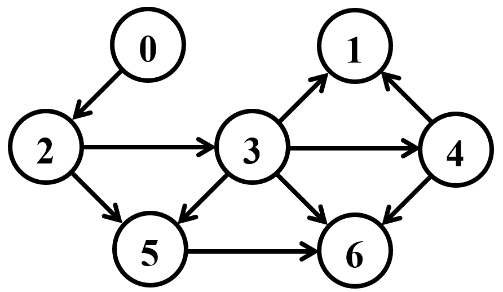
**Лабораторная работа №6. Алгоритмы на графах**

**Ход работы**

## **1. Разбор алгоритма поиска в ширину (BFS).**

Исходный граф:



Матрица смежности

{

{0, 0, 1, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 1, 0},

{0, 1, 0, 0, 1, 1, 1},

{0, 1, 0, 0, 0, 0, 1},

{0, 0, 0, 0, 0, 0, 1},

{0, 0, 0, 0, 0, 0, 0},

}

Матрица инцедентности

{

{-1, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0},

{-1, 0, 1, 0, 0, 0},

{0, -1, 0, 1, 1, 0},

{0, 0, -1, 0, 0, 1},

{0, 0, 0, 0, -1, 1},

{0, 0, 0, 0, 0, 0}

}

Список смежных вершин

0: 2

1:

2: 3 5

3: 1 4 5 6

4: 1 6

5: 6

6:

Алгоритм подразумевает, что задана исходная (стартовой) вершина, и основывается на простом правиле: при выборе очередной вершины предпочтение отдается ближайшей.

При этом считается, что все дуги графа имеют единичную длину.

Сначала посещается стартовая вершина, затем все вершины, смежные ей (т. е. находящиеся на расстоянии 1), после чего вершины, находящиеся на расстоянии 2 от стартовой и т.д.

Текущее состояние алгоритма хранится в следующих структурах памяти: Q – очередь вершин, С – массив окраски вершин, D – массив расстояний и P – массив предшествующих вершин.

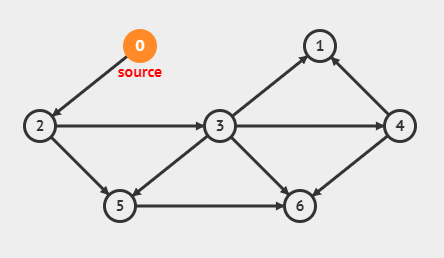
Очередь Q (структура памяти, реализующая алгоритм «первый вошел − первый вышел»), используется для промежуточного хранения номеров вершин. На каждом шаге алгоритма, в очередь помещаются номера вершин в порядке их обнаружения. На каждом шаге, кроме первого, из очереди извлекается очередной номер вершины, подлежащей отметке о посещении. На первом шаге алгоритма в очередь помещается номер стартовой вершины. На последнем шаге очередь пуста.

Массив C используется для хранения состояния вершин. С каждым из трех возможных состояний обычно связывают цвет: белый (W) – вершина не посещалась, серый (G) – вершина посещалась, черный (B) – фиксирован факт посещения вершины. На первом шаге алгоритма стартовая вершина окрашивается в серый цвет, а остальные – в белый. На последнем шаге все вершины становятся черными.

В массиве D для каждой вершины хранятся расстояния от стартовой вершины. На первом шаге для стартовой вершины в массиве D устанавливается значение 0, а для остальных вершин – значение «бесконечность» (I). На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные их расстоянию от стартовой вершины.

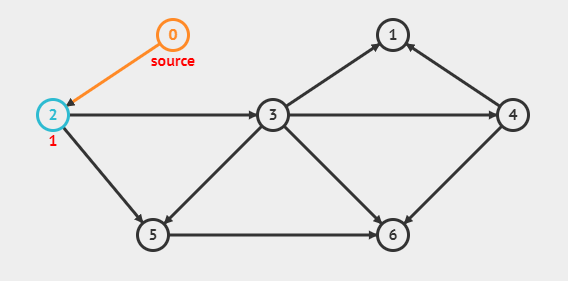
Массив P позволяет восстановить порядок обхода вершин и хранит для каждой вершины, кроме стартовой, предшествующую в обходе вершину. На первом шаге алгоритма всем элементам массива присваивается значение «пустота» (N). На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные номеру предшествующей вершины в порядке обхода.

**Шаг 1**

****

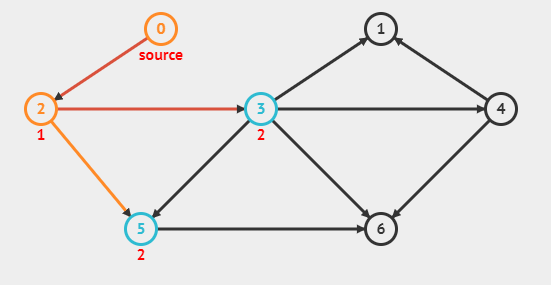
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 0 |  |  |  |  |  |  |
| C | G | W | W | W | W | W | W |
| D | 0 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |

**Шаг 2**

****

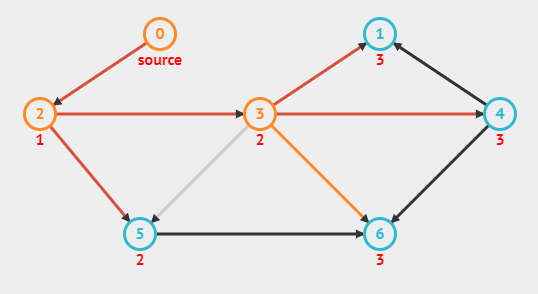
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 2 |  |  |  |  |  |  |
| C | B | W | G | W | W | W | W |
| D | 0 | I | 1 | I | I | I | I |
| P | N | N | 0 | N | N | N | N |

**Шаг 3**

****

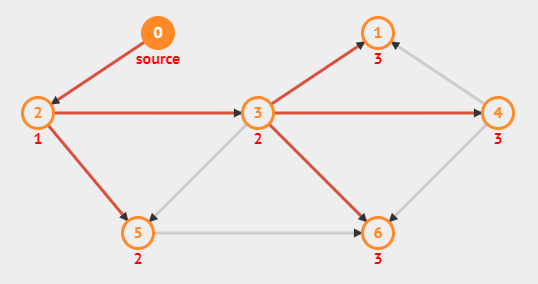
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 3 | 5 |  |  |  |  |  |
| C | B | W | B | G | W | G | W |
| D | 0 | I | 1 | 2 | I | 2 | I |
| P | N | N | 0 | 2 | N | 2 | N |

**Шаг 4**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 1 | 4 | 6 |  |  |  |  |
| C | G | G | B | B | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 3 | 0 | 2 | 3 | 2 | 3 |

**Шаг 5**

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | N | 3 | 0 | 2 | 3 | 2 | 3 |

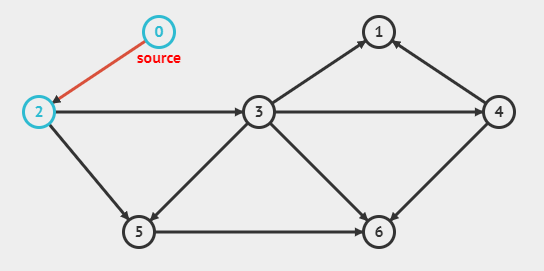
**Ответ: 0 2 3 5 1 4 6**

## **2. Разбор алгоритма поиска в глубину(DFS).**

Как и для поиска в ширину, задается стартовая вершина. Алгоритм описывается следующим образом: для каждой не пройденной вершины, начиная со стартовой, необходимо найти все смежные вершины и повторить поиск для каждой.

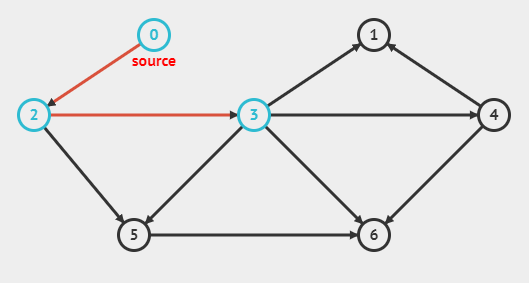
Назначение и размерность массивов С (массив окраски вершин) и P (массив предшествующих вершин) такие же, как и в алгоритме BFS. В массиве D для каждой вершины записывается время обнаружения (шаг окраски в серый цвет). Массив F предназначен для хранения времени фиксации (шага окраски в черный цвет) вершины. Кроме того, используется переменная t, текущее значение которой – номер шага алгоритма.

**Шаг 1**



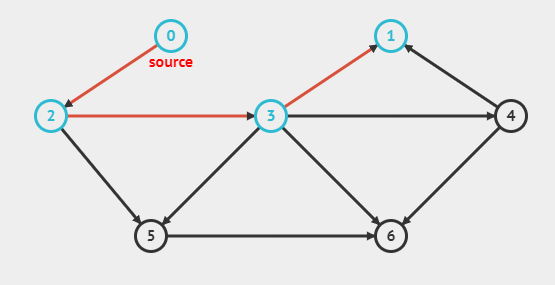
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | W | W | W | W |
| D | 0 | I | 1 | I | I | I | I |
| P | N | N | 0 | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 2**



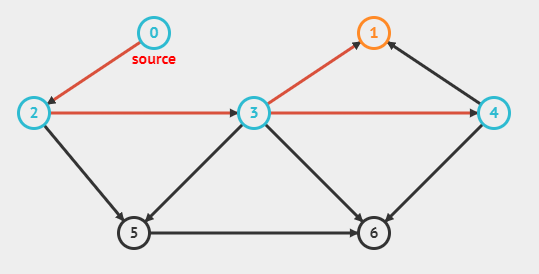
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | W | W |
| D | 0 | I | 1 | 2 | I | I | I |
| P | N | N | 0 | 1 | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 3**



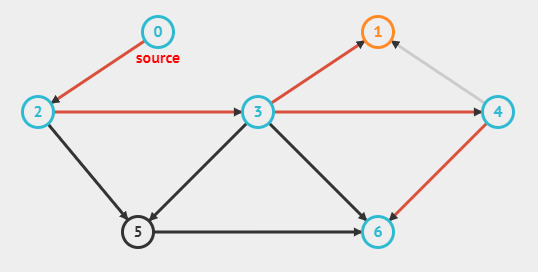
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | W | W | W |
| D | 0 | 3 | 1 | 2 | I | I | I |
| P | N | 2 | 0 | 1 | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 4**



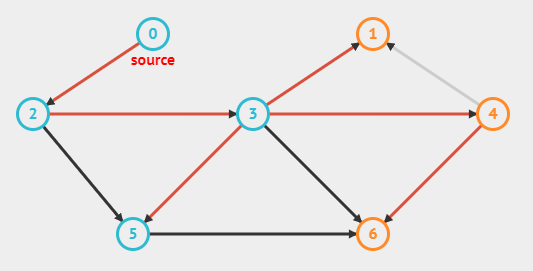
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | W | W |
| D | 0 | 3 | 1 | 2 | 3 | I | I |
| P | N | 2 | 0 | 1 | 2 | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 5**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | W | G |
| D | 0 | 3 | 1 | 2 | 3 | I | 3 |
| P | N | 2 | 0 | 1 | 2 | N | 3 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 6**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Шаг 7**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 0 | 0 | 0 | 0 | 7 | 0 |

**Шаг 8**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 0 | 0 | 0 | 0 | 7 | 8 |

**Шаг 9**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 0 | 0 | 0 | 9 | 7 | 8 |

**Шаг 10**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 10 | 0 | 0 | 9 | 7 | 8 |

**Шаг 11**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 10 | 0 | 11 | 9 | 7 | 8 |

**Шаг 12**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 0 | 10 | 12 | 11 | 9 | 7 | 8 |

**Шаг 13**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |
| F | 13 | 10 | 12 | 11 | 9 | 7 | 8 |

**Шаг 14**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | N | 2 | 0 | 1 | 2 | 2 | 3 |

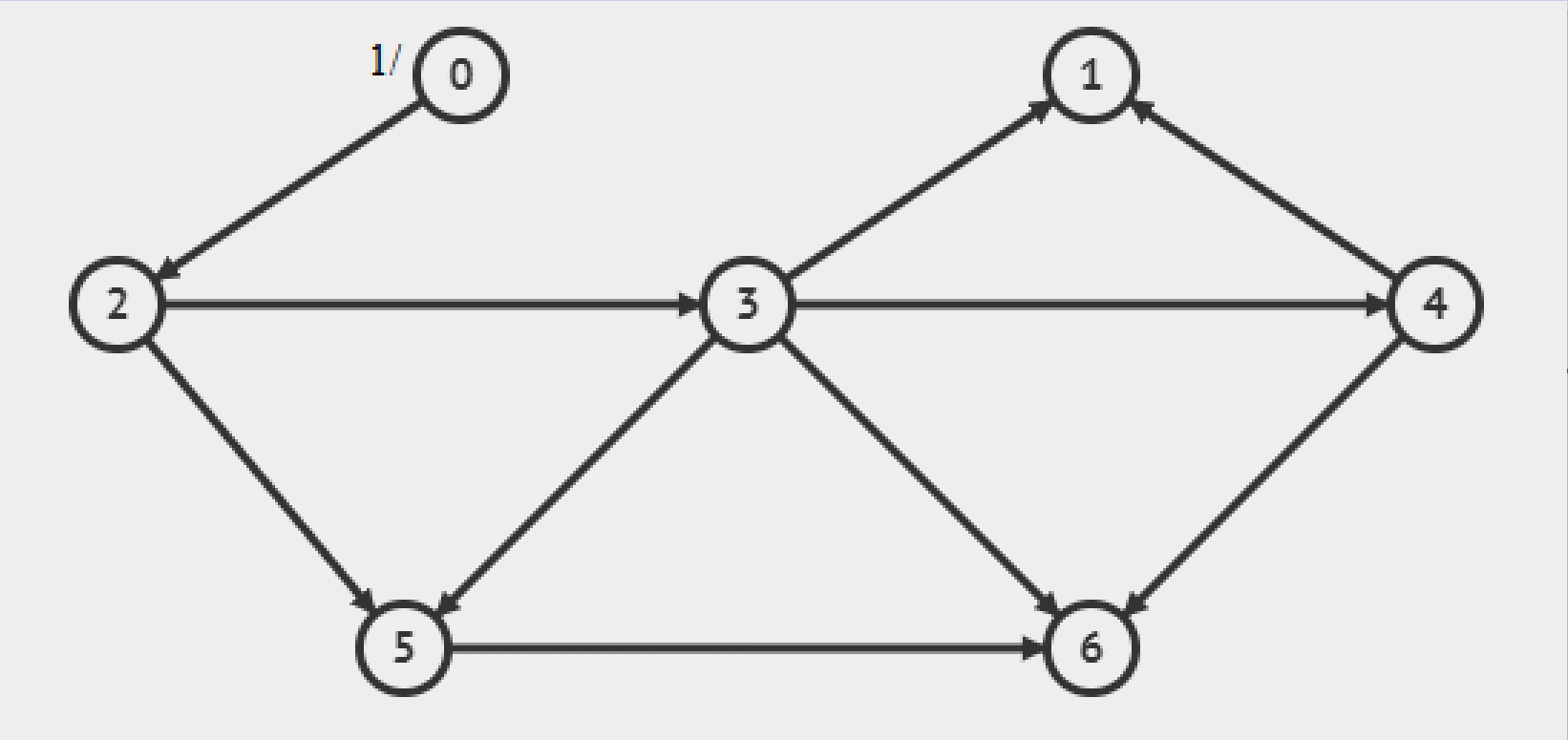
**Ответ: 0 2 3 5 1 4 6**

## **3. Разбор алгоритма топологической сортировки.**

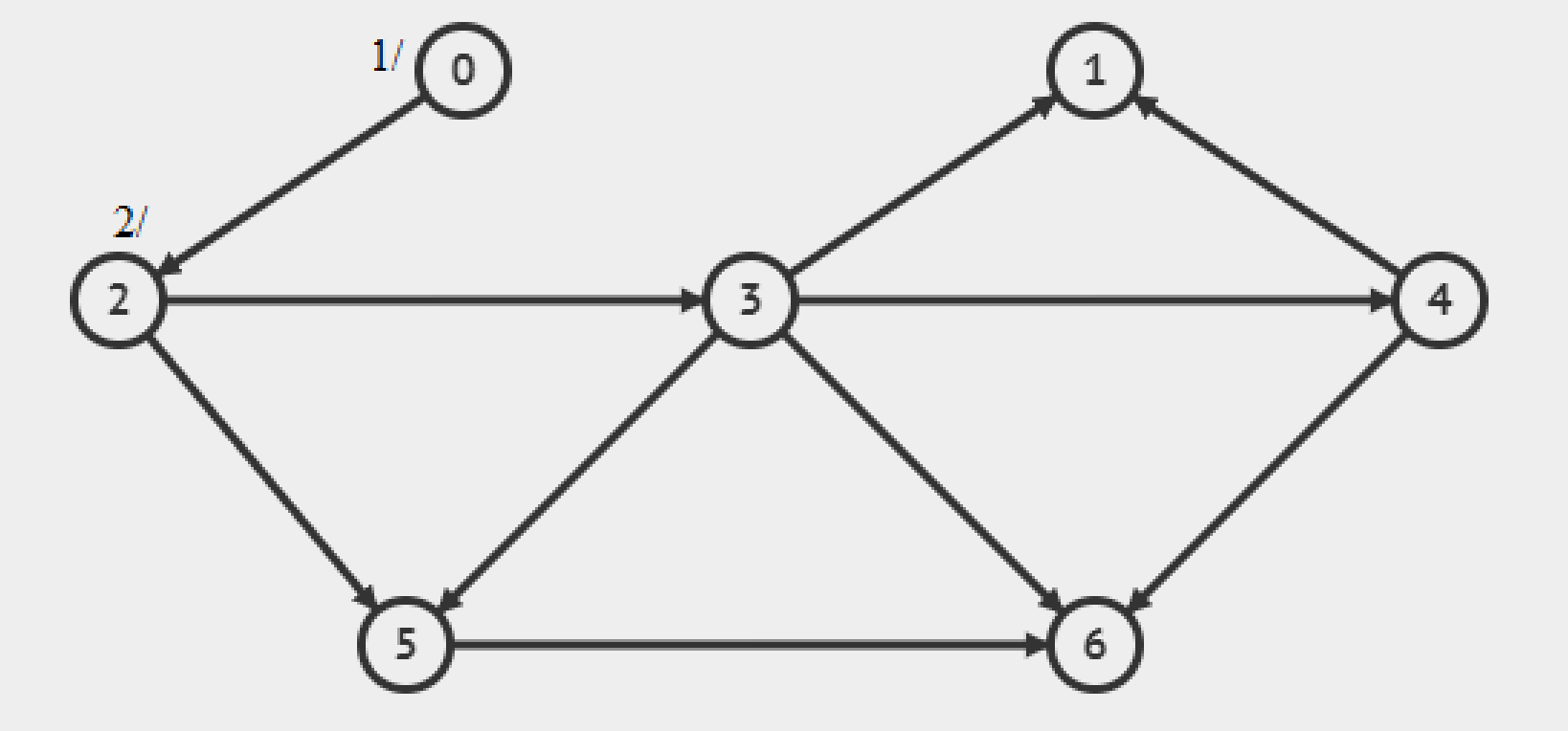
**Топологическая сортировка** — это процедура упорядочивания вершин ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним. Обычно после выполнения топологической сортировки вершины переименовываются (перенумеровываются) в соответствии с полученным порядком. После такого переименования граф обладает свойством: начальная вершина каждой дуги имеет номер (имя) меньший, чем номер конечной вершины этой дуги.

Наиболее известны два способа топологической сортировки графа: алгоритмы Демукрона и алгоритм, применяющий поиск в глубину.

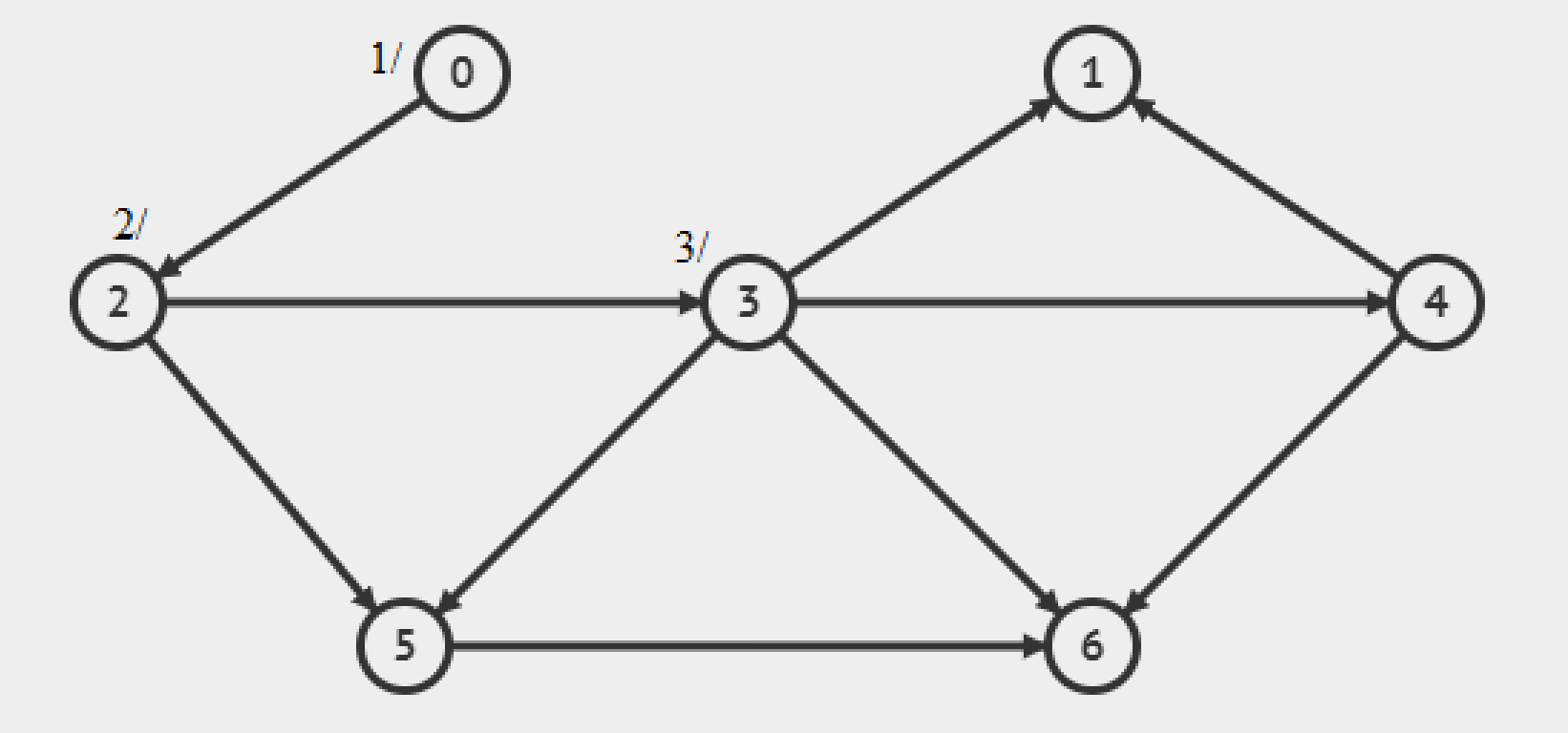
**Шаг 1**



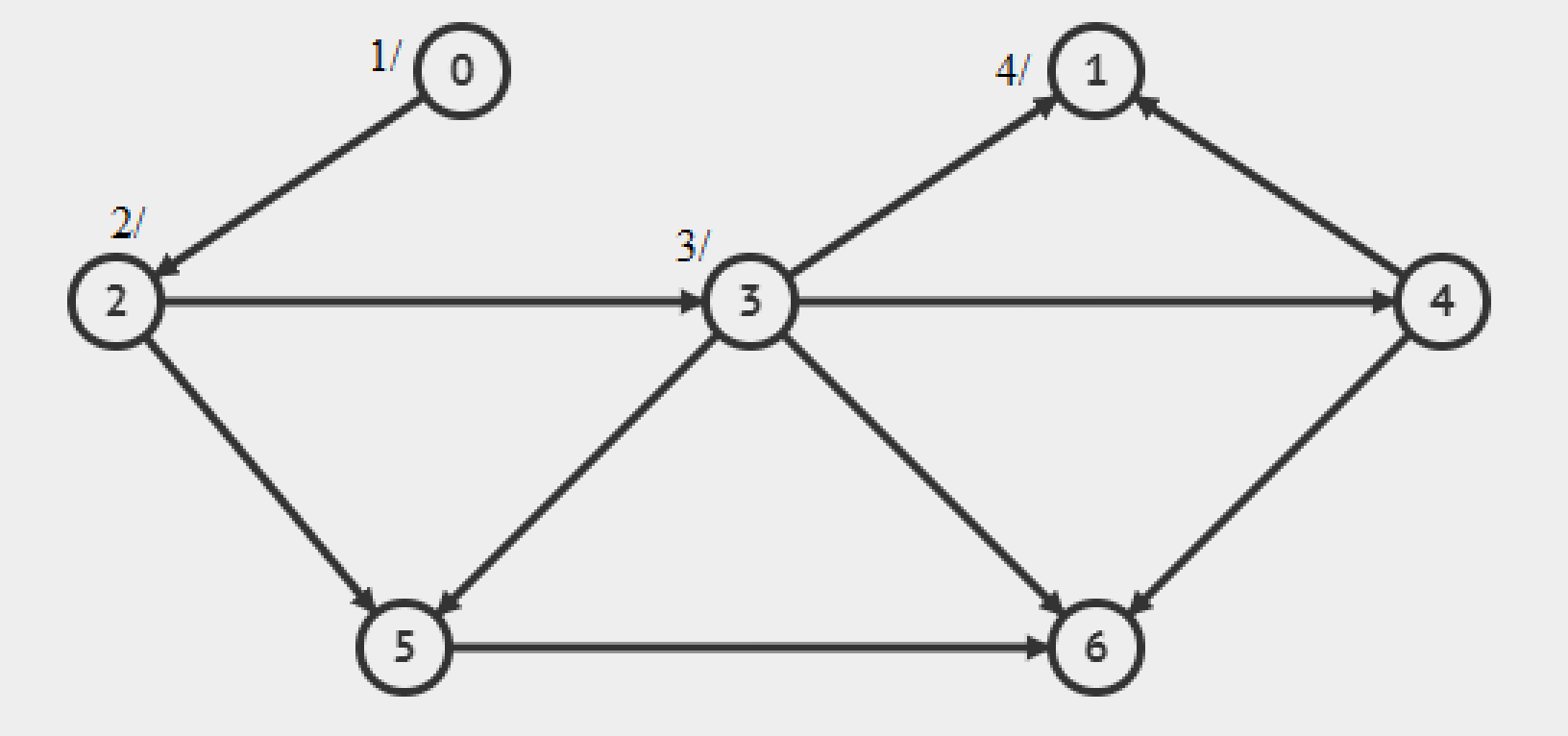
**Шаг 2**



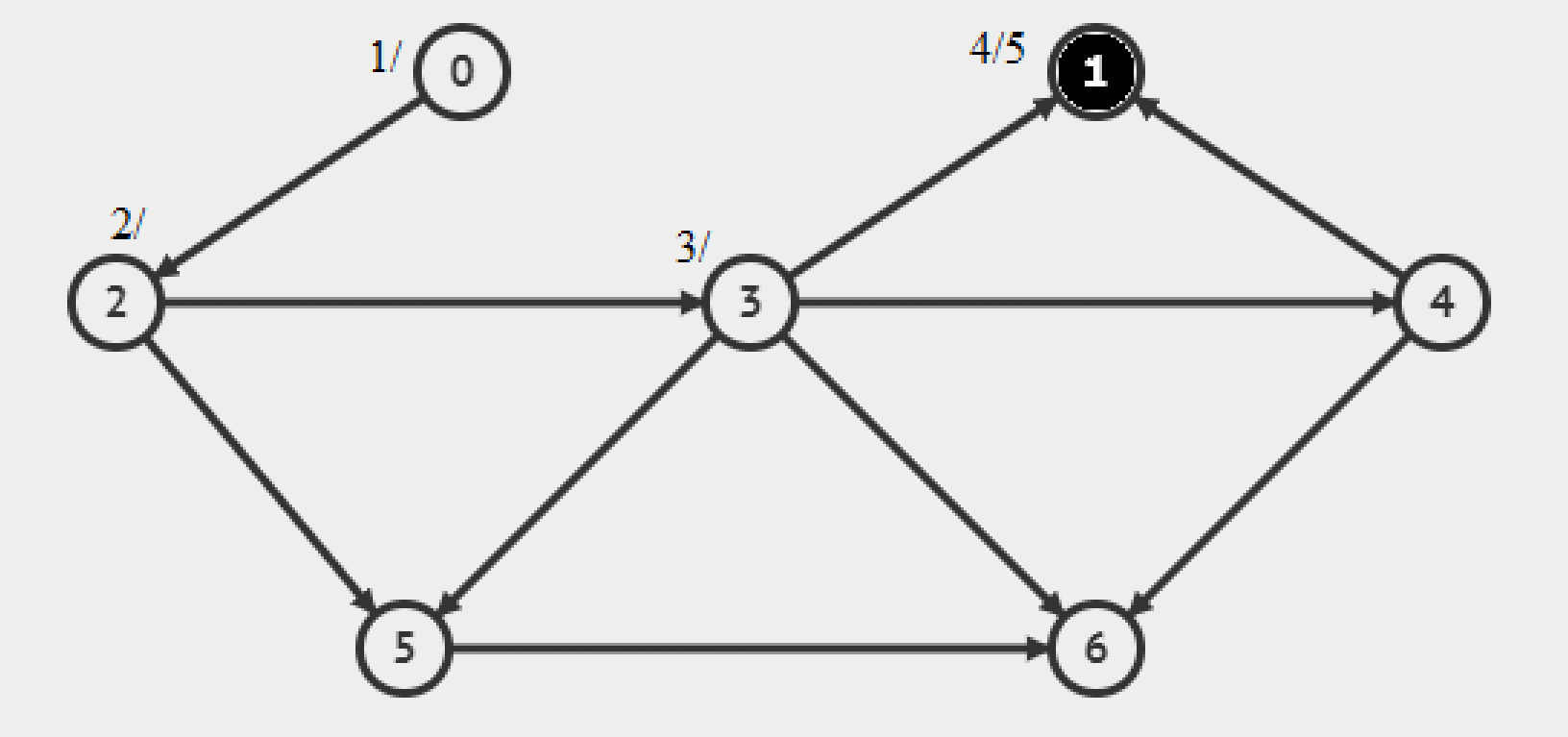
**Шаг 3**



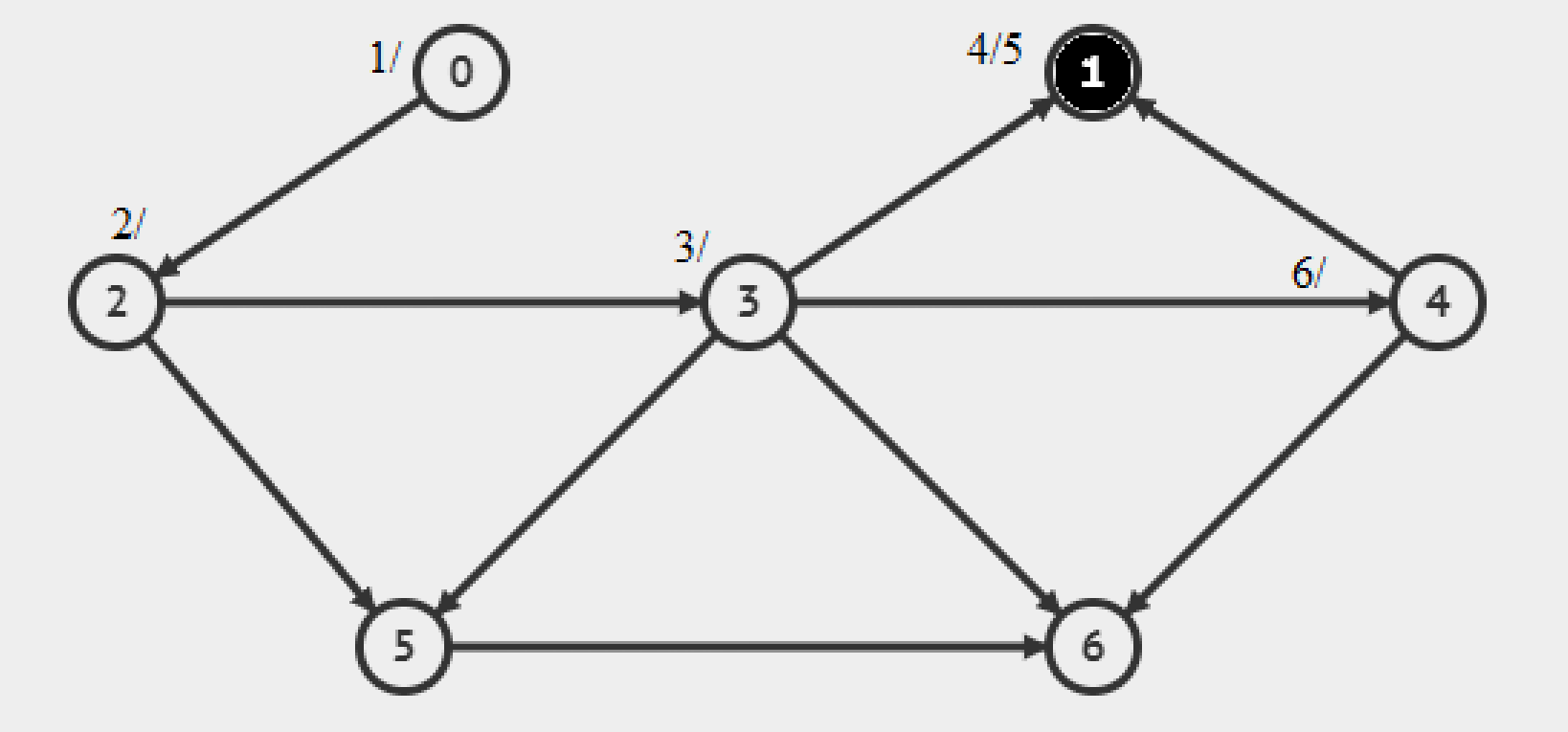
**Шаг 4**



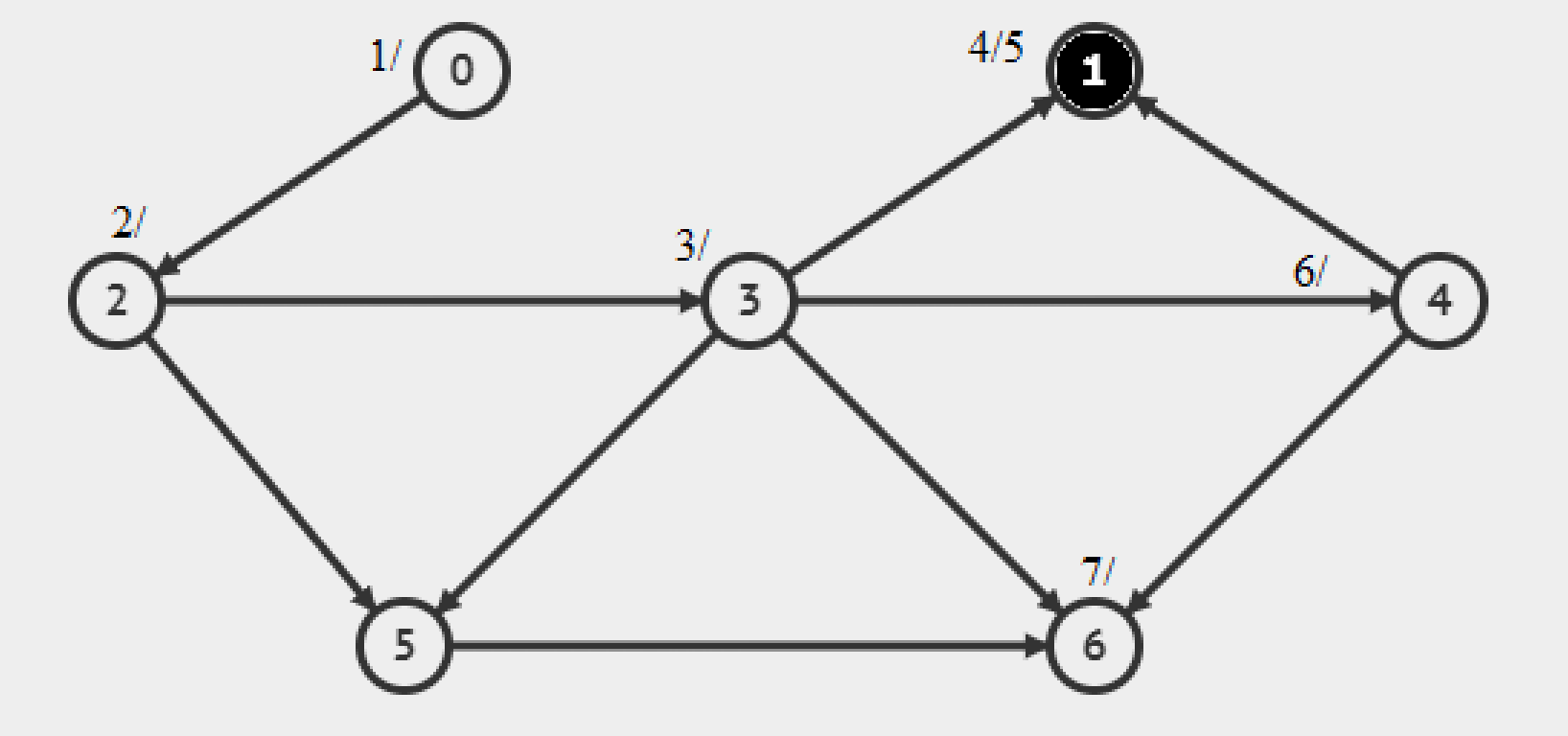
**Шаг 5** (1)



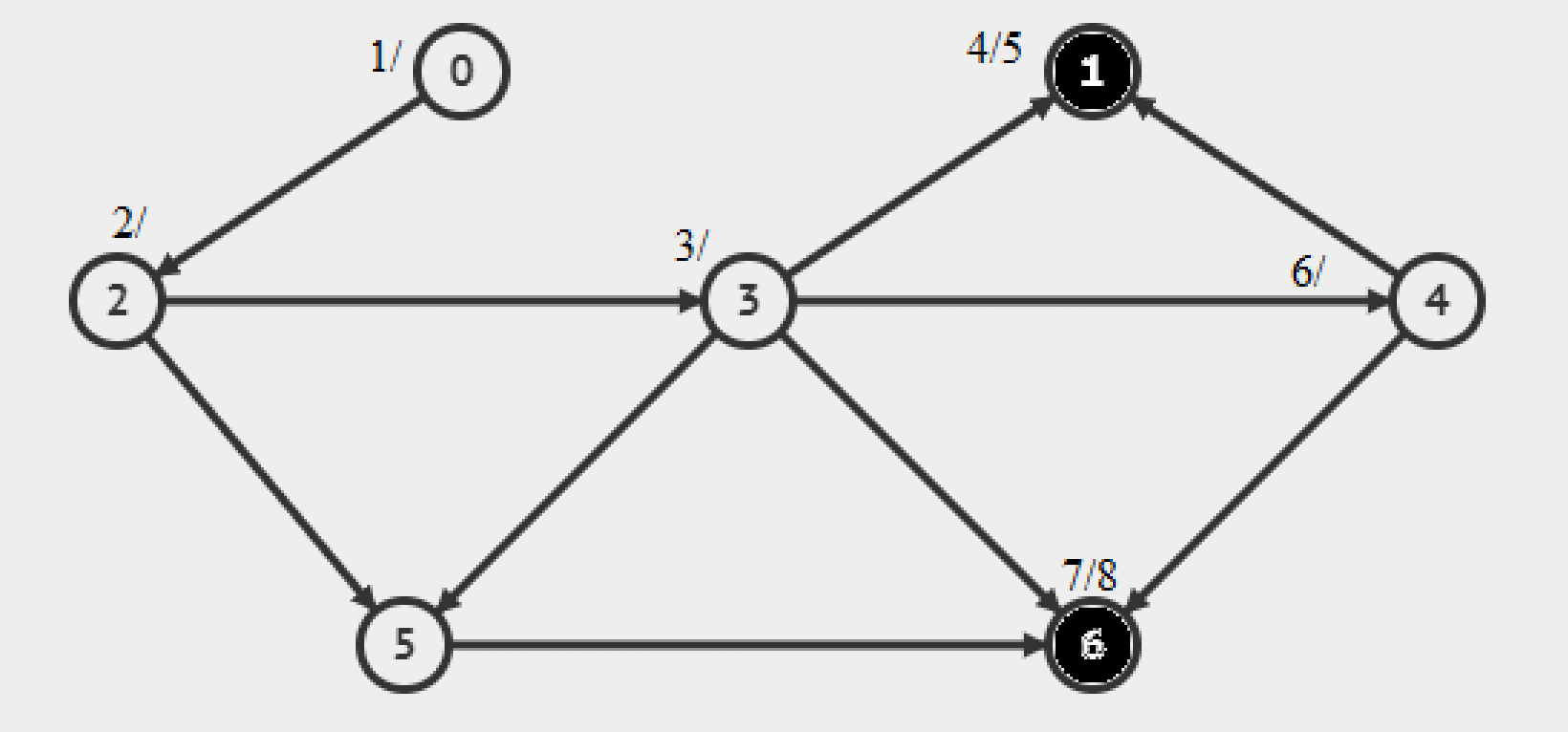
**Шаг 6**



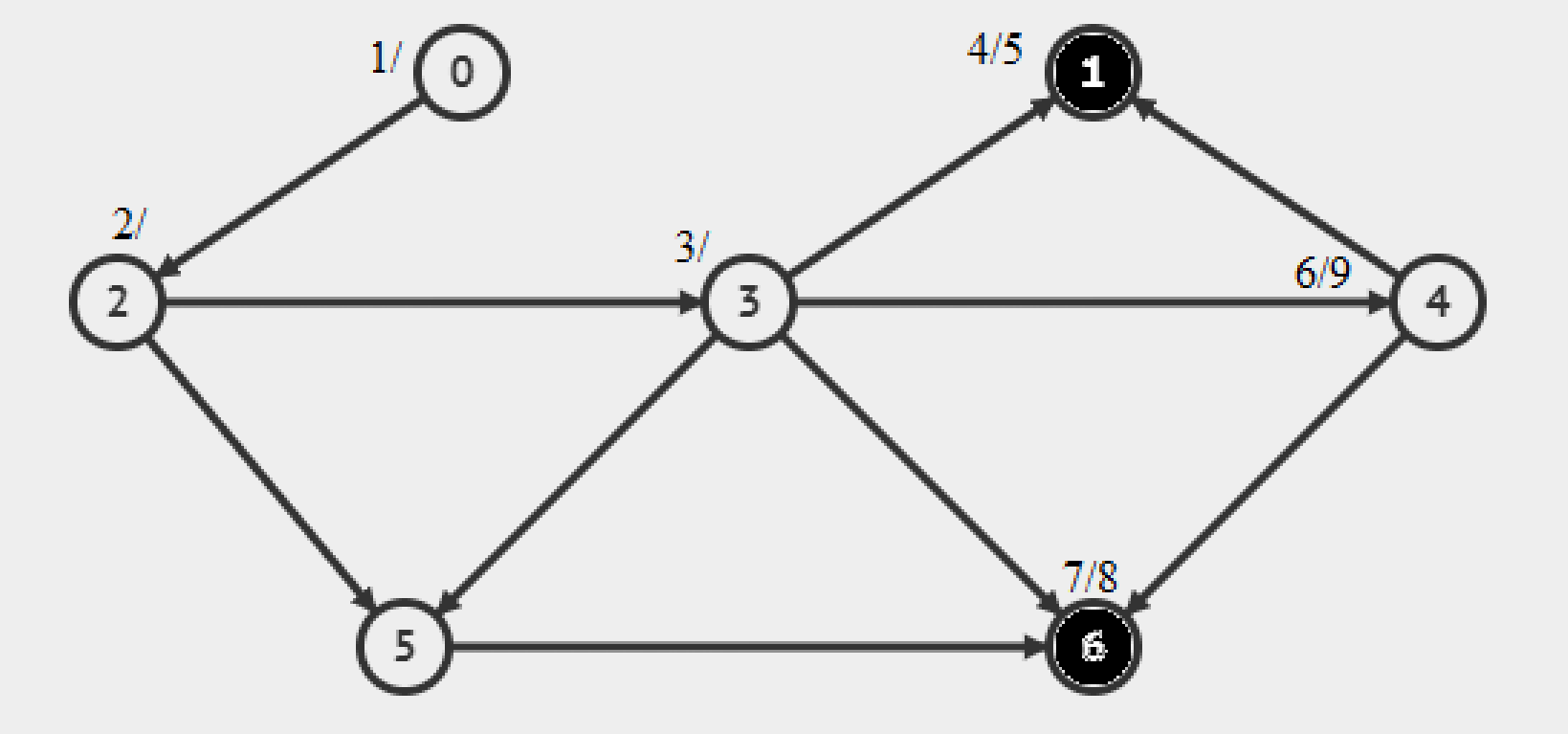
**Шаг 7**



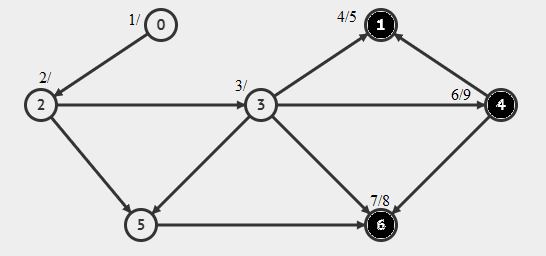
**Шаг 8** (6 1)



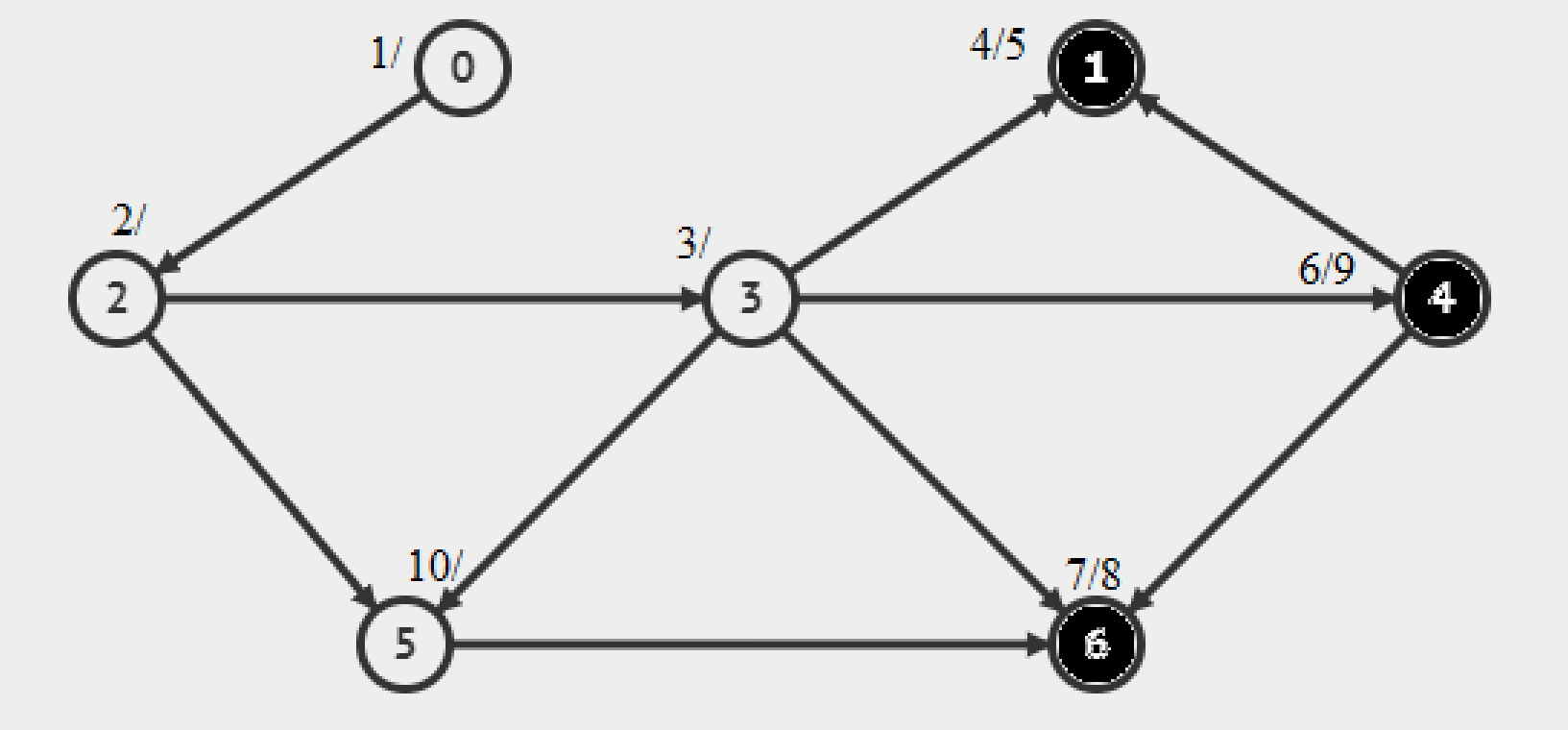
**Шаг 9**



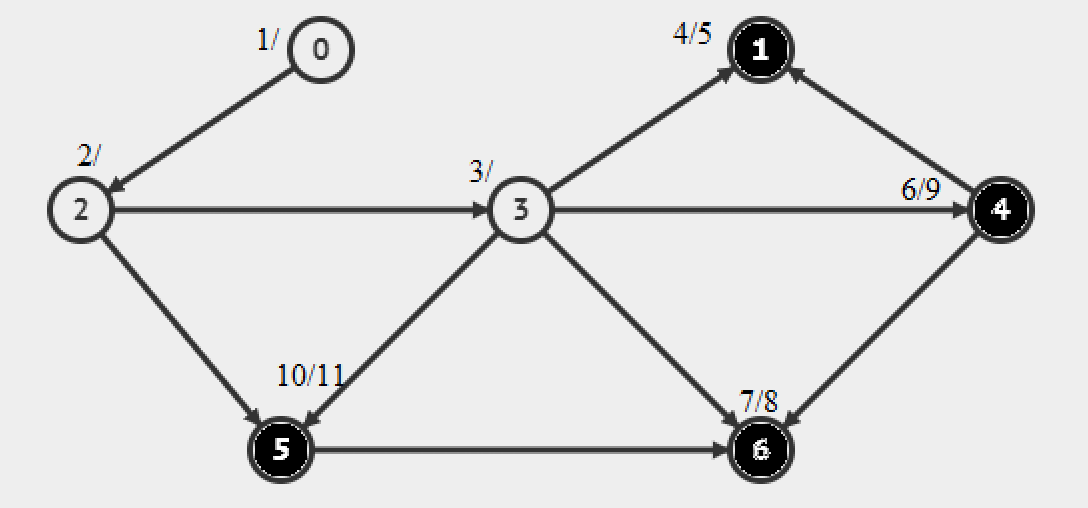
**Шаг 10** (4 6 1)



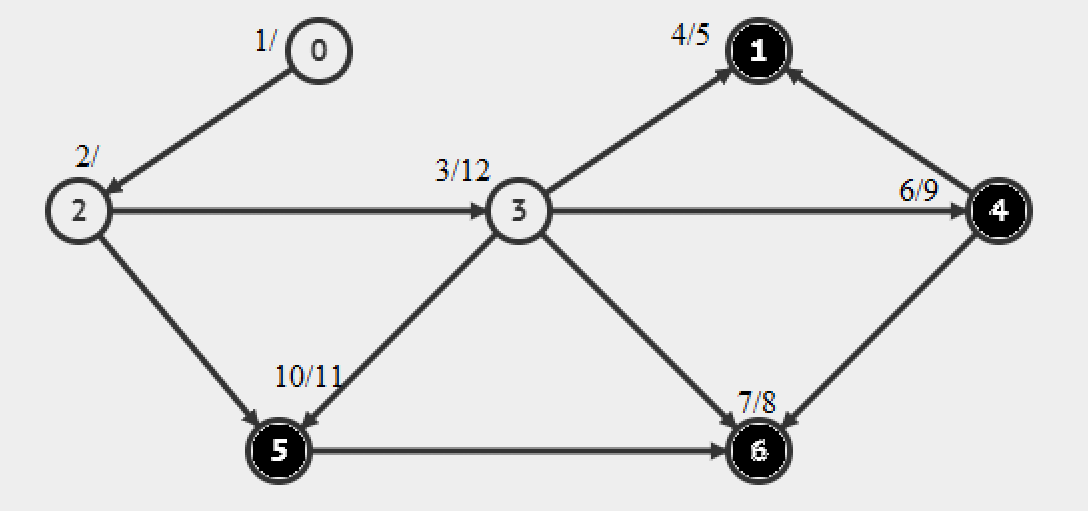
**Шаг 11**



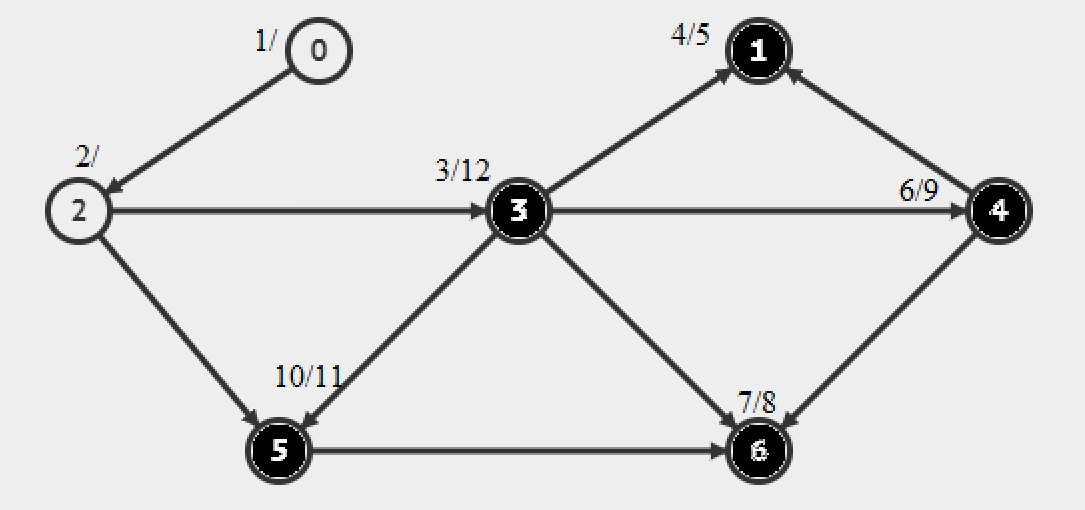
**Шаг 12** (5 4 6 1)



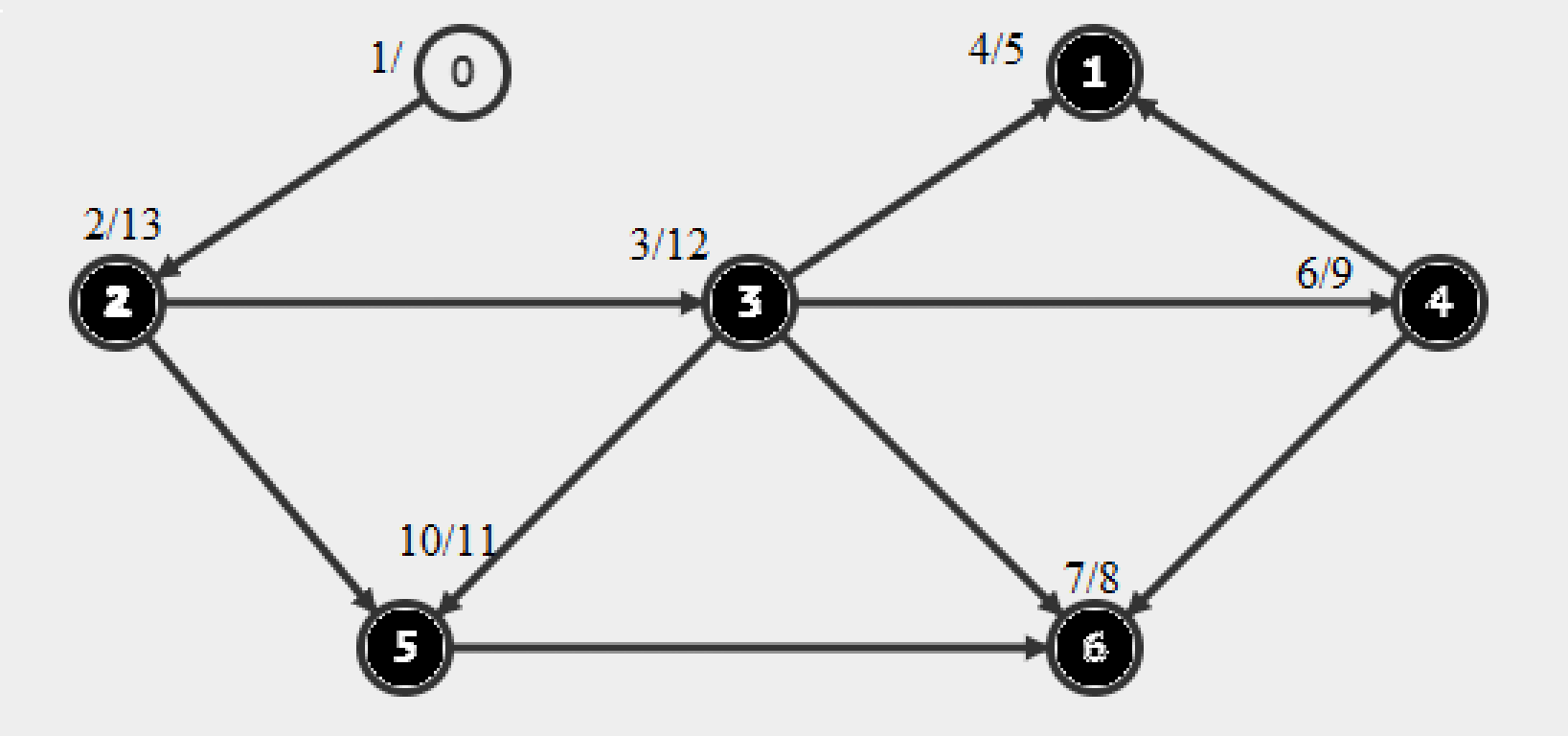
**Шаг 13**



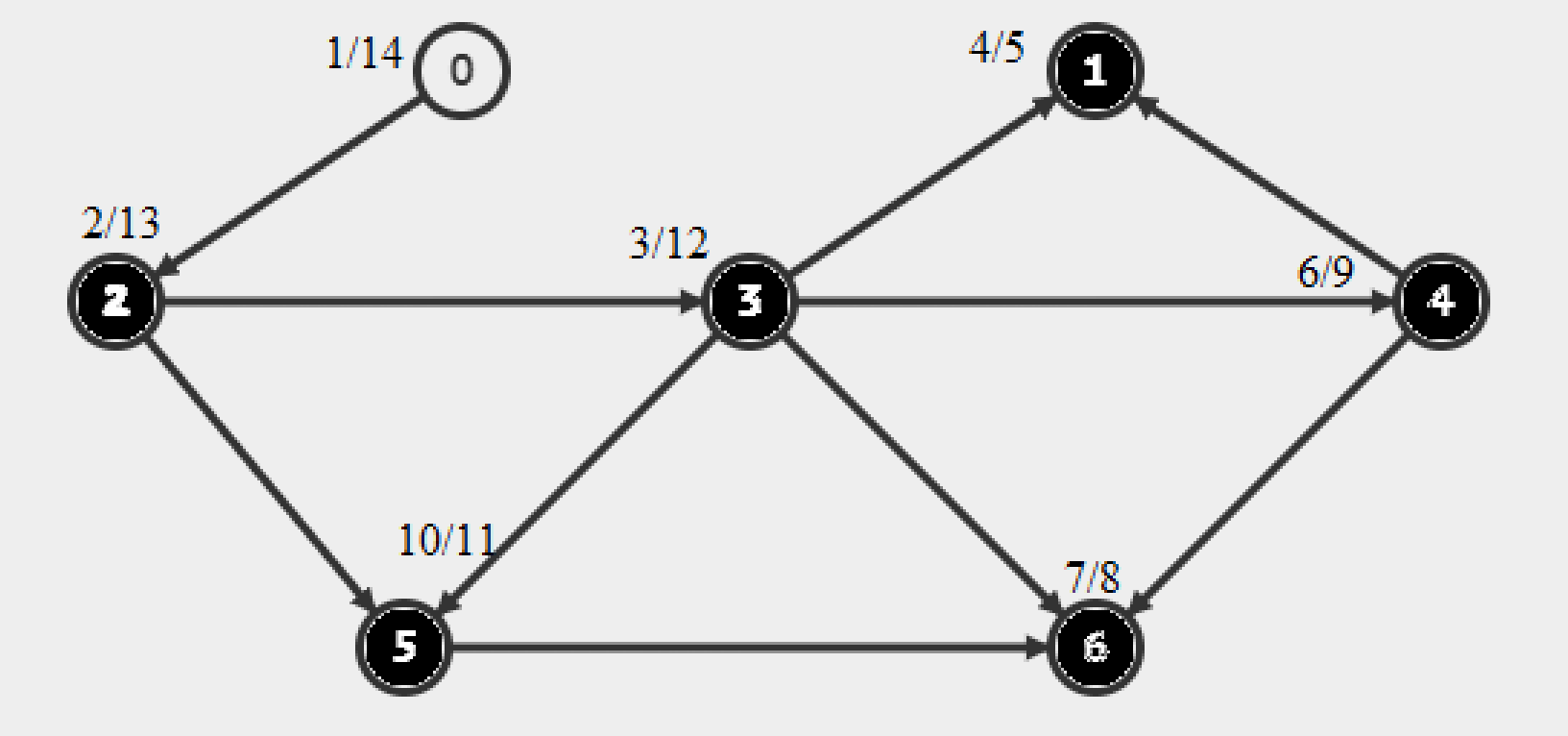
**Шаг 14** (3 5 4 6 1)



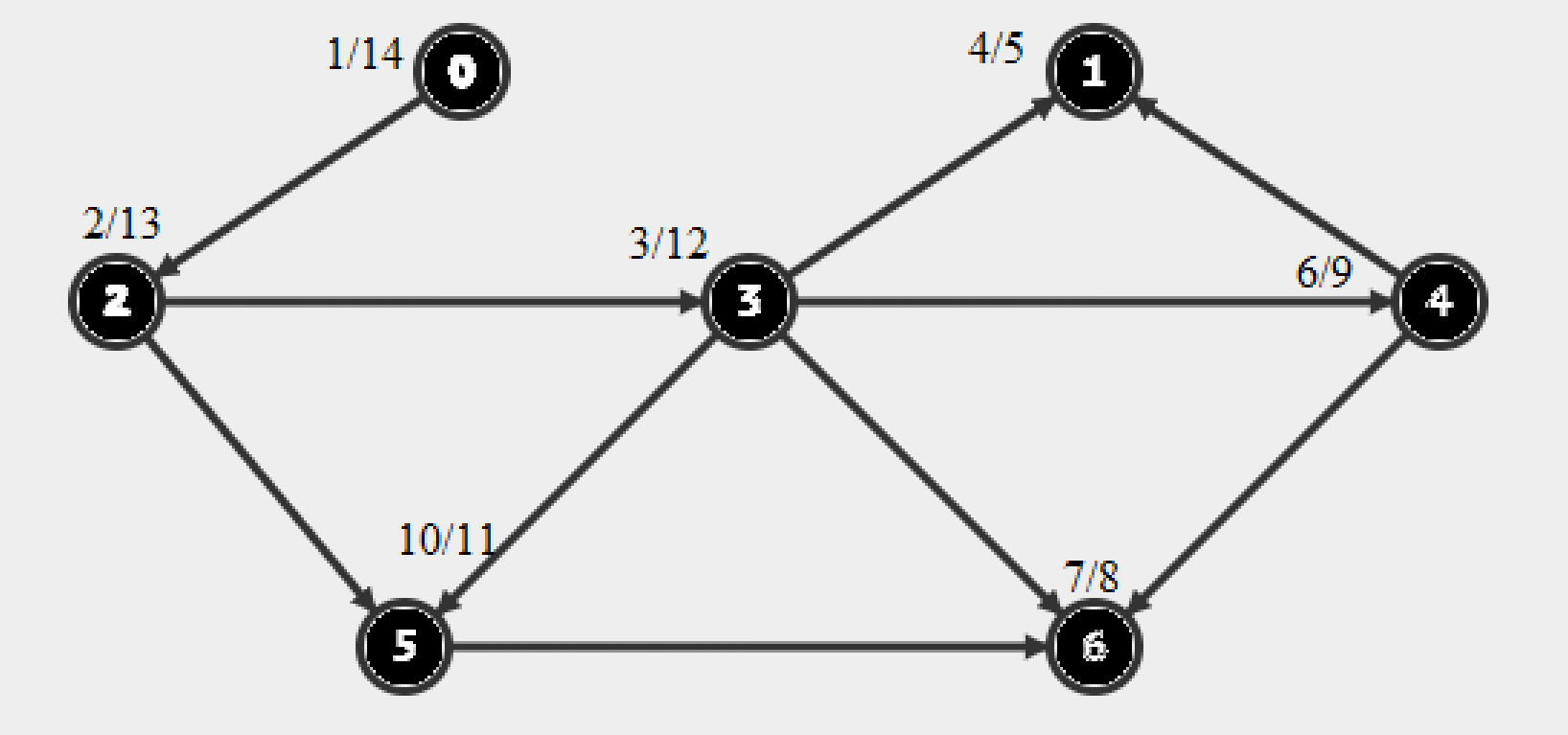
**Шаг 15** (2 3 5 4 6 1)



**Шаг 16**

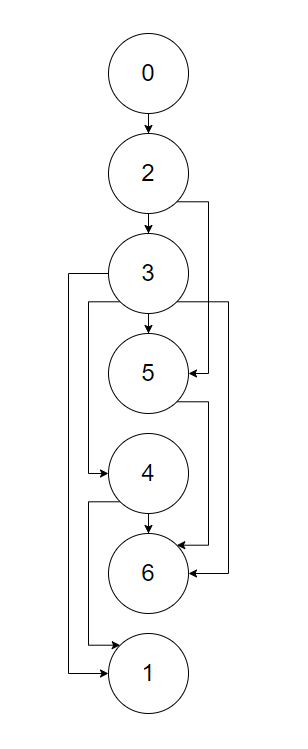


**Шаг 17** (0 2 3 5 4 6 1)



**Ответ:** 0 2 3 5 4 6 1

Полученное дерево:

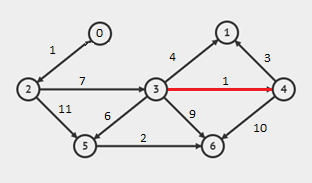
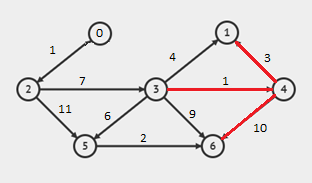


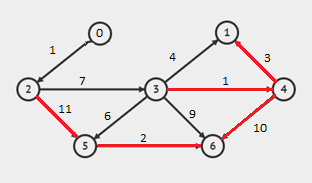
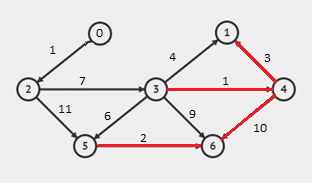
## **4. Алгоритм Прима**

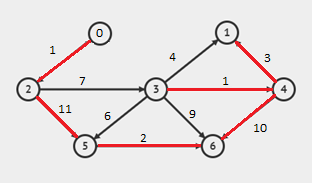
Суть самого алгоритма Прима тоже сводится к жадному перебору рёбер, но уже из определенного множества. На входе так же имеется пустой подграф, который и будем достраивать до потенциального минимального остовного дерева.

Изначально наш подграф состоит из одной любой вершины исходного графа.

Затем из рёбер инцидентных этой вершине, выбирается такое минимальное ребро, которое связала бы две абсолютно разные компоненты связности, одной из которых и является наш подграф. То есть, как только у нас появляется возможность добавить новую вершину в наш подграф, мы тут же включаем ее по минимальмально возможному весу.

Продолжаем выполнять предыдущий шаг до тех пор, пока не найдем искомое MST.





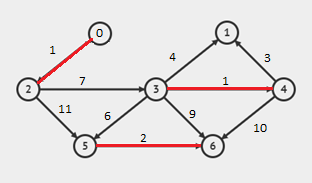
## **5. Алгоритм Краскала**

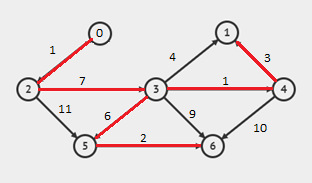
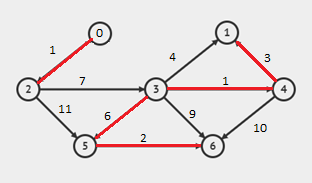
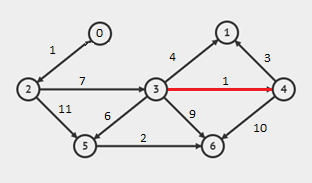
Механизм, по которому работает данный алгоритм, очень прост. На входе имеется пустой подграф, который и будем достраивать до потенциального минимального остовного дерева. Будем рассматривать только связные графы, в другом случае при применении алгоритма Краскала мы будем получать не минимальное остовное дерево, а просто остовной лес.

Вначале мы производим сортировку рёбер по неубыванию по их весам.

Добавляем i-ое ребро в наш подграф только в том случае, если данное ребро соединяет две разные компоненты связности, одним из которых является наш подграф. То есть, на каждом шаге добавляется минимальное по весу ребро, один конец которого содержится в нашем подграфе, а другой - еще нет.

Алгоритм завершит свою работу после того, как множество вершин нашего подграфа совпадет с множеством вершин исходного графа.

Данный алгоритм называется жадным из-за того, что мы на каждом шаге пытаемся найти оптимальный вариант, который приведет к оптимальному решению в целом.

****

## **6. Проверка алгоритмов на языках программирования.**

Результат выполнения алгоритмов, написанных на языке программирования C++ представлен на рисунке 4.1.

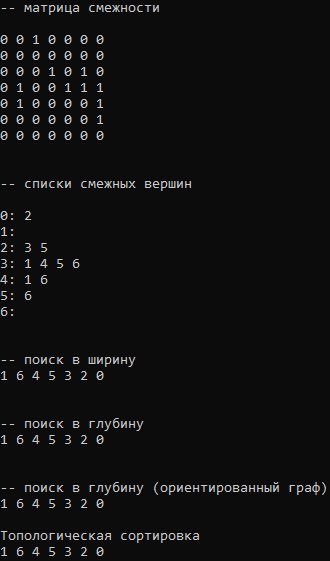


Рисунок 4.1. Проверка алгоритмов на языке программирования C++