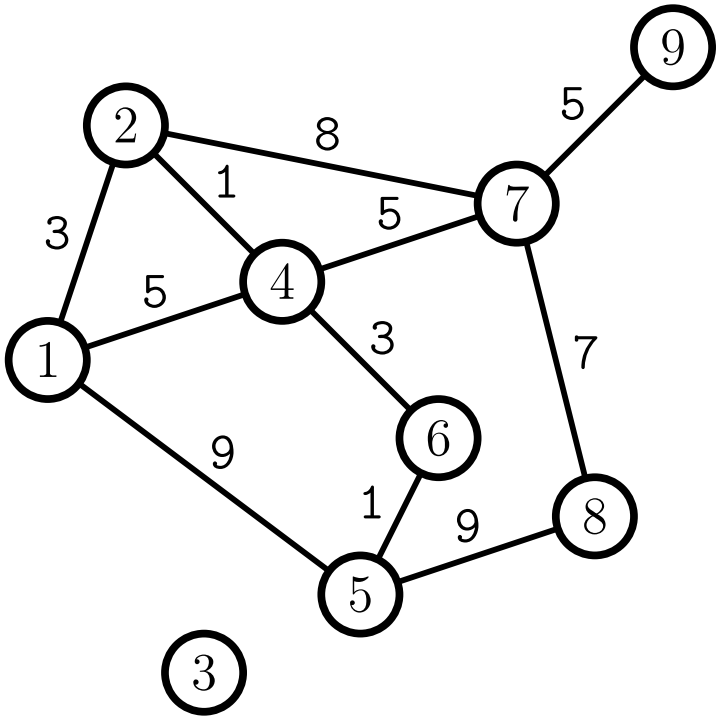
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	9	7	9	∞	∞
2	3	0	∞	1	12	4	6	∞	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	13	3	5	∞	∞
5	9	12	∞	13	0	1	18	9	∞
6	7	4	∞	3	1	0	8	∞	∞
7	9	6	∞	5	18	8	0	7	5
8	∞	∞	∞	∞	9	∞	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$k = 4$

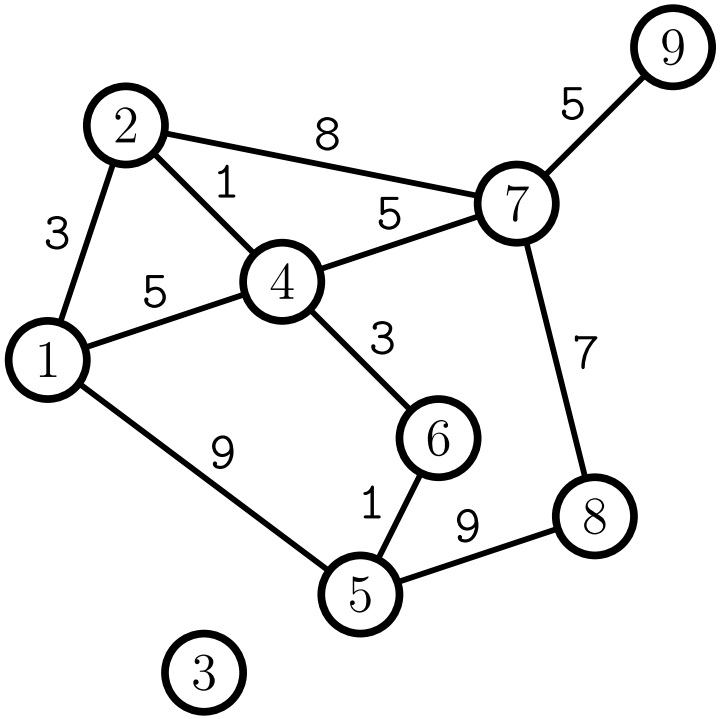
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	9	7	9	18	∞
2	3	0	∞	1	12	4	6	21	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	13	3	5	22	∞
5	9	12	∞	13	0	1	18	9	∞
6	7	4	∞	3	1	0	8	10	∞
7	9	6	∞	5	18	8	0	7	5
8	18	21	∞	22	9	10	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$k = 5$

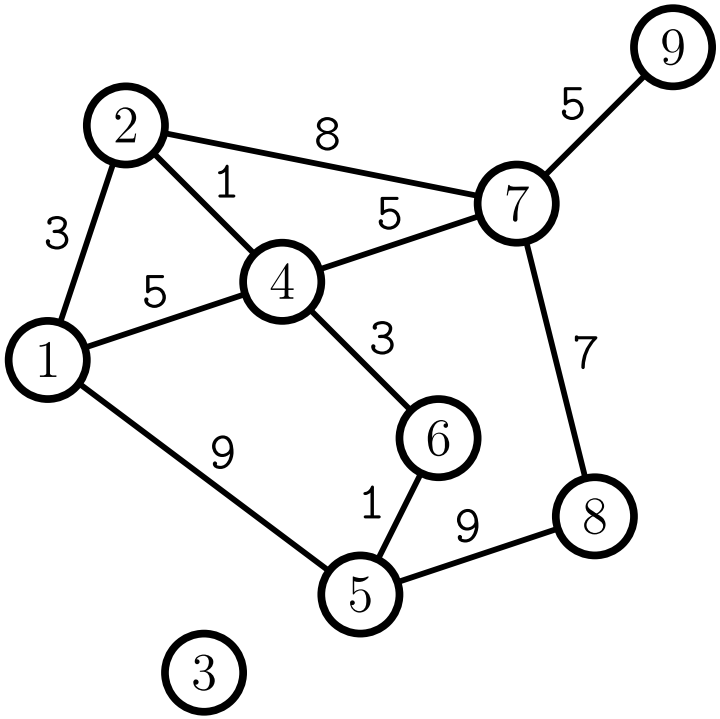
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	8	7	9	17	∞
2	3	0	∞	1	5	4	6	14	∞
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	4	3	5	13	∞
5	8	5	∞	4	0	1	9	9	∞
6	7	4	∞	3	1	0	8	10	∞
7	9	6	∞	5	9	8	0	7	5
8	17	14	∞	13	9	10	7	0	∞
9	∞	∞	∞	∞	∞	∞	5	∞	0

$k = 6$

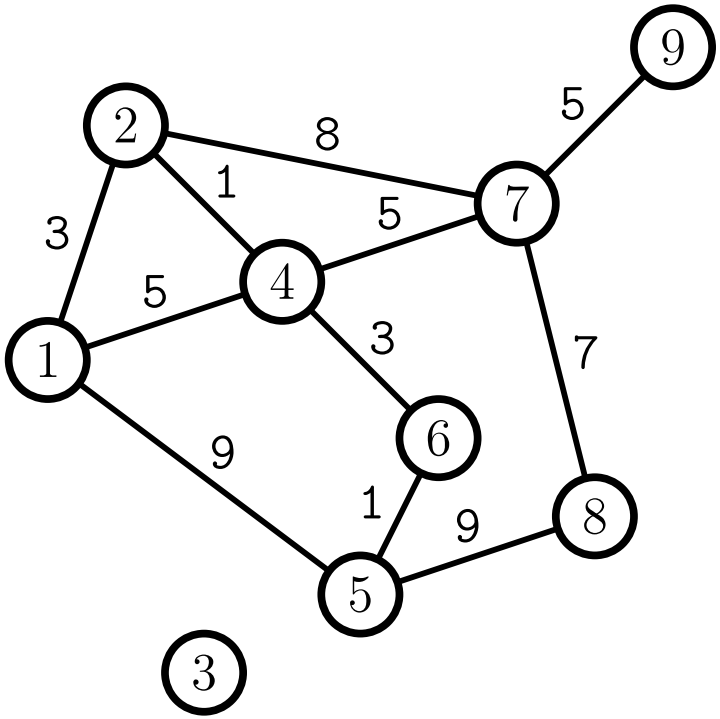
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	8	7	9	16	14
2	3	0	∞	1	5	4	6	13	11
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	4	3	5	12	10
5	8	5	∞	4	0	1	9	9	14
6	7	4	∞	3	1	0	8	10	13
7	9	6	∞	5	9	8	0	7	5
8	16	13	∞	12	9	10	7	0	12
9	14	11	∞	10	14	13	5	12	0

$k = 7$

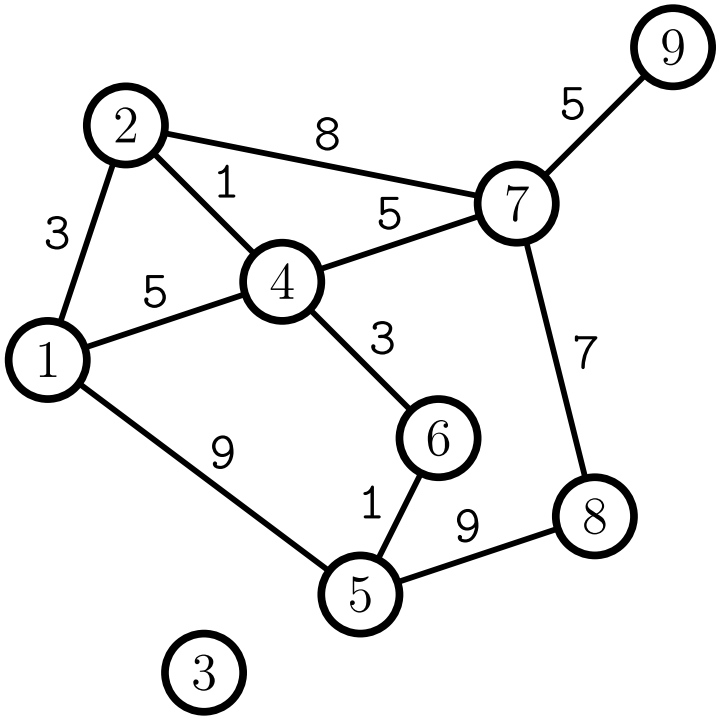
Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	8	7	9	16	14
2	3	0	∞	1	5	4	6	13	11
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	4	3	5	12	10
5	8	5	∞	4	0	1	9	9	14
6	7	4	∞	3	1	0	8	10	13
7	9	6	∞	5	9	8	0	7	5
8	16	13	∞	12	9	10	7	0	12
9	14	11	∞	10	14	13	5	12	0

$k = 8$

Алгоритм Флойда–Уоршолла: пример



	1	2	3	4	5	6	7	8	9
1	0	3	∞	4	8	7	9	16	14
2	3	0	∞	1	5	4	6	13	11
3	∞	∞	0	∞	∞	∞	∞	∞	∞
4	4	1	∞	0	4	3	5	12	10
5	8	5	∞	4	0	1	9	9	14
6	7	4	∞	3	1	0	8	10	13
7	9	6	∞	5	9	8	0	7	5
8	16	13	∞	12	9	10	7	0	12
9	14	11	∞	10	14	13	5	12	0

$$k = 9$$

Алгоритм Флойда–Уоршолла: доказательство

- Как доказать, что в получившейся матрице клетка (u, v) содержит длину кратчайшего пути из u в v?

Алгоритм Флойда–Уоршолла: доказательство

- Как доказать, что в получившейся матрице клетка (u, v) содержит длину кратчайшего пути из u в v?
- Индукция по k:
- После k итераций клетка (u, v) содержит кратчайший путь из тех, в которых промежуточные вершины имеют номера от 1 до k.

Алгоритм Флойда–Уоршолла: доказательство

- Как доказать, что в получившейся матрице клетка (u, v) содержит длину кратчайшего пути из u в v?
- Индукция по k:
- После k итераций клетка (u, v) содержит кратчайший путь из тех, в которых промежуточные вершины имеют номера от 1 до k.
  - База: $k = 0$, в матрице ребро, промежуточных вершин нет.
  - Предположение: после $k - 1$ итерации утверждение верно.
  - Переход: кратчайший путь с вершинами $1, 2, \ldots, k$ содержит k не более одного раза.
  - Если не содержит k, он уже найден.
  - Если содержит k, он найдётся как $A_{u,k} + A_{k,v}$, ведь эти части пути состоят из вершин с номерами строго меньше k.

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память:
- Время:
- Находит кратчайшие пути между всеми парами вершин.

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время:
- Находит кратчайшие пути между всеми парами вершин.

Алгоритм Флойда–Уоршолла: анализ

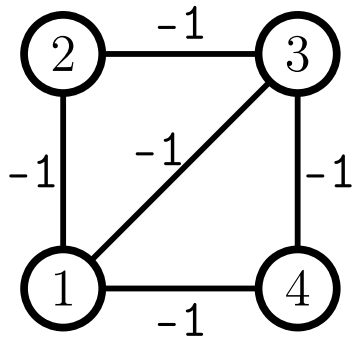
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:

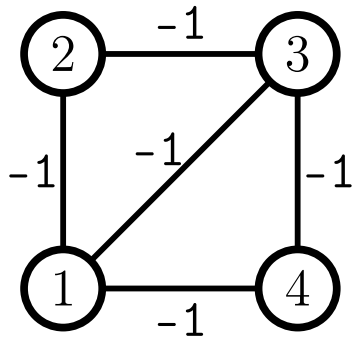


	1	2	3	4
1	0	-1	-1	-1
2	-1	0	-1	∞
3	-1	-1	0	-1
4	-1	∞	-1	0

$k = 0$

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:

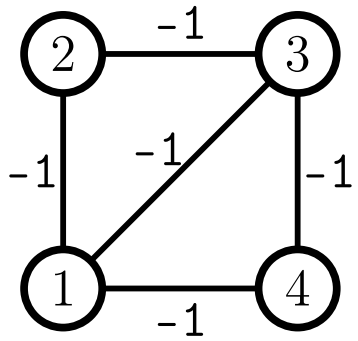


	1	2	3	4
1	0	-1	-1	-1
2	-1	-2	-2	-2
3	-1	-2	-2	-2
4	-1	-2	-2	-2

$k = 1$

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:

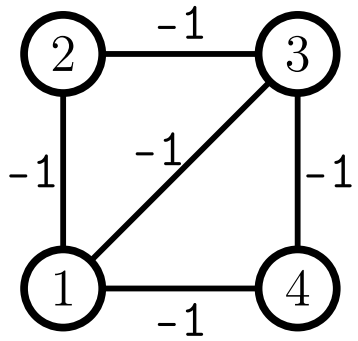


	1	2	3	4
1	-2	-3	-5	-5
2	-3	-4	-6	-6
3	-5	-6	-12	-12
4	-5	-6	-12	-12

$k = 2$

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:

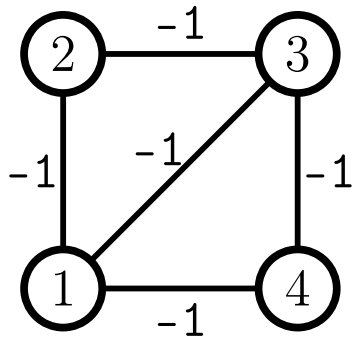


	1	2	3	4
1	-10	-11	-17	-29
2	-11	-12	-18	-30
3	-17	-18	-24	-36
4	-29	-30	-36	-72

$k = 3$

Алгоритм Флойда–Уоршолла: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n^2)$ (можно переиспользовать матрицу смежности).
- Время: $\Theta(n^3)$.
- Находит кратчайшие пути между всеми парами вершин.
- Не работает, если в графе есть циклы отрицательной длины:



	1	2	3	4
1	-58	-59	-65	-101
2	-59	-60	-66	-102
3	-65	-66	-72	-108
4	-101	-102	-108	-144

$k = 4$

Содержание

Алгоритм Беллмана–Форда: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от $start$ до v.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.

Алгоритм Беллмана–Форда: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от *start* до v.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.
- Сделаем $n - 1$ одинаковых шагов:
  - Рассмотрим каждое ребро (дугу) (u, v) веса w.
  - Релаксируем его: $d(v) := \min(d(v), d(u) + w)$.

Алгоритм Беллмана–Форда: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от *start* до v.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.
- Сделаем $n - 1$ одинаковых шагов:
  - Рассмотрим каждое ребро (дугу) (u, v) веса w.
  - Релаксируем его: $d(v) := \min(d(v), d(u) + w)$.
- После этого в d посчитаны кратчайшие расстояния.

Алгоритм Беллмана–Форда: код

```
1       vector <int> d (n, infinity);
2
3       d[0] = 0;
4       for (int k = 0; k < n - 1; k++) {
5           for (int u = 0; u < n; u++)
6               for (auto e : adj[u])
7                   if (d[e.v] > d[u] + e.w)
8                       d[e.v] = d[u] + e.w;
9       }
10
11
12
13      int y = n - 1;
14      cout << d[y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Алгоритм Беллмана–Форда: код

```
 1       vector <int> d (n, infinity);
 2       vector <int> p (n, infinity);
 3       d[0] = 0;
 4       for (int k = 0; k < n - 1; k++) {
 5           for (int u = 0; u < n; u++)
 6               for (auto e : adj[u])
 7                   if (d[e.v] > d[u] + e.w) {
 8                       d[e.v] = d[u] + e.w;
 9                       p[e.v] = u;
10                   }
11       }
12
13       int x = 0, y = n - 1;
14       cout << d[y] << endl;
15       vector <int> path;
16       while (x != y) {
17           y = p[y];
18           path.push_back (y + 1);
19       }
20       copy (path.rbegin (), path.rend (),
21           ostream_iterator <int> (cout, " "));
22       cout << n << endl;
```

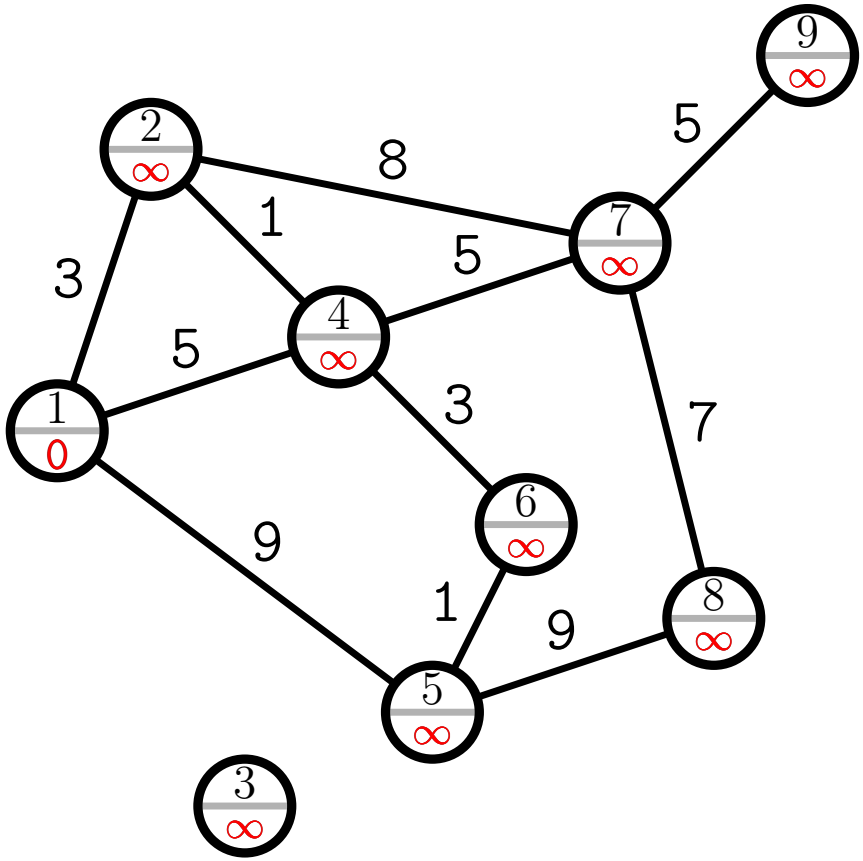
Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

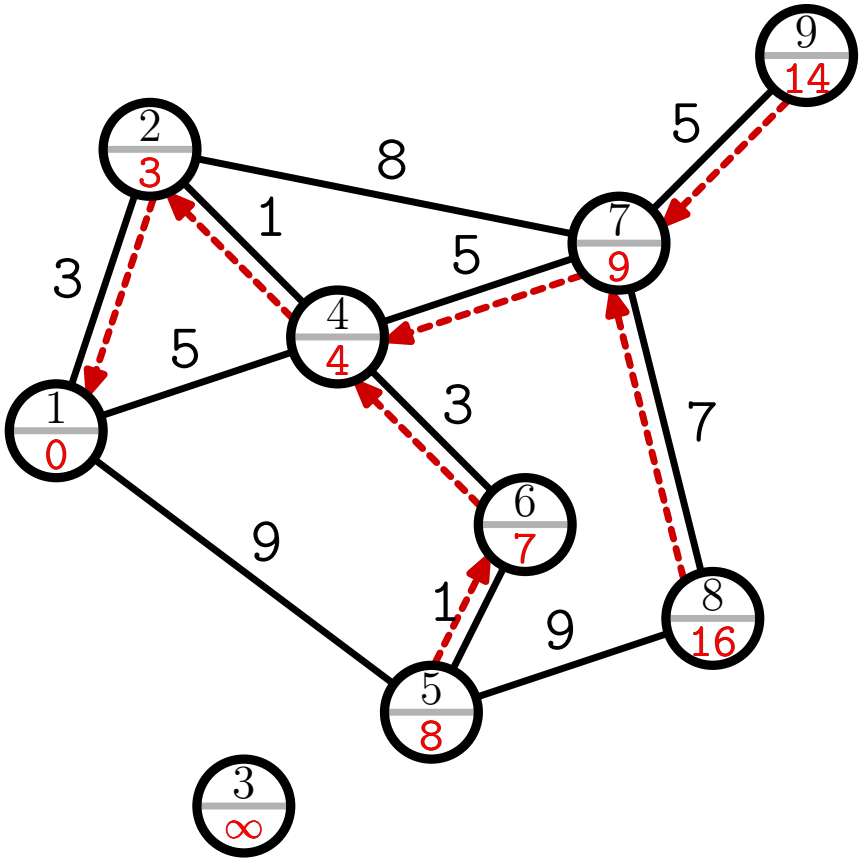
```
14
1 2 4 7 9
```

Алгоритм Беллмана–Форда: пример 1



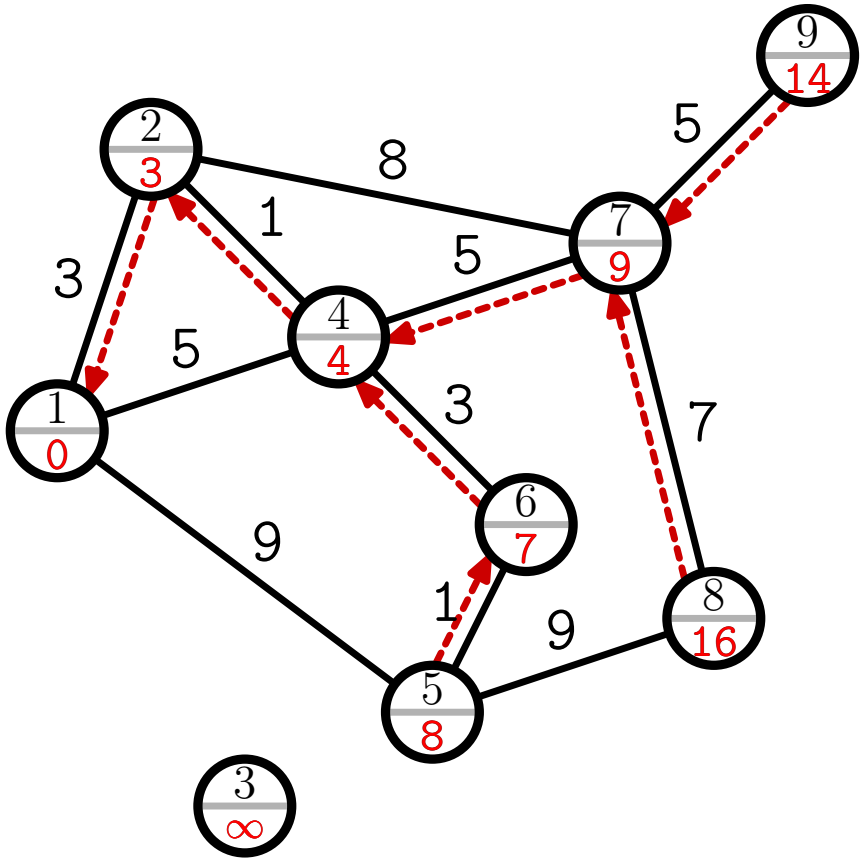
$k = 0$

Алгоритм Беллмана–Форда: пример 1



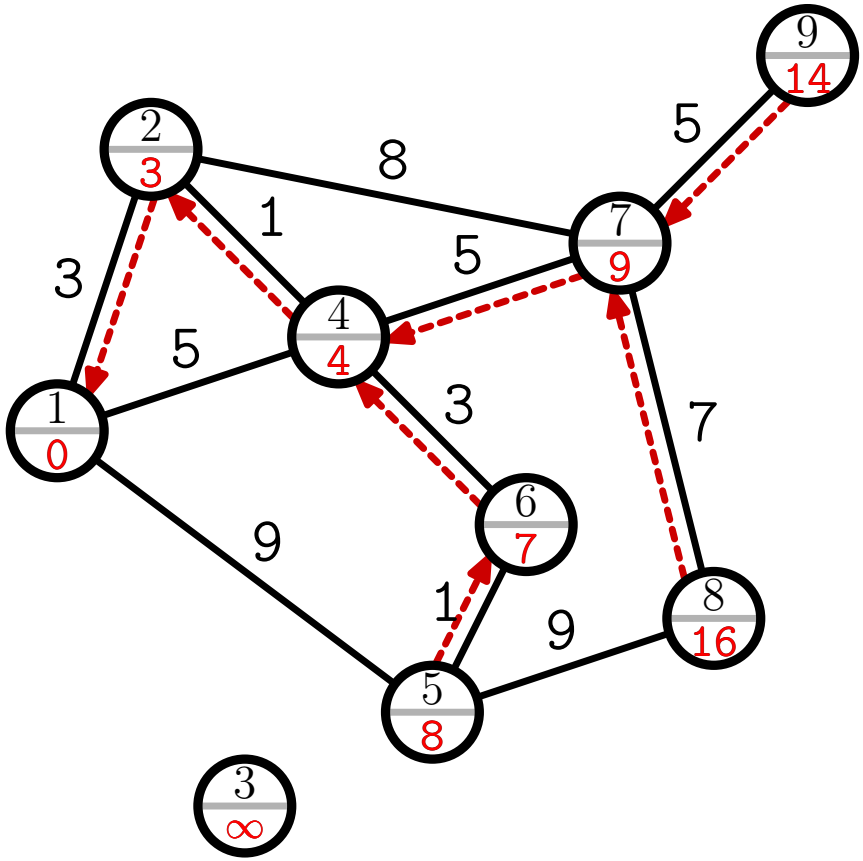
$k = 1$

Алгоритм Беллмана–Форда: пример 1



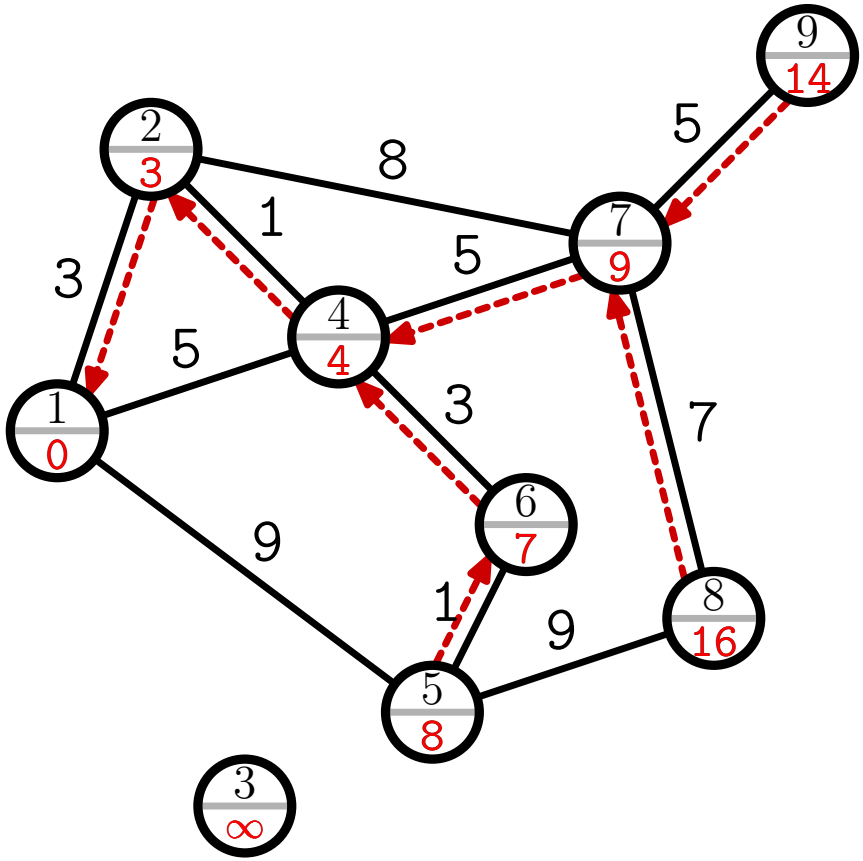
$k = 2$

Алгоритм Беллмана–Форда: пример 1



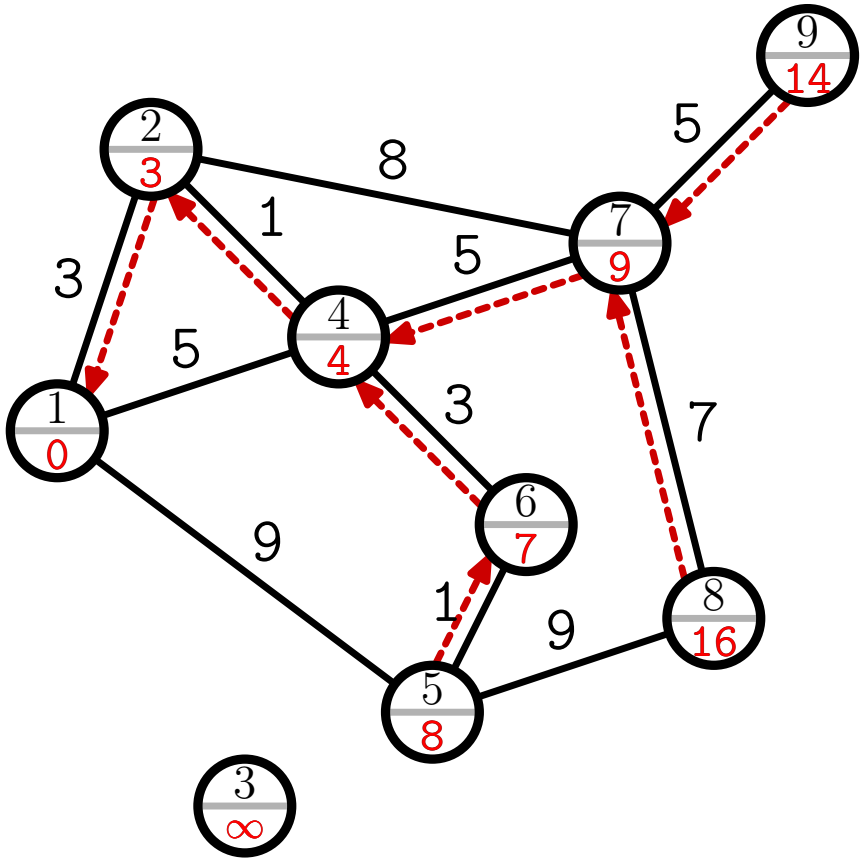
$k = 3$

Алгоритм Беллмана–Форда: пример 1



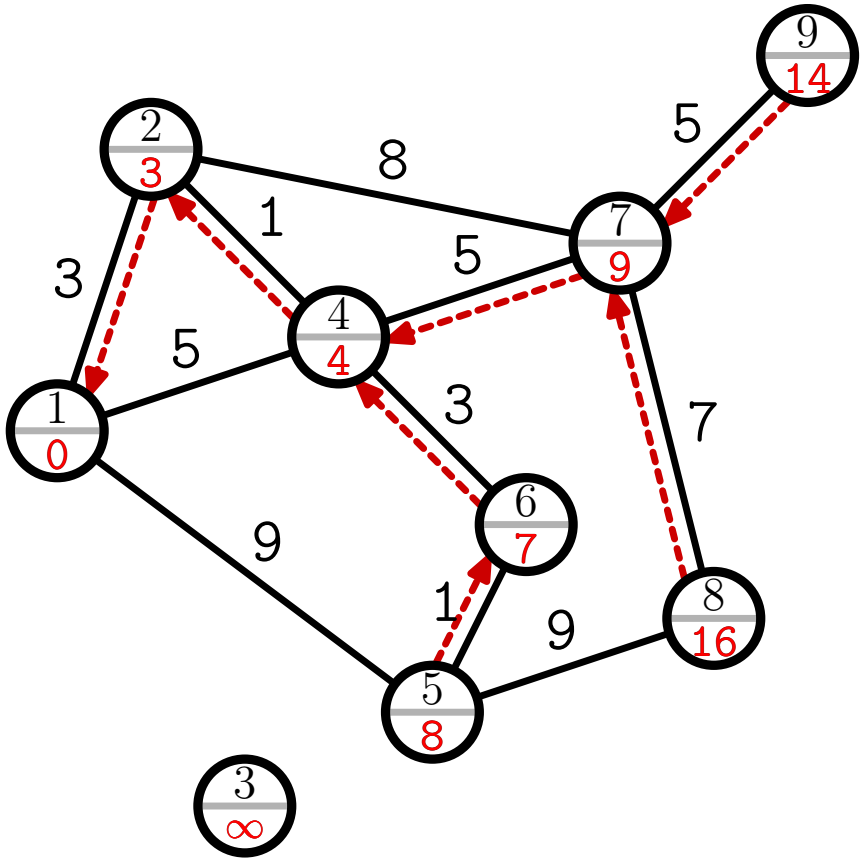
$k = 4$

Алгоритм Беллмана–Форда: пример 1



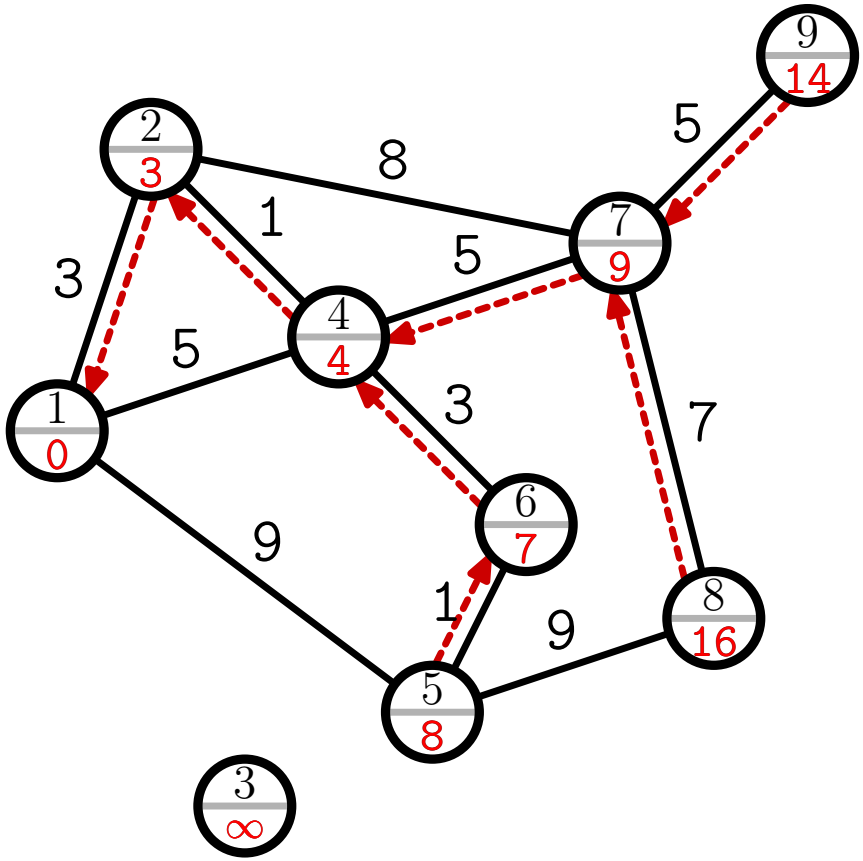
$k = 5$

Алгоритм Беллмана–Форда: пример 1



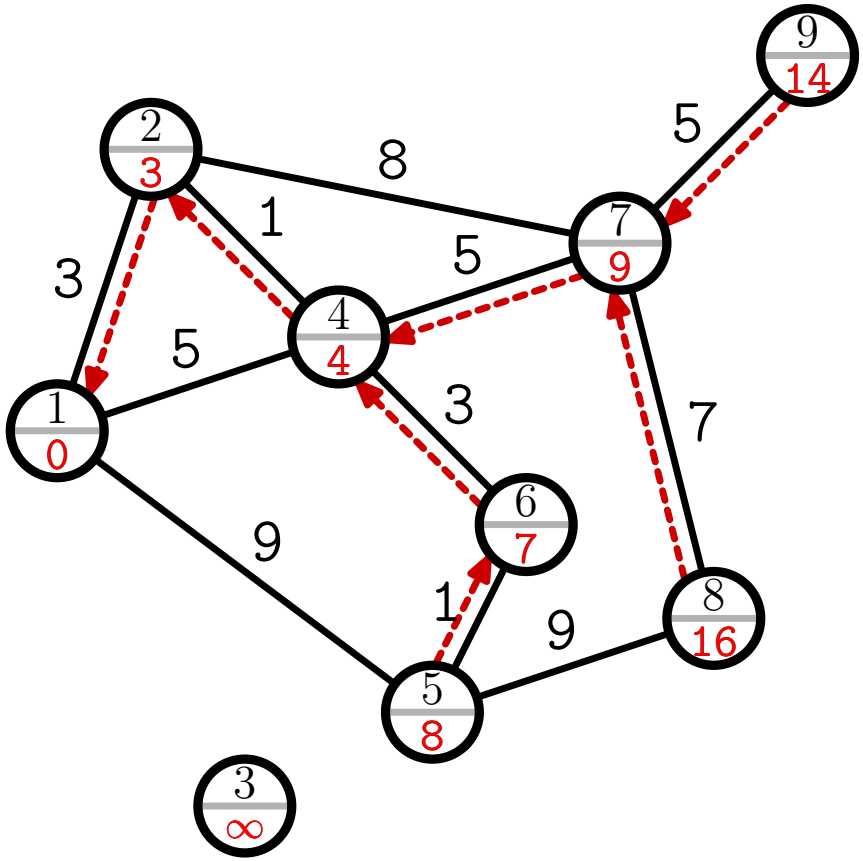
$k = 6$

Алгоритм Беллмана–Форда: пример 1



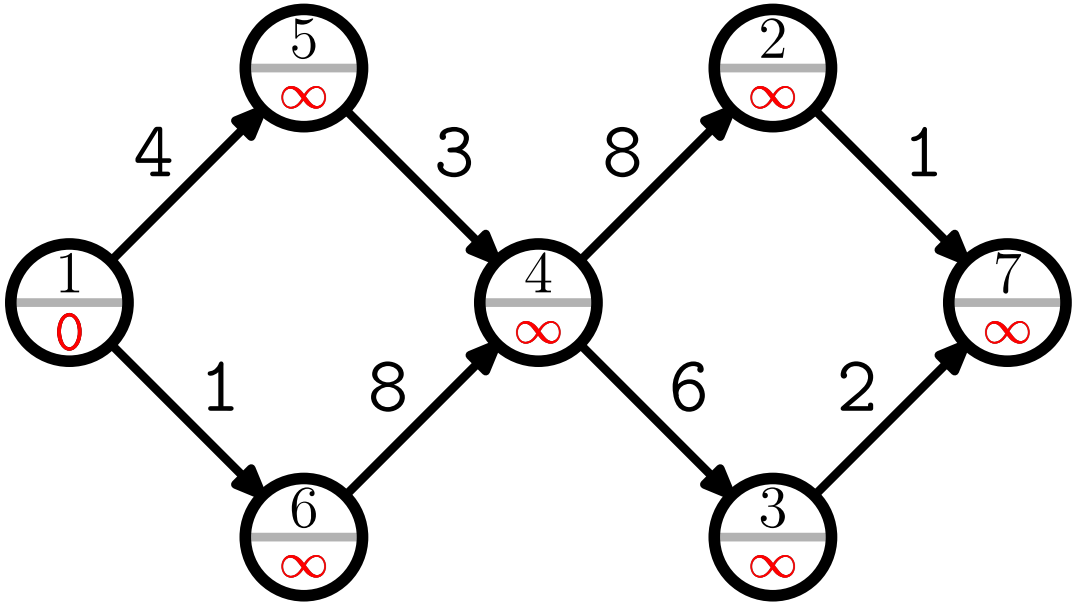
$k = 7$

Алгоритм Беллмана–Форда: пример 1



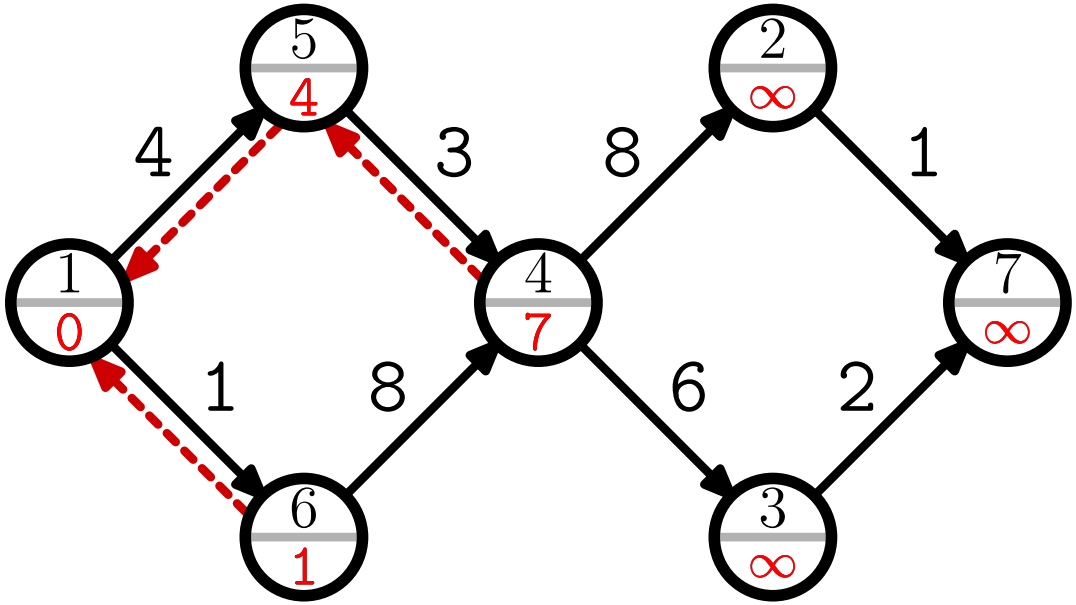
$k = 8$

Алгоритм Беллмана–Форда: пример 2



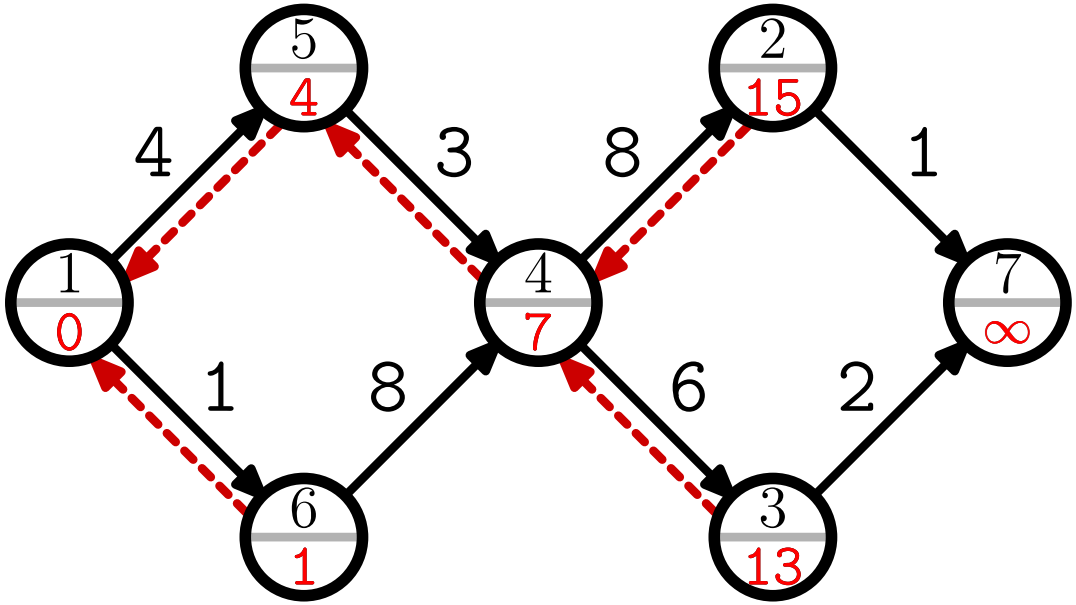
$k = 0$

Алгоритм Беллмана–Форда: пример 2



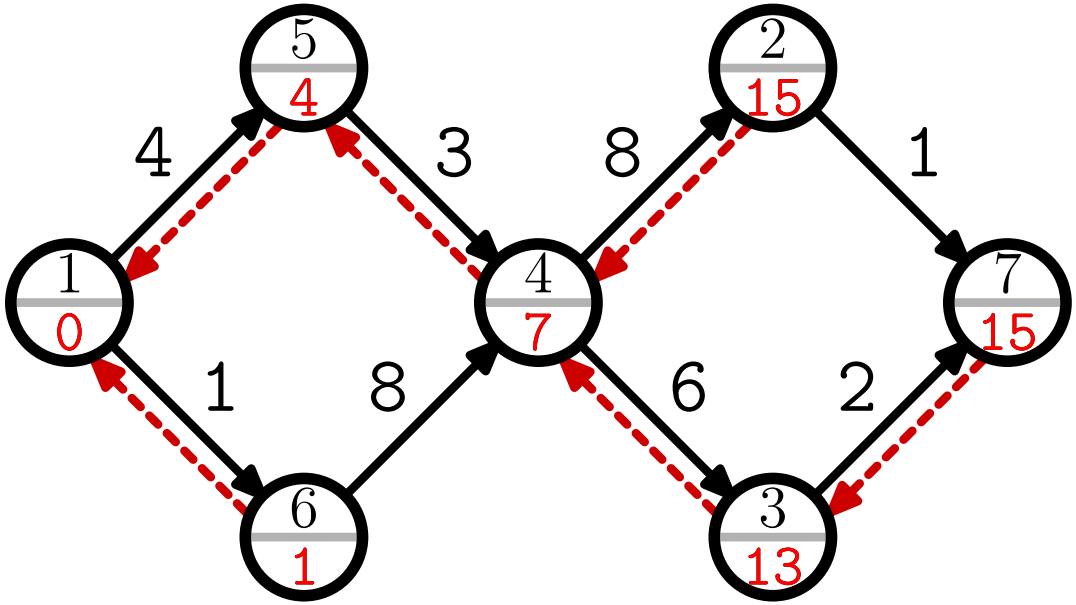
$k = 1$

Алгоритм Беллмана–Форда: пример 2



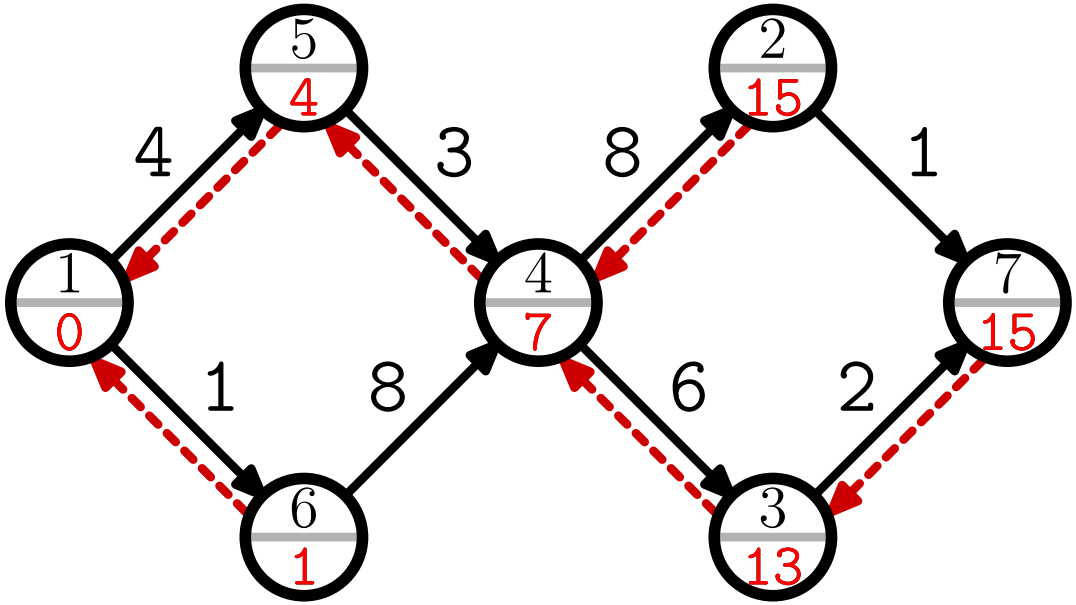
$k = 2$

Алгоритм Беллмана–Форда: пример 2



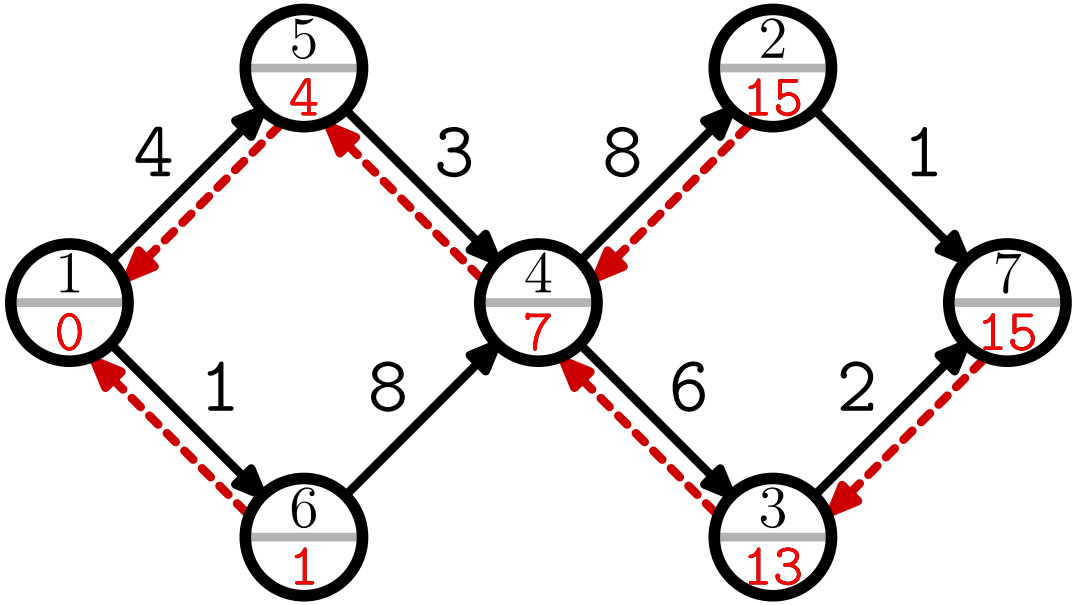
$k = 3$

Алгоритм Беллмана–Форда: пример 2



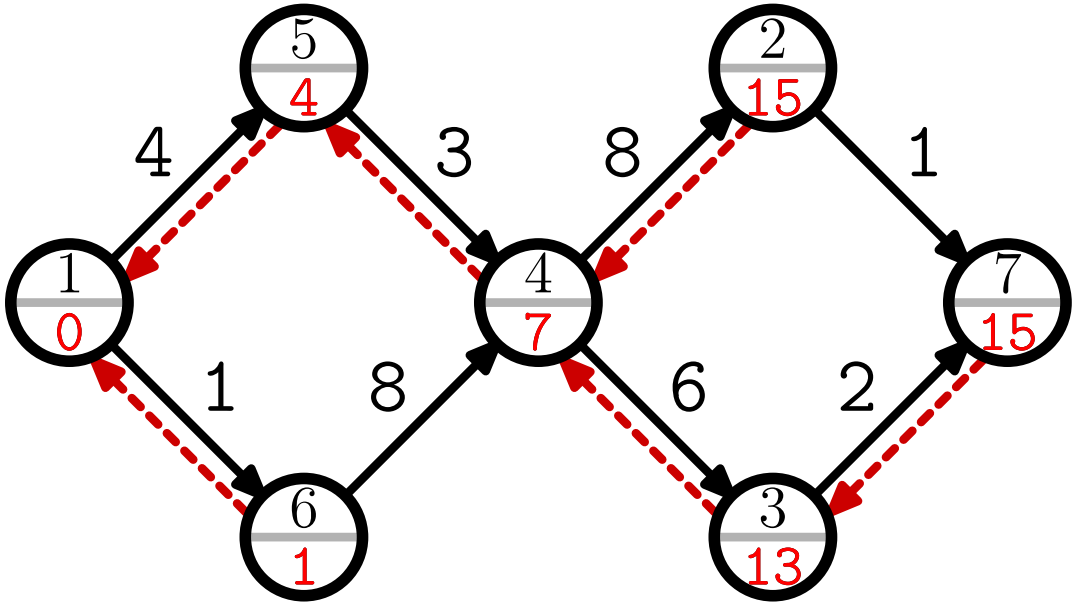
$k = 4$

Алгоритм Беллмана–Форда: пример 2



$k = 5$

Алгоритм Беллмана–Форда: пример 2



$k = 6$

Алгоритм Беллмана–Форда: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?

Алгоритм Беллмана–Форда: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Индукция по k:
- После k итераций найдены все кратчайшие пути, состоящие не более чем из k рёбер.

Алгоритм Беллмана–Форда: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Индукция по k:
- После k итераций найдены все кратчайшие пути, состоящие не более чем из k рёбер.
  - База: $k = 0$, найден только пустой путь.
  - Предположение: после $k - 1$ итерации утверждение верно.
  - Переход: пусть кратчайший путь до какой-то вершины v состоит из k рёбер.
  - Тогда этот же путь, но до предпоследней вершины $p(v)$, тоже кратчайший, и состоит из $k - 1$ ребра.
  - Значит, путь до $p(v)$ найден на предыдущем шаге.
  - И при релаксации ребра $(p(v), v)$ мы найдём путь до v.

Алгоритм Беллмана–Форда: анализ

- Сколько времени и памяти требуется алгоритму?
- Память:
- Время:
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Беллмана–Форда: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время:
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Беллмана–Форда: анализ

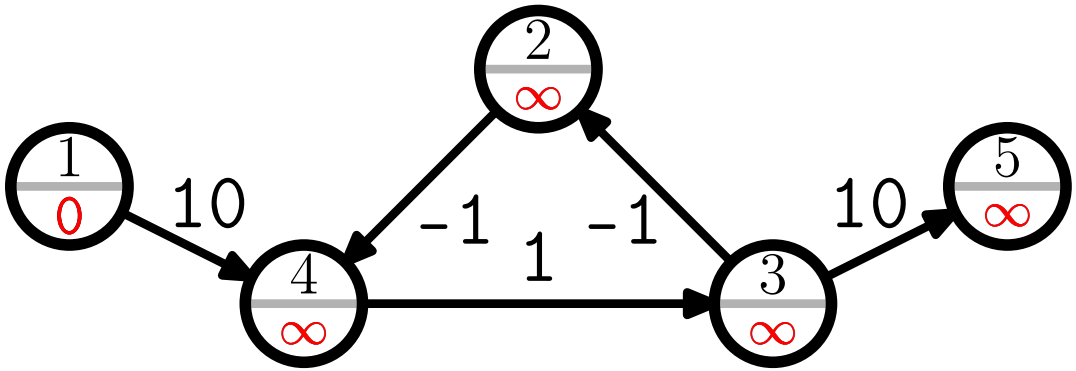
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Беллмана–Форда: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...

Алгоритм Беллмана–Форда: анализ

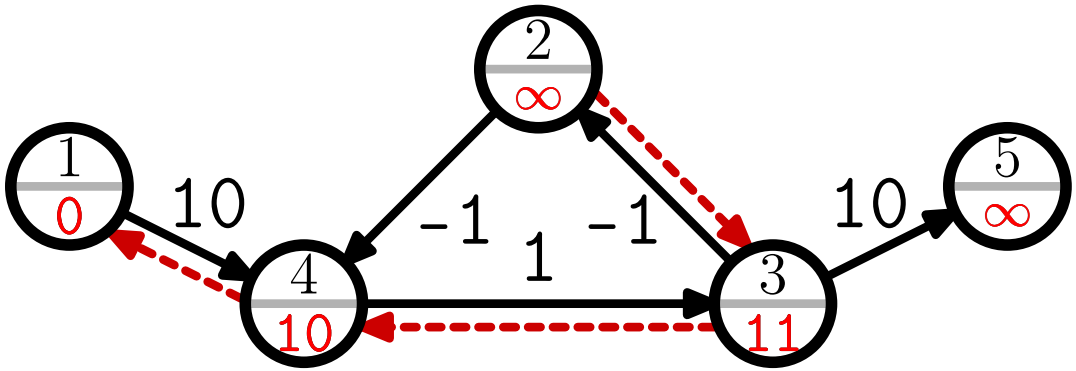
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 0$

Алгоритм Беллмана–Форда: анализ

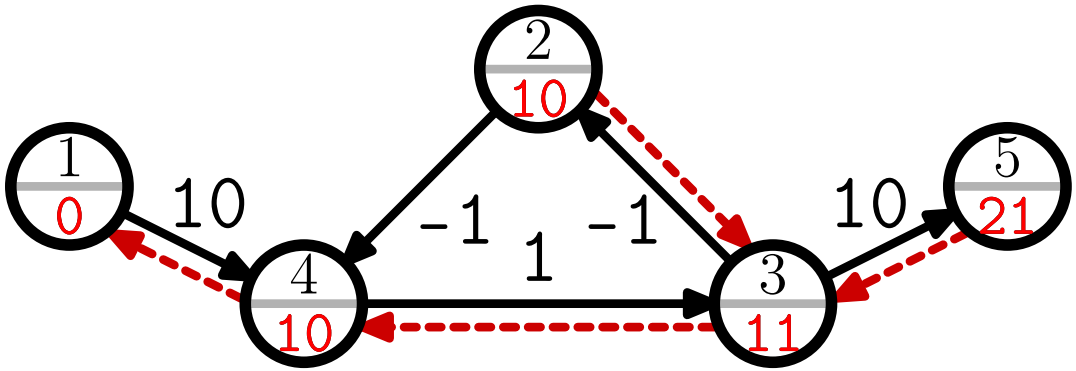
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 1$

Алгоритм Беллмана–Форда: анализ

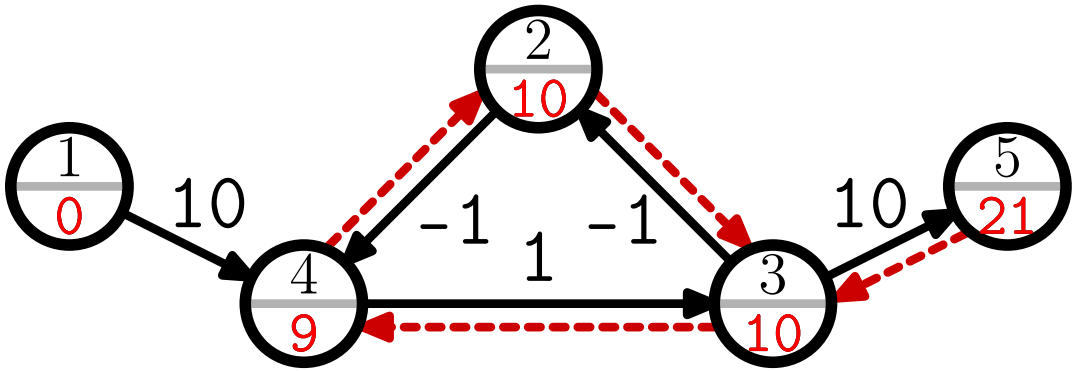
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 2$

Алгоритм Беллмана–Форда: анализ

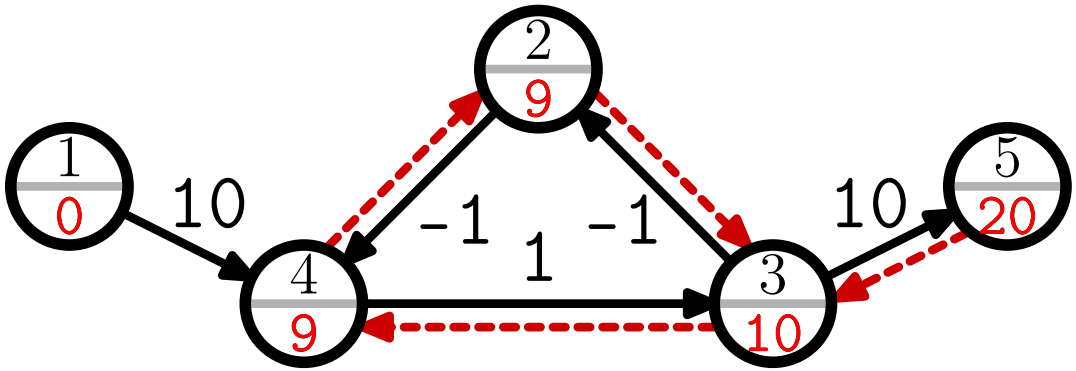
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 3$

Алгоритм Беллмана–Форда: анализ

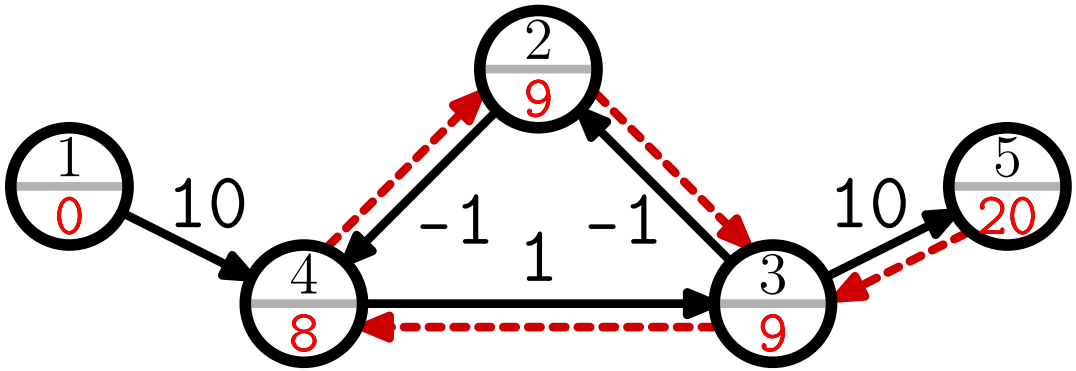
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 4$

Алгоритм Беллмана–Форда: анализ

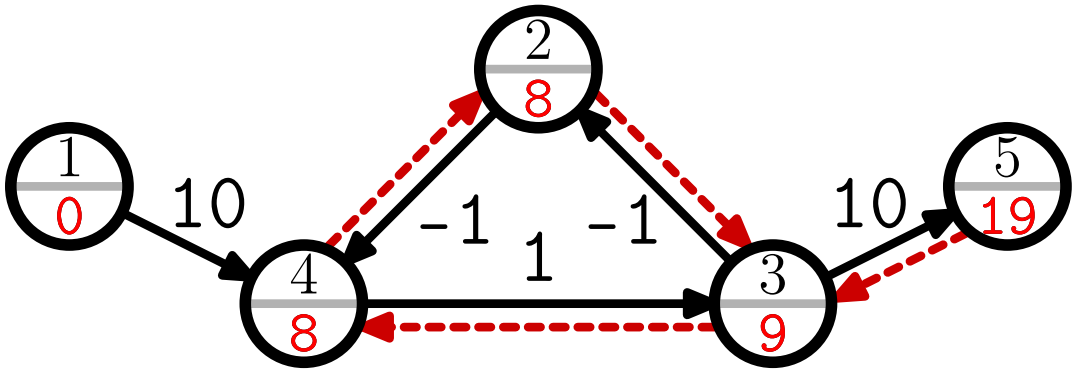
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 5$

Алгоритм Беллмана–Форда: анализ

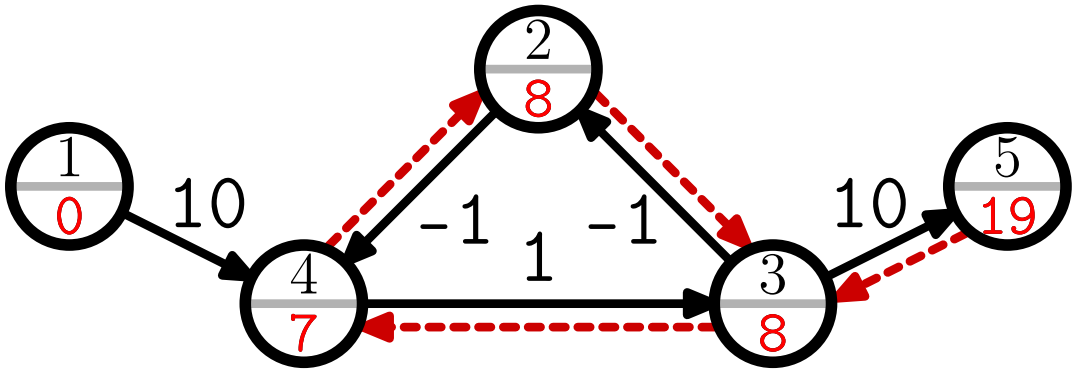
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 6$

Алгоритм Беллмана–Форда: анализ

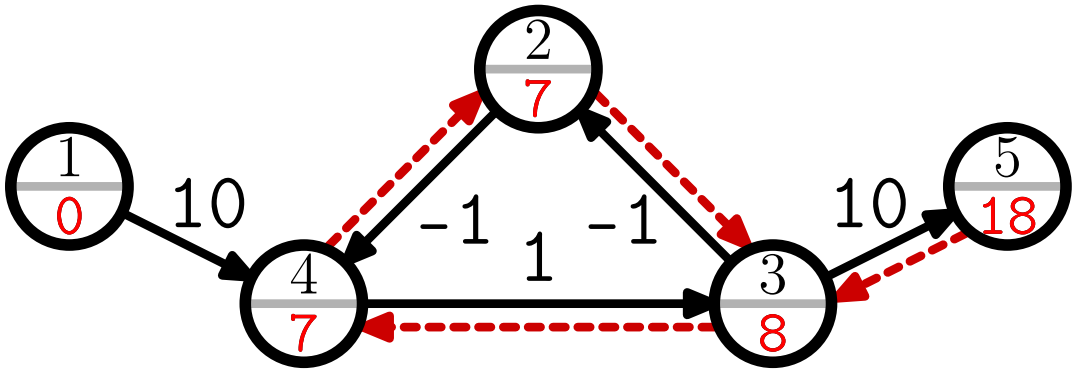
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 7$

Алгоритм Беллмана–Форда: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...



$k = 8$

Алгоритм Беллмана–Форда: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $\Theta(mn)$.
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть циклы отрицательной длины...
- Но помогает их обнаружить, если сделать ещё итерацию!

Содержание

Алгоритм Дейкстры: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от $start$ до v.
- Также будем строить дерево кратчайших путей.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.
- А в дерево изначально добавим одну вершину $start$.

Алгоритм Дейкстры: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от *start* до v.
- Также будем строить дерево кратчайших путей.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.
- А в дерево изначально добавим одну вершину *start*.
- Сделаем не более $n - 1$ следующих шагов:
  - Из только что добавленной вершины u рассмотрим каждую дугу (u, v) веса w и релаксируем её: $d(v) := \min(d(v), d(u) + w)$.
  - Изо всех вершин вне дерева выберем для добавления ту, расстояние до которой минимально.
  - Если таких вершин не осталось, завершим работу.

Алгоритм Дейкстры: идея

- Будем для каждой вершины v считать $d(v)$: кратчайшее расстояние от *start* до v.
- Также будем строить дерево кратчайших путей.
- Начнём с $d(start) = 0$ и $d(v \neq start) = \infty$.
- А в дерево изначально добавим одну вершину *start*.
- Сделаем не более $n - 1$ следующих шагов:
  - Из только что добавленной вершины u рассмотрим каждую дугу (u, v) веса w и релаксируем её: $d(v) := \min(d(v), d(u) + w)$.
  - Изо всех вершин вне дерева выберем для добавления ту, расстояние до которой минимально.
  - Если таких вершин не осталось, завершим работу.
- После этого в d посчитаны кратчайшие расстояния, а в дерево добавлены все достижимые вершины.

Алгоритм Дейкстры: код

```
1       vector <int> d (n, infinity);
2
3       vector <bool> vis (n);
4       int u = 0;
5       d[u] = 0;
6       while (u != -1) {
7           vis[u] = true;
8           for (auto e : adj[u])
9               if (d[e.v] > d[u] + e.w)
10                  d[e.v] = d[u] + e.w;
11
12
13          u = -1;
14          for (int v = 0; v < n; v++)
15              if (!vis[v])
16                  if (u == -1 || d[u] > d[v])
17                      u = v;
18      }
19
20      int y = n - 1;
21      cout << d[y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Алгоритм Дейкстры: код

```
 1       vector <int> d (n, infinity);
 2       vector <int> p (n, infinity);
 3       vector <bool> vis (n);
 4       int u = 0;
 5       d[u] = 0;
 6       while (u != -1) {
 7           vis[u] = true;
 8           for (auto e : adj[u])
 9               if (d[e.v] > d[u] + e.w) {
10                   d[e.v] = d[u] + e.w;
11                   p[e.v] = u;
12               }
13           u = -1;
14           for (int v = 0; v < n; v++)
15               if (!vis[v])
16                   if (u == -1 || d[u] > d[v])
17                       u = v;
18       }
19
20       int x = 0, y = n - 1;
21       cout << d[y] << endl;
22       /* ... */
```

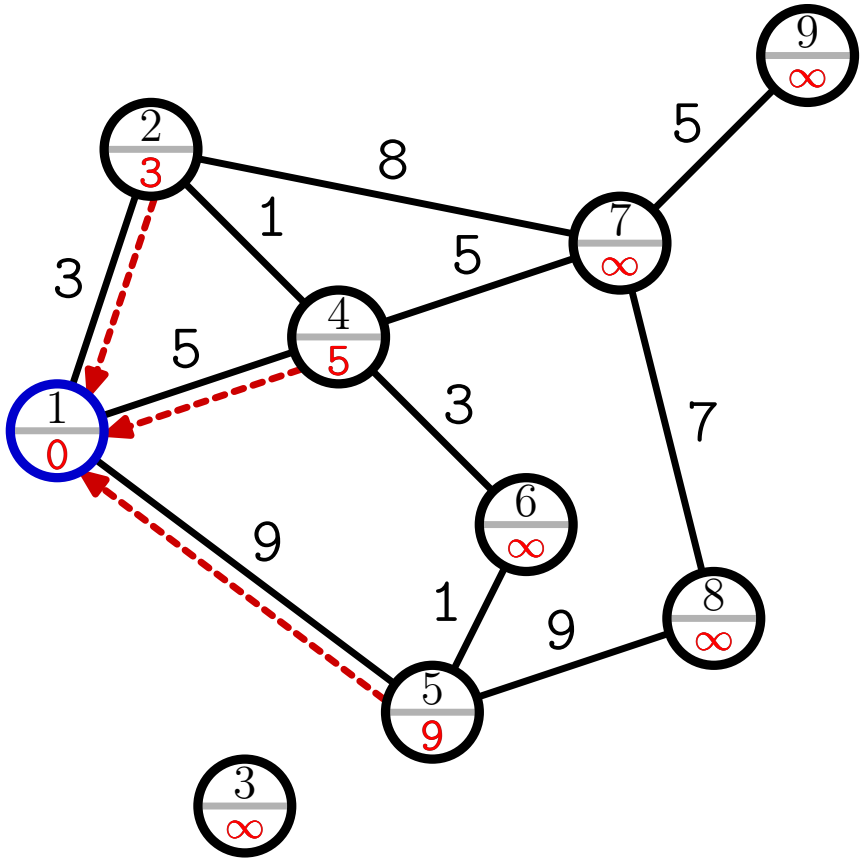
Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

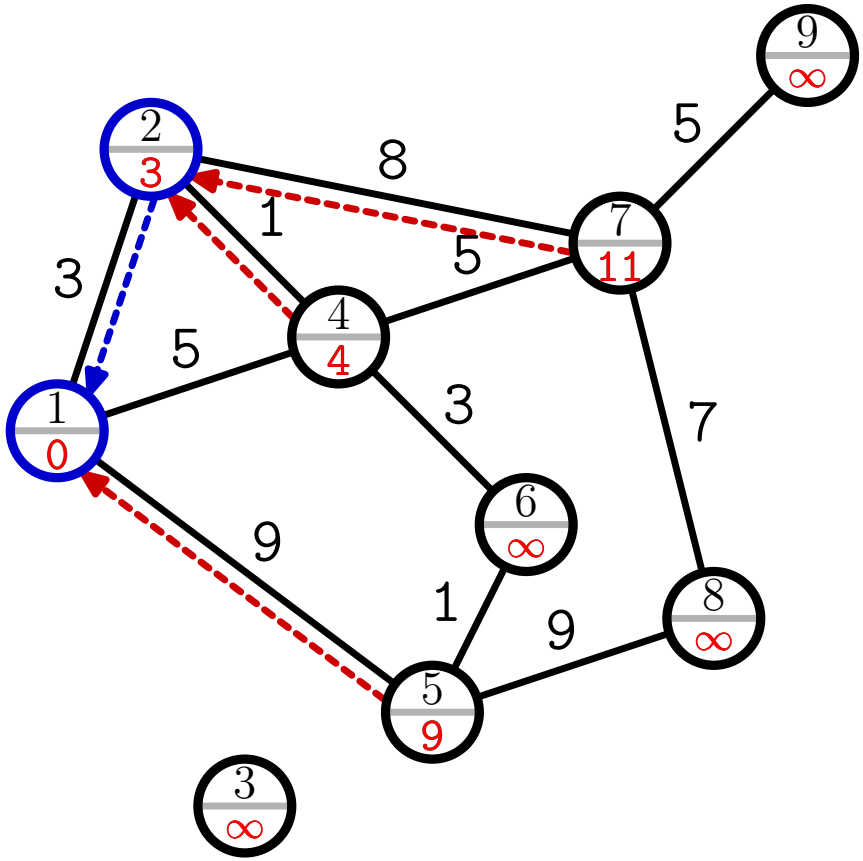
```
14
1 2 4 7 9
```

Алгоритм Дейкстры: пример



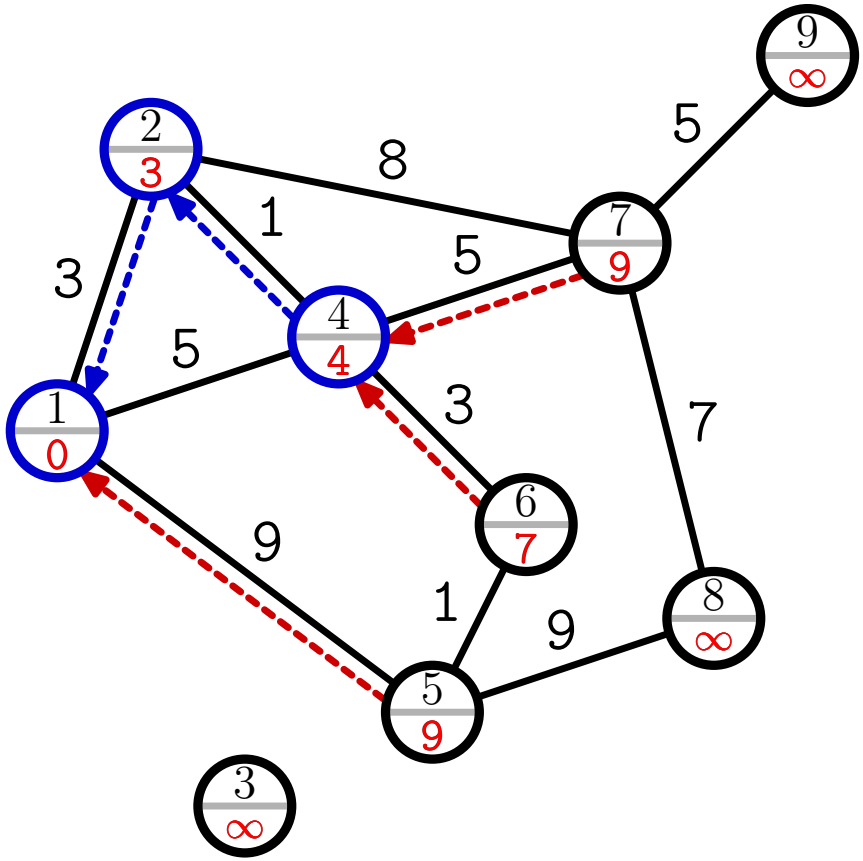
step 1

Алгоритм Дейкстры: пример



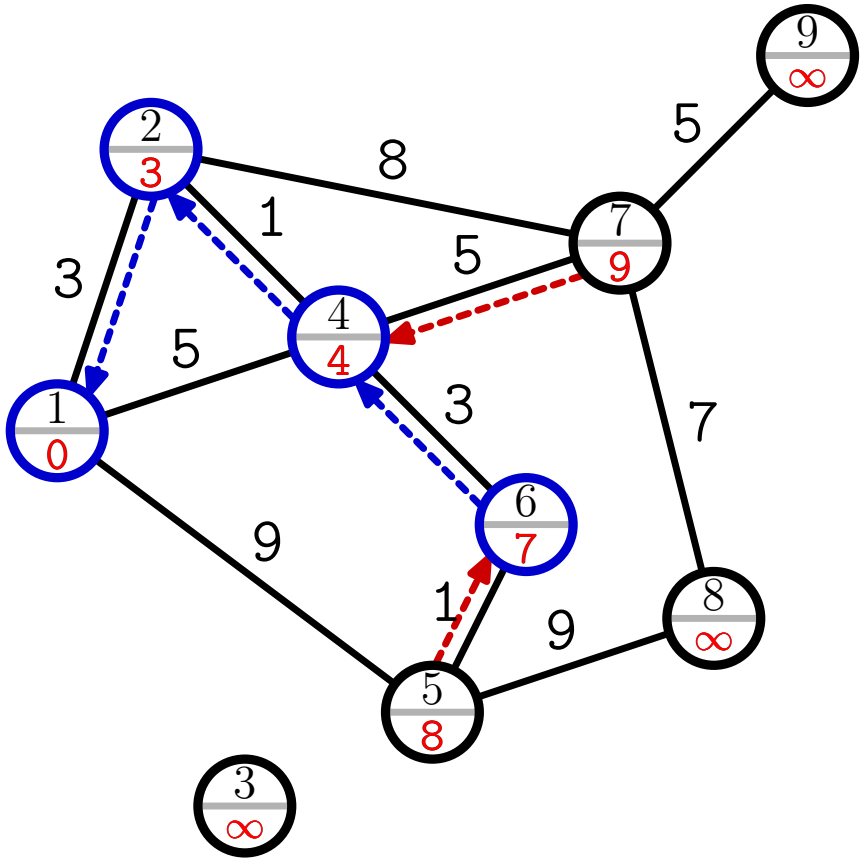
step 2

Алгоритм Дейкстры: пример



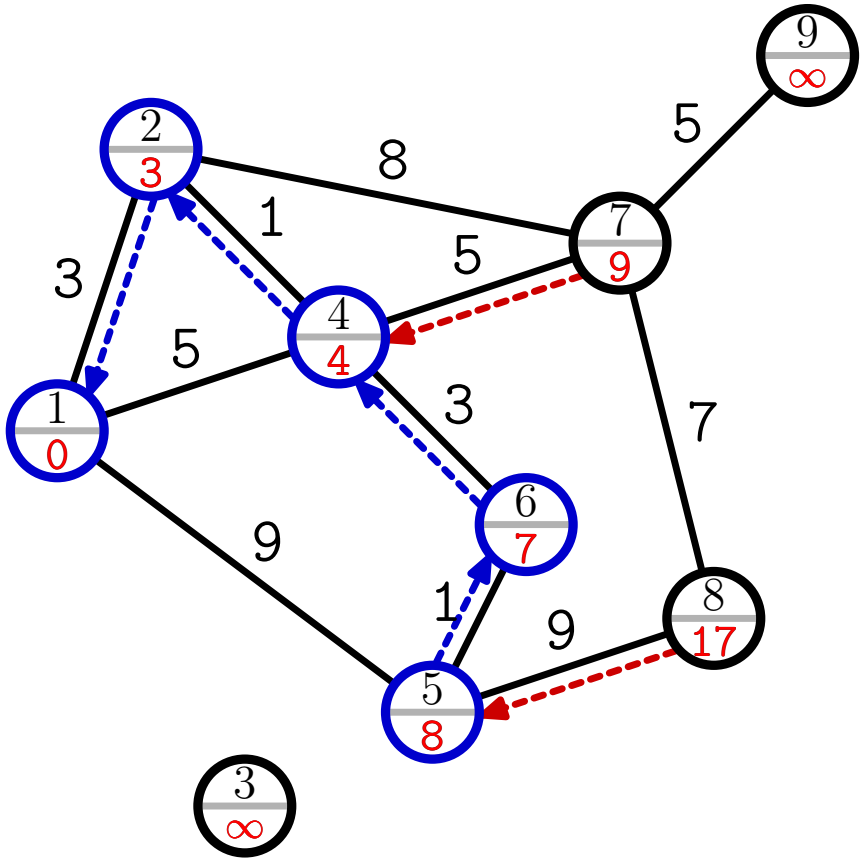
step 3

Алгоритм Дейкстры: пример



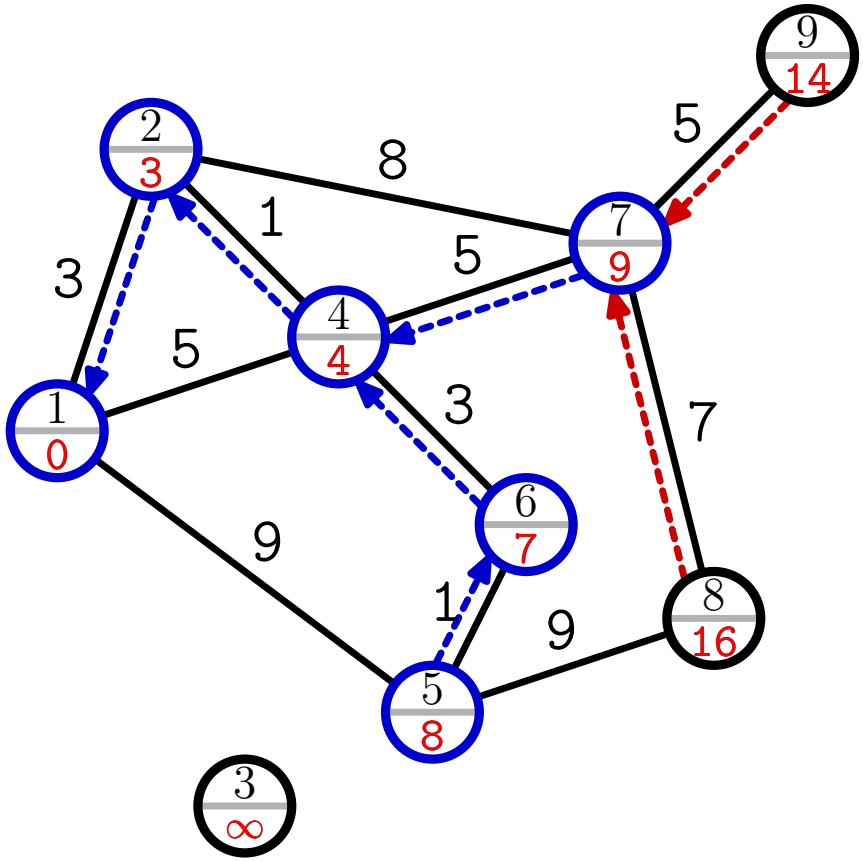
step 4

Алгоритм Дейкстры: пример



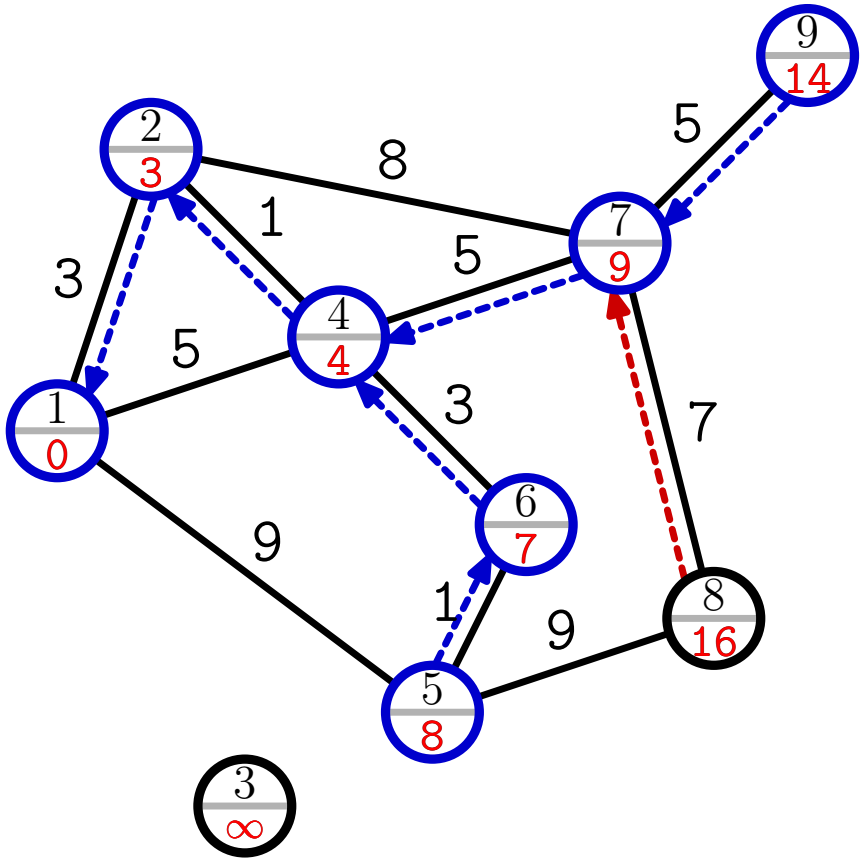
step 5

Алгоритм Дейкстры: пример



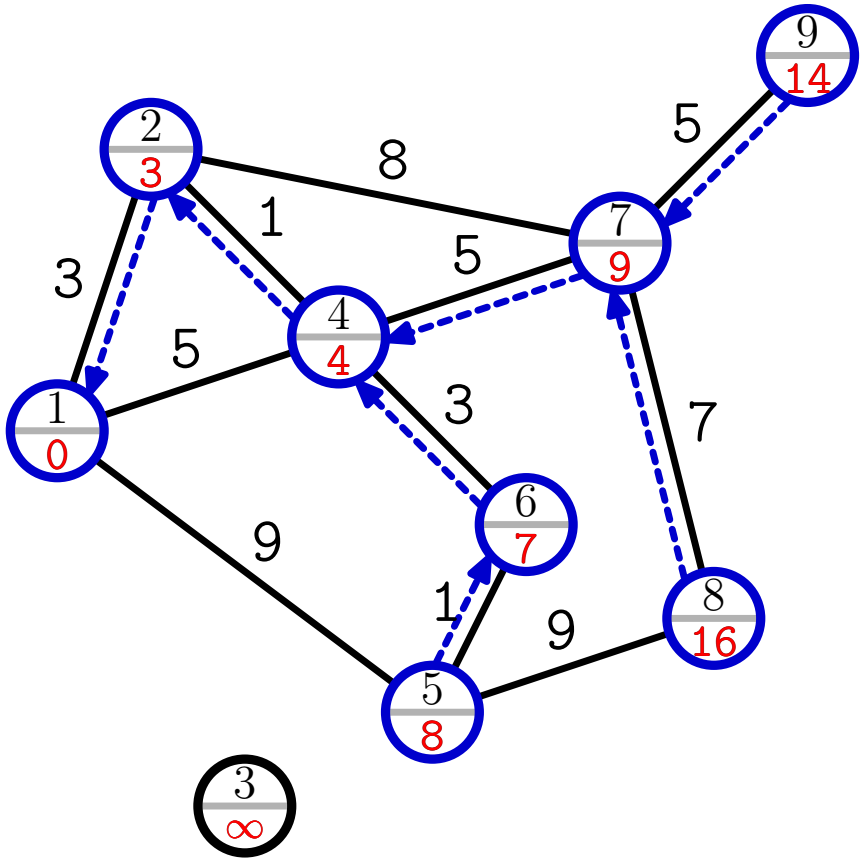
step 6

Алгоритм Дейкстры: пример



step 7

Алгоритм Дейкстры: пример



step 8

Алгоритм Дейкстры: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Алгоритму нужно, чтобы рёбра были неотрицательны.

Алгоритм Дейкстры: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Алгоритму нужно, чтобы рёбра были неотрицательны.
- На каждом шаге мы добавляем в дерево кратчайших путей ближайшую к старту вершину из оставшихся.
- От противного: пусть расстояние до этой вершины равно x, и предположим, что мы потом сможем найти до этой вершины более короткий путь p.

Алгоритм Дейкстры: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Алгоритму нужно, чтобы рёбра были неотрицательны.
- На каждом шаге мы добавляем в дерево кратчайших путей ближайшую к старту вершину из оставшихся.
- От противного: пусть расстояние до этой вершины равно x, и предположим, что мы потом сможем найти до этой вершины более короткий путь p.
- Рассмотрим дерево на этом шаге.
- Путь p когда-то выходит из этого дерева.
- В этот момент длина p не меньше, чем x: ведь это ближайшая из вершин вне дерева.
- Все следующие рёбра в пути p неотрицательны.

Алгоритм Дейкстры: доказательство

- Как доказать, что получившиеся в d числа — кратчайшие расстояния?
- Алгоритму нужно, чтобы рёбра были неотрицательны.
- На каждом шаге мы добавляем в дерево кратчайших путей ближайшую к старту вершину из оставшихся.
- От противного: пусть расстояние до этой вершины равно x, и предположим, что мы потом сможем найти до этой вершины более короткий путь p.
- Рассмотрим дерево на этом шаге.
- Путь p когда-то выходит из этого дерева.
- В этот момент длина p не меньше, чем x: ведь это ближайшая из вершин вне дерева.
- Все следующие рёбра в пути p неотрицательны.
- Значит, общая длина p не меньше x — противоречие!

Алгоритм Дейкстры: анализ

- Сколько времени и памяти требуется алгоритму?
- Память:
- Время:
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Дейкстры: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время:
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Дейкстры: анализ

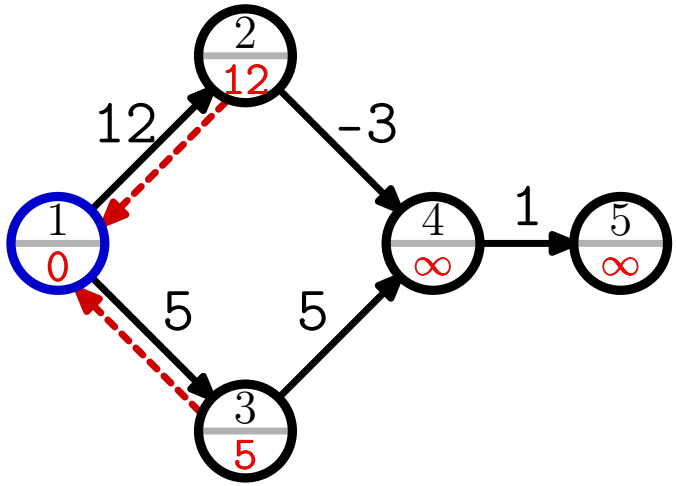
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.

Алгоритм Дейкстры: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.

Алгоритм Дейкстры: анализ

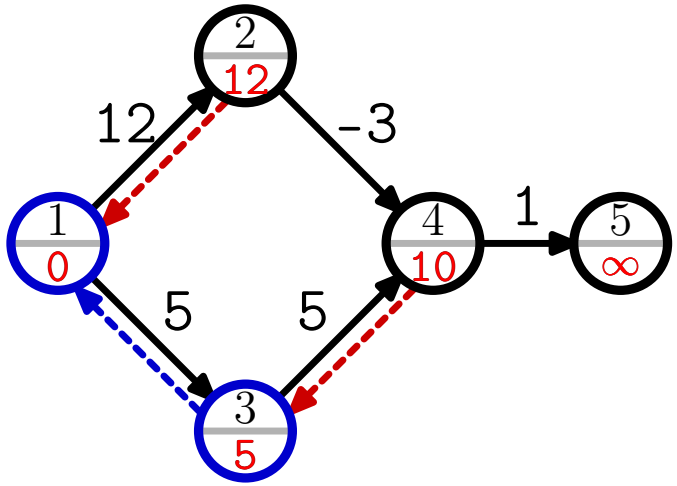
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.



step 1

Алгоритм Дейкстры: анализ

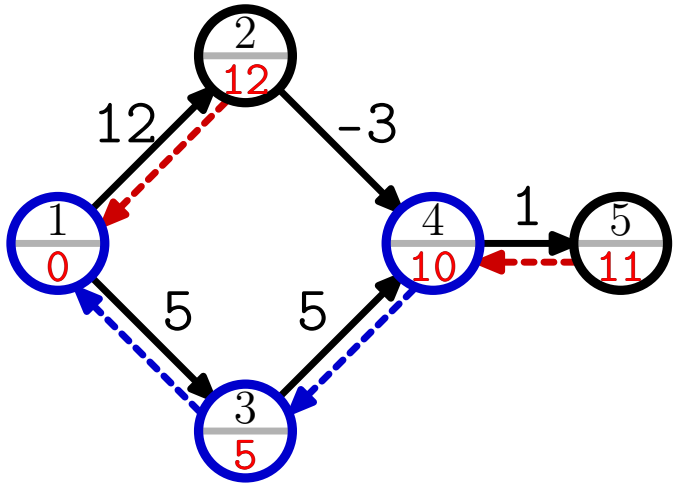
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.



step 2

Алгоритм Дейкстры: анализ

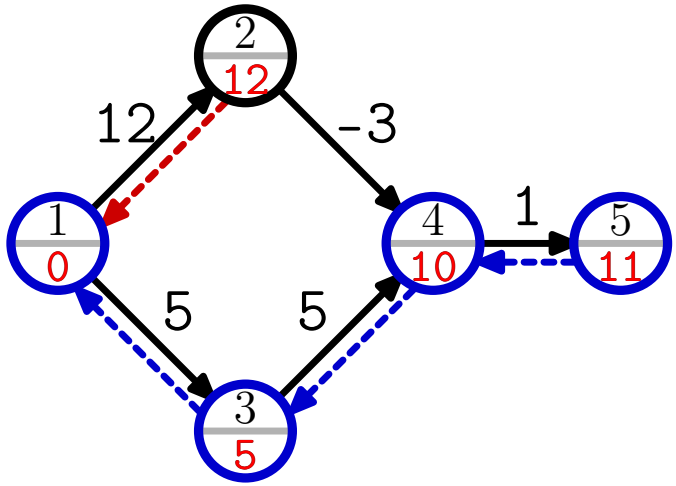
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.



step 3

Алгоритм Дейкстры: анализ

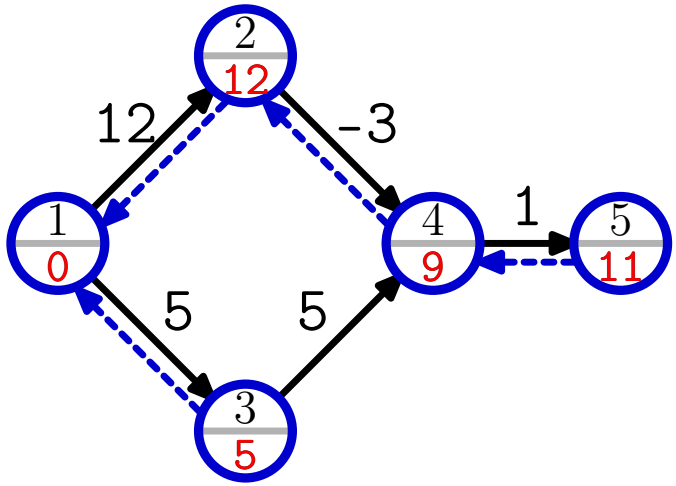
- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.



step 4

Алгоритм Дейкстры: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.



step 5

Алгоритм Дейкстры: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: пока что $O(n^2)$...
- Находит кратчайшие пути из одной вершины во все остальные.
- Не работает, если в графе есть **рёбра** отрицательной длины.
- Давайте ускорим до $O(m \log n)$.

Алгоритм Дейкстры: код 2

```
1       vector <int> d (n, infinity);
2
3       vector <bool> vis (n);
4       int u = 0;
5       d[u] = 0;
6       while (u != -1) {
7           vis[u] = true;
8           for (auto e : adj[u])
9               if (d[e.v] > d[u] + e.w)
10                  d[e.v] = d[u] + e.w;
11
12
13          u = -1;
14          for (int v = 0; v < n; v++)
15              if (!vis[v])
16                  if (u == -1 || d[u] > d[v])
17                      u = v;
18      }
19
20      int y = n - 1;
21      cout << d[y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Алгоритм Дейкстры: код 2

```
1       vector <int> d (n, infinity);
2       vector <bool> vis (n);
3       set <pair <int, int> > s;
4       int x = 0;
5       d[x] = 0;
6       s.insert ({d[x], x});
7       while (!s.empty ()) {
8           auto u = s.begin () -> second;
9           s.erase (s.begin ());
10          vis[u] = true;
11          for (auto e : adj[u])
12              if (!vis[e.v])
13                  if (d[e.v] > d[u] + e.w) {
14                      s.erase ({d[e.v], e.v});
15                      d[e.v] = d[u] + e.w;
16                      s.insert ({d[e.v], e.v});
17                  }
18      }
19
20      int y = n - 1;
21      cout << d[y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Алгоритм Дейкстры: код 2

```
1       vector <int> d (n, infinity);
2       vector <bool> vis (n);
3       priority_queue <pair <int, int> > pq;
4       int x = 0;
5       d[x] = 0;
6       pq.push ({-d[x], x});
7       while (!pq.empty ()) {
8           auto u = pq.top ().second;
9           pq.pop ();
10          if (vis[u])
11              continue;
12          vis[u] = true;
13          for (auto e : adj[u])
14              if (!vis[e.v])
15                  if (d[e.v] > d[u] + e.w) {
16                      d[e.v] = d[u] + e.w;
17                      pq.push ({-d[e.v], e.v});
18                  }
19      }
20
21      int y = n - 1;
22      cout << d[y] << endl;
```

Ввод:

```
9 11
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
5 6 1
7 8 7
1 5 9
```

Вывод:

```
14
```

Содержание

Задача

- Дан связный взвешенный граф.
- Мы хотим выбрать некоторые рёбра графа:
  - рёбра должны составлять дерево,
  - это дерево должно содержать все вершины графа,
  - суммарная длина этих рёбер должна быть минимальна.

Содержание

Алгоритм Прима: идея

- Будем строить дерево, начиная с одной любой вершины.
- Будем для каждой вершины v вне дерева поддерживать $d(v)$: кратчайшее расстояние от v до какой-либо вершины дерева.
- А также $p(v)$ номер этой вершины дерева.

Алгоритм Прима: идея

- Будем строить дерево, начиная с одной любой вершины.
- Будем для каждой вершины v вне дерева поддерживать $d(v)$: кратчайшее расстояние от v до какой-либо вершины дерева.
- А также $p(v)$ номер этой вершины дерева.
- Сделаем ровно $n - 1$ следующих шагов:
  - Из только что добавленной в дерево вершины u рассмотрим каждое ребро (u, v) веса w и релаксируем её:
  - Если $d(v) > w$, то заменим $d(v)$ на w и $p(v)$ на u.
  - Изо всех вершин вне дерева выберем для добавления ту, расстояние до которой минимально.
  - Если таких вершин не осталось, завершим работу.

Алгоритм Прима: идея

- Будем строить дерево, начиная с одной любой вершины.
- Будем для каждой вершины v вне дерева поддерживать $d(v)$: кратчайшее расстояние от v до какой-либо вершины дерева.
- А также $p(v)$ номер этой вершины дерева.
- Сделаем ровно $n - 1$ следующих шагов:
  - Из только что добавленной в дерево вершины u рассмотрим каждое ребро (u, v) веса w и релаксируем её:
  - Если $d(v) > w$, то заменим $d(v)$ на w и $p(v)$ на u.
  - Изо всех вершин вне дерева выберем для добавления ту, расстояние до которой минимально.
  - Если таких вершин не осталось, завершим работу.
- Минимальное остовное дерево — это рёбра $(v, p(v))$ для всех v, кроме самой первой вершины.

Алгоритм Прима: код

```
1       vector <int> d (n, infinity);
2
3       vector <bool> vis (n);
4       int u = 0;
5       d[u] = 0;
6       int res = 0;
7       while (u != -1) {
8           res += d[u];
9           vis[u] = true;
10          for (auto e : adj[u])
11              if (d[e.v] > e.w)
12                  d[e.v] = e.w;
13          u = -1;
14          for (int v = 0; v < n; v++)
15              if (!vis[v])
16                  if (u == -1 || d[u] > d[v])
17                      u = v;
18      }
19      cout << res << endl;
```

Ввод:

```
9 12
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
6 5 1
7 8 7
1 5 9
5 3 2
```

Вывод:

```
27
```

Алгоритм Прима: код

```
1       vector <int> d (n, infinity);
2       vector <int> p (n, infinity);
3       vector <bool> vis (n);
4       int u = 0;
5       d[u] = 0;
6       int res = 0;
7       while (u != -1) {
8           if (u != 0)
9               cout << u + 1 << " " <<
10                  p[u] + 1 << " " << d[u] << endl;
11          res += d[u];
12          vis[u] = true;
13          for (auto e : adj[u])
14              if (d[e.v] > e.w) {
15                  d[e.v] = e.w;
16                  p[e.v] = u;
17              }
18          u = -1;
19          for (int v = 0; v < n; v++)
20              if (!vis[v])
21                  if (u == -1 || d[u] > d[v])
22                      u = v;
23      }
24      cout << res << endl;
```

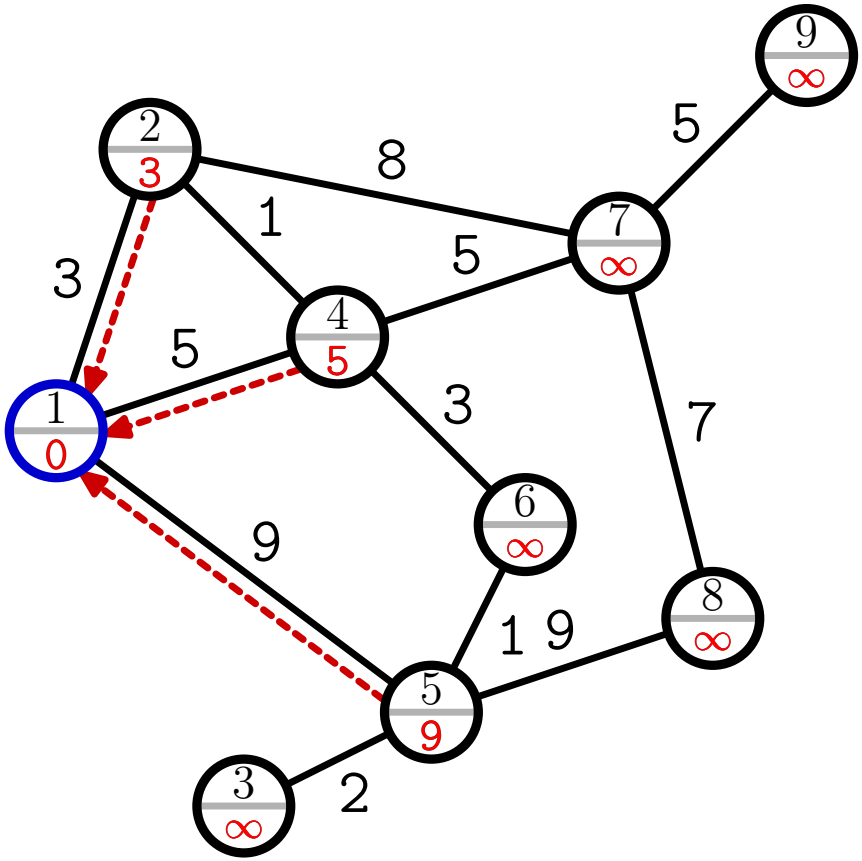
Ввод:

```
9 12
1 2 3
1 4 5
2 7 8
2 4 1
4 6 3
4 7 5
5 8 9
7 9 5
6 5 1
7 8 7
1 5 9
5 3 2
```

Вывод:

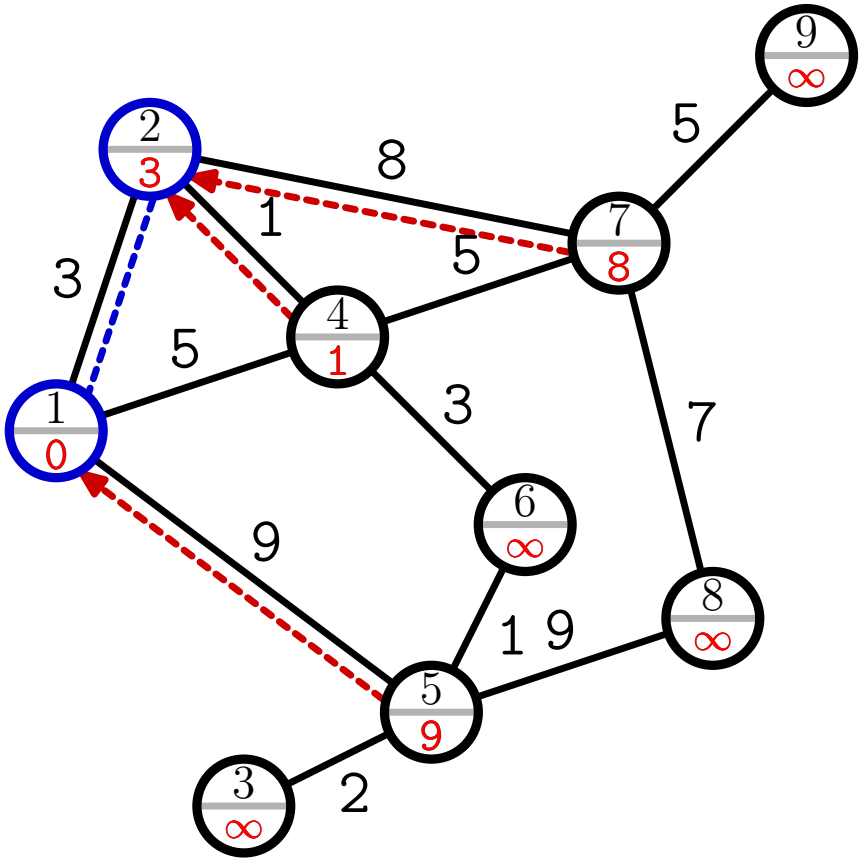
```
2 1 3
4 2 1
6 4 3
5 6 1
3 5 2
7 4 5
9 7 5
8 7 7
27
```

Алгоритм Прима: пример



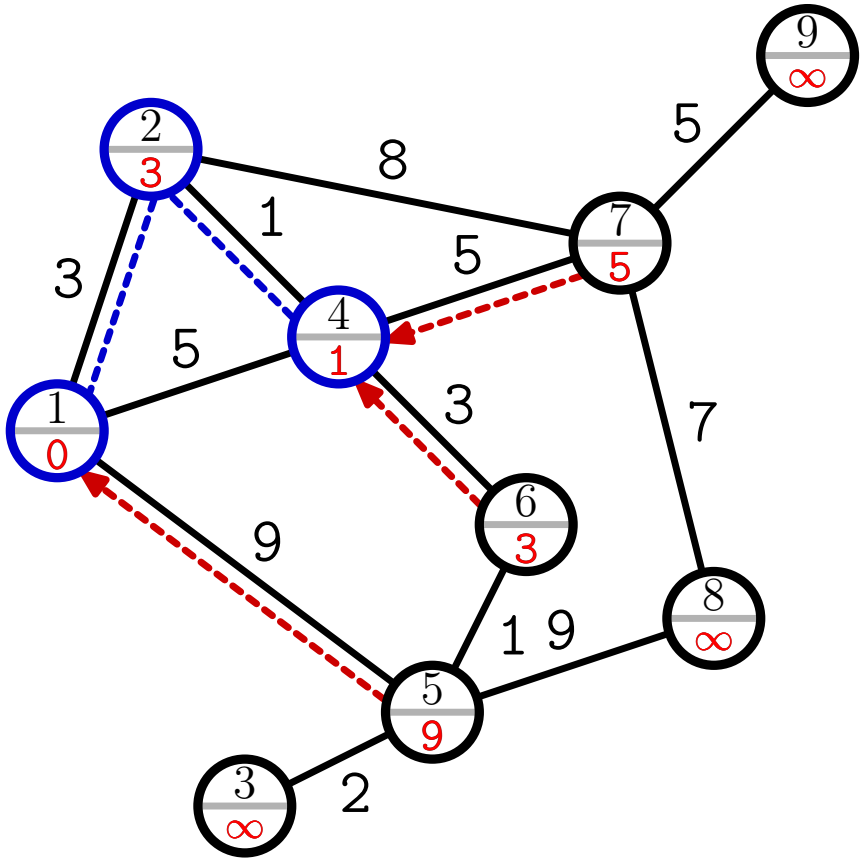
step 1

Алгоритм Прима: пример



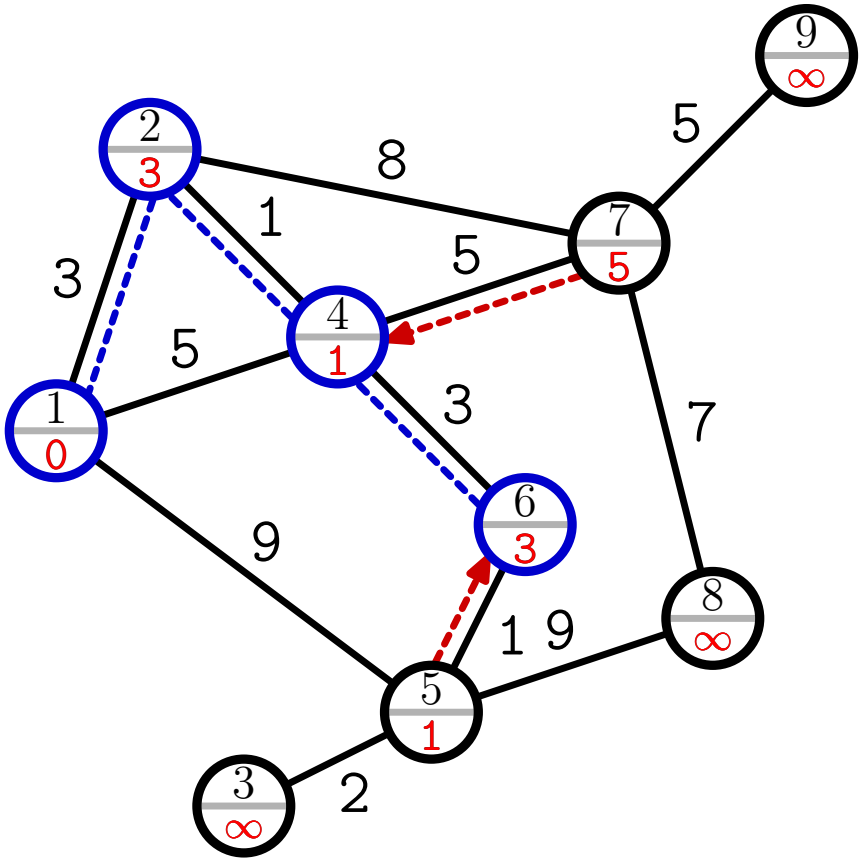
step 2

Алгоритм Прима: пример



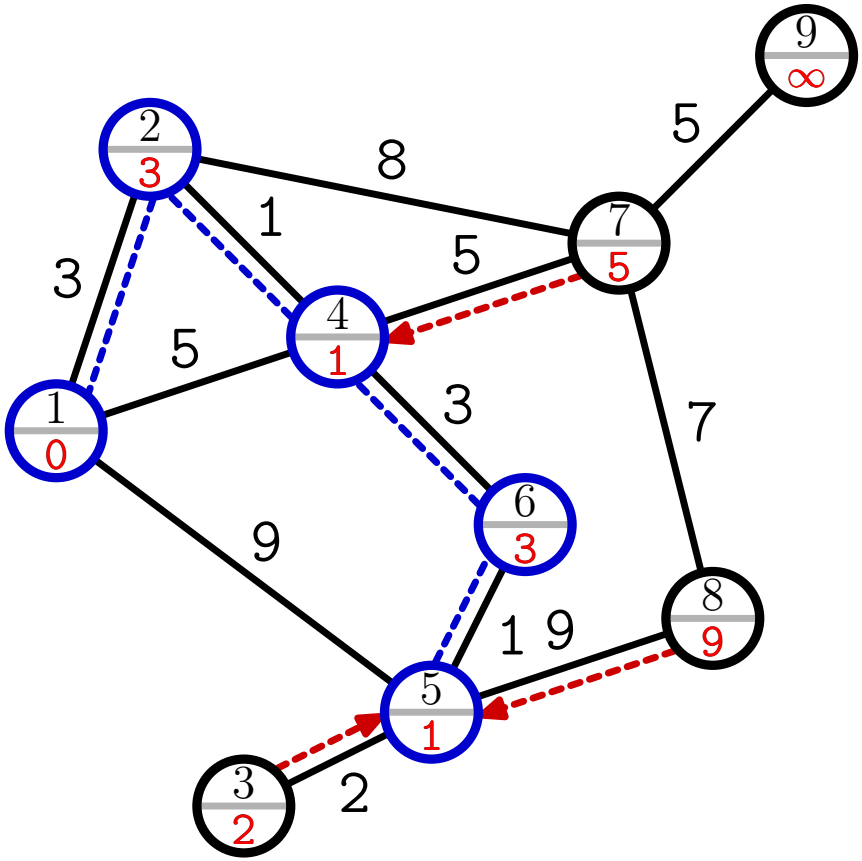
step 3

Алгоритм Прима: пример



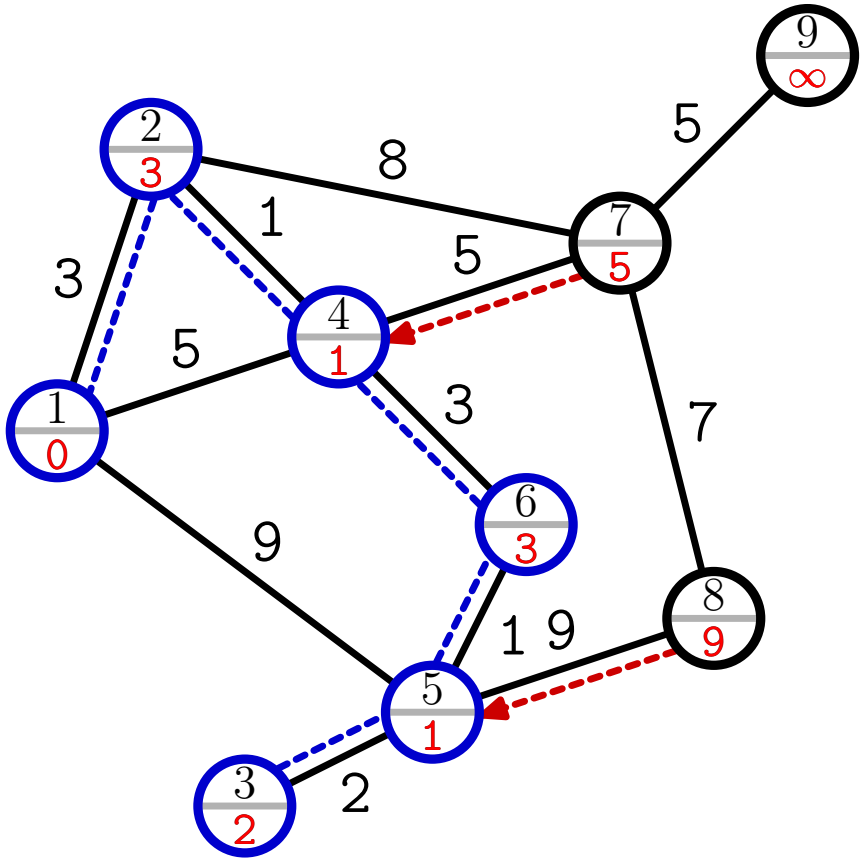
step 4

Алгоритм Прима: пример



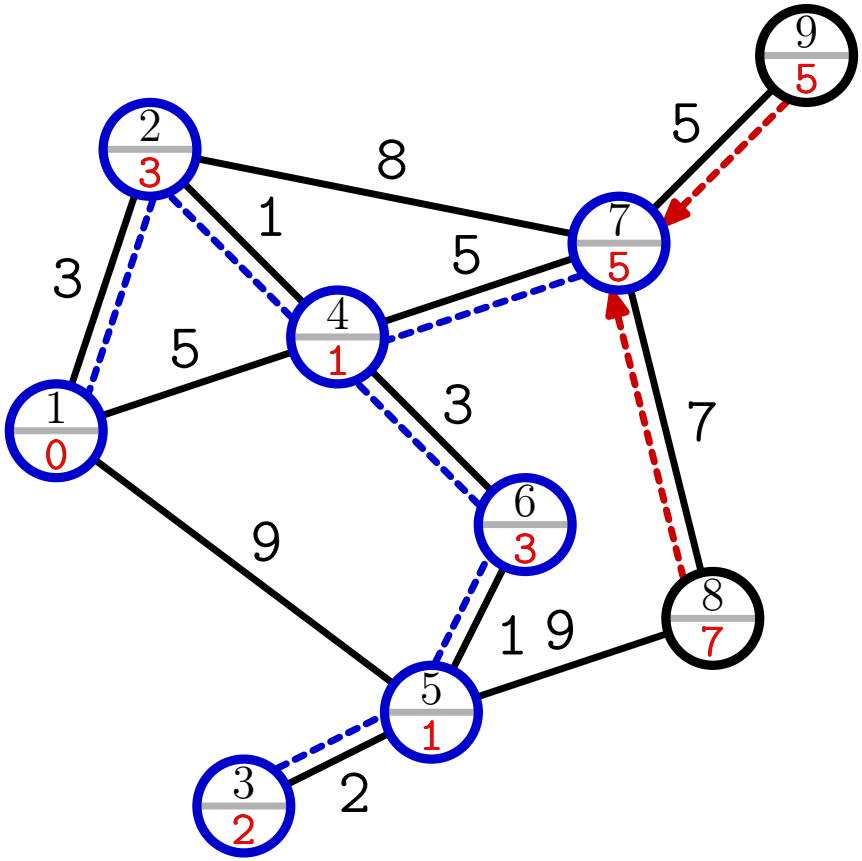
step 5

Алгоритм Прима: пример



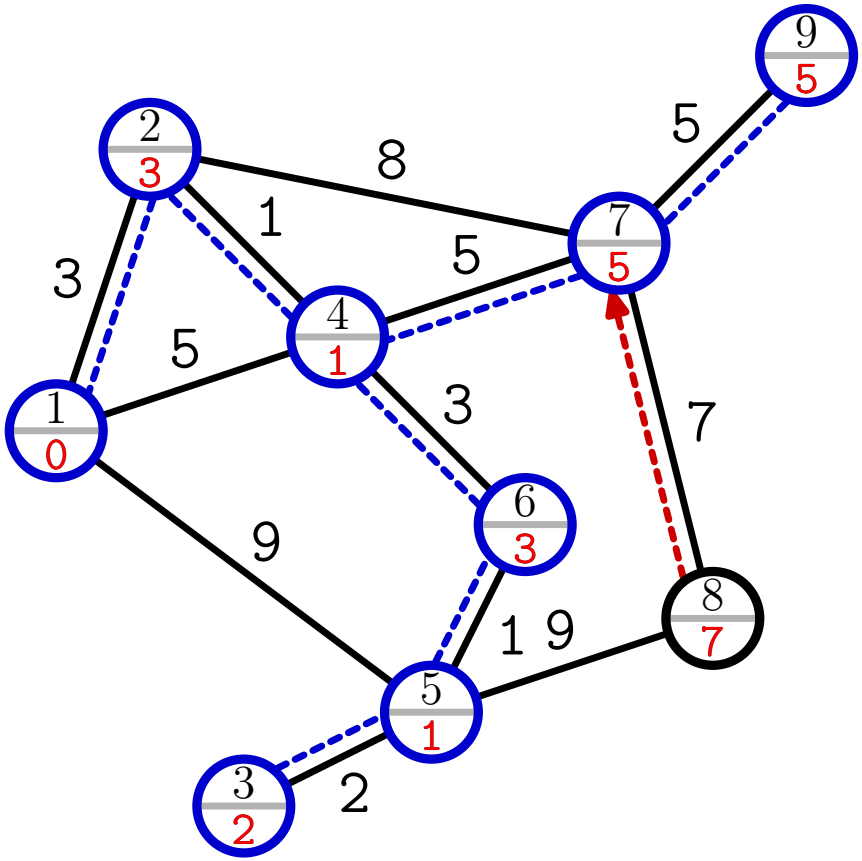
step 6

Алгоритм Прима: пример



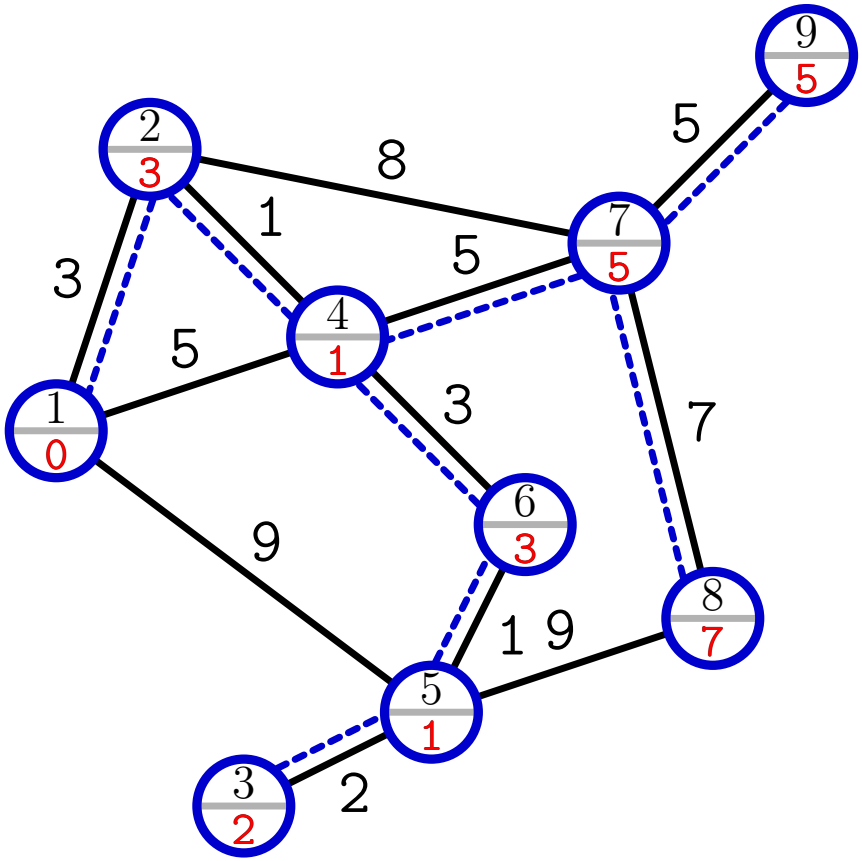
step 7

Алгоритм Прима: пример



step 8

Алгоритм Прима: пример



step 9

Алгоритм Прима: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?

Алгоритм Прима: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.

Алгоритм Прима: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.
  - Добавим его к дереву T, получится цикл.
  - Посмотрим на дерево S, которое мы построили к этому моменту.
  - В цикле есть ребро (u, v), которое пересекает границу S.
  - Значит, в цикле есть другое ребро (u', v'), которое также пересекает границу S.
  - Рассмотрим дерево T с добавленным (u, v) и удалённым (u', v').
  - Это тоже остовное дерево, и его вес не больше веса T, так как вес (u, v) минимальный среди всех рёбер, пересекающих границу S.

Алгоритм Прима: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.
  - Добавим его к дереву T, получится цикл.
  - Посмотрим на дерево S, которое мы построили к этому моменту.
  - В цикле есть ребро (u, v), которое пересекает границу S.
  - Значит, в цикле есть другое ребро (u', v'), которое также пересекает границу S.
  - Рассмотрим дерево T с добавленным (u, v) и удалённым (u', v').
  - Это тоже остовное дерево, и его вес не больше веса T, так как вес (u, v) минимальный среди всех рёбер, пересекающих границу S.
- Заменяя так рёбра, получаем, что наше дерево не хуже T.

Алгоритм Прима: анализ

- Сколько времени и памяти требуется алгоритму?
- Память:
- Время:

Алгоритм Прима: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время:

Алгоритм Прима: анализ

- Сколько времени и памяти требуется алгоритму?
- Память: $\Theta(n)$.
- Время: $O(n^2)$, можно ускорить до $O(m \log n)$.

Дейкстра и Прим: сравнение кода

```
 1       vector <int> d (n, infinity);
 2
 3       vector <bool> vis (n);
 4       int u = 0;
 5       d[u] = 0;
 6       while (u != -1) {
 7           vis[u] = true;
 8           for (auto e : adj[u])
 9               if (d[e.v] > d[u] + e.w)
10                   d[e.v] = d[u] + e.w;
11
12
13           u = -1;
14           for (int v = 0; v < n; v++)
15               if (!vis[v])
16                   if (u == -1 || d[u] > d[v])
17                       u = v;
18       }
19
20       int y = n - 1;
21       cout << d[y] << endl;
```

Дейкстра и Прим: сравнение кода

```
 1       vector <int> d (n, infinity);
 2
 3       vector <bool> vis (n);
 4       int u = 0;
 5       d[u] = 0;
 6       int res = 0;
 7       while (u != -1) {
 8           res += d[u];
 9           vis[u] = true;
10           for (auto e : adj[u])
11               if (d[e.v] > e.w)
12                   d[e.v] = e.w;
13           u = -1;
14           for (int v = 0; v < n; v++)
15               if (!vis[v])
16                   if (u == -1 || d[u] > d[v])
17                       u = v;
18       }
19       cout << res << endl;
```

Содержание

Алгоритм Краскала: идея

- Будем строить дерево, начиная с графа из n вершин и 0 рёбер.
- Будем рассматривать рёбра в порядке возрастания веса.
- И будем поддерживать компоненты связности в графе.

Алгоритм Краскала: идея

- Будем строить дерево, начиная с графа из n вершин и 0 рёбер.
- Будем рассматривать рёбра в порядке возрастания веса.
- И будем поддерживать компоненты связности в графе.
- Если очередное ребро соединяет различные компоненты связности, добавим его в дерево и объединим компоненты связности.
- Если очередное ребро соединяет вершины из одной компоненты связности, пропустим это ребро.

Алгоритм Краскала: идея

- Будем строить дерево, начиная с графа из n вершин и 0 рёбер.
- Будем рассматривать рёбра в порядке возрастания веса.
- И будем поддерживать компоненты связности в графе.
- Если очередное ребро соединяет различные компоненты связности, добавим его в дерево и объединим компоненты связности.
- Если очередное ребро соединяет вершины из одной компоненты связности, пропустим это ребро.
- Основная часть кода — система непересекающихся множеств с прошлого занятия.

Алгоритм Краскала: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?

Алгоритм Краскала: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.

Алгоритм Краскала: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.
  - Добавим его к дереву T, получится цикл.
  - Поскольку в нашем решении u и v лежат в разных компонентах связности, в этом цикле есть другое ребро (u', v'), которое мы ещё не рассмотрели.
  - Рассмотрим дерево T с добавленным (u, v) и удалённым (u', v').
  - Это тоже остовное дерево, и его вес не больше веса T, так как вес (u, v) не больше веса любого другого ребра, которое мы рассмотрим позже.

Алгоритм Краскала: доказательство

- Как доказать, что получившееся остовное дерево — минимальное?
- Пусть правильный ответ — минимальное дерево T.
- Рассмотрим первый шаг, на котором мы взяли ребро (u, v), которого нет в T.
  - Добавим его к дереву T, получится цикл.
  - Поскольку в нашем решении u и v лежат в разных компонентах связности, в этом цикле есть другое ребро (u', v'), которое мы ещё не рассмотрели.
  - Рассмотрим дерево T с добавленным (u, v) и удалённым (u', v').
  - Это тоже остовное дерево, и его вес не больше веса T, так как вес (u, v) не больше веса любого другого ребра, которое мы рассмотрим позже.
- Заменяя так рёбра, получаем, что наше дерево не хуже T.

Вопросы?

Вопросы?