Графы

Гаевой, Казменко, Макаров

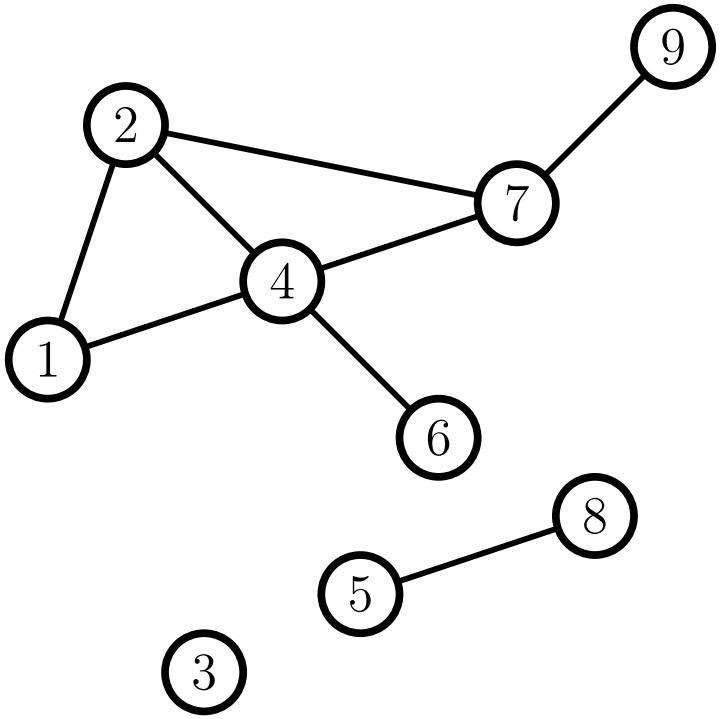
Санкт-Петербургский Государственный Университет

Четверг, 19 ноября 2020 года

Содержание

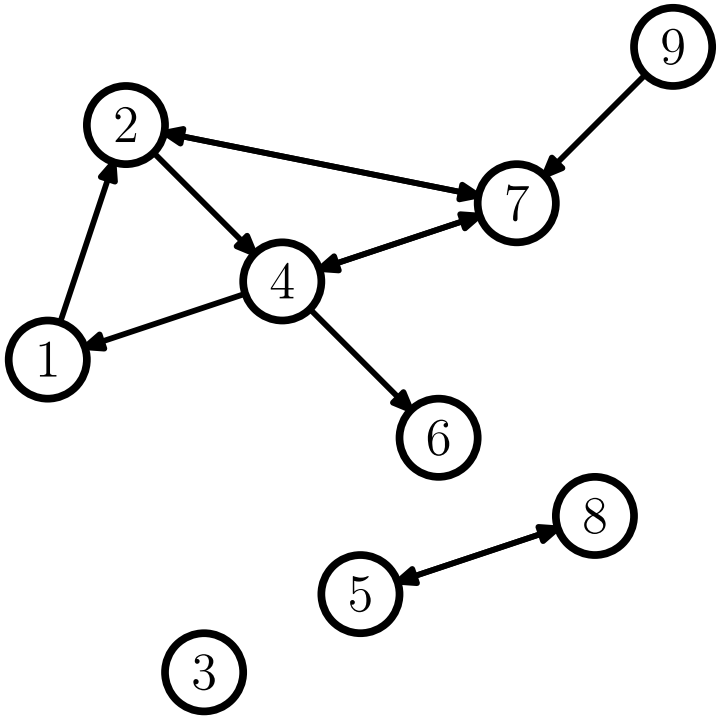
Содержание

Определения



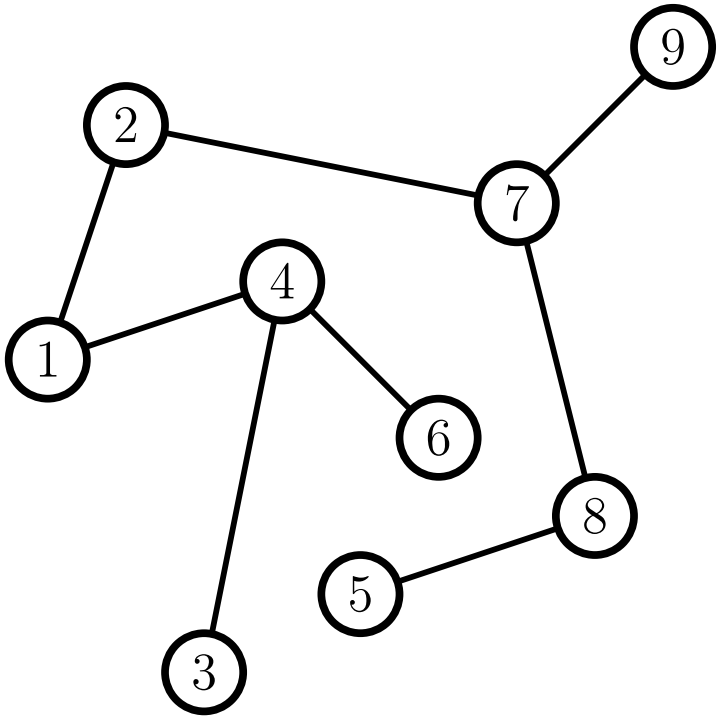
Определение: граф (неориентированный) — это пара (V, E), где V — множество вершин, а E — множество (неориентированных) рёбер. Каждое ребро — это неупорядоченная пара вершин.

Определения



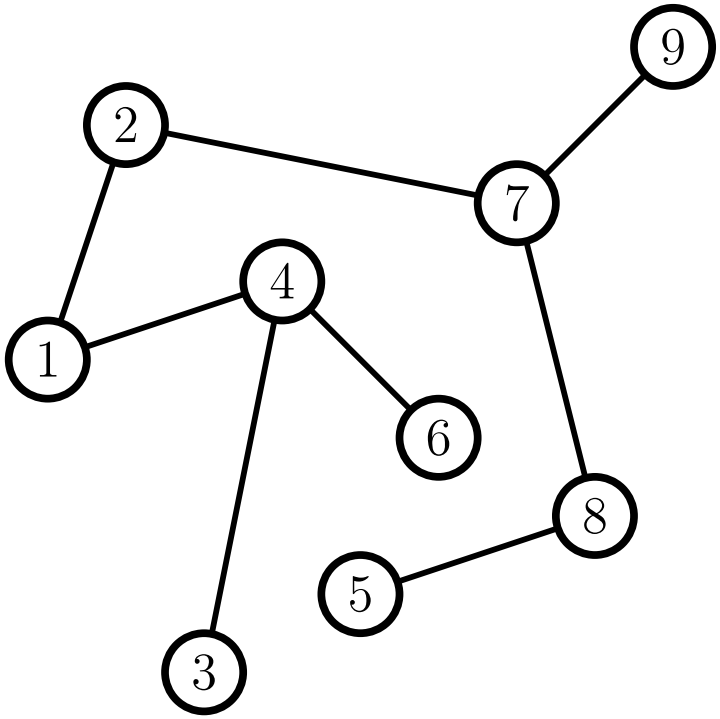
Определение: ориентированный граф (орграф) — это пара (V, E), где V — множество вершин, а E — множество дуг (ориентированных рёбер). Каждое ребро — это упорядоченная пара вершин.

Определения



Определение: дерево — это связный граф без циклов.
Необходимые определения: компонента связности, связный граф, цикл.

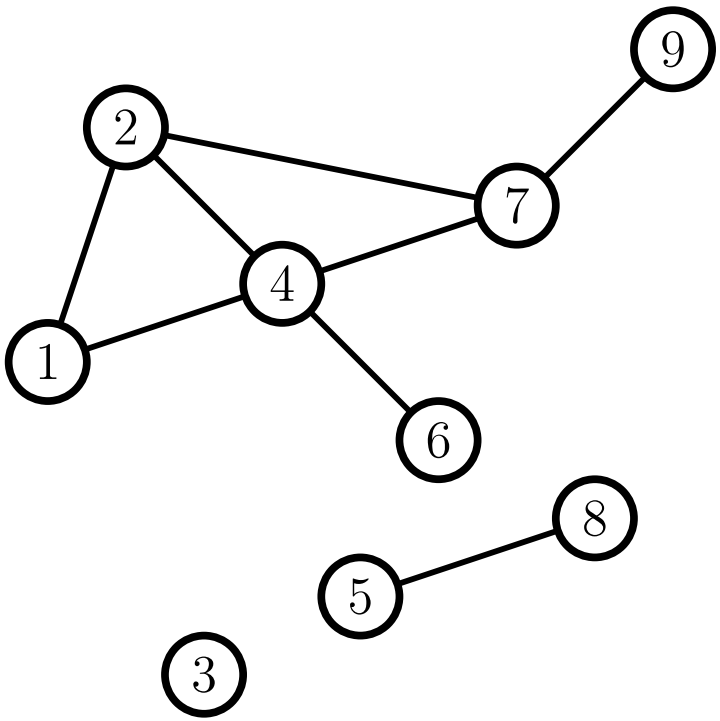
Определения



Критерий: обыкновенный граф является деревом, если он связен и $|E| = |V| - 1$.

Необходимое определение: обыкновенный граф.

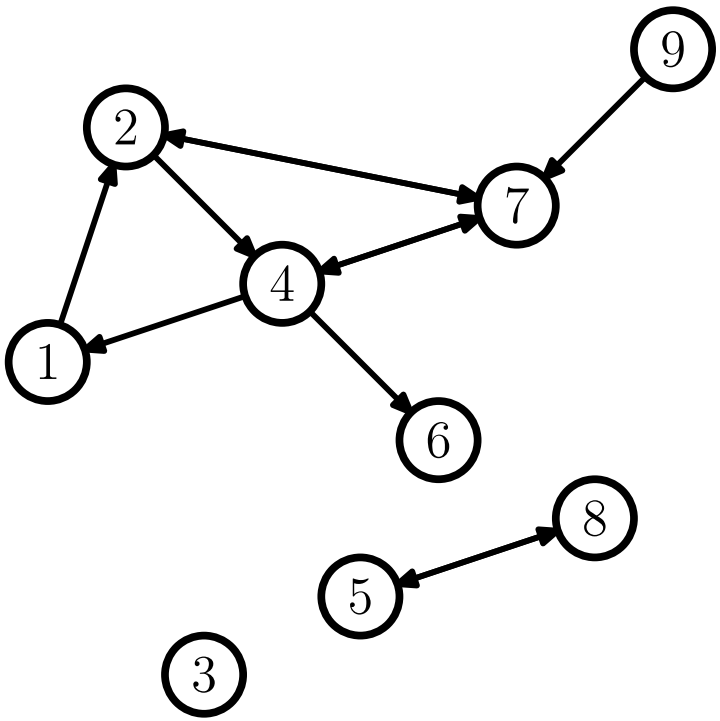
Задание графа



Ввод:

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

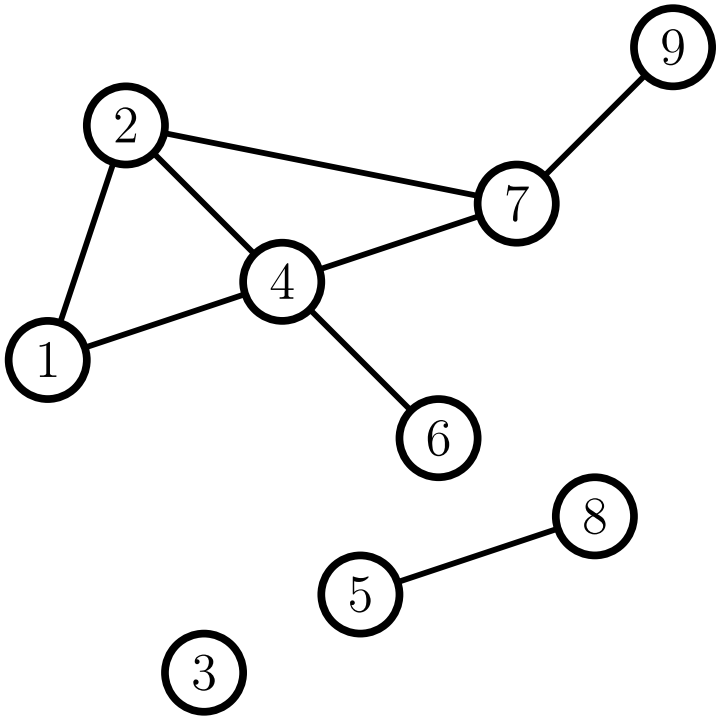
Задание графа



Ввод:

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

Матрица смежности

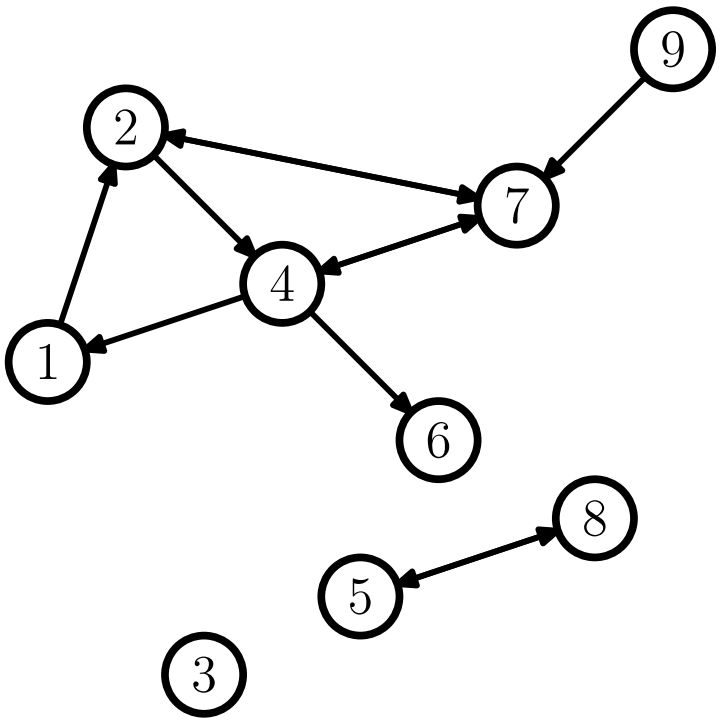


Ввод:

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

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

Матрица смежности

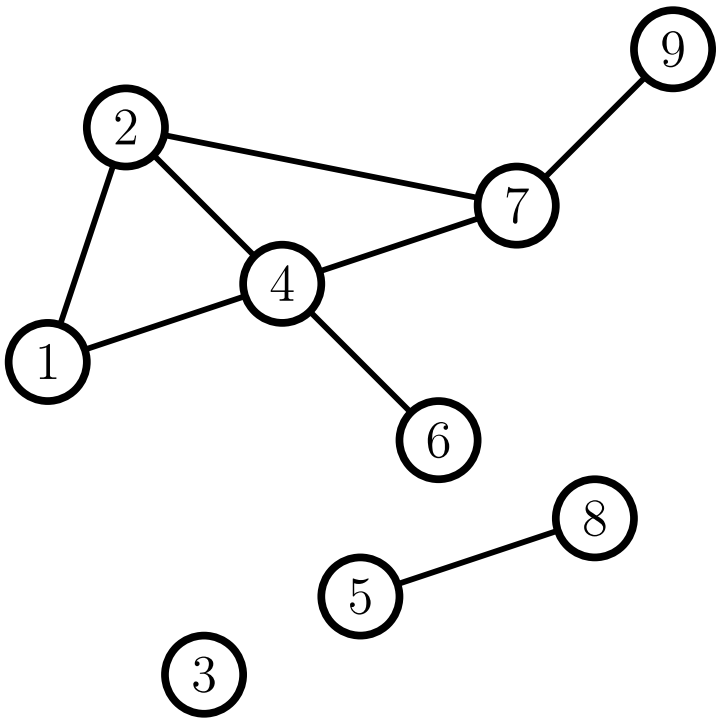


Ввод:

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

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

Списки смежных вершин

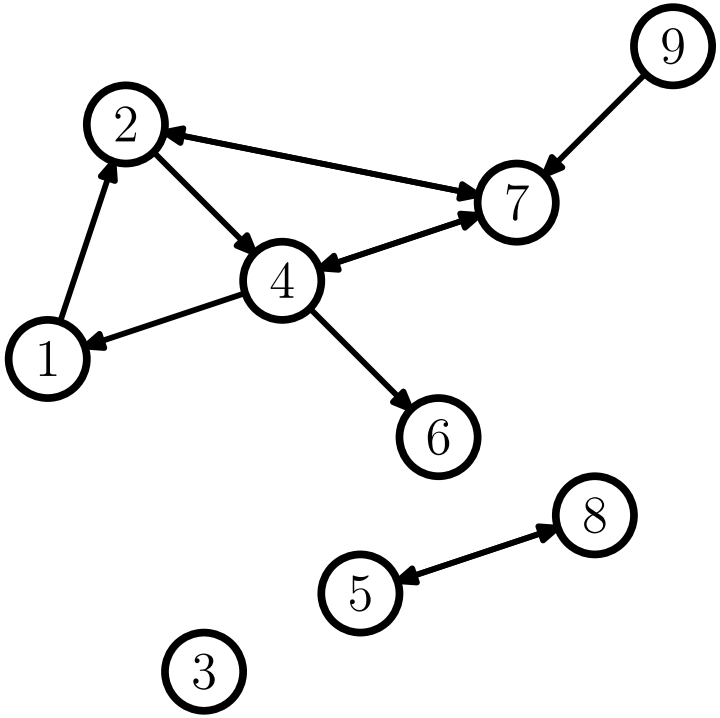


Ввод:

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

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

Списки смежных вершин



Ввод:

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

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

Матрица смежности

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

Ввод:

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

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

Матрица смежности

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

Ввод:

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

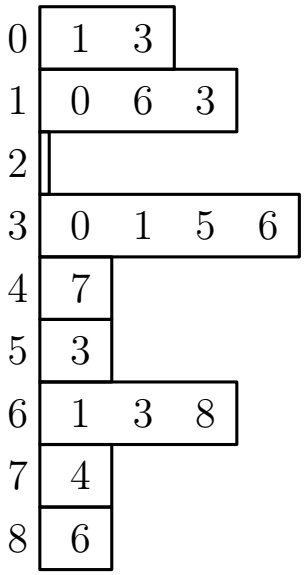
	0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0
2	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	1	1	0	0
4	0	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	0	0
6	0	1	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	1	0	0

Списки смежных вершин

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

Ввод:

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

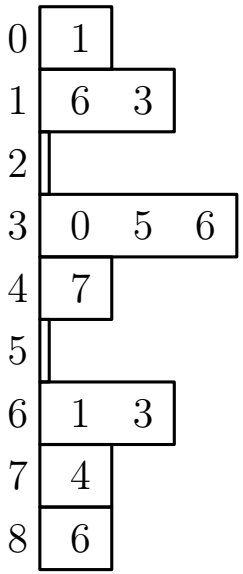


Списки смежных вершин

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

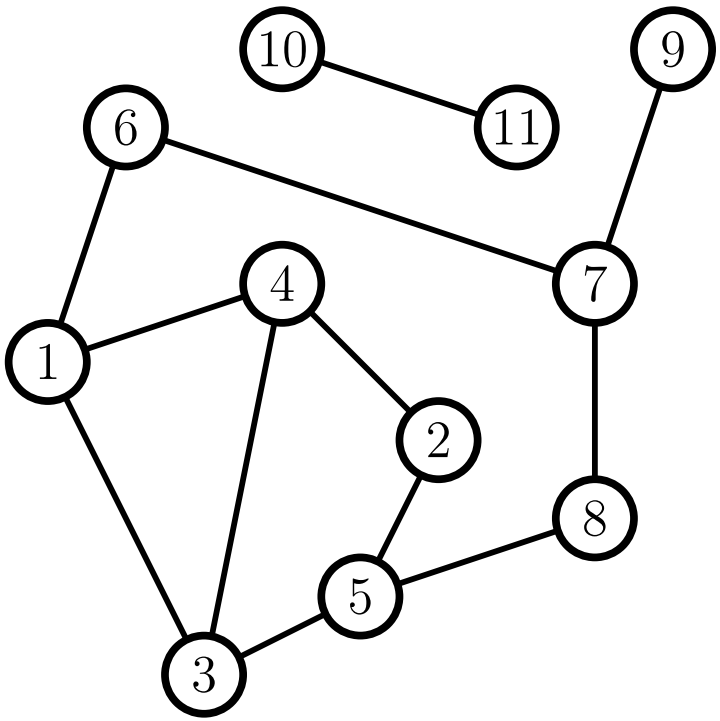
Ввод:

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



Содержание

Пример



Ввод:

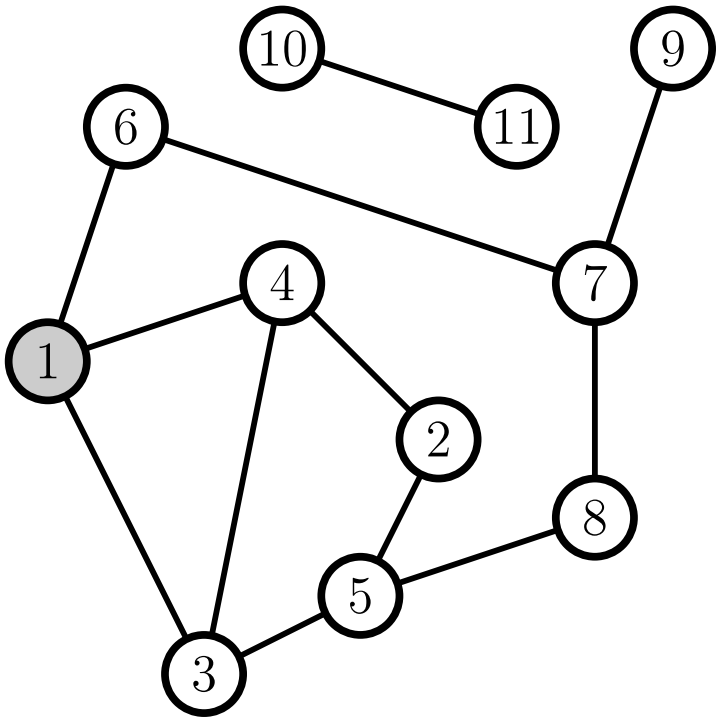
```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Задача: найдите кратчайшие расстояния от вершины 1 до всех остальных вершин.

Пример



Ввод:

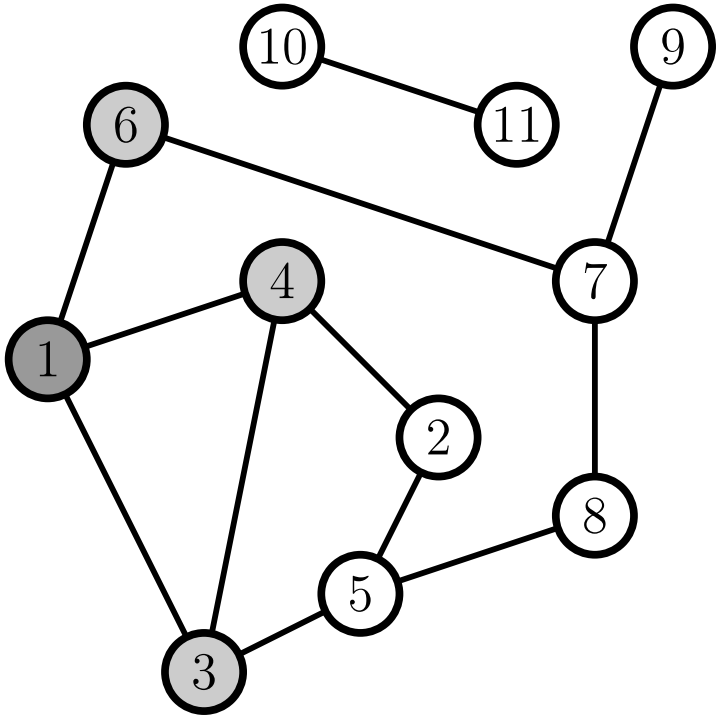
```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

1

Пример



1	3	4	6

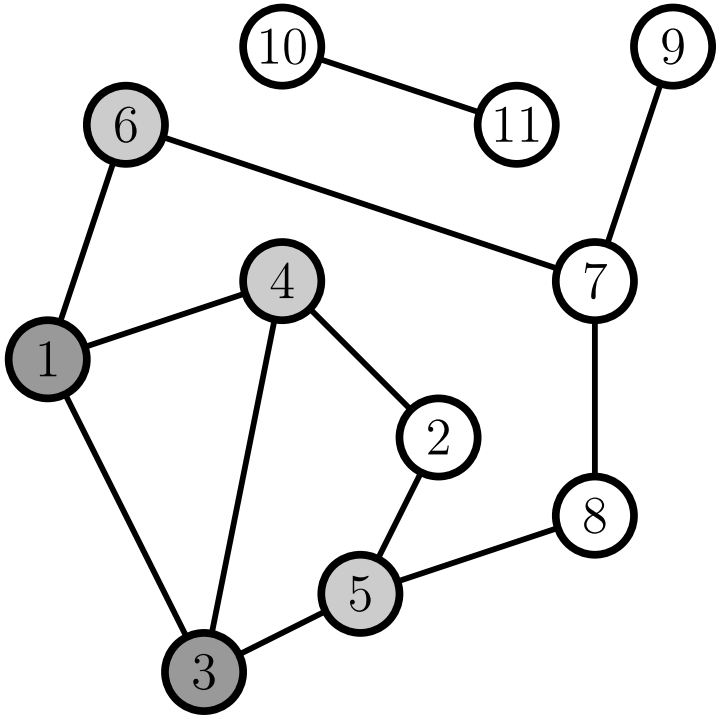
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5

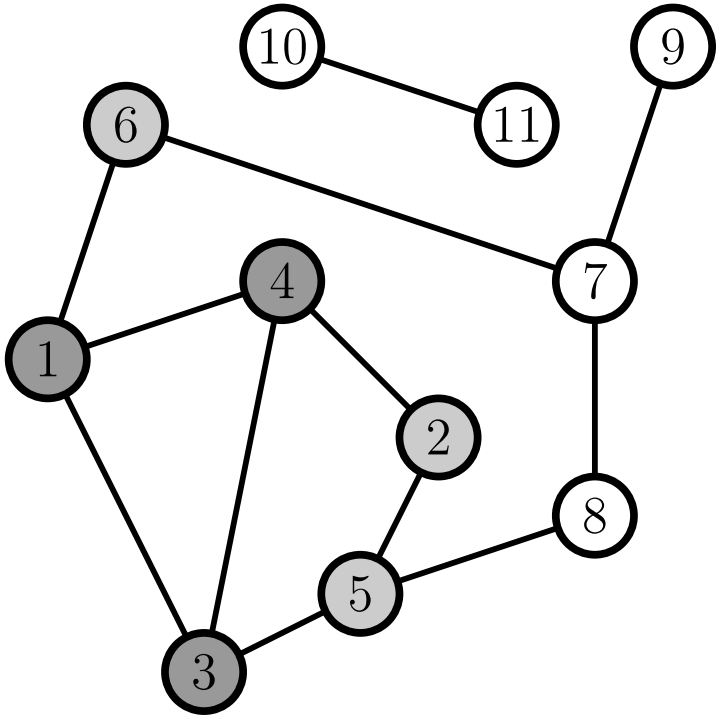
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5	2

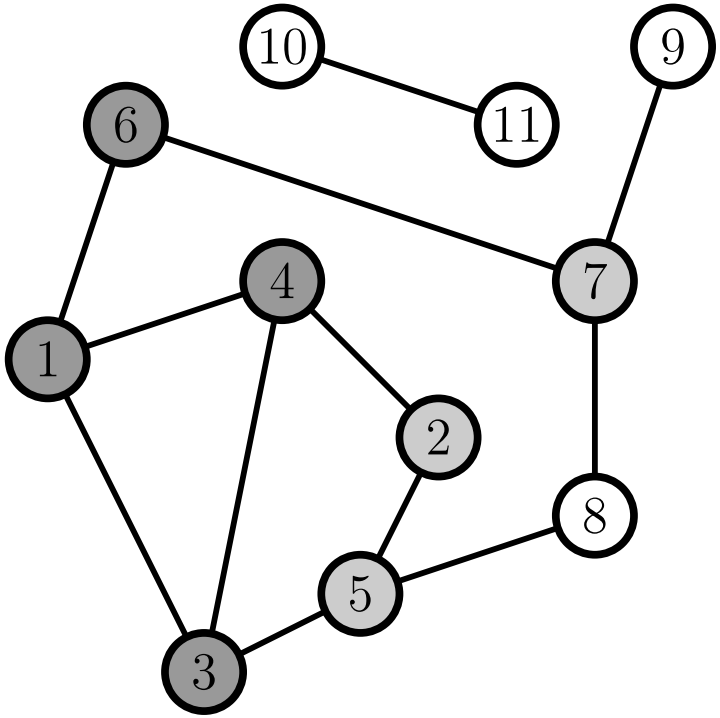
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5	2	7

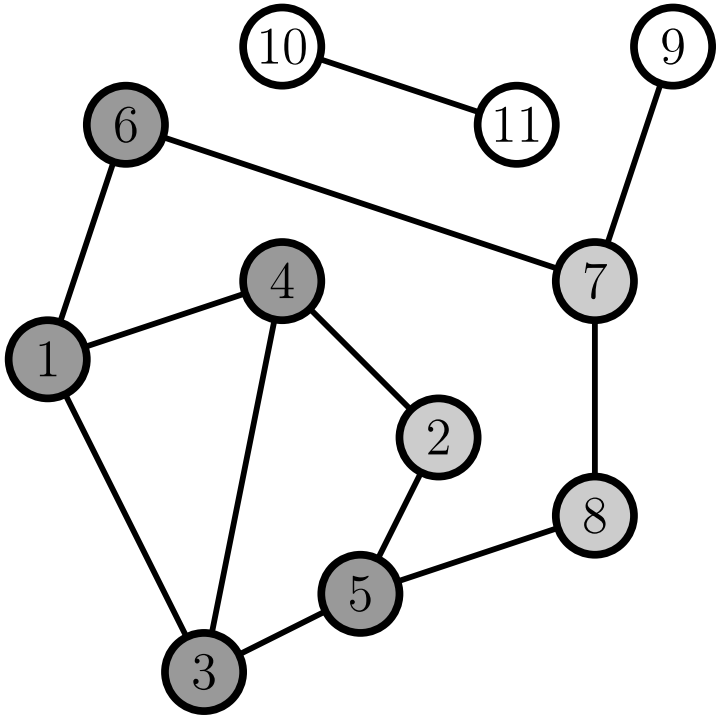
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5	2	7	8

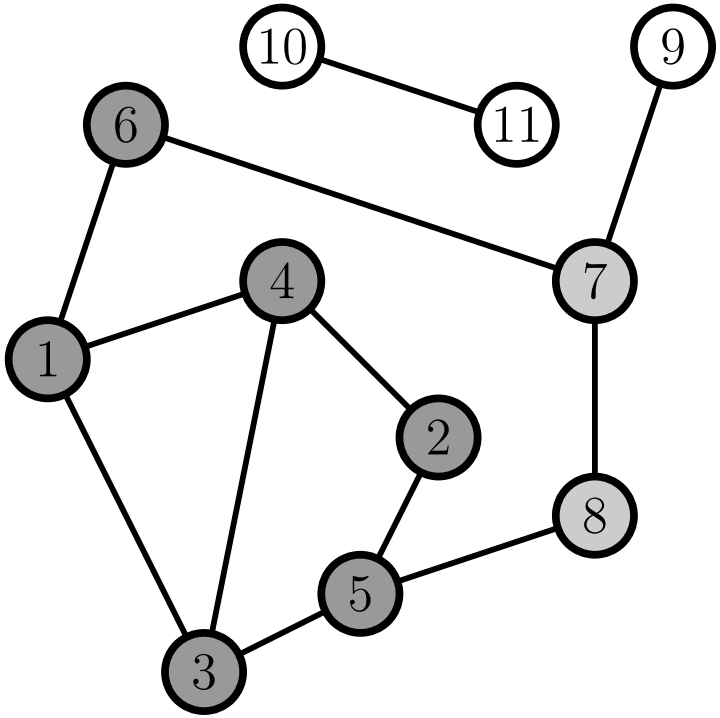
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример

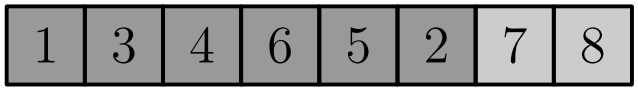


Ввод:

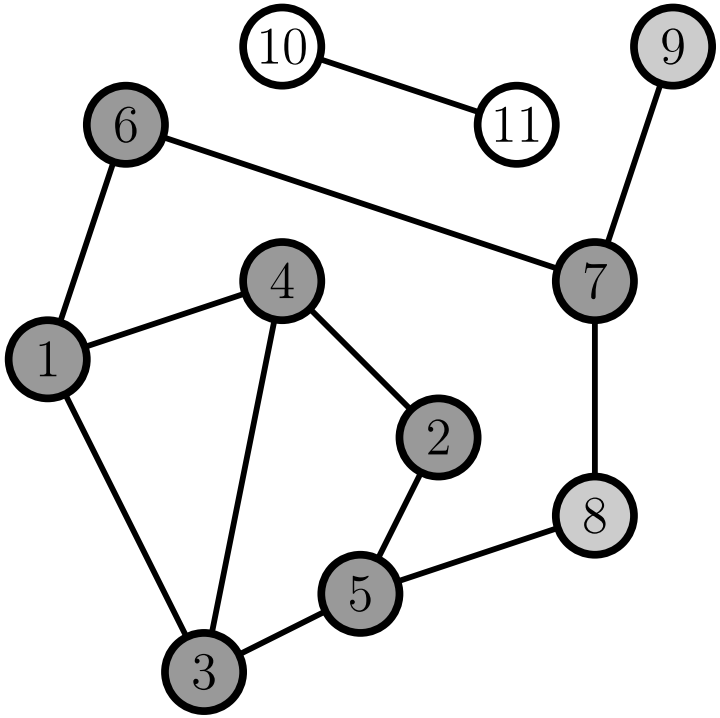
```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```



Пример



1	3	4	6	5	2	7	8	9

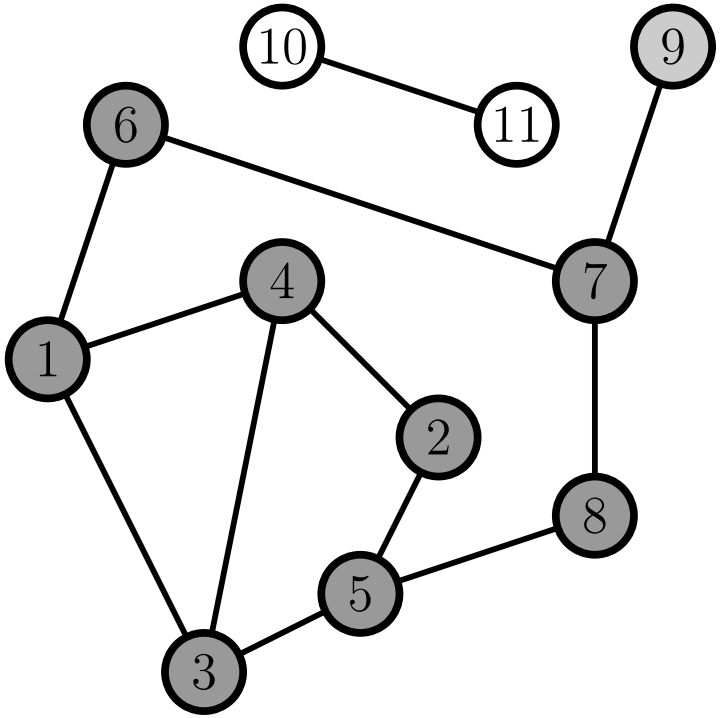
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5	2	7	8	9

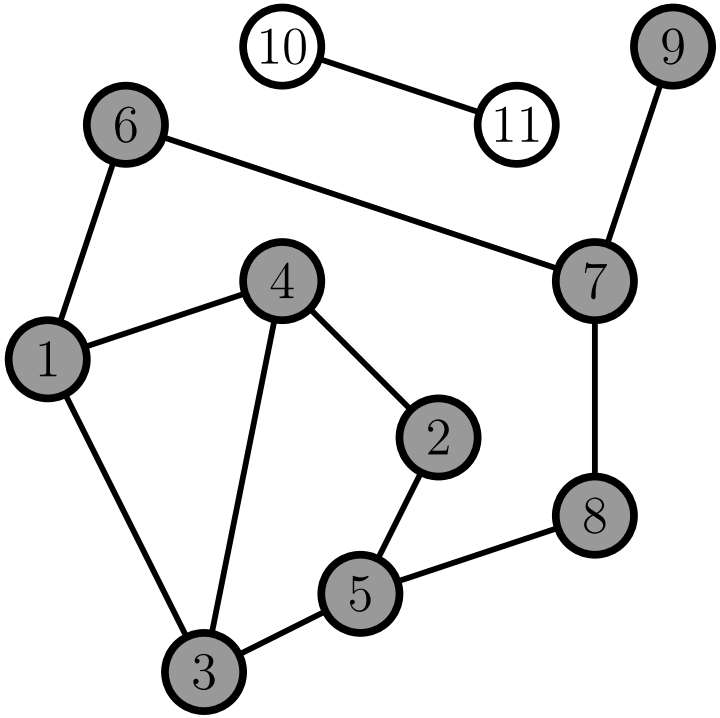
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Пример



1	3	4	6	5	2	7	8	9

Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Код breadth-first search

```
 1  /* ... */
 2  int const infinity = 1000000001;
 3  /* ... */
 4      vector <int> d (n, infinity);
 5      queue <int> q;
 6      d[0] = 0;
 7      q.push (0);
 8      while (!q.empty ()) {
 9          auto v = q.front ();
10          q.pop ();
11          cout << v + 1 << ": " << d[v] << endl;
12          for (int u = 0; u < n; u++) if (a[v][u]) {
13              if (d[u] == infinity) {
14                  d[u] = d[v] + 1;
15                  q.push (u);
16              }
17          }
18      }
19  /* ... */
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Код breadth-first search

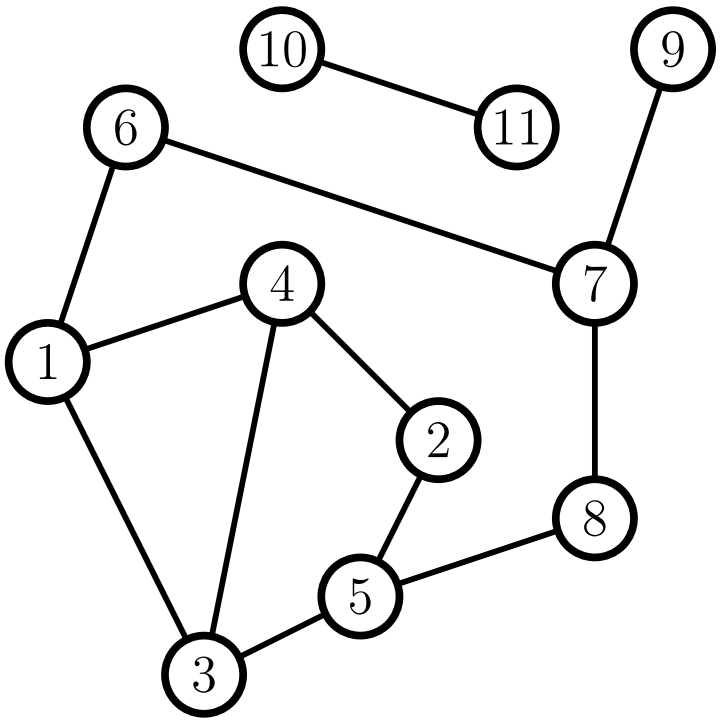
```
 1  /* ... */
 2  int const infinity = 1000000001;
 3  /* ... */
 4      vector <int> d (n, infinity);
 5      queue <int> q;
 6      d[0] = 0;
 7      q.push (0);
 8      while (!q.empty ()) {
 9          auto v = q.front ();
10          q.pop ();
11          cout << v + 1 << ": " << d[v] << endl;
12          for (auto u : adj[v]) {
13              if (d[u] == infinity) {
14                  d[u] = d[v] + 1;
15                  q.push (u);
16              }
17          }
18      }
19  /* ... */
```

Вывод:

```
1: 0
3: 1
4: 1
6: 1
5: 2
2: 2
7: 2
8: 3
9: 3
```

Содержание

Пример



Ввод:

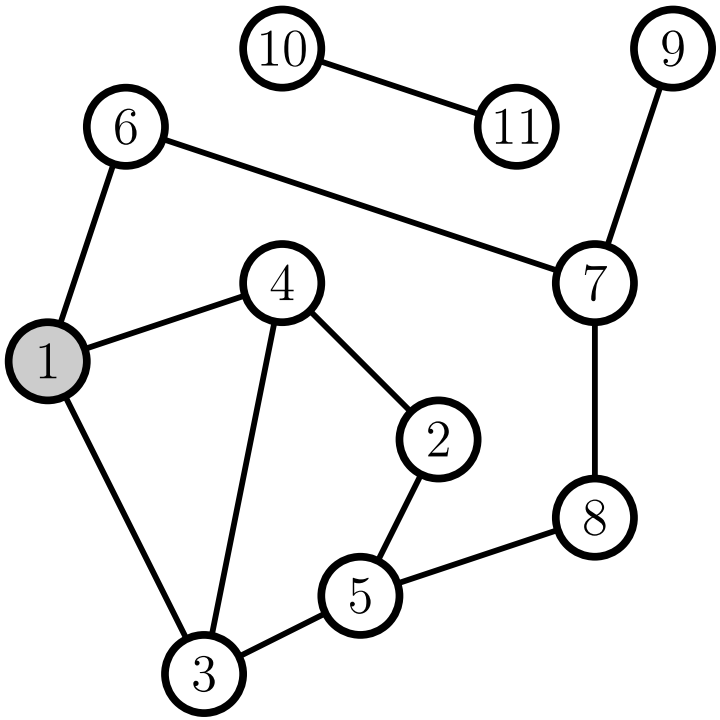
```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Задача: посетите все вершины, достижимые из вершины 1.

Пример



1

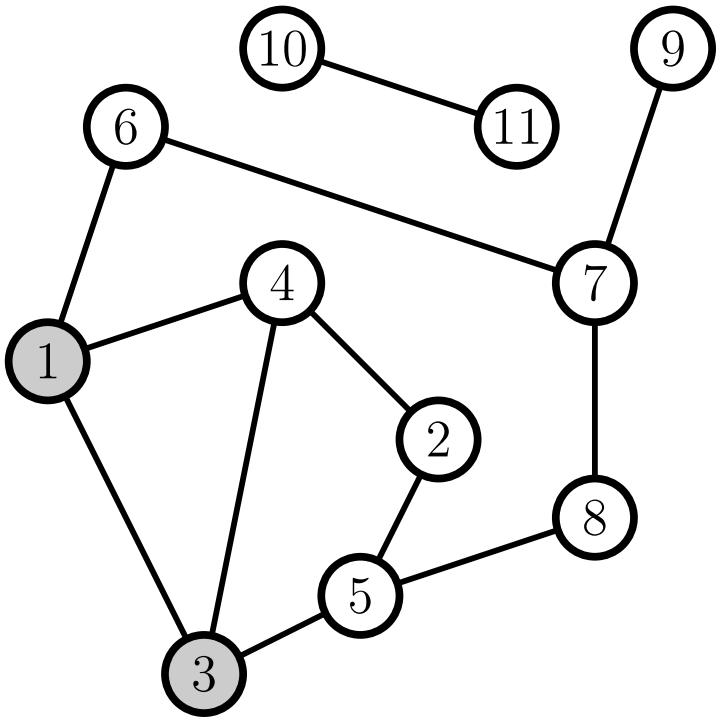
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3

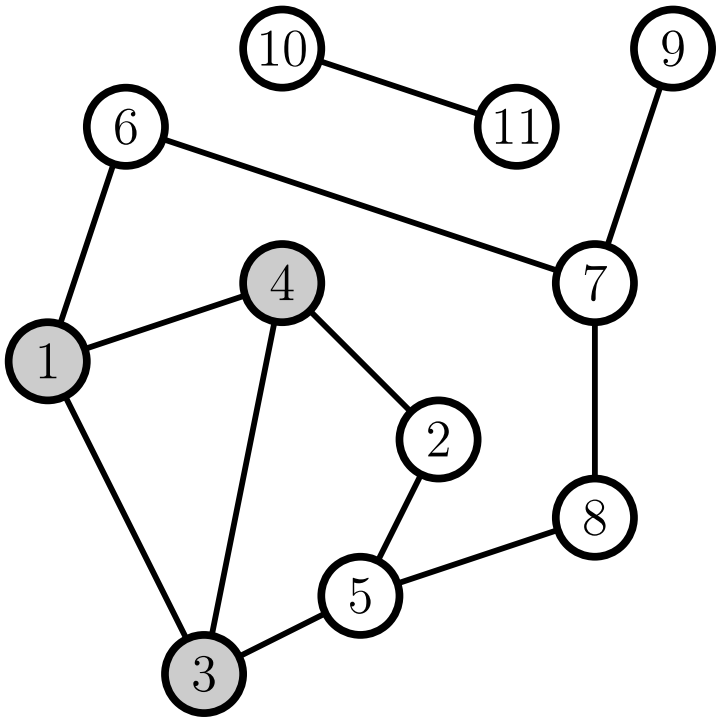
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4

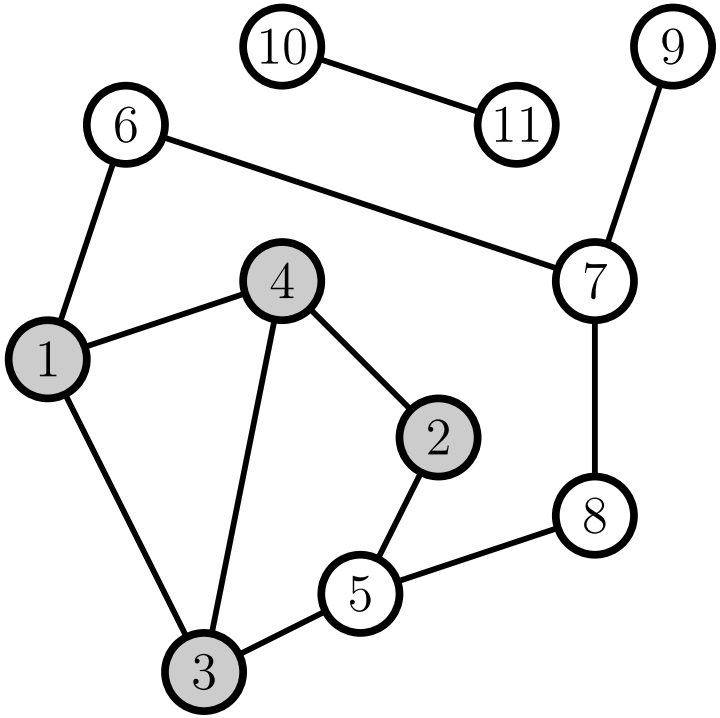
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2

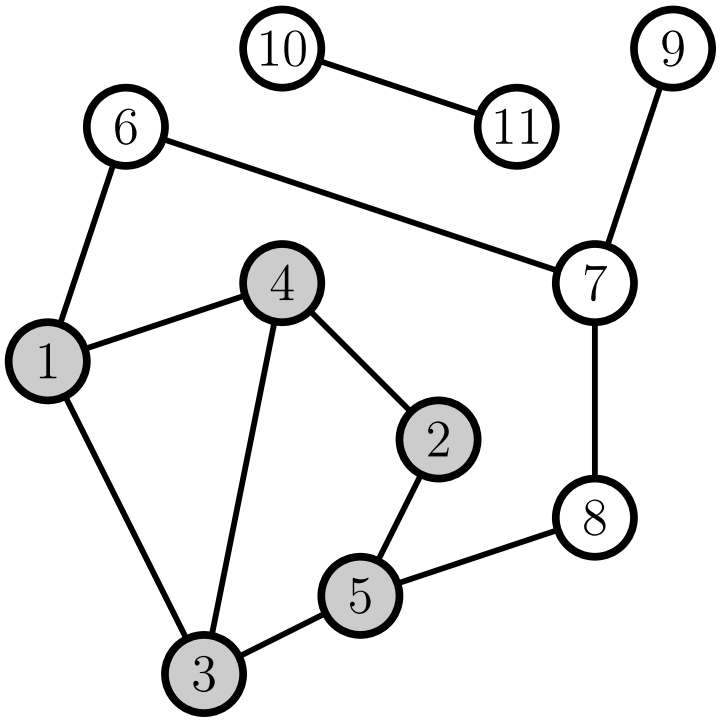
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5

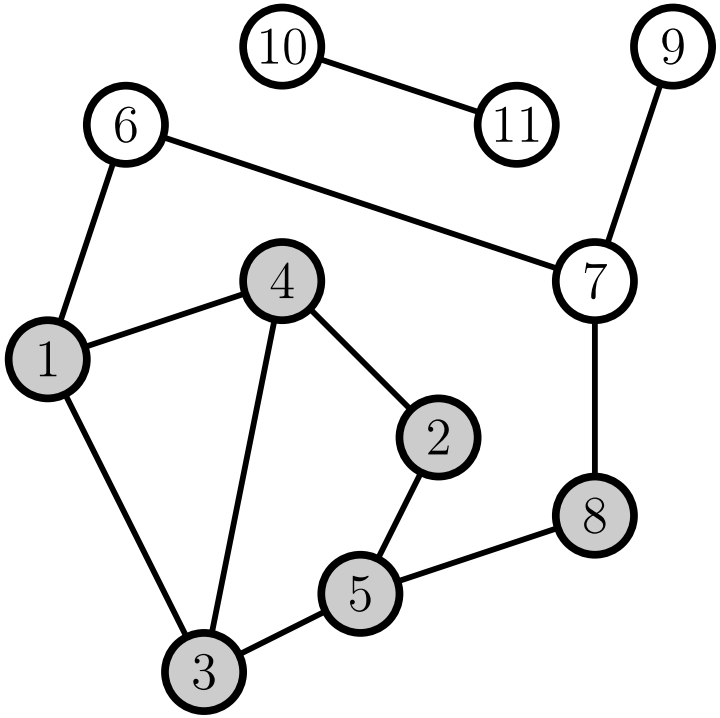
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8

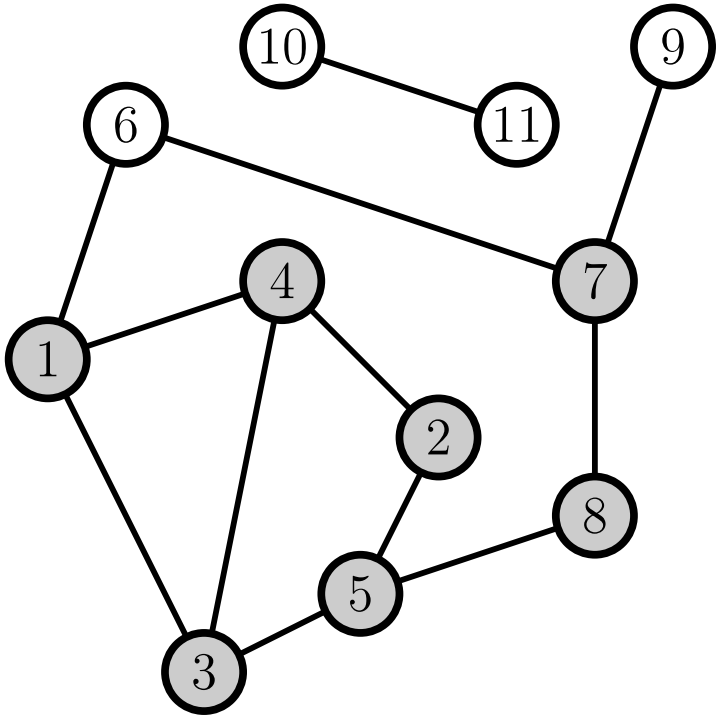
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8	7

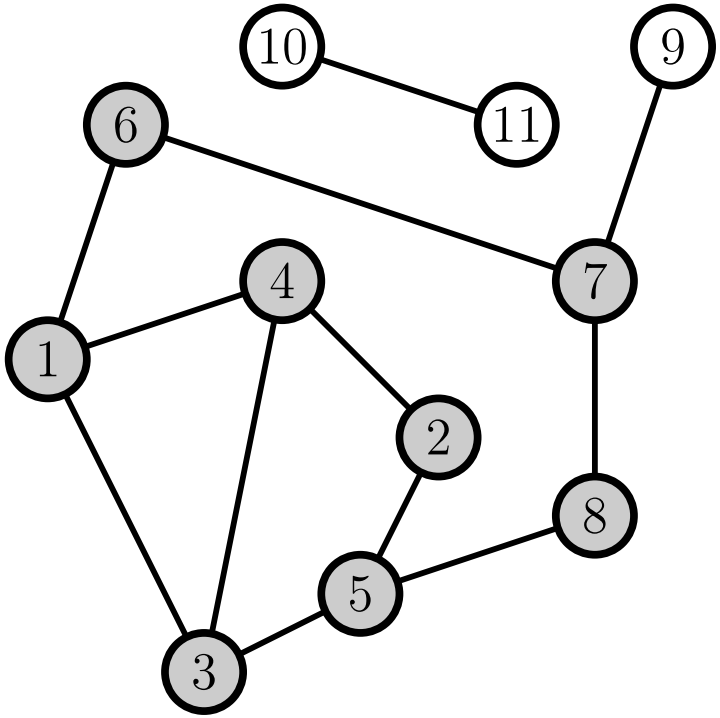
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8	7	6

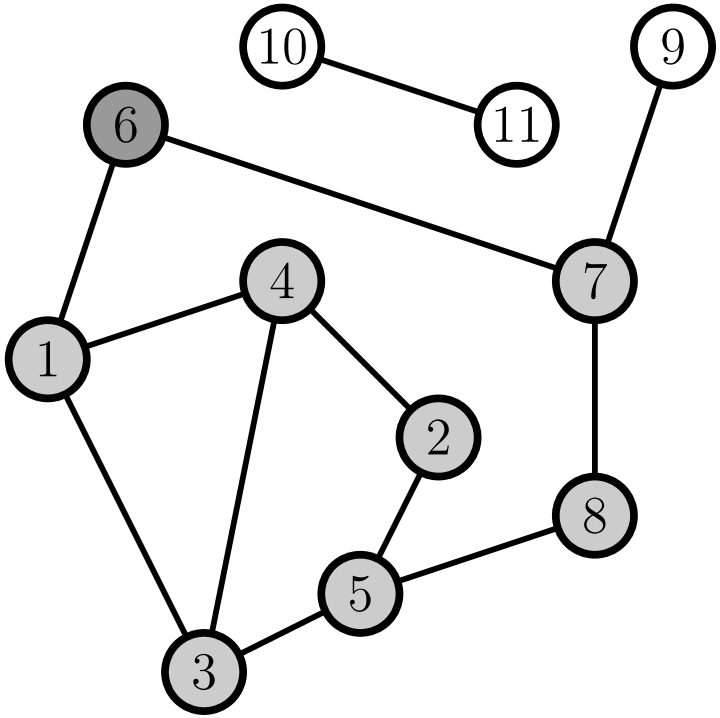
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8	7

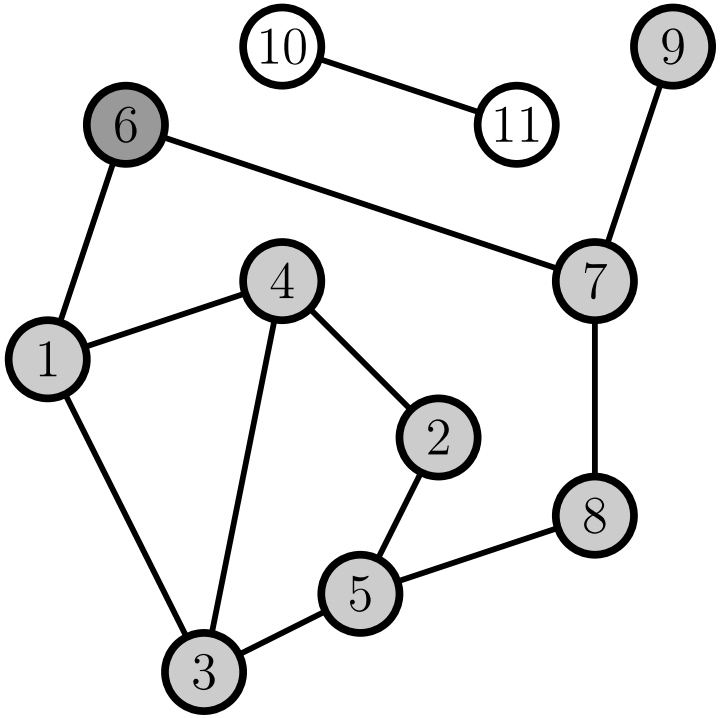
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8	7	9

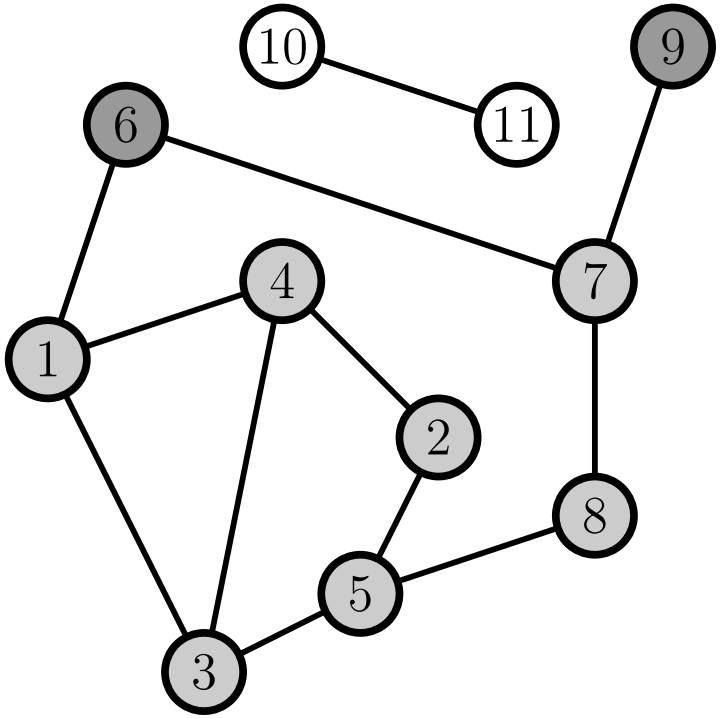
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8	7

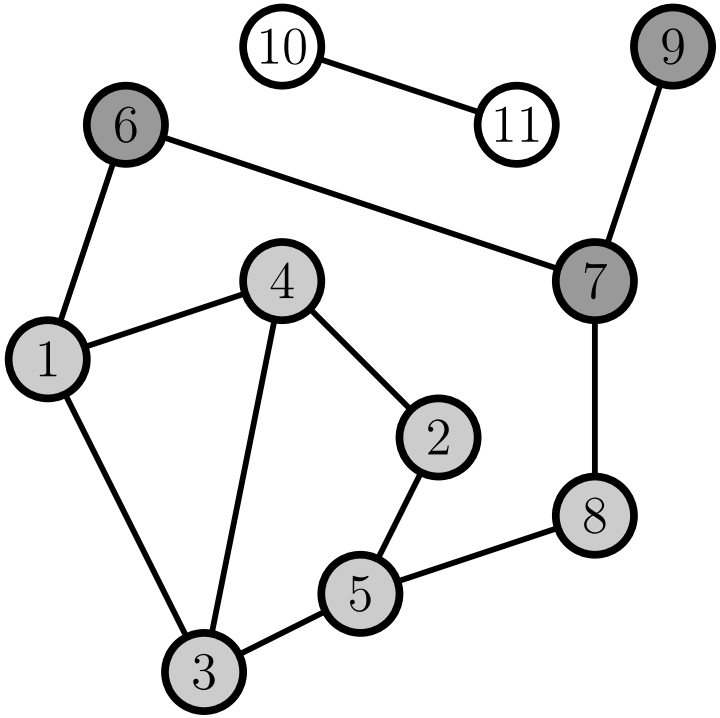
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5	8

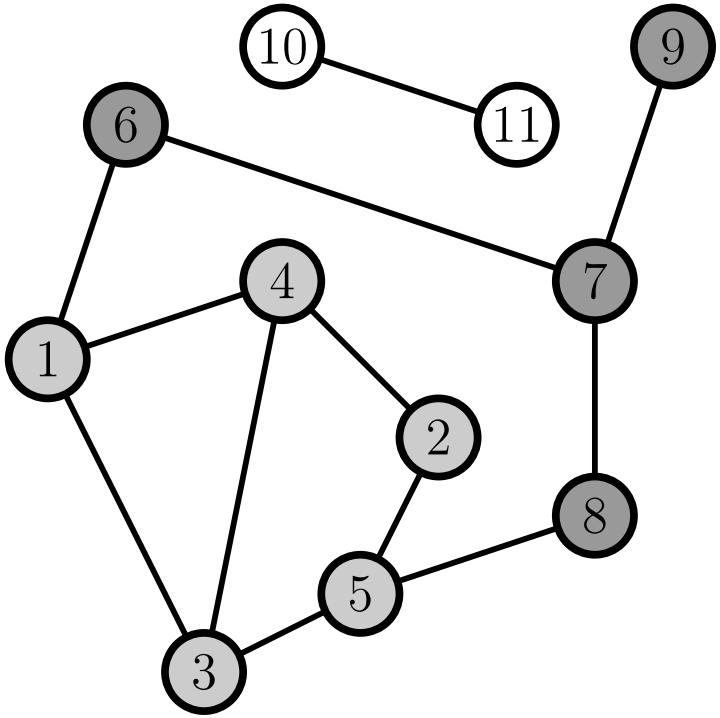
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2	5

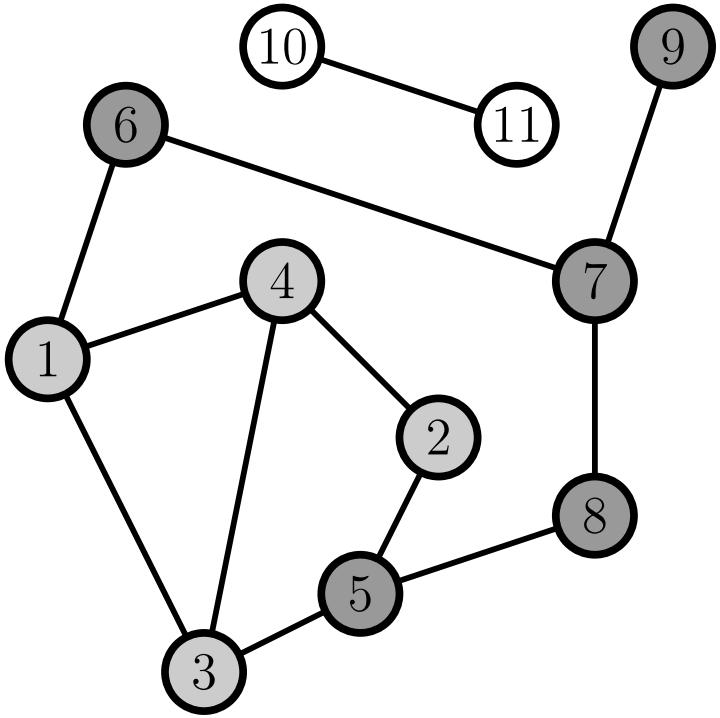
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4	2

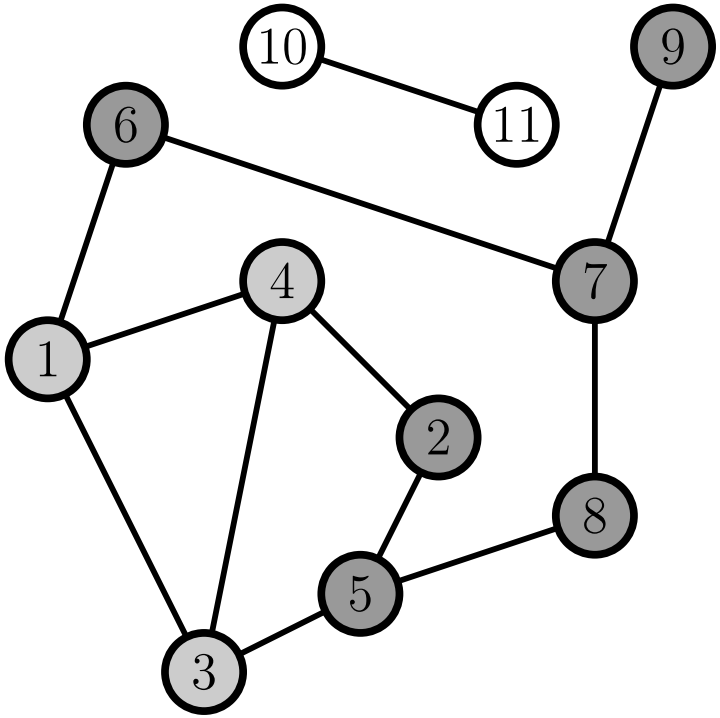
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3	4

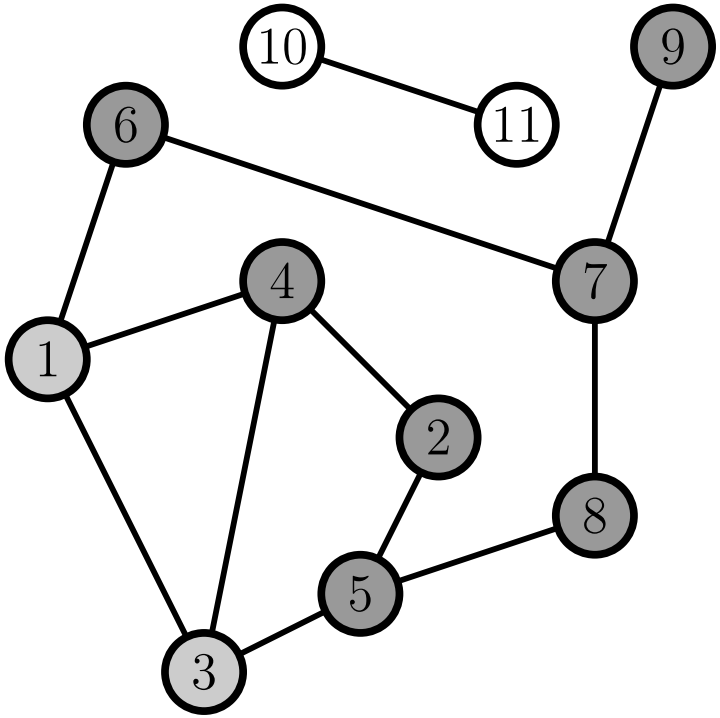
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1	3

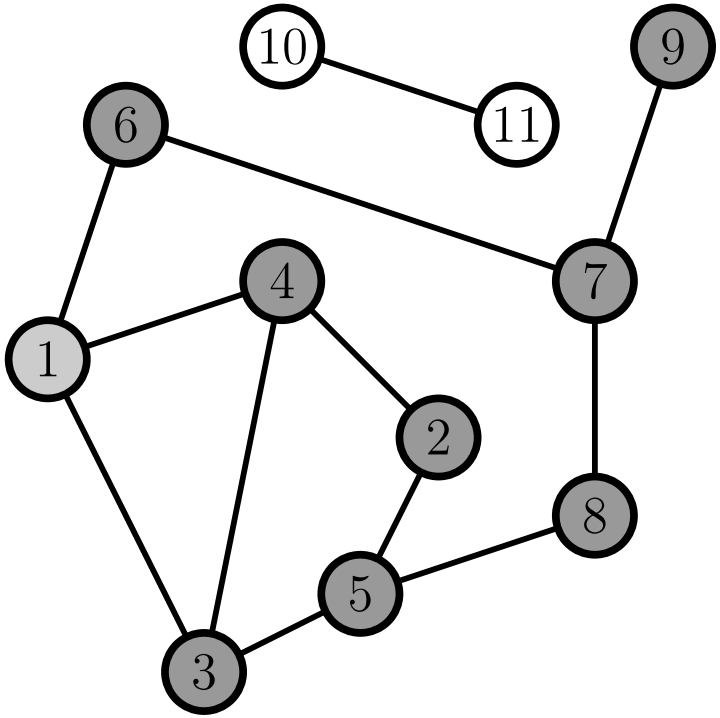
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



1

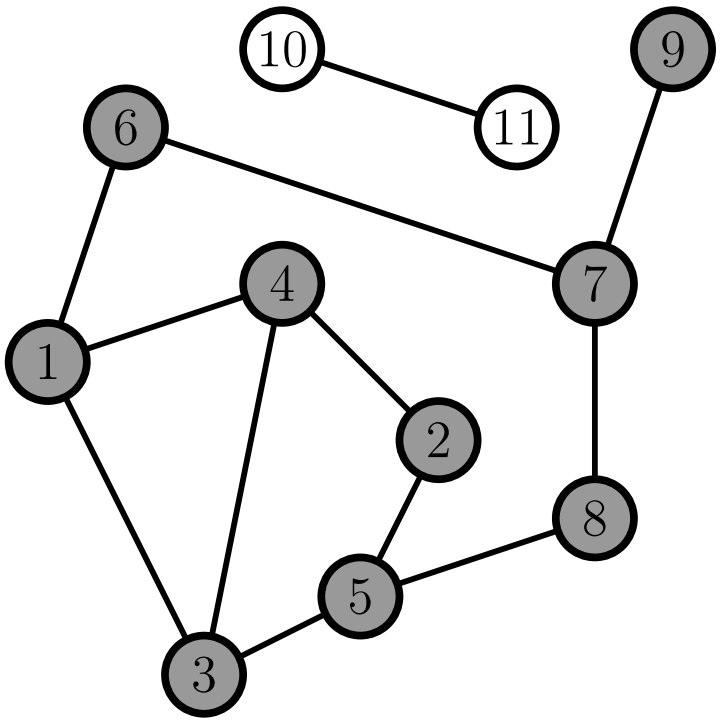
Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Пример



Ввод:

```
11 12
1 6
1 4
6 7
2 5
4 3
4 2
5 8
7 8
7 9
1 3
3 5
10 11
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Код depth-first search

```
 1  /* ... */
 2  vector <vector <int> > a;
 3  vector <bool> vis;
 4  int n, m;
 5
 6  void dfs (int v) {
 7      if (vis[v])
 8          return;
 9      vis[v] = true;
10      cout << "in  " << v + 1 << endl;
11      for (int u = 0; u < n; u++) if (a[v][u]) {
12          dfs (u);
13      }
14      cout << "out " << v + 1 << endl;
15  }
16  /* ... */
17      vis = vector <bool> (n);
18      dfs (0);
19  /* ... */
```

Вывод:

```
in  1
in  3
in  4
in  2
in  5
in  8
in  7
in  6
out 6
in  9
out 9
out 7
out 8
out 5
out 2
out 4
out 3
out 1
```

Код depth-first search

```
 1  /* ... */
 2  vector <vector <int> > adj;
 3  vector <bool> vis;
 4  int n, m;
 5
 6  void dfs (int v) {
 7      if (vis[v])
 8          return;
 9      vis[v] = true;
10      cout << "in  " << v + 1 << endl;
11      for (auto u : adj[v]) {
12          dfs (u);
13      }
14      cout << "out " << v + 1 << endl;
15  }
16  /* ... */
17      vis = vector <bool> (n);
18      dfs (0);
19  /* ... */
```

Вывод:

```
in  1
in  6
in  7
in  8
in  5
in  2
in  4
in  3
out 3
out 4
out 2
out 5
out 8
in  9
out 9
out 7
out 6
out 1
```

Содержание

Клетчатая доска

Пример 1:

- Есть клетчатая доска.
- Некоторые клетки заняты.
- Соседними считаются клетки, имеющие общую сторону.
- За один шаг можно перемещаться на соседнюю незанятую клетку.
- Определите, докуда можно добраться из начальной клетки.

Клетчатая доска

Пример 1:

- Есть клетчатая доска.
- Некоторые клетки заняты.
- Соседними считаются клетки, имеющие общую сторону.
- За один шаг можно перемещаться на соседнюю незанятую клетку.
- Определите, докуда можно добраться из начальной клетки.

Как будем решать:

- Граф: вершины — клетки, рёбра — переходы в соседей.

Клетчатая доска

Пример 1:

- Есть клетчатая доска.
- Некоторые клетки заняты.
- Соседними считаются клетки, имеющие общую сторону.
- За один шаг можно перемещаться на соседнюю незанятую клетку.
- Определите, докуда можно добраться из начальной клетки.

Как будем решать:

- Граф: вершины — клетки, рёбра — переходы в соседей.
- Нужно найти компоненту связности начальной клетки.

Клетчатая доска

Пример 1:

- Есть клетчатая доска.
- Некоторые клетки заняты.
- Соседними считаются клетки, имеющие общую сторону.
- За один шаг можно перемещаться на соседнюю незанятую клетку.
- Определите, докуда можно добраться из начальной клетки.

Как будем решать:

- Граф: вершины — клетки, рёбра — переходы в соседей.
- Нужно найти компоненту связности начальной клетки.
- Будем нумеровать вершины и рёбра?

Клетчатая доска — решение

- Граф: вершины — клетки, рёбра — переходы в соседей.
- Нужно найти компоненту связности начальной клетки.
- Вершина — это координаты $(\mathrm{row}, \mathrm{col})$.

Клетчатая доска — решение

- Граф: вершины — клетки, рёбра — переходы в соседей.
- Нужно найти компоненту связности начальной клетки.
- Вершина — это координаты (row, col).
- Рёбра:
  - в $(\text{row} - 1, \text{col})$,
  - в $(\text{row}, \text{col} - 1)$,
  - в $(\text{row} + 1, \text{col})$,
  - в $(\text{row}, \text{col} + 1)$.

Клетчатая доска — решение

- Граф: вершины — клетки, рёбра — переходы в соседей.
- Нужно найти компоненту связности начальной клетки.

```
 1  int const dirs = 4;
 2  int const dRow [dirs] = {-1,  0, +1,  0};
 3  int const dCol [dirs] = { 0, -1,  0, +1};
 4  /* ... */
 5  void dfs (int row, int col) {
 6      if (vis[row][col]) return;
 7      vis[row][col] = true;
 8      for (int dir = 0; dir < dirs; dir++) {
 9          int nRow = row + dRow[dir];
10          int nCol = col + dCol[dir];
11          if (0 <= nRow && nRow < rows &&
12              0 <= nCol && nCol < cols &&
13              !wall[nRow][nCol]) {
14                  dfs (nRow, nCol);
15          }
16      }
17  }
```

Шахматный конь

Пример 2:

- На шахматной доске $n \times n$ в клетке A стоит конь.
- За какое минимальное количество ходов он может попасть в клетку B?

Шахматный конь

Пример 2:

- На шахматной доске $n \times n$ в клетке A стоит конь.
- За какое минимальное количество ходов он может попасть в клетку B?

Как будем решать:

- Граф: вершины — клетки, рёбра — ходы коня.

Шахматный конь

Пример 2:

- На шахматной доске $n \times n$ в клетке A стоит конь.
- За какое минимальное количество ходов он может попасть в клетку B?

Как будем решать:

- Граф: вершины — клетки, рёбра — ходы коня.
- Нужно запустить поиск в ширину из клетки A.

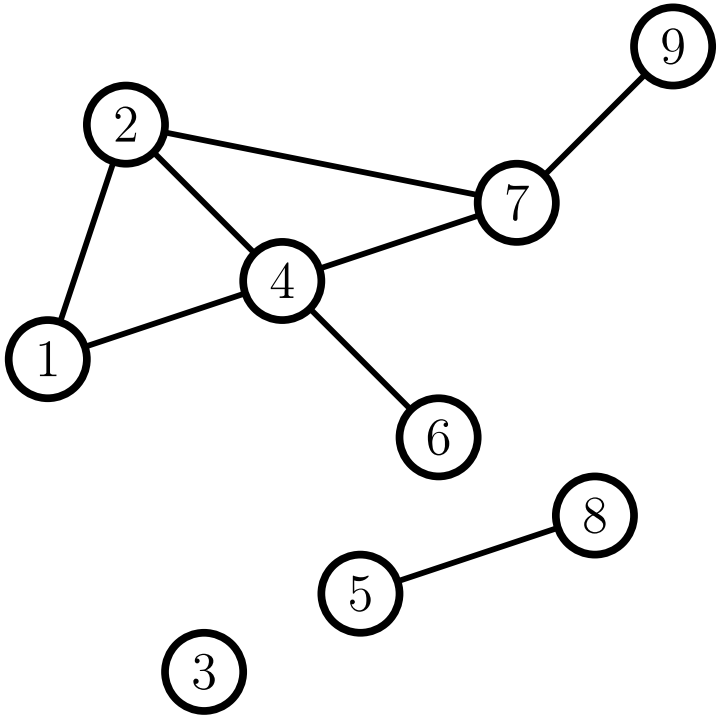
Шахматный конь — решение

- Граф: вершины — клетки, рёбра — ходы коня.
- Нужно запустить поиск в ширину из клетки A.

```
 1  int const dirs = 8;
 2  int const dRow [dirs] = {-1, -2, -2, -1, +1, +2, +2, +1};
 3  int const dCol [dirs] = {-2, -1, +1, +2, +2, +1, -1, -2};
 4  /* ... */
 5      while (!q.empty ()) {
 6          int row = q.front ().first;
 7          int col = q.front ().second;
 8          q.pop ();
 9          for (int dir = 0; dir < dirs; dir++) {
10              int nRow = row + dRow[dir];
11              int nCol = col + dCol[dir];
12              if (0 <= nRow && nRow < rows &&
13                  0 <= nCol && nCol < cols &&
14                  d[nRow][nCol] == infinity) {
15                      d[nRow][nCol] = d[row][col] + 1;
16                      q.push (make_pair (nRow, nCol));
17              }
18      }
```

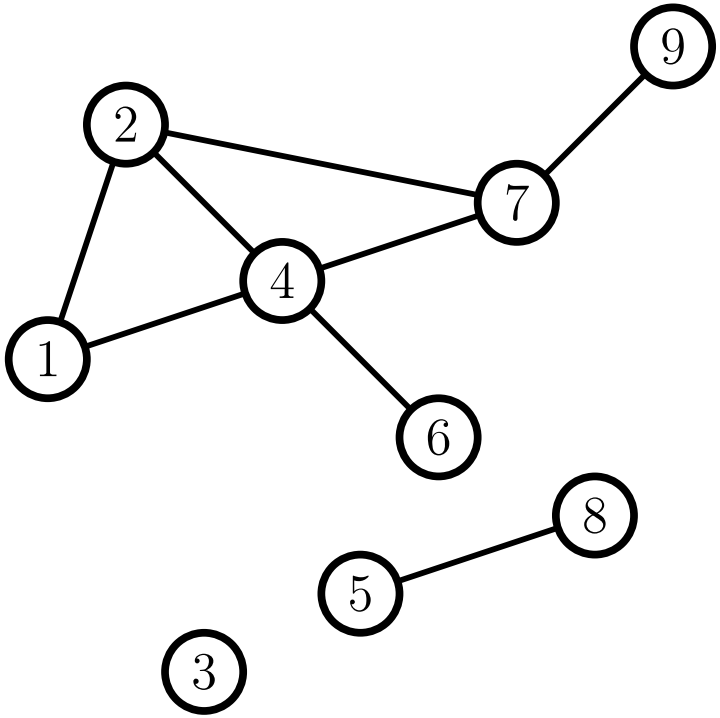
Содержание

Связность



Дан неориентированный граф. Проверьте, связен ли он.

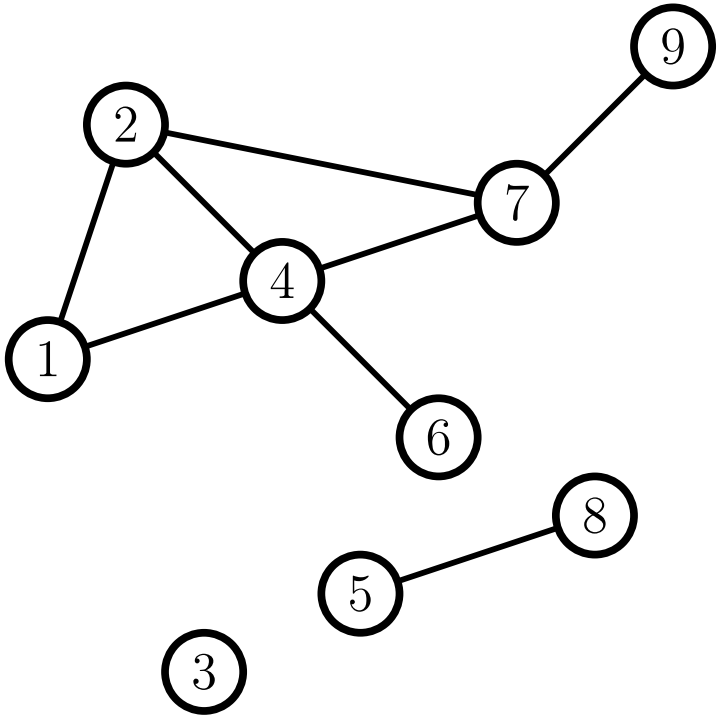
Связность



Дан неориентированный граф. Проверьте, связен ли он.
Как будем решать:

- Запустим поиск из любой вершины.

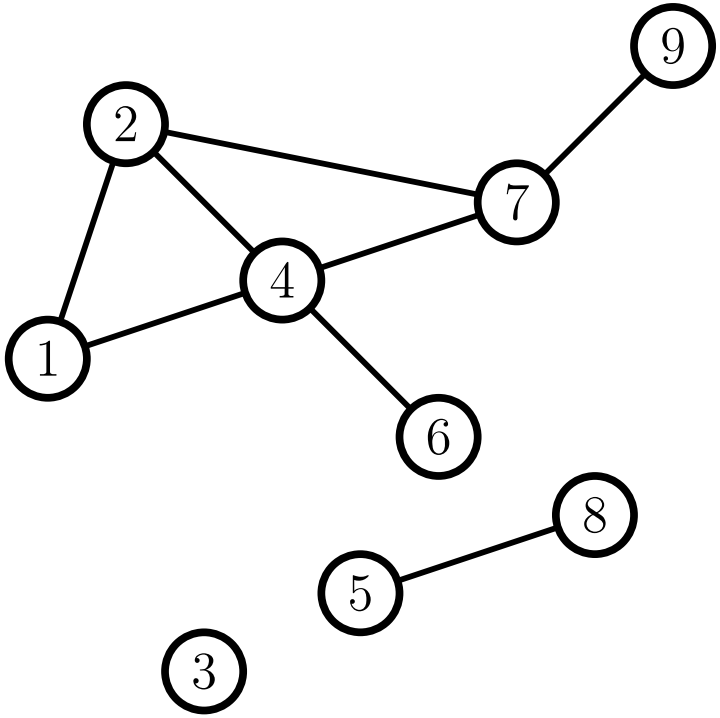
Связность



Дан неориентированный граф. Проверьте, связен ли он.
Как будем решать:

- Запустим поиск из любой вершины.
- Можно поиск в ширину, можно поиск в глубину.

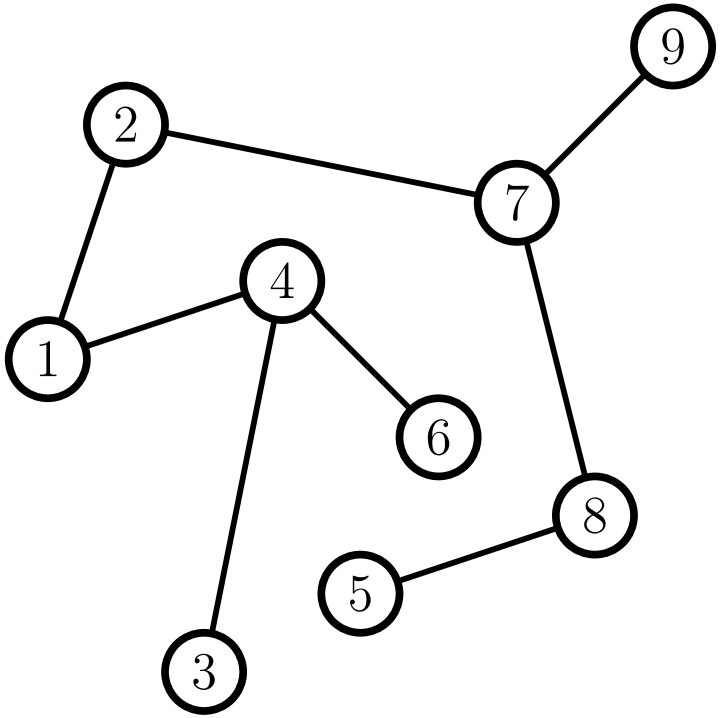
Связность



Дан неориентированный граф. Проверьте, связен ли он.
Как будем решать:

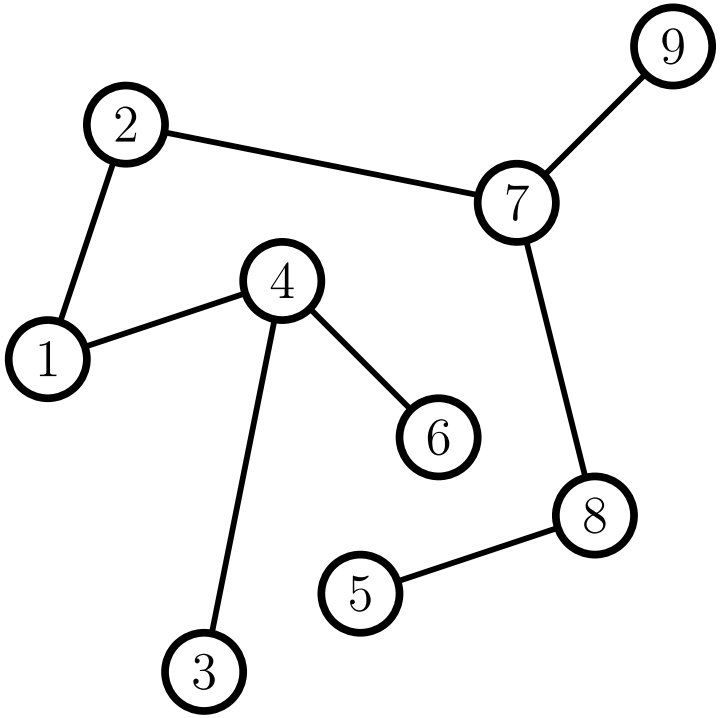
- Запустим поиск из любой вершины.
- Можно поиск в ширину, можно поиск в глубину.
- Проверим, что мы посетили все вершины.

Дерево



Дан неориентированный граф. Проверьте, является ли он деревом.

Дерево

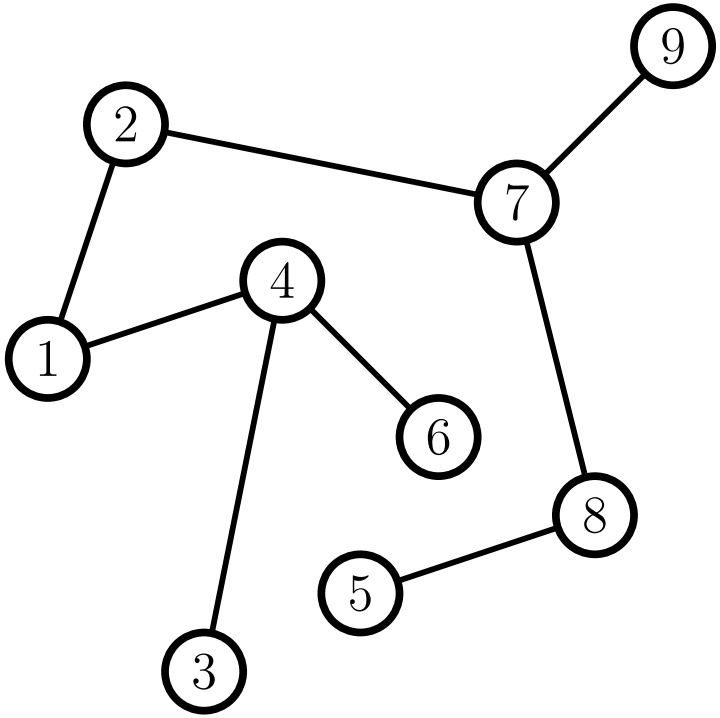


Дан неориентированный граф. Проверьте, является ли он деревом.

Как будем решать:

- Проверим связность.

Дерево



Дан неориентированный граф. Проверьте, является ли он деревом.

Как будем решать:

- Проверим связность.
- Проверим, что $|V| = |E| + 1$.

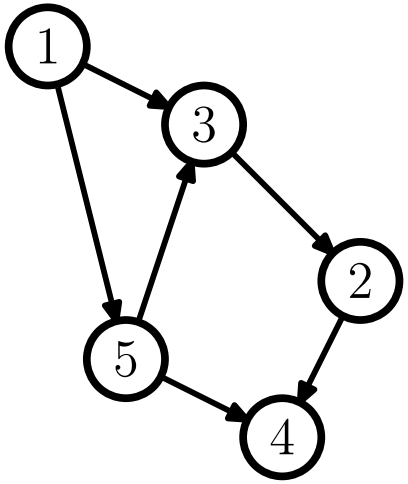
Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.

Пример 1:



Ввод:

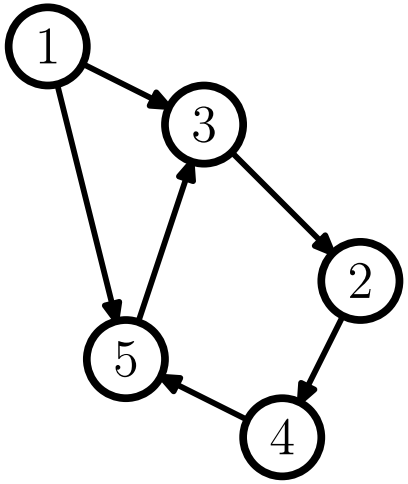
```
5 6
1 5
5 3
3 2
2 4
5 4
1 3
```

Вывод:

```
YES
1 5 3 2 4
```

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Пример 2:



Ввод:

```
5 6
1 5
5 3
3 2
2 4
4 5
1 3
```

Вывод:

```
NO
```

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Как будем решать:

- Посещаем вершины поиском в глубину.

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Как будем решать:

- Посещаем вершины поиском в глубину.
- Пусть мы находимся в какой-то вершине v.

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Как будем решать:

- Посещаем вершины поиском в глубину.
- Пусть мы находимся в какой-то вершине v.
- Условимся, что, когда вершина посещена, она выведена, то есть добавлена в порядок.

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Как будем решать:

- Посещаем вершины поиском в глубину.
- Пусть мы находимся в какой-то вершине v.
- Условимся, что, когда вершина посещена, она выведена, то есть добавлена в порядок.
- Посетим все вершины u, из которых есть дуга $u \to v$.

Топологическая сортировка

Дан ориентированный граф. Расположите его вершины в таком порядке, чтобы все дуги следовали от более ранних вершин к более поздним. Или выясните, что это невозможно.
Как будем решать:

- Посещаем вершины поиском в глубину.
- Пусть мы находимся в какой-то вершине v.
- Условимся, что, когда вершина посещена, она выведена, то есть добавлена в порядок.
- Посетим все вершины u, из которых есть дуга $u \to v$.
- После этого выведем вершину v, тем самым выполнив наше условие.

Топологическая сортировка

Как найти все такие u, для которых $u \to v$?

Топологическая сортировка

Как найти все такие u, для которых $u \to v$?

- Можно хранить не сами рёбра $(u \to v)$, а обратные к ним $(v \to u)$.

Топологическая сортировка

Как найти все такие u, для которых $u \to v$?

- Можно хранить не сами рёбра $(u \to v)$, а обратные к ним $(v \to u)$.
- Можно найти обратный порядок, а потом его перевернуть.

Топологическая сортировка

Как выяснить, что ответ не существует?

Топологическая сортировка

Как выяснить, что ответ не существует?

- У вершины три состояния:
  0. ещё не посещали (fresh),
  1. начали посещать и не закончили (busy),
  2. уже посетили (done).

Топологическая сортировка

Как выяснить, что ответ не существует?

- У вершины три состояния:
  0. ещё не посещали (fresh),
  1. начали посещать и не закончили (busy),
  2. уже посетили (done).
- Если мы пришли посещать вершину в состоянии fresh, запускаем алгоритм в ней.

Топологическая сортировка

Как выяснить, что ответ не существует?

- У вершины три состояния:
  0. ещё не посещали (fresh),
  1. начали посещать и не закончили (busy),
  2. уже посетили (done).
- Если мы пришли посещать вершину в состоянии fresh, запускаем алгоритм в ней.
- Если мы пришли посещать вершину в состоянии done, ничего не делаем с ней.

Топологическая сортировка

Как выяснить, что ответ не существует?

- У вершины три состояния:
  0. ещё не посещали (fresh),
  1. начали посещать и не закончили (busy),
  2. уже посетили (done).
- Если мы пришли посещать вершину в состоянии fresh, запускаем алгоритм в ней.
- Если мы пришли посещать вершину в состоянии done, ничего не делаем с ней.
- Если мы пришли посещать вершину в состоянии busy, значит, мы нашли цикл, и ответ не существует.

Топологическая сортировка — код

```
1   vector <vector <int> > adj;
2   vector <int> vis;
3   vector <int> order;
4   int n, m;
5
6   enum {fresh, busy, done};
7
8   bool dfs (int v) {
9       if (vis[v] != fresh)
10          return vis[v] == done;
11      vis[v] = busy;
12      for (auto u : adj[v])
13          if (!dfs (u))
14              return false;
15      order.push_back (v);
16      vis[v] = done;
17      return true;
18  }
```

Топологическая сортировка — код

```
 1       cin >> n >> m;
 2       adj = vector <vector <int> > (n);
 3       for (int j = 0; j < m; j++) {
 4           int u, v;
 5           cin >> u >> v;
 6           u -= 1;
 7           v -= 1;
 8           adj[v].push_back (u);
 9       }
10
11       vis = vector <int> (n);
12       bool ok = true;
13       for (int v = 0; v < n; v++)
14           ok &= dfs (v);
15
16       if (ok) {
17           cout << "YES" << endl;
18           for (auto v : order)
19               cout << v + 1 << " ";
20           cout << endl;
21       }
22       else
23           cout << "NO" << endl;
```

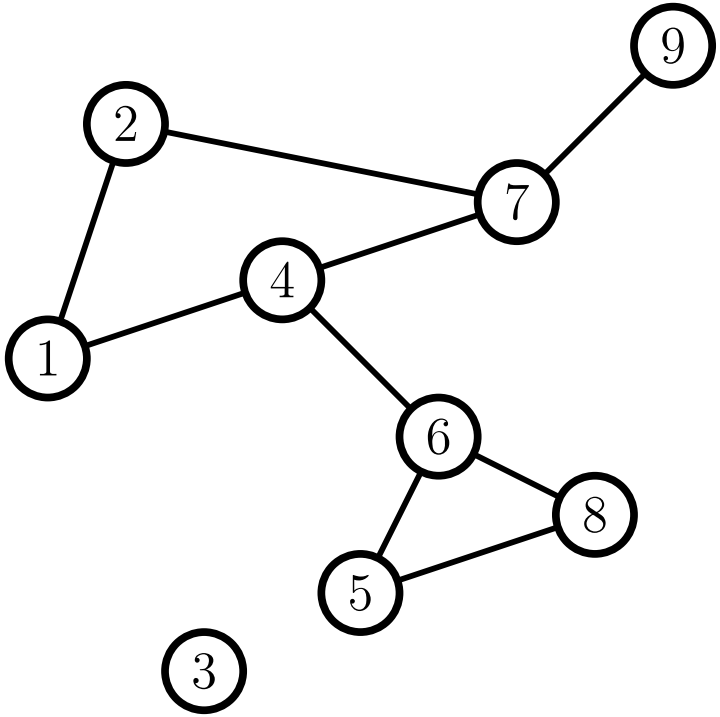
Мосты

Дан неориентированный граф. Найдите все мосты в нём: рёбра, при удалении которых количество компонент связности увеличивается.

Мосты

Дан неориентированный граф. Найдите все мосты в нём: рёбра, при удалении которых количество компонент связности увеличивается.

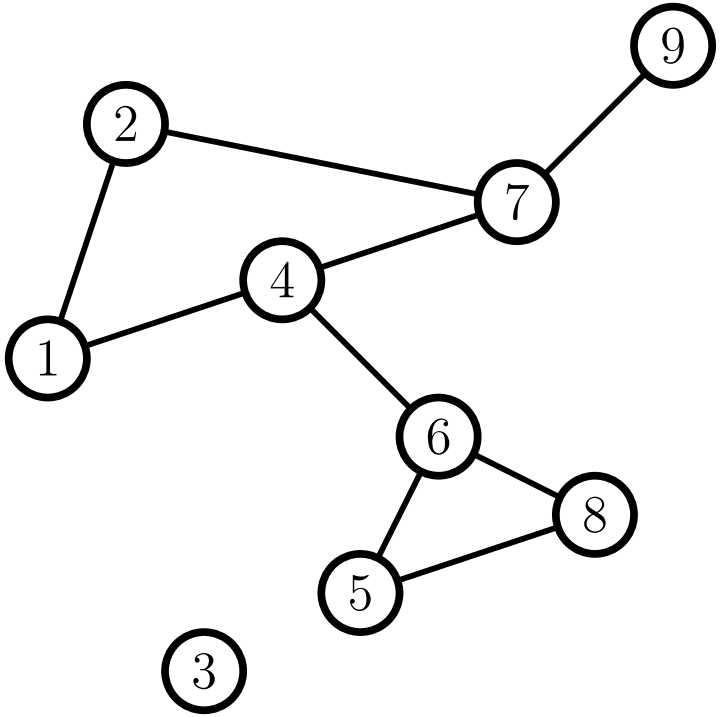
Пример:



Мосты

Дан неориентированный граф. Найдите все мосты в нём: рёбра, при удалении которых количество компонент связности увеличивается.

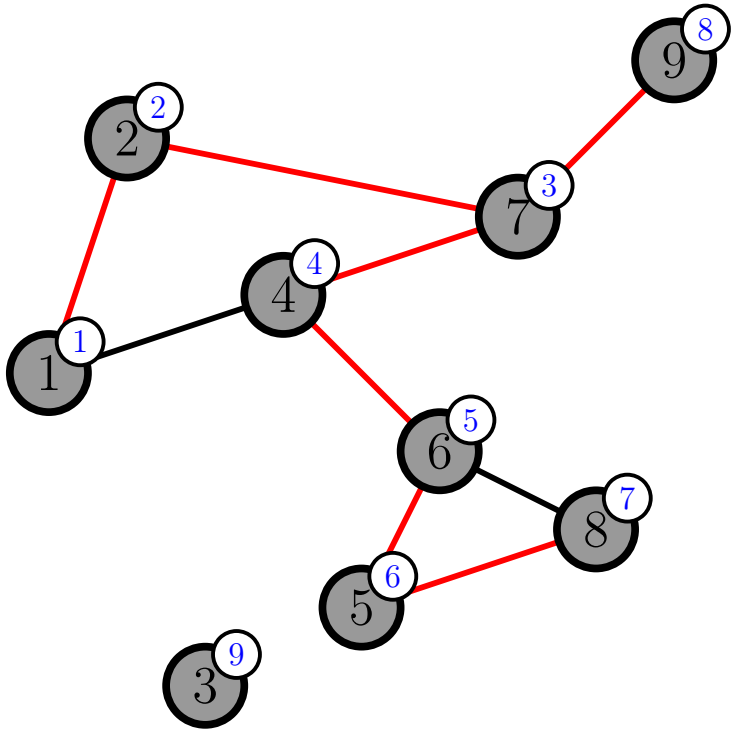
Пример:



Здесь мосты — рёбра 4–6 и 7–9.

Мосты

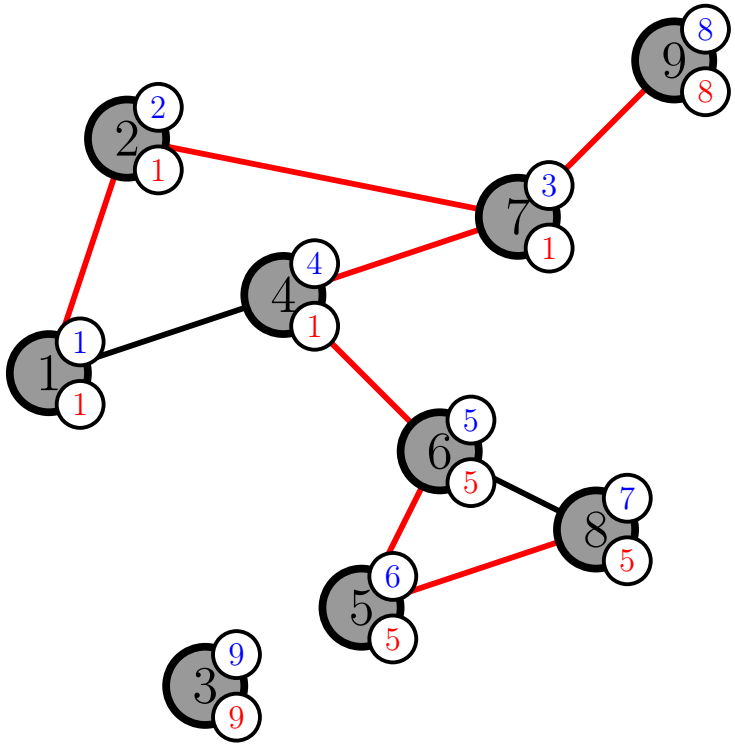
Идея решения:



Запустим поиск в глубину от всех вершин, запишем время входа в каждой вершине (синее число).

Мосты

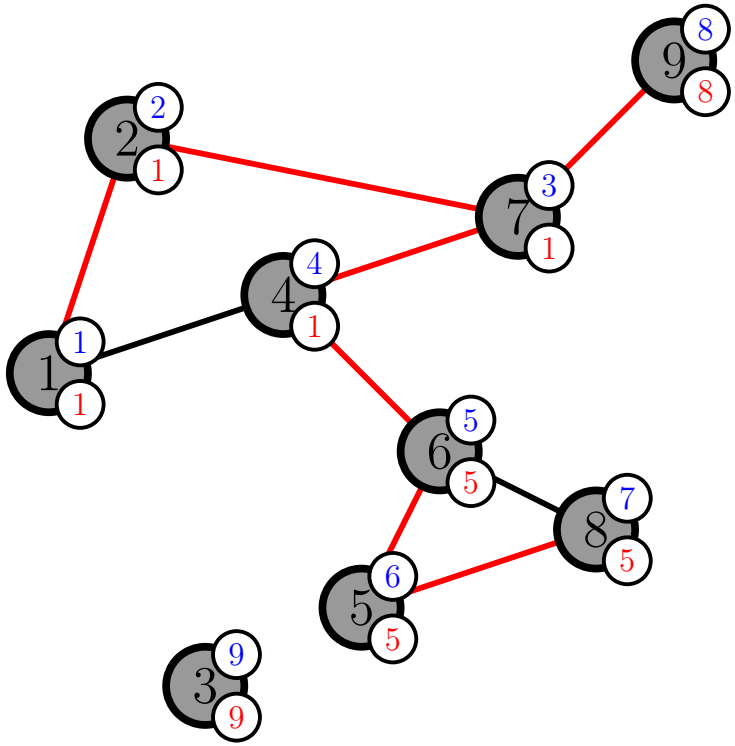
Идея решения:



Для каждой вершины рассмотрим её поддерево в поиске в глубину. Все рёбра, не вошедшие в дерево — обратные рёбра.

Мосты

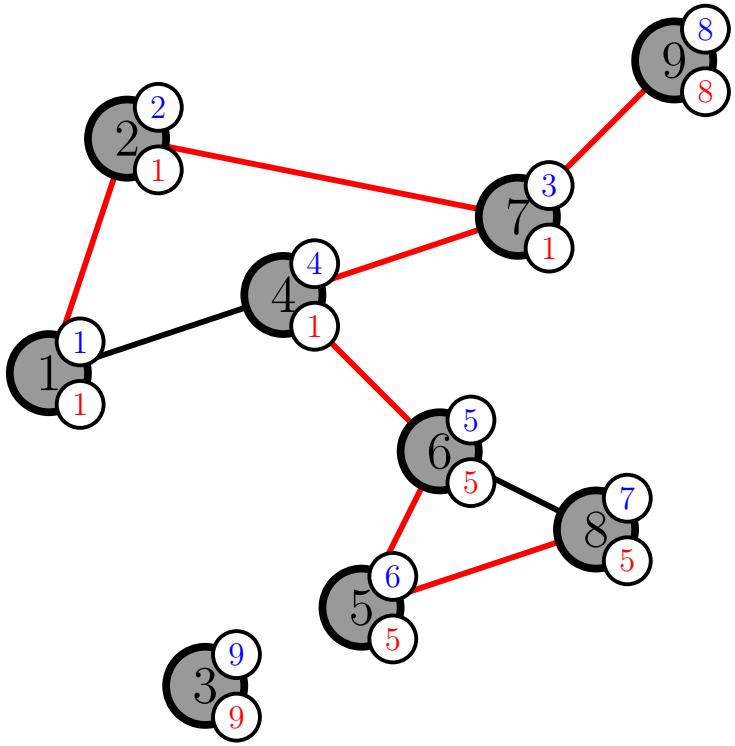
Идея решения:



Для каждой вершины рассмотрим её поддерево в поиске в глубину. Все рёбра, не вошедшие в дерево — обратные рёбра. Посмотрим, какие времена входа достижимы, если ходить по этому поддереву и в конце, может быть, пройти по обратному ребру. Запишем минимальное из них (красное число).

Мосты

Идея решения:



Если в вершине синее число (время входа) равно красному (минимум достижимых), то ребро дерева, идущее в эту вершину, является мостом (если оно есть).

Вопросы?

Вопросы?