# Лабораторная работа №6. Алгоритмы на графах

**Цель работы:** Освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов. Разобрать алгоритм Прима и алгоритм Крускала.

***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.

|  |  |
| --- | --- |
| Вариант | Граф G |
| 12 |  |

Матрица смежности - это квадратная матрица размера n x n, где n - число вершин в графе. Эта матрица используется для представления графа в виде таблицы, где каждый элемент матрицы a\_ij показывает, существует ли ребро между вершинами i и j.

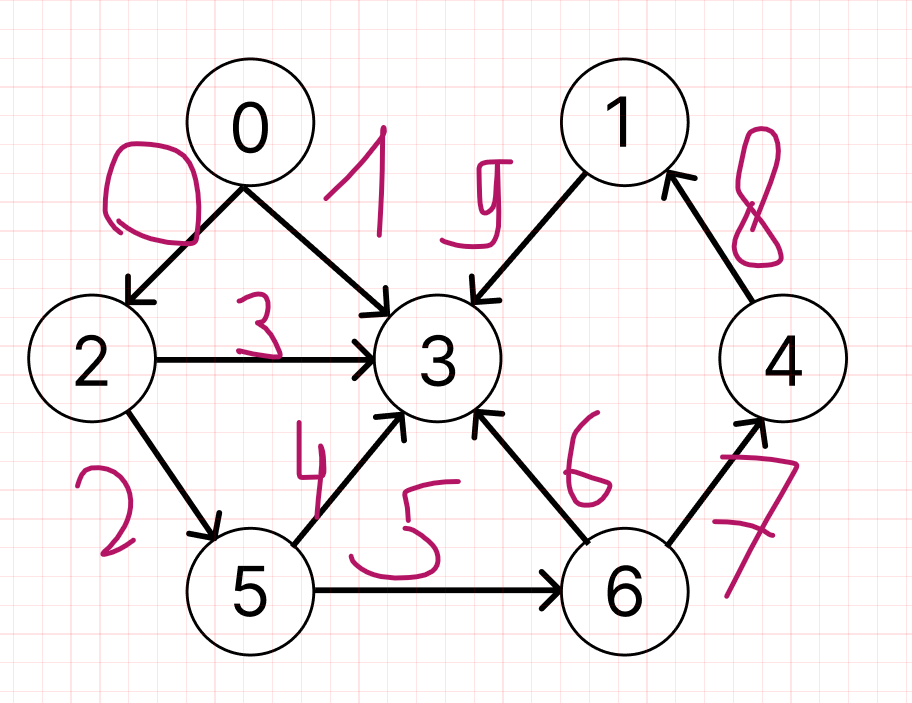
Если в графе есть ребро между вершинами i и j, то a\_ij равно 1. Если ребра между этими вершинами нет, то a\_ij равно 0. Так как граф ориентированный, матрица смежности не будет симметричной относительно главной диагонали.

**Матрица смежности:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Вершина** | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **0** | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **6** | 0 | 0 | 0 | 1 | 1 | 0 | 0 |



G = (V, E), V = {0, 1, 2, 3, 4, 5, 6}, E = { <0, 2>0, <0, 3>1, <2, 5>2, <2, 3>3, <5, 3>4, <5, 6>5, <6, 3>6, <6, 4>7, <1, 4>8, <1, 3>9}

****

\*если в двух словах, если ребро исходит от вершины, то 1, если входит, то -1, 0 – если ничего

**Матрица инцидентности:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Вершина** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |
| **2** | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 | 0 | -1 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 0 |
| **5** | 0 | 0 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 0 |

Список смежных вершин - это список всех вершин, с которыми заданная вершина в графе имеет ребра. Другими словами, это список вершин, которые непосредственно соединены с данной вершиной ребрами.

Например, если в графе есть вершина A, которая имеет ребра, соединяющие ее с вершинами B, C и D, то список смежных вершин для вершины A будет содержать вершины B, C и D.

Список смежных вершин может быть представлен в виде массива, списка или любой другой структуры данных, в которой хранятся вершины, с которыми данная вершина имеет ребра.

**Список смежных вершин:**

S0 = {2, 3}

S1 = {3}

S2 = {3,5}

S3 = ∅

S4 = {1,6}

S5 = {3,6}

S6 = {3,4}

***Задание 2.*** Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях. Оформить отчет, включив в него **каждый** шаг выполнения алгоритмов.

**Алгоритм поиска в ширину (англ. breadth-first search, BFS)** позволяет найти кратчайшие пути из одной вершины невзвешенного графа до всех остальных вершин

**Поиск в ширину:**

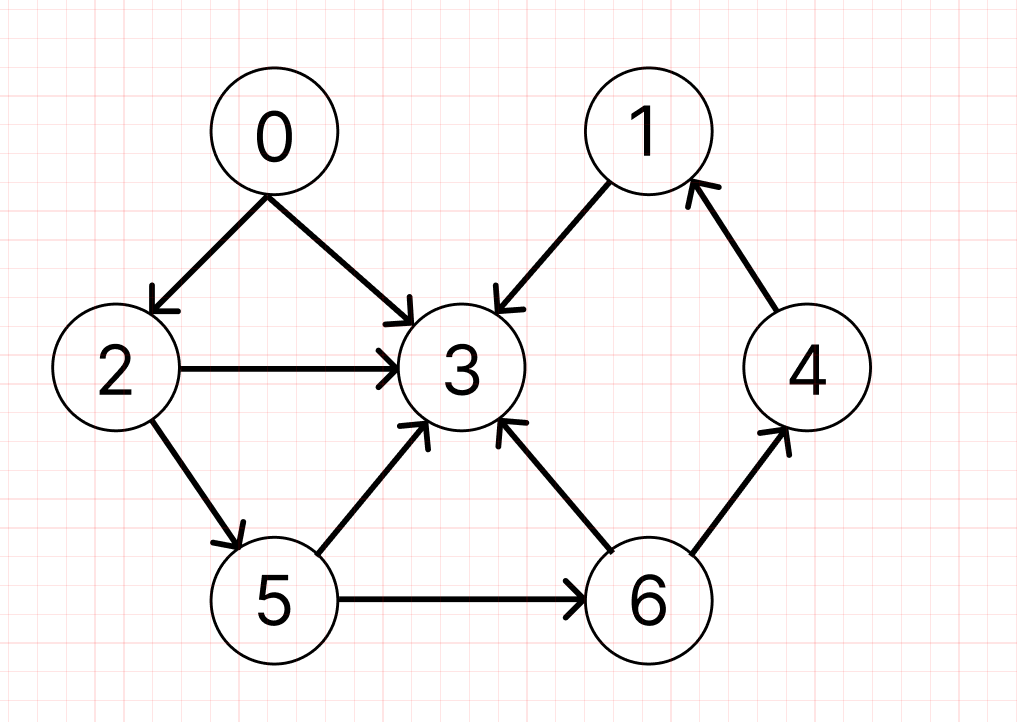
Q- для промежуточного хранения вершин(очередь)

массивы:

С- массив окраски вершин ( Б – не добавлена в очередь , С – добавлена в очередь , Ч – вышла из очереди )

D- массив расстояний

Р- массив предшествующих вершин

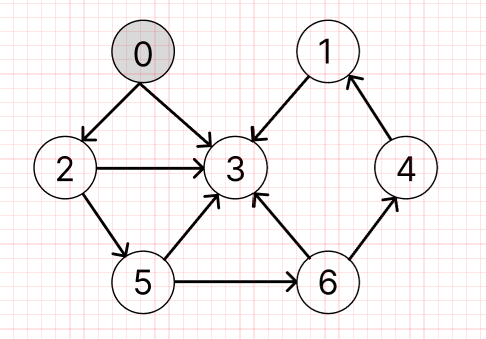
****

**Шаг 1**

В качестве стартовой вершины выбираем вершину с номером 0.

Очередь : 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | Б | Б | Б | Б | Б |
| Расстояния | 0 | I | I | I | I | I | I |
| Массив предыдущих вершин | N | N | N | N | N | N | N |

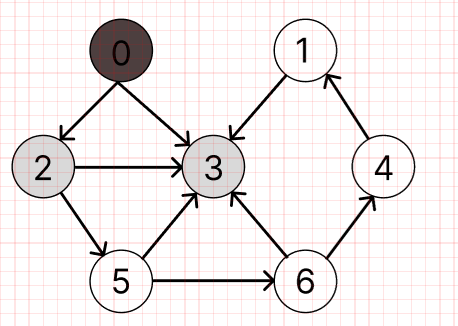
****

**Шаг 2**

У вершины 0 две смежные вершины. Для последующего пути выбираем вершину с наименьшим весом – 2, а 3-ю добавляем в начало очереди. Закрашиваем 0 в черный цвет как пройденную. Смежные в серый.

Очередь : 2 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | С | С | Б | Б | Б |
| Расстояния | 0 | I | 1 | 1 | I | I | I |
| Массив предыдущих вершин | N | N | 0 | 0 | N | N | N |

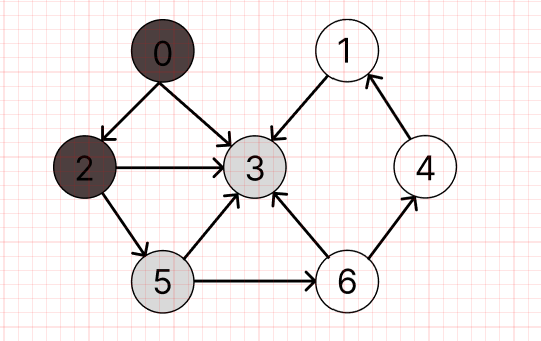
****

**Шаг 3**

Переходим в смежную вершину, вершину номер 2. С ней смежные вершины номер 3 и 5. Закрашиваем 3 и 5 в серый цвет, а вершину номер 2 в черный, как пройденную.

Очередь : 3 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | С | Б | С | Б |
| Расстояния | 0 | I | 1 | 1 | I | 2 | I |
| Массив предыдущих вершин | N | N | 0 | 0 | 1 | N | N |

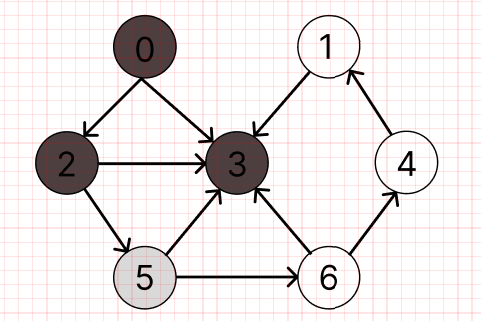
****

**Шаг 4**

Переходим в смежную вершину, вершину номер 3. Вершина номер 3 не имеет смежных вершин поэтому просто окрашиваем ее в черный, как пройденную.

Очередь : 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | Ч | Б | С | Б |
| Расстояния | 0 | I | 1 | 1 | I | 2 | I |
| Массив предыдущих вершин | N | N | 0 | 0 | 1 | 3 | N |

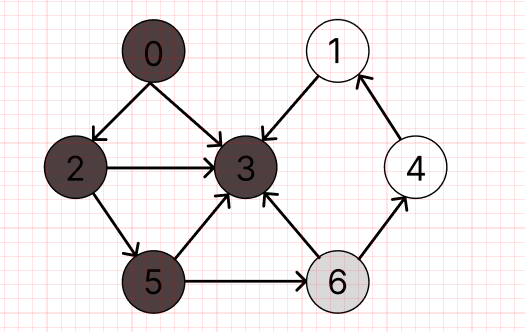
****

**Шаг 5**

Переходим в смежную вершину, вершину номер номер 5. Вершина номер 5 имеет одну смежную вершину, номер 6, закраиваем ее в серый цвет, а вершину 5 в черный.

Очередь : 6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | Ч | Б | Ч | С |
| Расстояния | 0 | I | 1 | 1 | I | 2 | 3 |
| Массив предыдущих вершин | N | N | 0 | 0 | 1 | 3 | 4 |

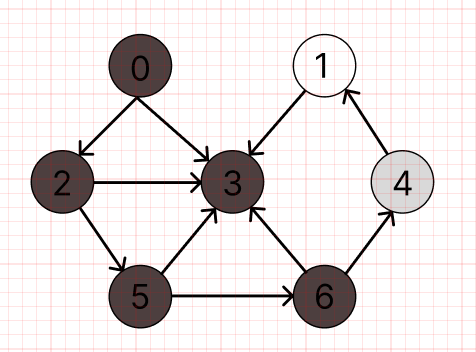
****

**Шаг 6**

Переходим смежную вершину 6, закрашиваем ее черным цветом, как пройденную, а смежную вершину 4 в серый как смежную

Очередь : 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | Ч | Б | Ч | С |
| Расстояния | 0 | I | 1 | 1 | 4 | 2 | 3 |
| Массив предыдущих вершин | N | N | 0 | 0 | 5 | 3 | 4 |

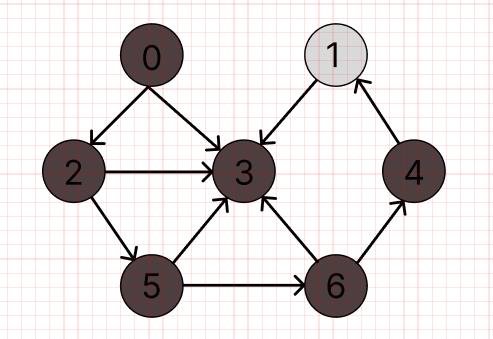
****

**Шаг 7**

Закрашиваем вершину номер 4 в черный, как пройденную, а вершину номер 1 в серый как смежную.

Очередь : 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | Ч | Б | Ч | С |
| Расстояния | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| Массив предыдущих вершин | 0 | 6 | 0 | 0 | 5 | 3 | 4 |

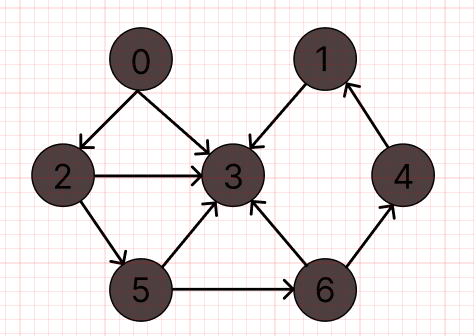
****

**Шаг 8**

Закрашиваем вершину номер 1 в черный цвет, как пройденную. Больше вершин не осталось.

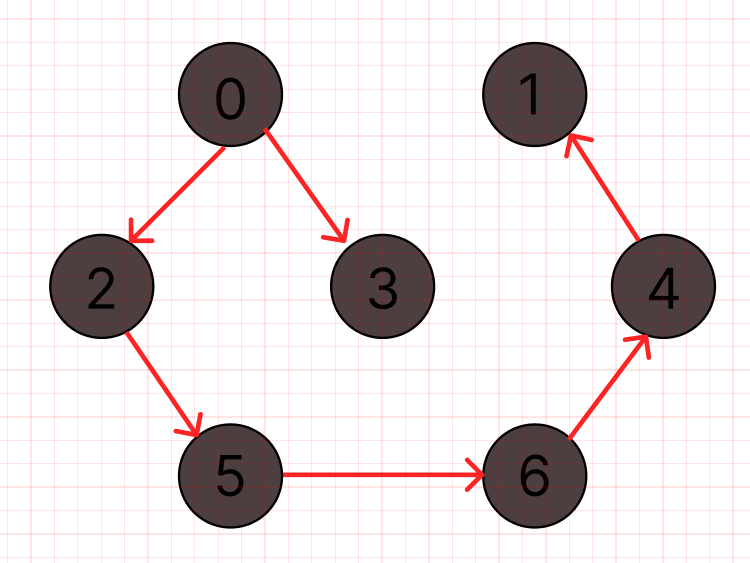
Очередь :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Б | Ч | Ч | Б | Ч | С |
| Расстояния | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| Массив предыдущих вершин | N | 6 | 0 | 0 | 5 | 3 | 4 |

****

Очередь пуста – следует конец обхода.

***BFS-дерево*** – это дерево, множество вершин которого является подмножеством вершин исходного графа, связанных дугами в порядке их посещения (в соответствии с массивом **P**), а корнем – стартовая вершина.

****

Порядок обхода:

0, 2, 3, 5, 6, 4, 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Массив предыдущих вершин | - | 6 | 0 | 0 | 5 | 3 | 4 |

**Алгоритм BFS** сводится к следующей последовательности шагов.

1. Инициализировать массивы **С**, **D**, **P**. Стартовую вершину **s** поместить в очередь **Q**. и окрасить в серый цвет: **C[s] = G**. Для стартовой вершины установить расстояние, равное нулю: **D[s] = 0**.
2. Если очередь **Q** пуста, то работа алгоритма завершена, в противном случае перейти к следующему шагу.
3. Выбрать из очереди **Q** вершину **k** и окрасить ее в черный цвет: **С[k] = B**.
4. Построить множества **J** вершин белого цвета смежных вершине **k**. Если таких вершин нет, то перейти к шагу 2, иначе – к следующему шагу.
5. Каждую вершину **j** из множества **J** поместить в очередь **Q**. Обычно (но не обязательно) в очередь вершины помещаются в порядке возрастания номеров.
6. Каждую вершину **j** из множества **J** окрасить в серый цвет: **С[j] = G**.
7. Для каждой вершины **j** из множества **J** вычислитьрасстояние: **D[j] = D[k] + 1**.
8. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
9. Перейти к шагу 3.

**Алгоритм поиска (или обхода) в глубину** (англ. depth-first search, DFS) позволяет построить обход графа, при котором посещаются все вершины, доступные из начальной вершины.

**Алгоритм поиска в глубину:**

T-шаг

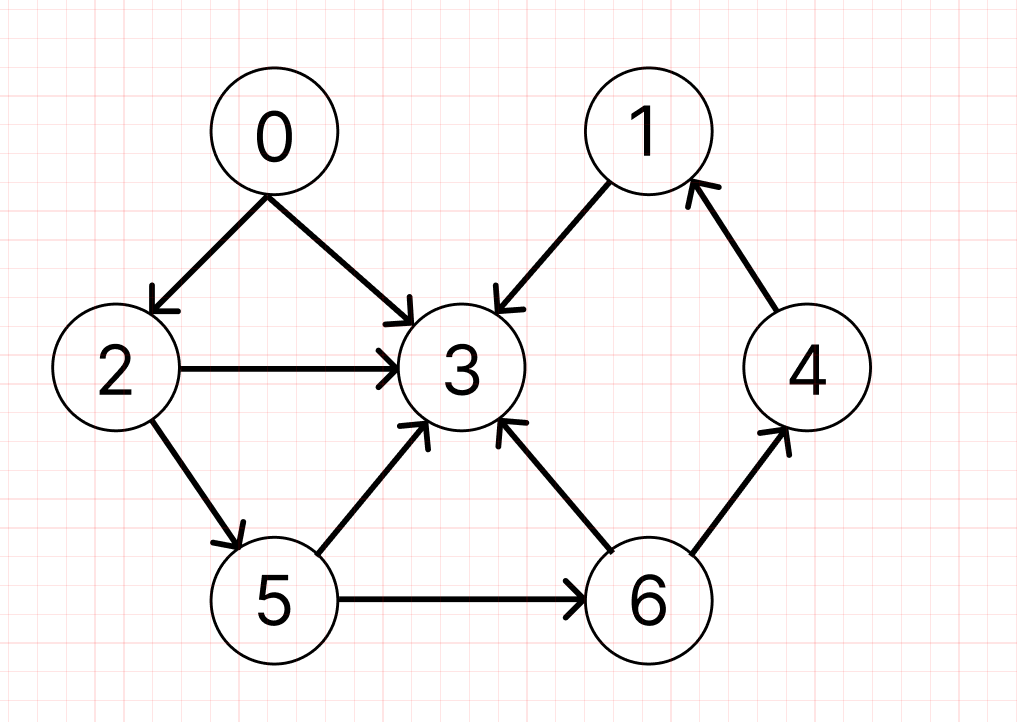
Стек посещённых вершин

Массив окраски вершин ( Б – не посещена , С – в стеке , Ч – вышла из стека)

Массив расстояний

Массив предшествующих вершин

Массив шаг на котором вершина окрашивается в черный цвет

****

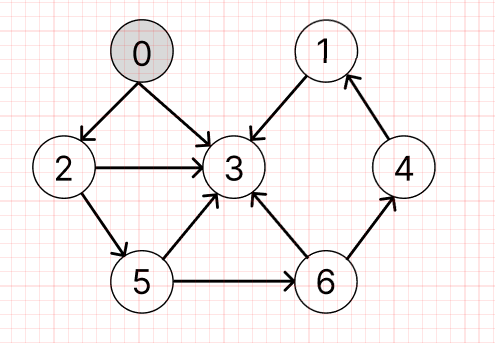
**Шаг 1**

По условию граф имеет 7 вершин, пронумерованных с нуля. В качестве стартовой вершины возьмем вершину с номером 0.

Стек : 0

t(шаг) = 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | Б | Б | Б | Б | Б |
| Время обнаружения (Шаг окраски в серый) | 1 | I | I | I | I | I | I |
| Номера предыдущих вершин | N | N | N | N | N | N | N |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 2**

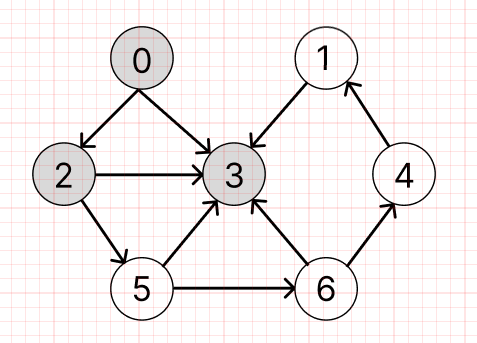
Вершина 0 имеет 2 смежные вершины 2 и 3. Закрашиваем их серым цветом.

И выбираем т что с наименьшим весом.

Стек : 0 2 3

t(шаг) = 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | C | С | Б | Б | Б |
| Время обнаружения (Шаг окраски в серый) | 1 | I | 2 | 2 | I | I | I |
| Номера предыдущих вершин | N | N | 0 | 0 | N | N | N |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

****

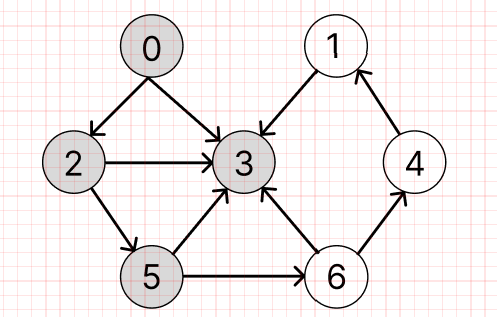
**Шаг 3**

Вершина 2 имеет одну не отмеченную смежную вершину номер 5, закрашиваем ее в серый цвет.

Стек : 0 2 3 5

t(шаг) = 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | C | С | Б | C | Б |
| Время обнаружения (Шаг окраски в серый) | 1 | I | 2 | I | I | 3 | I |
| Номера предыдущих вершин | N | N | 0 | 0 | N | 3 | N |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

****

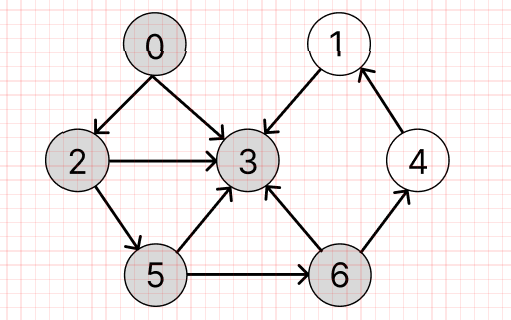
**Шаг 4**

Вершина номер 5 имеет, одну смежную неокрашенную вершину, это 6. Закрашиваем ее в серый цвет.

Стек : 0 2 3 5 6

t(шаг) = 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | C | С | Б | С | С |
| Время обнаружения (Шаг окраски в серый) | 1 | I | 2 | 2 | I | 3 | 4 |
| Номера предыдущих вершин | N | N | 0 | 0 | N | 3 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

****

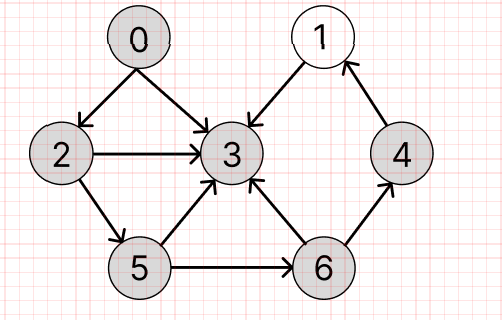
**Шаг 5**

Вершина номер 6 имеет одну смежную не закрашенную вершину, это номер 4, закрашиваем ее, в серый цвет.

Стек : 0 2 3 5 6 4

t(шаг) = 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | С | С | С | С | С |
| Время обнаружения (Шаг окраски в серый) | 1 | I | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | N | 0 | 0 | 5 | 3 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

****

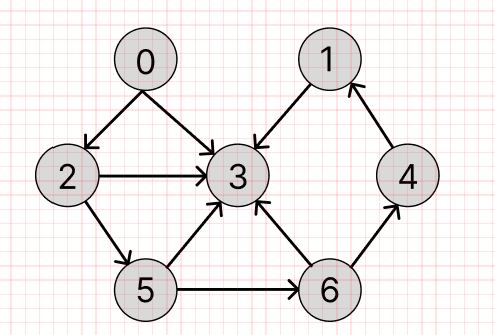
**Шаг 6**

Вершина 4 имеет одну смежную вершину, это один, закрашиваем ее в серый цвет.

Стек : 2 3 5 6 4 1

t(шаг) = 6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | С | С | С | С | С |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



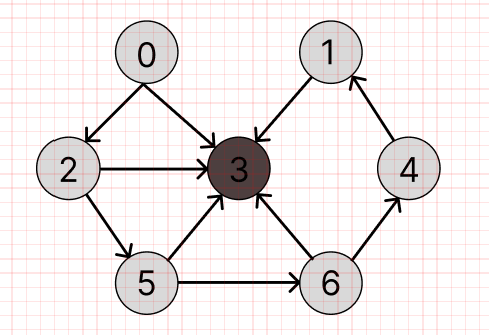
**Шаг 7**

Вершина 1 имеет смежную вершину номер 3, но она уже закрашена в серый цвет и не имеет смежную вершин следовательно закрашиваем вершину номер 3 в черный цвет и таким образом двигаемся по обратному пути.

Стек : 0

t(шаг) = 7

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | С | Ч | С | С | С |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 0 | 0 | 7 | 0 | 0 | 0 |

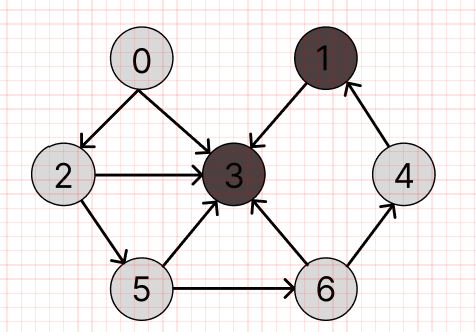


**Шаг 8**

Стек : 0

t(шаг) = 8

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | Ч | С | Ч | Ч |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 8 | 0 | 7 | 0 | 0 | 0 |

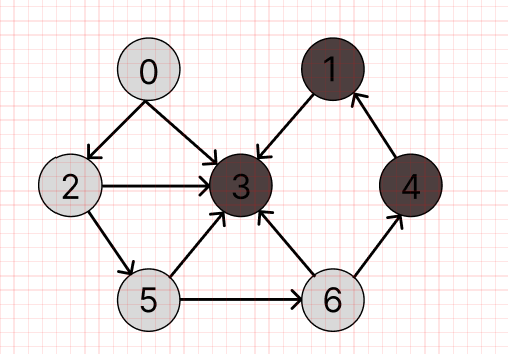


**Шаг 9**

Стек : 0

t(шаг) = 9

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | Ч | Ч | С | С |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 8 | 0 | 7 | 9 | 0 | 0 |

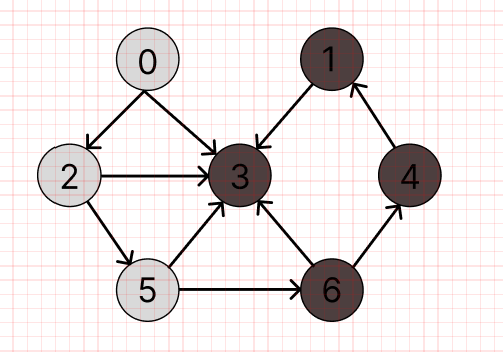
****

**Шаг 10**

Стек : 0

t(шаг) = 10

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | Ч | Ч | С | Ч |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 8 | 0 | 7 | 9 | 0 | 10 |

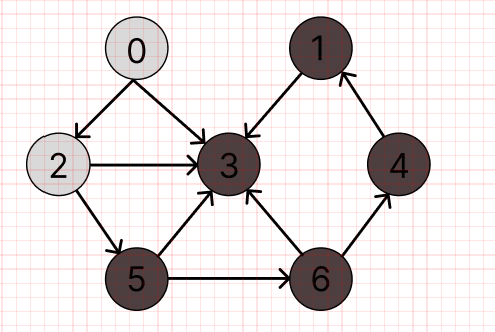
****

**Шаг 11**

Стек : 0

t(шаг) = 11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | Ч | Ч | Ч | Ч |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 8 | 0 | 7 | 9 | 11 | 10 |

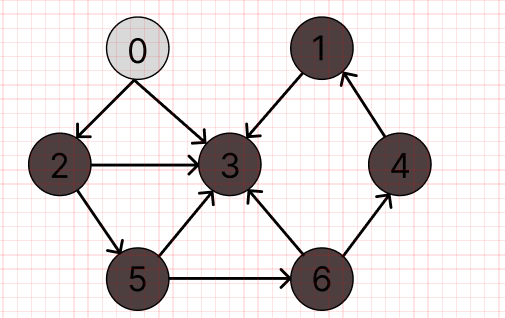
****

**Шаг 12**

Стек : 0

t(шаг) = 12

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Ч | Ч | Ч | Ч | Ч |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 0 | 8 | 12 | 7 | 9 | 11 | 10 |

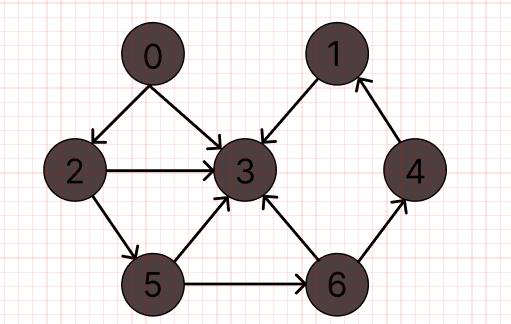
****

**Шаг 13**

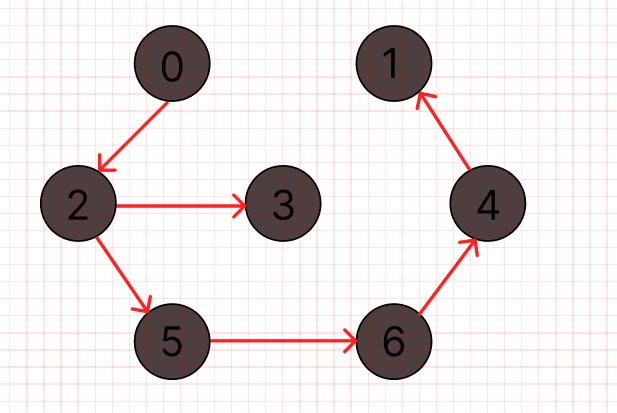
Стек : 0

t(шаг) = 13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | Ч | Ч | Ч | Ч |
| Время обнаружения (Шаг окраски в серый) | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| Номера предыдущих вершин | N | 6 | 0 | 0 | 5 | 2 | 4 |
| Время фиксации  (Шаг окраски в чёрный цыет) | 13 | 8 | 12 | 7 | 9 | 11 | 10 |

****

В результате получили DFS-дерево:

****

Порядок обхода: 1,4,6,5,3,2,0.

Результат

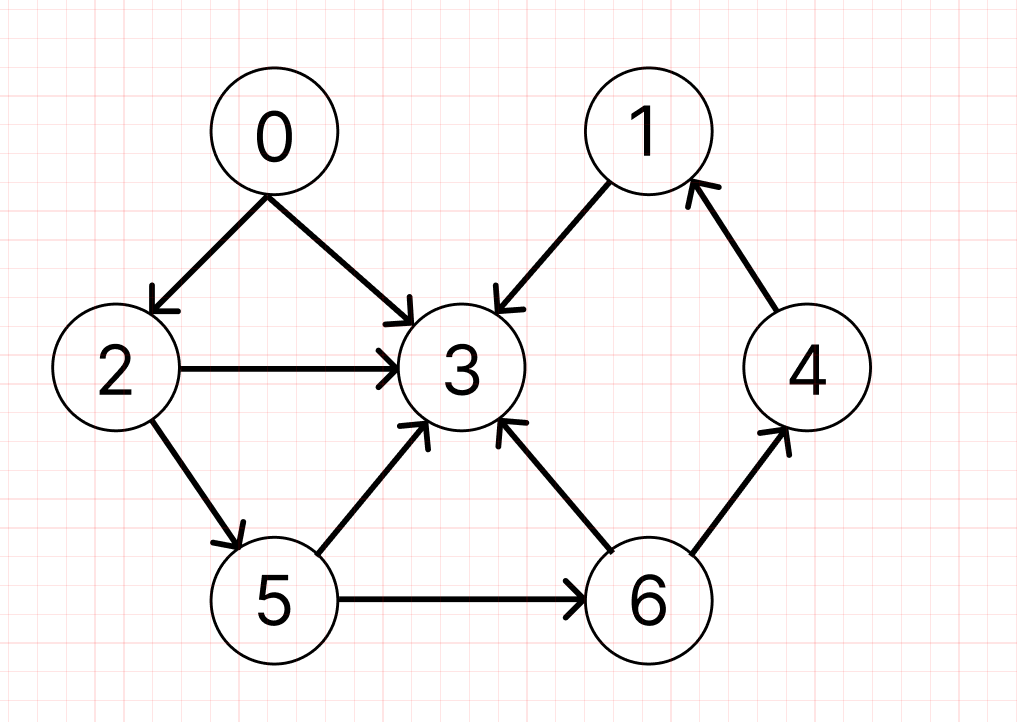
В основе алгоритма DFS лежит рекурсивная процедура **Visit**, имеющая один входной параметр **k** – вершину графа.

Опишем пошагово процедуру **Visit**.

1. Принять параметр **k** – вершину графа.
2. Вершину **k**  окрасить в серый цвет: **C[k] = G**.
3. Увеличить номер шага: **t = t + 1**.
4. Подсчитать расстояние до вершины: **D[k] = t**. Расстояние до вершины в алгоритме DFS совпадает с номером шага, на котором эта вершина была обнаружена (окрашена в серый цвет).
5. Построить множества **J** вершин белого цвета, смежных вершине **k**. Если таких вершин нет, то перейти к шагу 8.
6. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
7. Для каждой вершины **j** из множества **J** выполнить процедуру **Visit**.
8. Вершину **k**  окрасить в черный цвет: **C[k] = B**.
9. Увеличить номер шага: **t = t + 1**.
10. Отметить время фиксации вершины: **F[k] = t**.

**Топологическая сортировка**

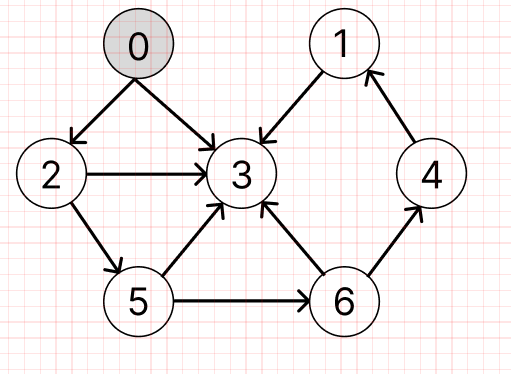
**Топологическая сортировка −** это процедура упорядочивания вершин бесконтурного ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним**.**

****

По условию, граф имеет 7 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

**Шаг 1.**

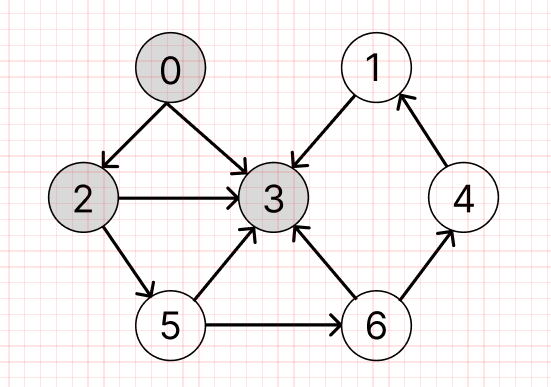
В качестве стартовой вершины выбираем вершину с номером 0. Окрашиваем ее в серый цвет.

****

**Шаг 2.**

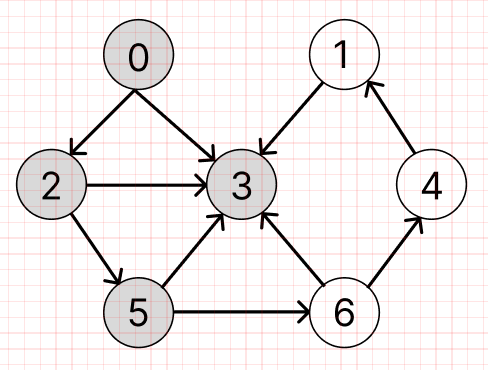
Вершина 0 имеет 2 смежные вершины 2 и 3. Закрашиваем их серым цветом.

И выбираем т что с наименьшим весом.

****

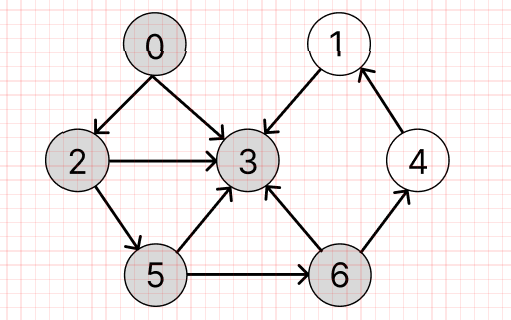
**Шаг 3.**

Вершина 2 имеет одну не отмеченную смежную вершину номер 5, закрашиваем ее в серый цвет.

****

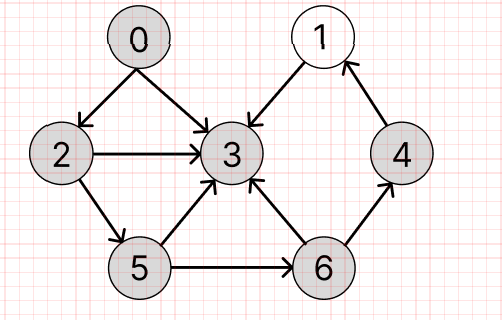
**Шаг 4.**

Вершина номер 5 имеет, одну смежную неокрашенную вершину, это 6. Закрашиваем ее в серый цвет.

****

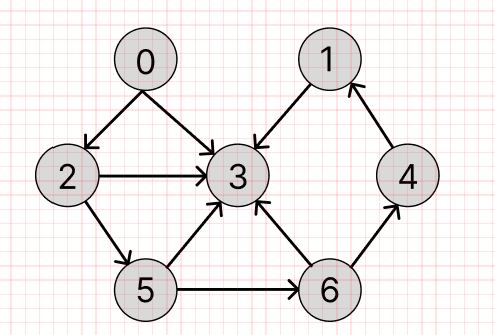
**Шаг 5.**

Вершина номер 6 имеет одну смежную не закрашенную вершину, это номер 4, закрашиваем ее, в серый цвет.

****

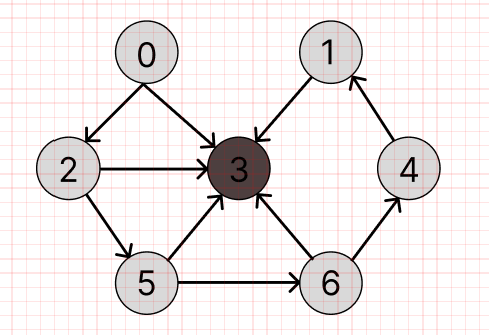
**Шаг 6.**

Вершина 4 имеет одну смежную вершину, это один, закрашиваем ее в серый цвет.



**Шаг 7.**

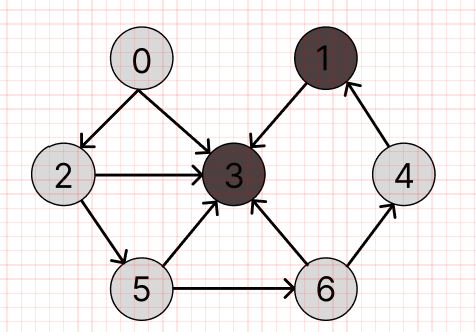
Вершина 1 имеет смежную вершину номер 3, но она уже закрашена в серый цвет и не имеет смежную вершин, следовательно закрашиваем вершину номер 3 в черный цвет и таким образом двигаемся по обратному пути. Добавляем вершину 3 в стек.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** |  |  |  |  |  |  |

**Шаг 8.**

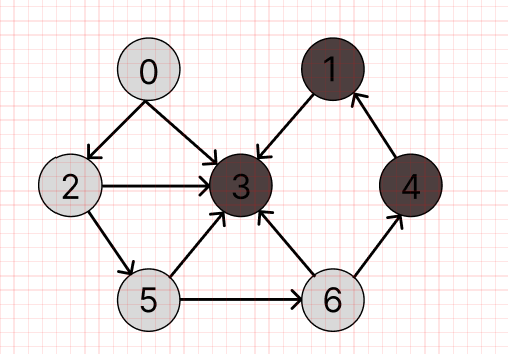
Добавляем вершину 1 в стек.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** |  |  |  |  |  |

**Шаг 9.**

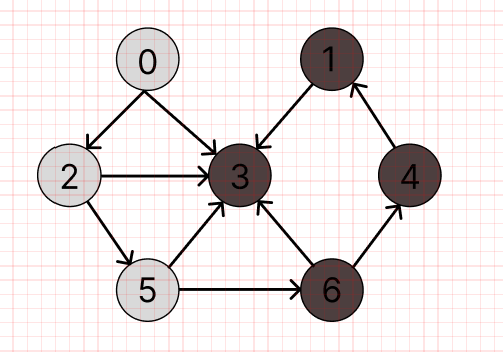
Добавляем вершину 4 в стек.

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** | **4** |  |  |  |  |

**Шаг 10.**

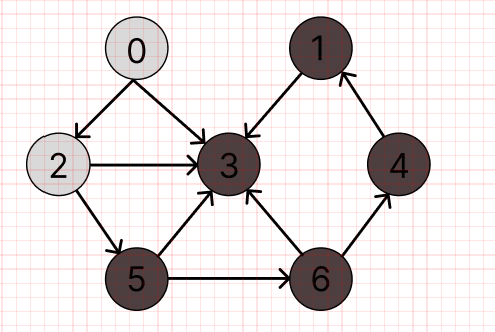
Добавляем вершину 6 в стек.

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** | **4** | **6** |  |  |  |

**Шаг 11.**

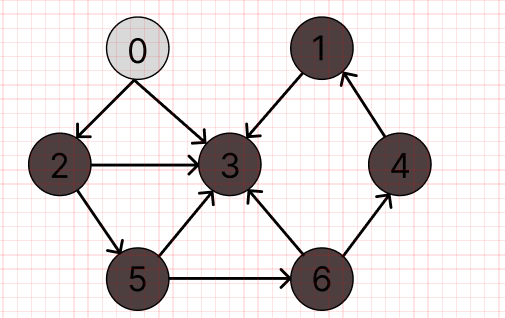
Добавляем вершину 5 в стек.

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** | **4** | **6** | **5** |  |  |

**Шаг 12.**

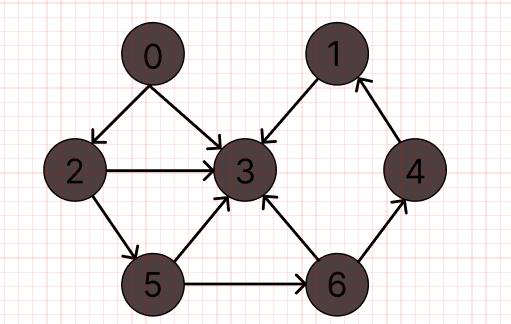
Добавляем вершину 2 в стек.

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** | **4** | **6** | **5** | **2** |  |

**Шаг 13.**

Добавляем вершину 0 в стек.

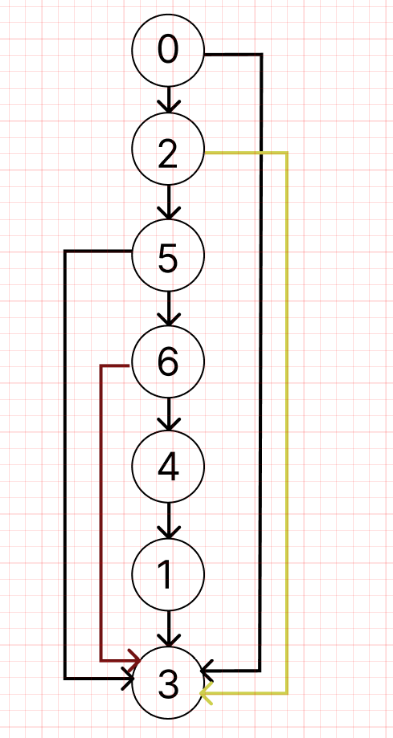
****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **3** | **1** | **4** | **6** | **5** | **2** | **0** |

Таким образом имеем сортировку:

3, 1, 4, 6, 5, 2, 0.

Схема:

****

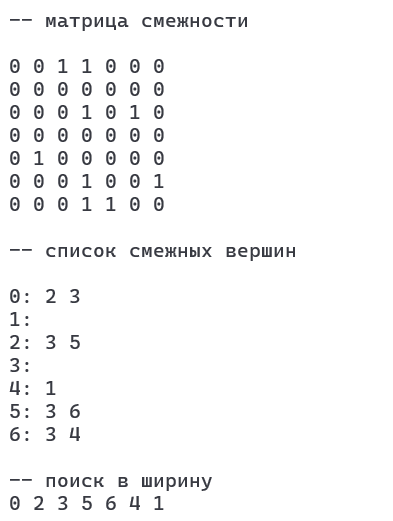
***Задание 3.*** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| #include "BFS.h"  void BFS::init(const graph::AList& al, int s)  {  this->al = &al;  this->c = new Color[this->al->n\_vertex];  this->d = new int[this->al->n\_vertex];  this->p = new int[this->al->n\_vertex];  for (int i = 0; i < this->al->n\_vertex; i++)  {  this->c[i] = WHITE;  this->d[i] = INF;  this->p[i] = NIL;  };  this->c[s] = GRAY;  this->q.push(s);  };  BFS::BFS(const graph::AList& al, int s) { this->init(al, s); };  BFS::BFS(const graph::AMatrix& am, int s)  {  this->init(\*(new graph::AList(am)), s);  };  int BFS::get()  {  int rc = NIL, v = NIL;  if (!this->q.empty())  {  rc = this->q.front();  for (int j = 0; j < this->al->size(rc); j++)  if (this->c[v = this->al->get(rc, j)] == WHITE)  {  this->c[v] = GRAY;  this->d[v] = this->d[rc] + 1;  this->p[v] = rc;  this->q.push(v);  };  this->q.pop();  this->c[rc] = BLACK;  };  return rc;  } |

Листинг 3.1 — файл BFS.cpp

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <queue>  struct BFS  {  const static int INF = 0x7fffffff;  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK }; //  const graph::AList\* al; // исходный граф  Color\* c; // цвет вершины  int\* d; // расстояние до вершины  int\* p; // предшествующая вершина  std::queue<int> q; // очередь  BFS(const graph::AList& al, int s);  BFS(const graph::AMatrix& am, int s);  void init(const graph::AList& al, int s);  int get(); // получить следующую вершину  }; |

Листинг 3.2 — файл BFS.h

****

***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| #include "DFS.h"  #define NINF 0x80000000  #define INF 0x7fffffff  void DFS::init(const graph::AList& al)  {  this->al = &al;  this->c = new Color[this->al->n\_vertex];  this->d = new int[this->al->n\_vertex];  this->f = new int[this->al->n\_vertex];  this->p = new int[this->al->n\_vertex];  this->t = 0;  for (int i = 0; i < this->al->n\_vertex; i++)  {  this->c[i] = WHITE;  this->d[i] = this->f[i] = 0;  this->p[i] = NIL;  };  for (int i = 0; i < this->al->n\_vertex; i++)  if (this->c[i] == WHITE)  {  this->visit(i);  this->topological\_sort.push\_back(i);  }  };  DFS::DFS(const graph::AList& al) { this->init(al); };  DFS::DFS(const graph::AMatrix& am)  {  this->init(\*(new graph::AList(am)));  };  void DFS::visit(int u)  {  int v = NIL;  this->c[u] = GRAY;  this->d[u] = ++(this->t);  for (int j = 0; j < this->al->size(u); j++)  if (this->c[v = this->al->get(u, j)] == WHITE)  {  this->p[v] = u;  this->visit(v);  this->topological\_sort.push\_back(v);  }  this->c[u] = BLACK;  this->f[u] = ++(this->t);  };  int DFS::get(int i)  {  int j = 0, min1 = INF, min2 = NINF, ntx = NIL;  for (int j = 0; j <= i; j++)  {  for (int k = 0; k < this->al->n\_vertex; k++)  if (this->f[k] < min1 && this->f[k] > min2)  {  min1 = this->f[k]; ntx = k;  };  min2 = min1; min1 = INF;  };  return ntx;  }; |

Листинг 4.1 — файл DFS.cpp

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <vector>  struct DFS  {  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK }; //  const graph::AList\* al; // исходный граф  Color\* c; // цвет вершины  int\* d; // время обнаружения  int\* f; // время завершения обработки  int\* p; // предшествующая вершина  int t; // текущее время  DFS(const graph::AList& al);  DFS(const graph::AMatrix& am);  std::vector <int> topological\_sort; //результат топологической сортировки  void visit(int v);  void init(const graph::AList& al);  int get(int i); // получить вершину  }; |

Листинг 4.2 — файл DFS.h



***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| // Функция для топологической сортировки  void topologicalSort(int v, bool visited[], std::stack<int>& Stack, graph::AList& g)  {  // Помечаем текущую вершину как посещенную  visited[v] = true;  // Рекурсивно вызываем функцию для всех смежных вершин  // Если смежная вершина не была посещена, то рекурсивно вызываем функцию  for (int i = 0; i < g.size(v); i++)  if (!visited[g.get(v, i)])  topologicalSort(g.get(v, i), visited, Stack, g);  // Помещаем текущую вершину в стек  Stack.push(v);  }  . . .  std::cout << std::endl << "-- топологическая сортировка" << std::endl;  // Создаем стек для хранения топологической сортировки  std::stack<int> Stack;  // Массив для хранения информации о посещении вершин  bool \*visited = new bool[g5.n\_vertex];  for (int i = 0; i < g5.n\_vertex; i++) visited[i] = false;  // Вызываем функцию для топологической сортировки  for (int i = 0; i < g5.n\_vertex; i++)  if (visited[i] == false)  topologicalSort(i, visited, Stack, g5);  // Выводим топологически отсортированный граф  while (Stack.empty() == false)  {  std::cout << Stack.top() << " ";  Stack.pop();  }  std::cout << std::endl; |

Листинг 5.1— функция топологической сортировки

***Задание 6.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

Веса ребер принять:

W:

W(e0,1)=8; W(e1,0)=5;

W(e0,2)=1; W(e2,0)=3;

W(e0,3)=2; W(e3,0)=8;

W(e1,3)=11; W(e3,1)=4;

W(e1,4)=5; W(e4,1)=3;

W(e2,3)=7; W(e3,2)=9;

W(e2,5)=11; W(e5,2)=10;

W(e4,3)=4; W(e3,4)=1;

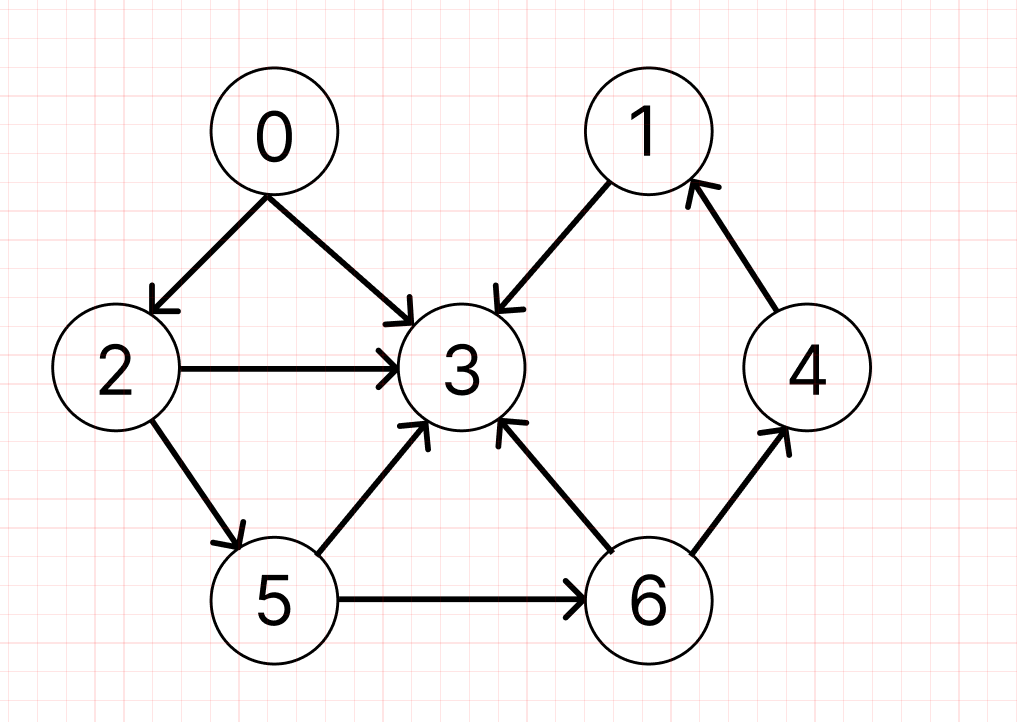
W(e4,6)=10; W(e6,4)=2;

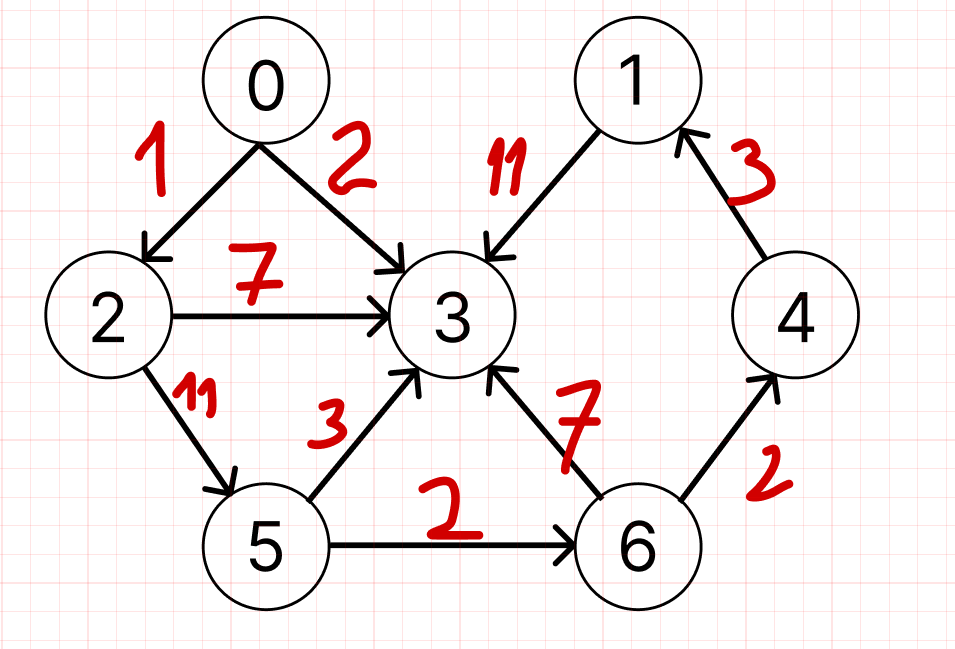
W(e5,6)=2; W(e6,5)=6;

W(e5,3)=3; W(e3,5)=6;

W(e6,3)=7; W(e3,6)=9;

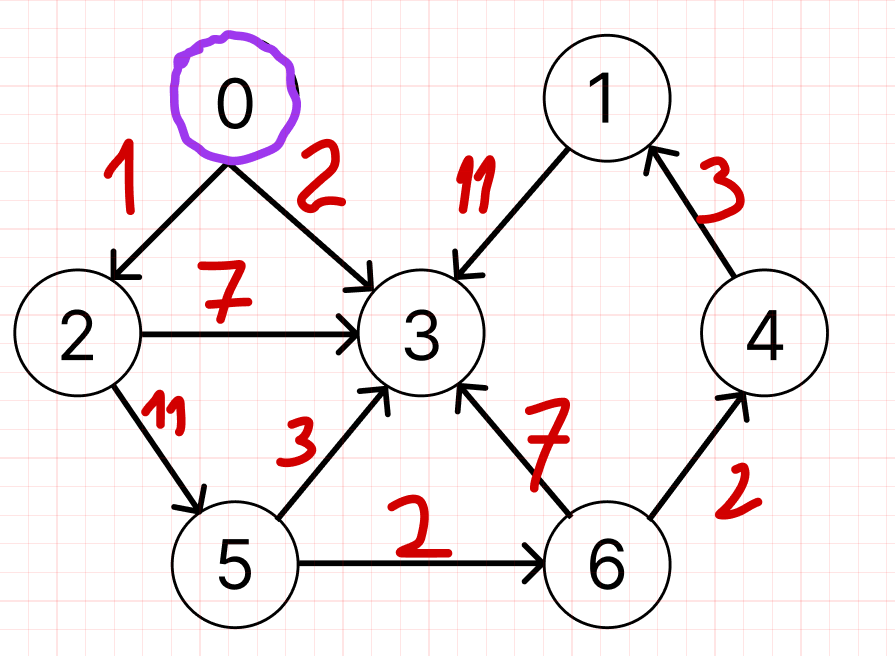
Алгоритм Прима - жадный алгоритм, который находит минимальное остовное дерево для взвешенного неориентированного графа. Это означает, что он находит подмножество ребер, которое формирует дерево, включающее каждую вершину, где общий вес всех ребер в дереве минимизирован.

****

****

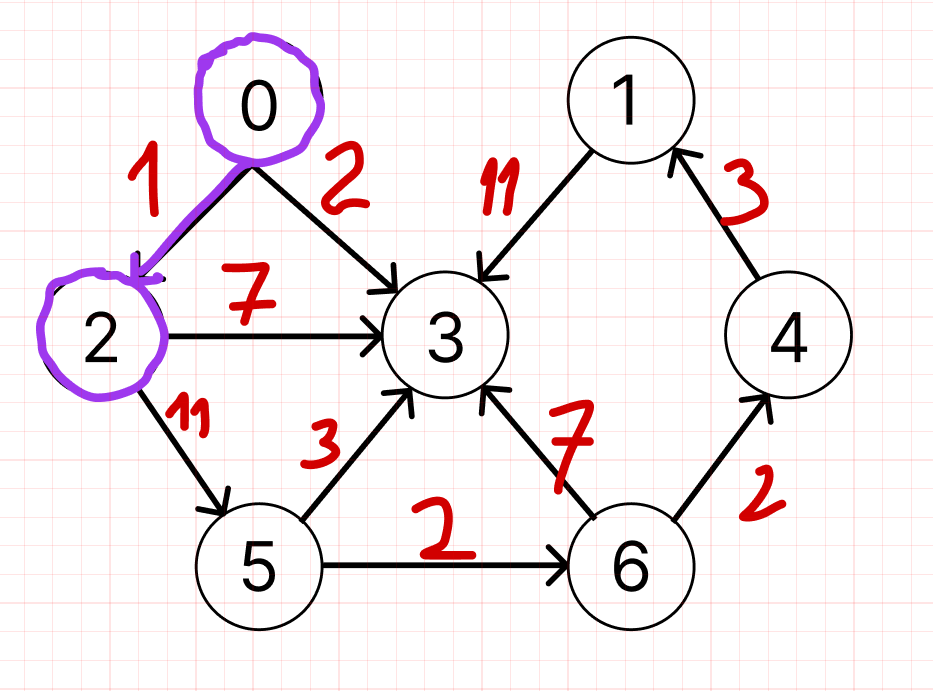
**Шаг 1:**

На первом шаге выбирается стартовая вершина (вершина 0) и окрашивается.

****

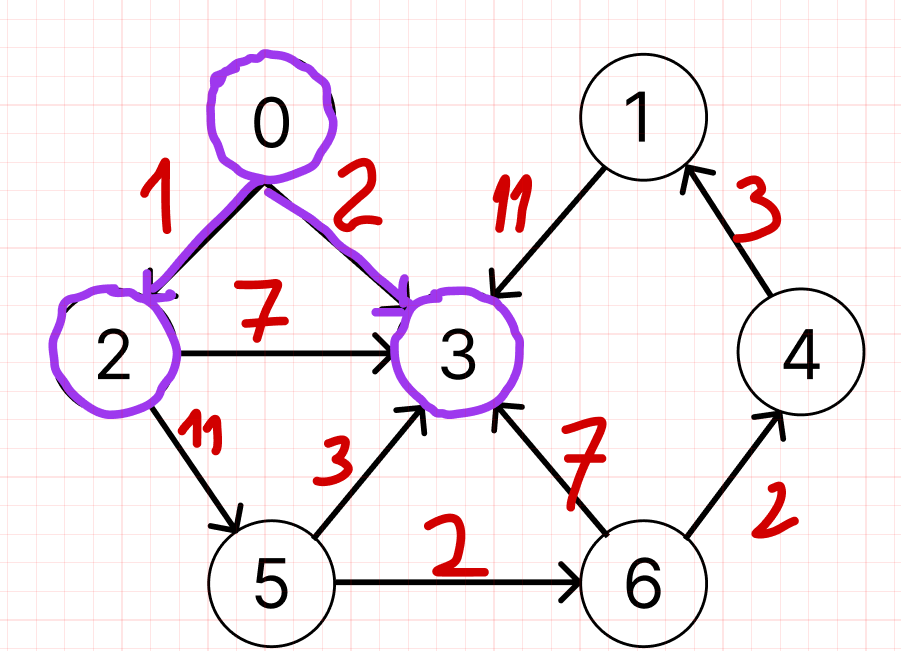
**Шаг 2:**

Среди всех ребер, инцидентных стартовой вершине, отыскивается ребро, имеющее наименьшую длину (на рисунке – ребро (0, 2)). Вторая (неокрашенная) вершина ребра окрашивается, а само ребро вместе с концевыми вершинами включается в будущее минимальное остовное дерево.

****

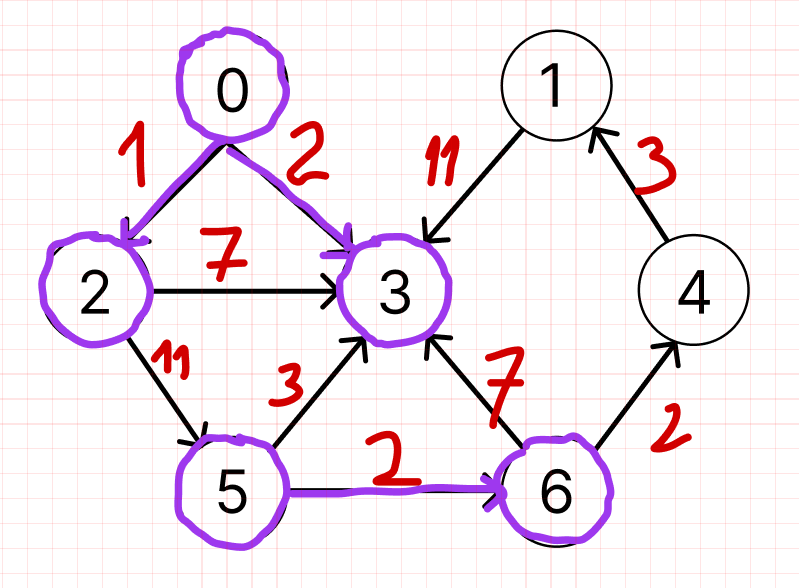
**Шаг 3:**

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (0,3) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

****

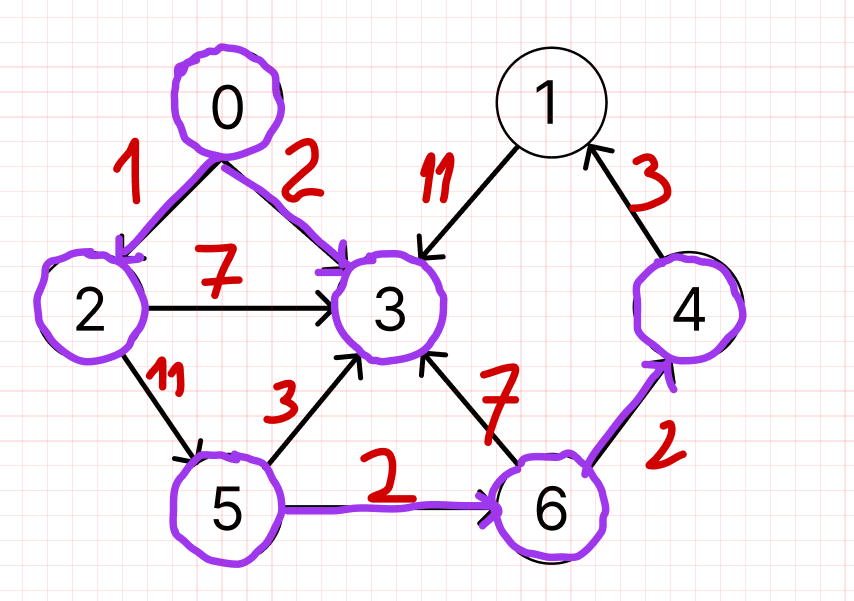
**Шаг 4:**

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (5, 6) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

****

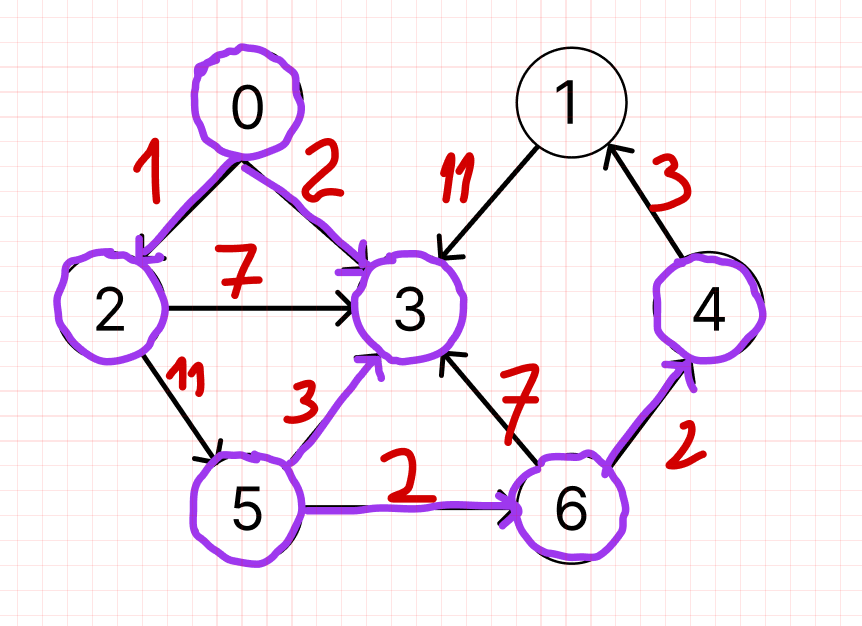
**Шаг 5:**

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (6, 4) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

****

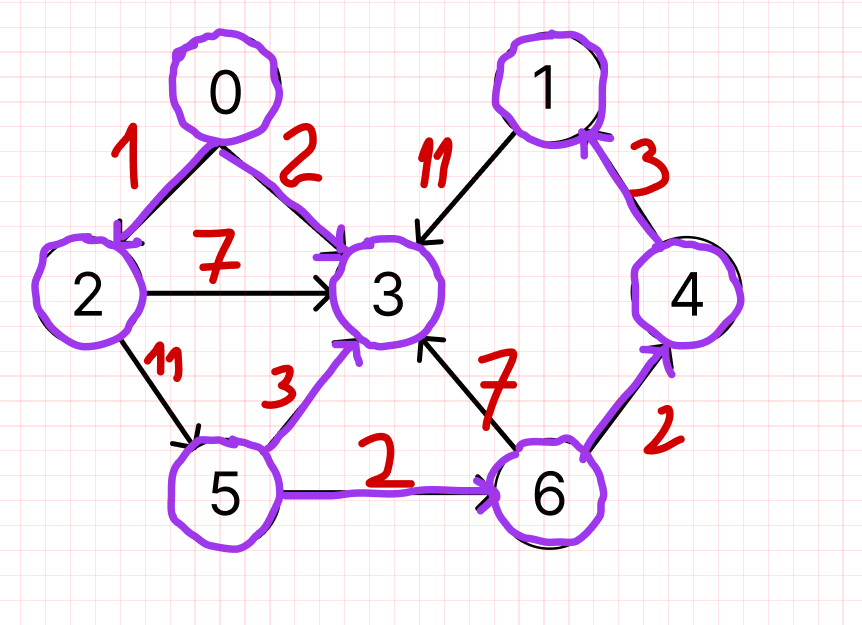
**Шаг 6:**

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (5, 3) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

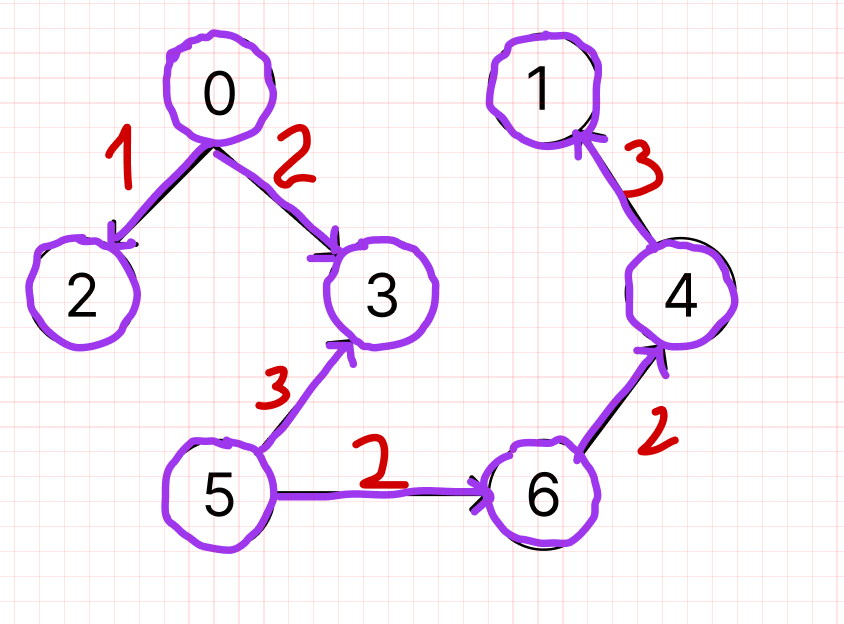
****

**Шаг 7:**

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (4,1) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

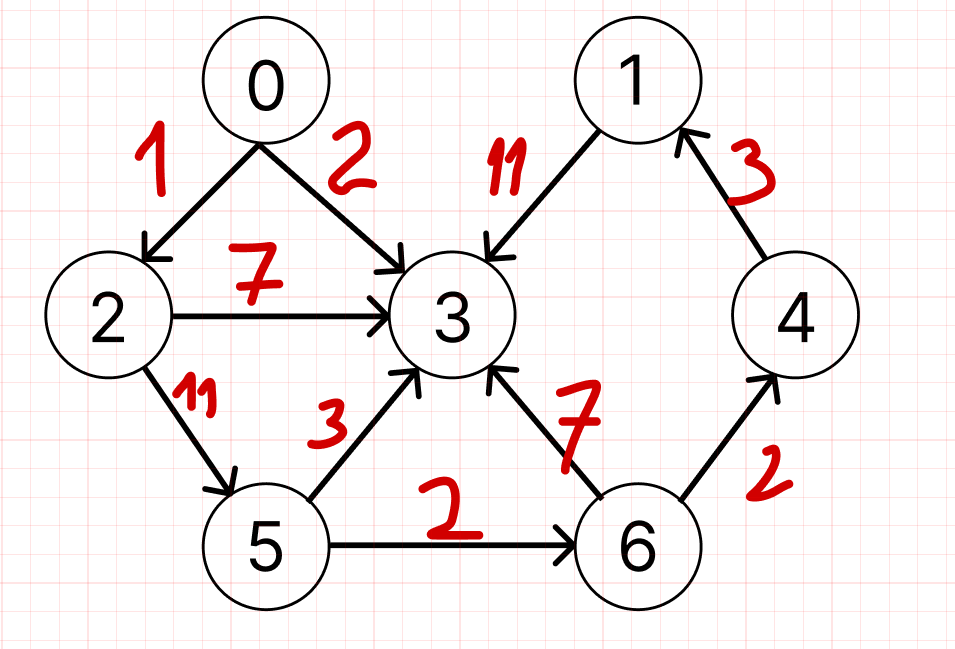
****

**Результат:**

****

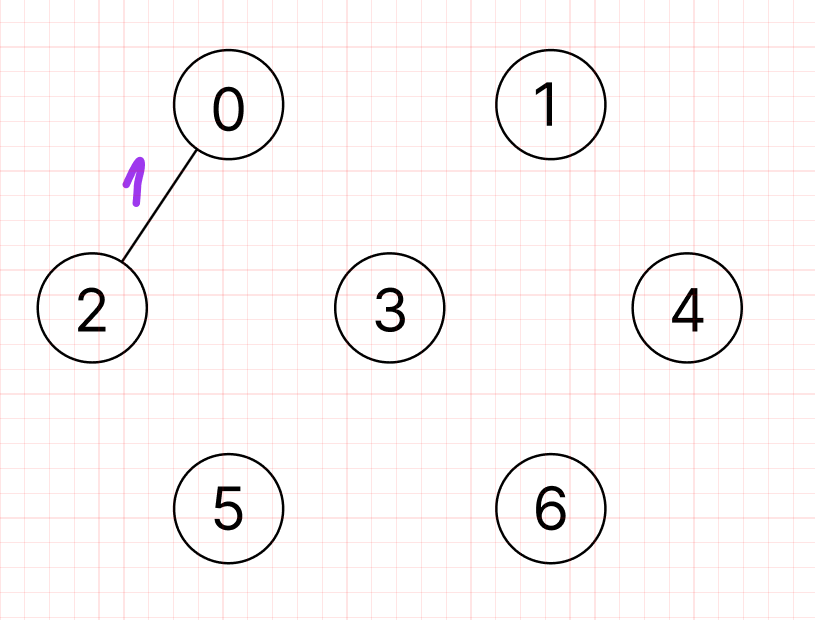
Вес минимального остовного дерева: 13

***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете

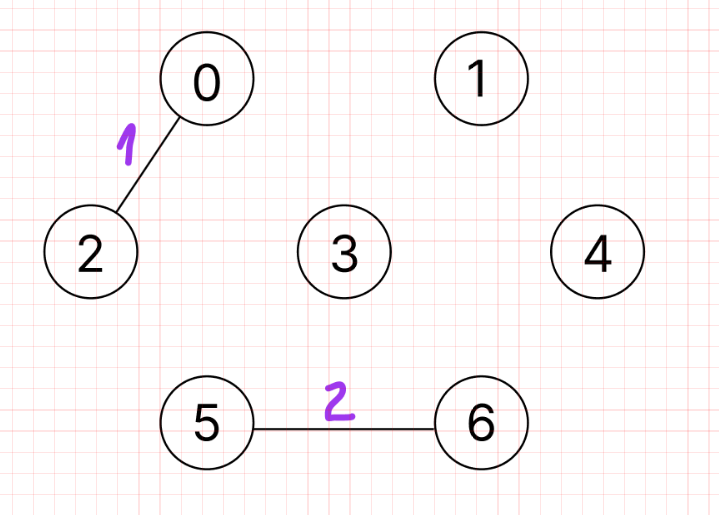
****

**Шаг 1:**

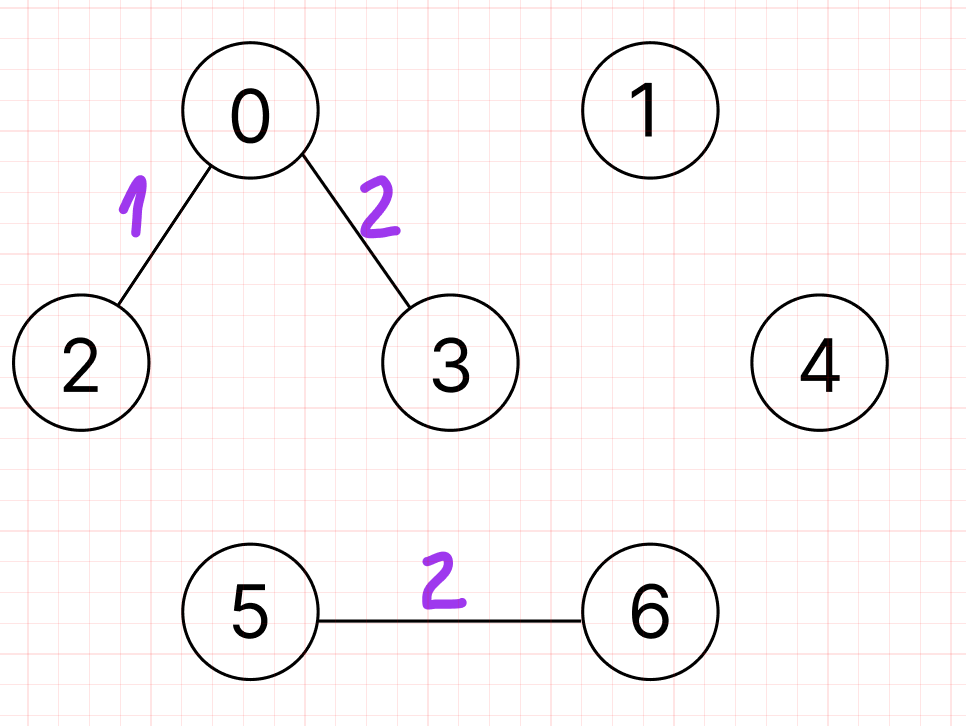
Из всех ребер выбираем ребро с минимальной стоимостью (весом) и включаем это ребро в остовное дерево. Повторяем так далее, однако смотрим на то, чтобы следующее ребро соединяло вершины из разных компонент (чтобы не было циклов)

****

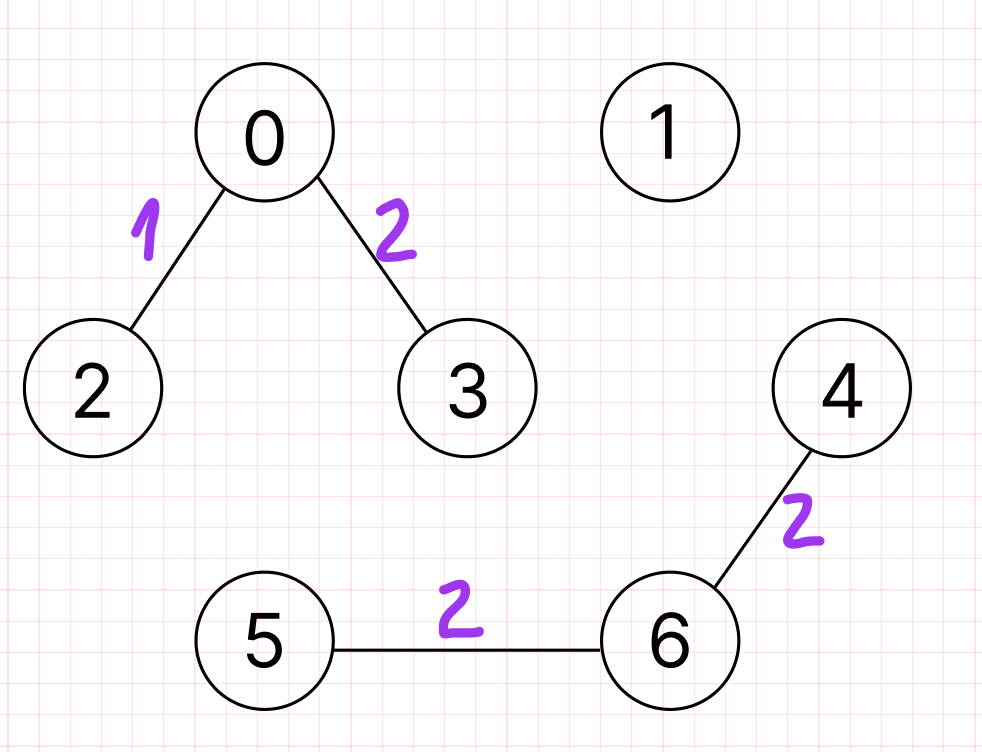
**Шаг 2:**

****

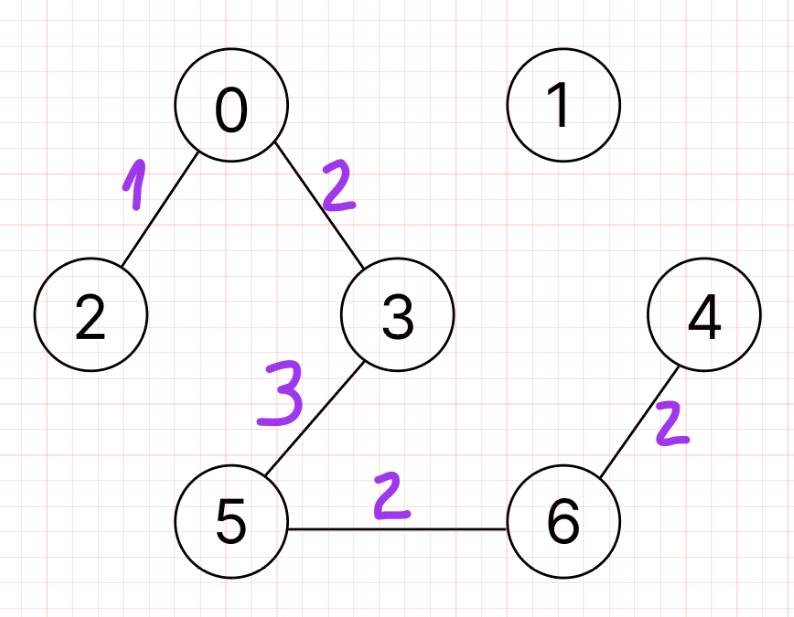
**Шаг 3:**

****

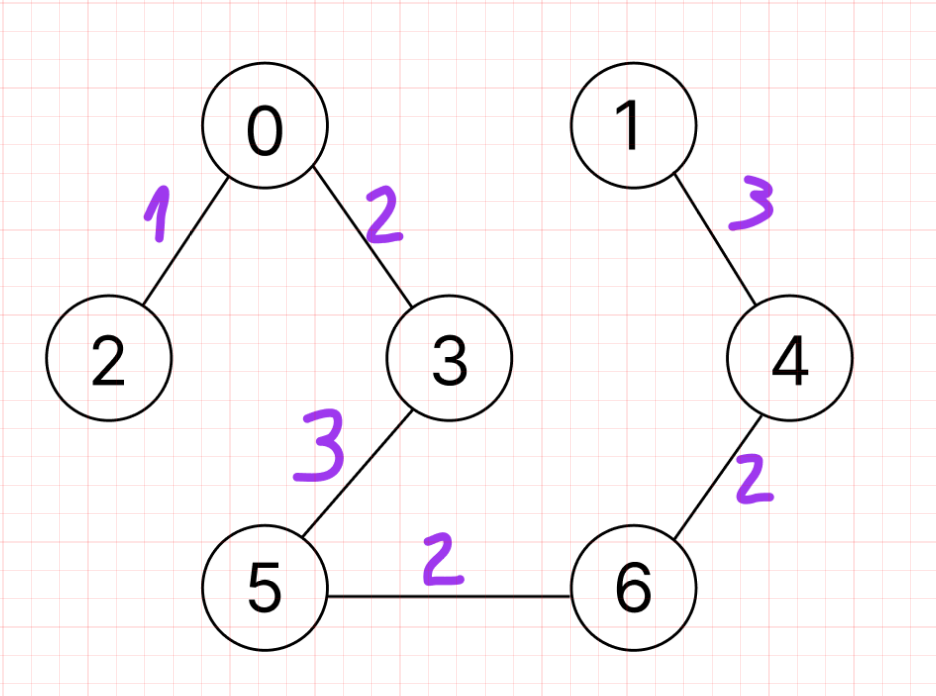
**Шаг 4:**

****

**Шаг 5:**

****

**Шаг 6:**

****

Вес минимального остовного дерева: 13

**Вывод:** была освоена сущность и программная реализация: способов представления графов; алгоритмов поиска в ширину и глубину; алгоритма топологической сортировки графов. Был разобран алгоритм Прима и алгоритм Крускала.