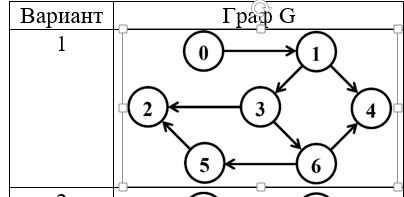
# Лабораторная работа №6. Алгоритмы на графах

**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.

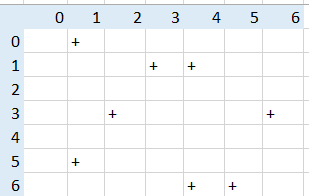
***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.

****

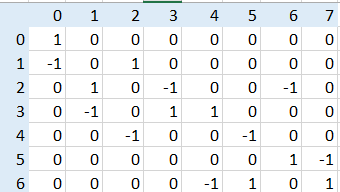
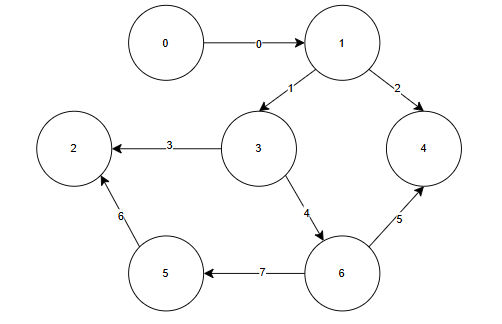
Матрица смежности - это квадратная матрица размера n x n, где n - число вершин в графе. Эта матрица используется для представления графа в виде таблицы, где каждый элемент матрицы a\_ij показывает, существует ли ребро между вершинами i и j.

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:**

****





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:**

0 {1} 1 {3,4} 2 {-} 3 {2,6} 4 {-}

5 {2} 6 {4, 5}

***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.

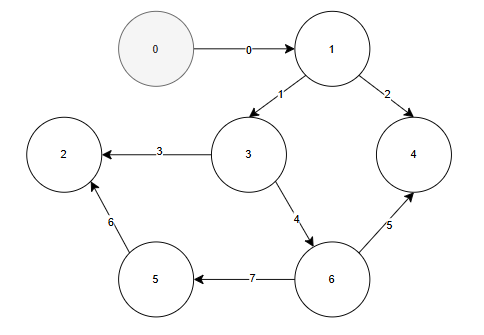
**Алгоритм поиска в ширину (англ. breadth-first search, BFS)** позволяет найти кратчайшие пути из одной вершины невзвешенного графа до всех остальных вершин

**Поиск в ширину:**

1.Посещённые вершины: {}

Очередь: {0}

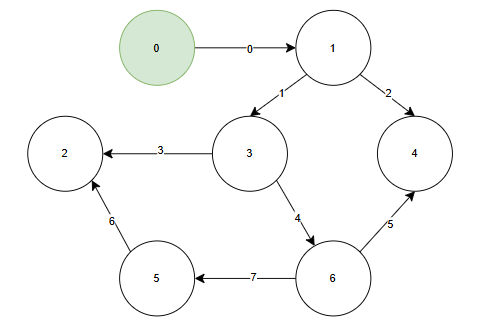
Текущая вершина: {}



2.Посещённые вершины: {0}

Очередь: {}

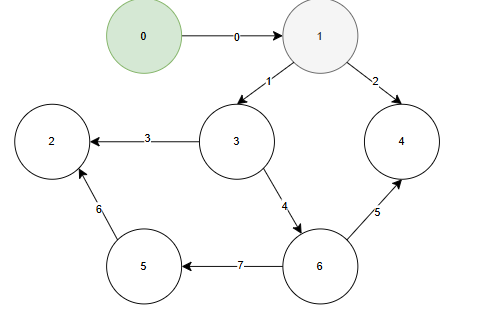
Текущая вершина: {0}



3.Посещённые вершины: {0}

Очередь: {1}

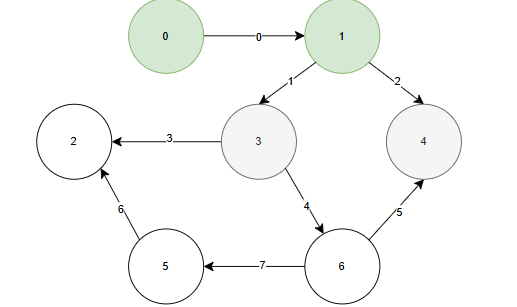
Текущая вершина: {0}



4.Посещённые вершины: {0, 1}

Очередь: {3,4}

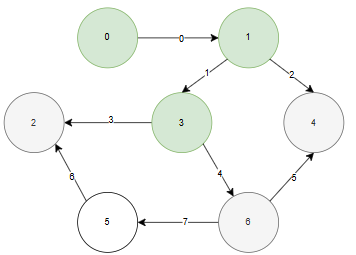
Текущая вершина: {1}



5.Посещённые вершины: {0, 1,3}

Очередь: {4,2,6}

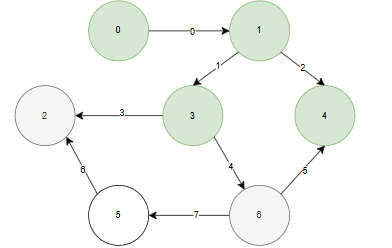
Текущая вершина: {3}



6.Посещённые вершины: {0, 1, 3, 4}

Очередь: {2,6}

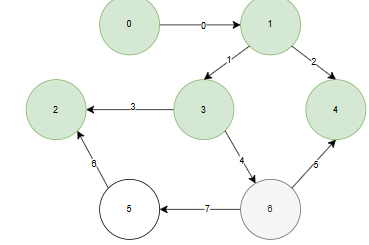
Текущая вершина: {4}



7.Посещённые вершины: {0, 1, 3, 4,2}

Очередь: {6}

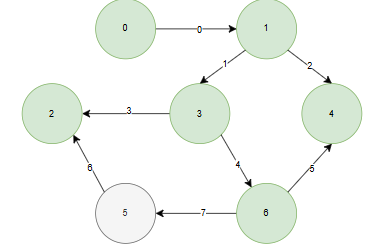
Текущая вершина: {2}



8.Посещённые вершины: {0, 1, 3, 4, 2,6}

Очередь: {5}

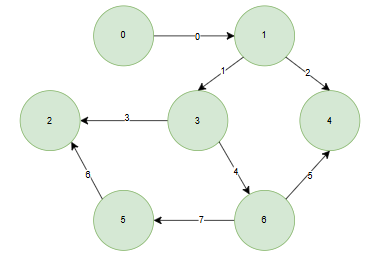
Текущая вершина: {6}



9.Посещённые вершины: {0, 1, 3, 4, 2, 6,5}

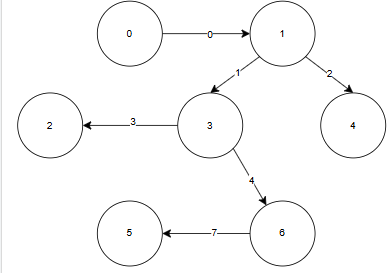
Очередь: {}

Текущая вершина: {5}



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.



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

**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.

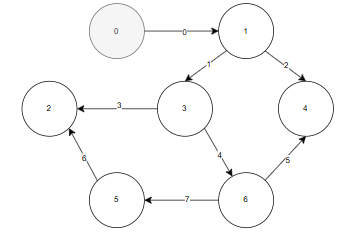
**Алгоритм поиска (или обхода) в глубину** (англ. depth-first search, DFS) позволяет построить обход графа, при котором посещаются все вершины, доступные из начальной вершины.

**Алгоритм поиска в глубину:**

1.Посещённые вершины: {}

Cтек: {0}

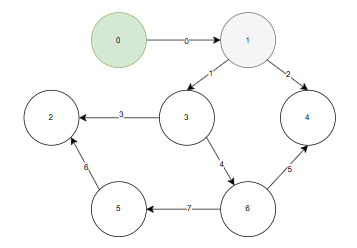
Текущая вершина: {}



2.Посещённые вершины: {0}

Cтек: {1}

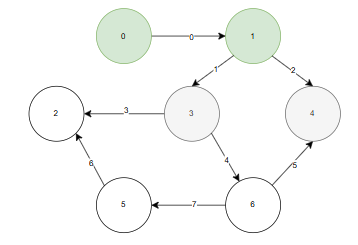
Текущая вершина: {0}



3.Посещённые вершины: {0,1}

Cтек: {3,4}

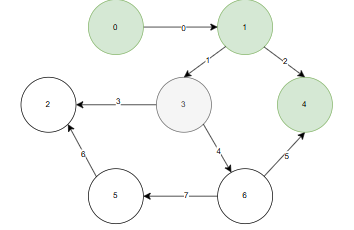
Текущая вершина: {1}



4.Посещённые вершины: {0, 1,4}

Cтек: {3}

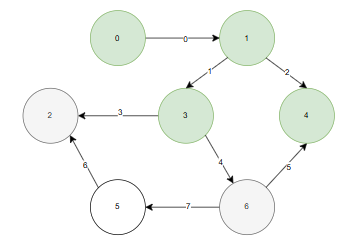
Текущая вершина: {4}



5.Посещённые вершины: {0, 1, 4,3}

Cтек: {2,6}

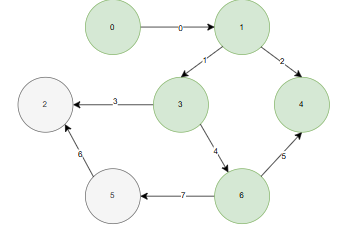
Текущая вершина: {3}



6.Посещённые вершины: {0, 1, 4, 3,6}

Cтек: {2,5}

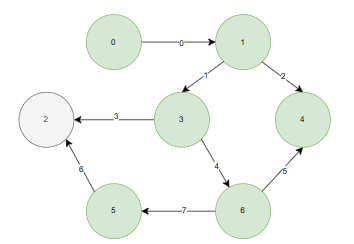
Текущая вершина: {6}



7.Посещённые вершины: {0, 1, 4, 3, 6, 5}

Cтек: {2}

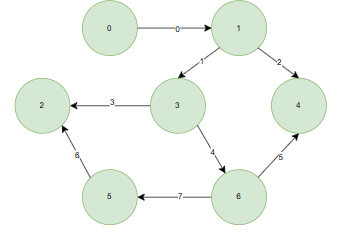
Текущая вершина: {5}



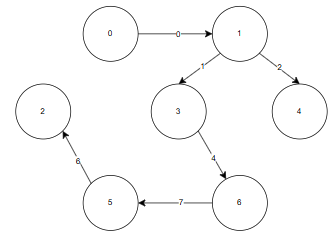
8.Посещённые вершины: {0, 1, 4, 3, 6, 5, 2}

Cтек: {}

Текущая вершина: {2}



Результирующий граф



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

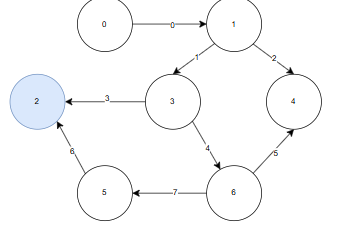
В основе алгоритма DFS лежит рекурсивная процедура **Visit**, имеющая один входной параметр **k** – вершину графа.

Опишем пошагово процедуру **Visit**.

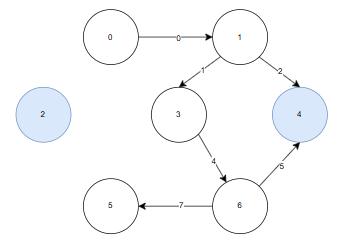
1. Принять параметр **k** – вершину графа.
2. Вершину **k**  окрасить в серый цвет: **C[k] = G**.
3. Увеличить номер шага: **t = t + 1**.
4. Подсчитать расстояние до вершины: **D[k] = t**. Расстояние до вершины в алгоритме DFS совпадает с номером шага, на котором эта вершина была обнаружена (окрашена 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**

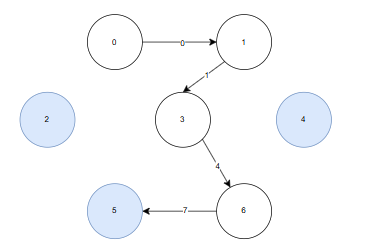
**Топологическая сортировка −** это процедура упорядочивания вершин бесконтурного ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху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



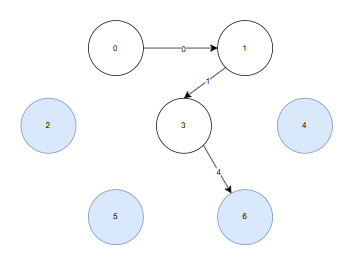
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | **2** |



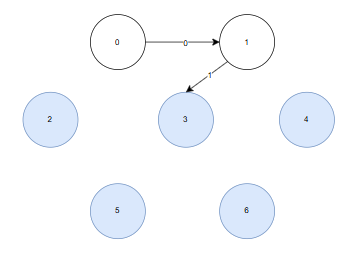
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | 4 | **2** |



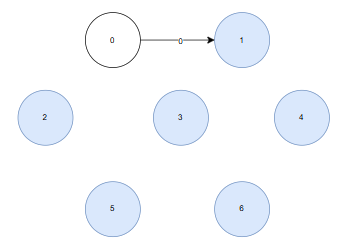
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 5 | 4 | **2** |



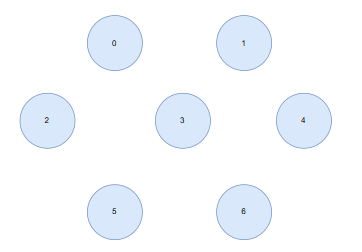
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 6 | 5 | 4 | **2** |



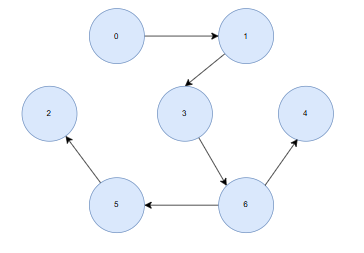
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **3** | 6 | 5 | 4 | **2** |



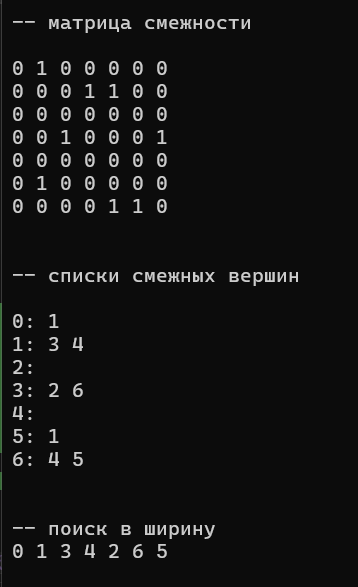
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **1** | **3** | 6 | 5 | 4 | **2** |



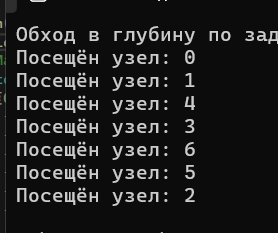
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | **1** | **3** | 6 | 5 | 4 | **2** |



***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.



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



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



***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;

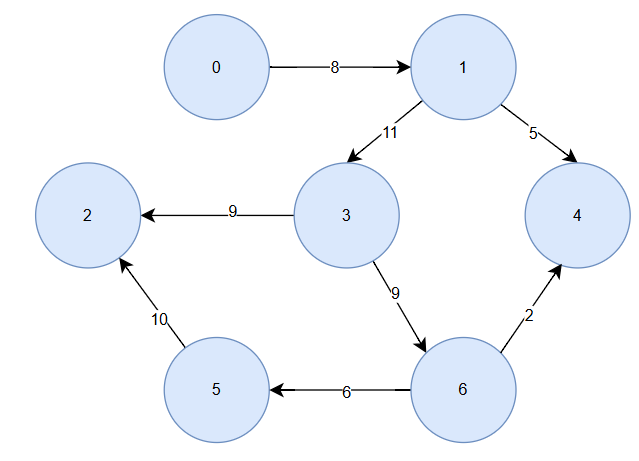
W(e4,3)=4; W(e3,4)=1;

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

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

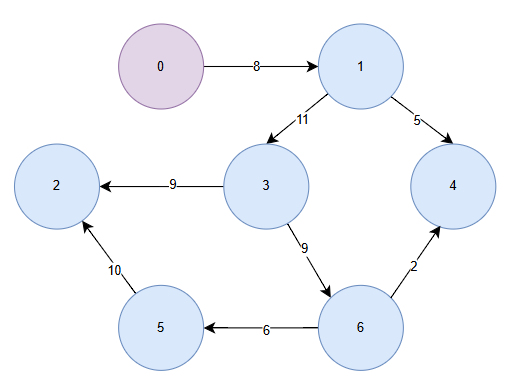
W(e5,3)=3; W(e3,5)=6;

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



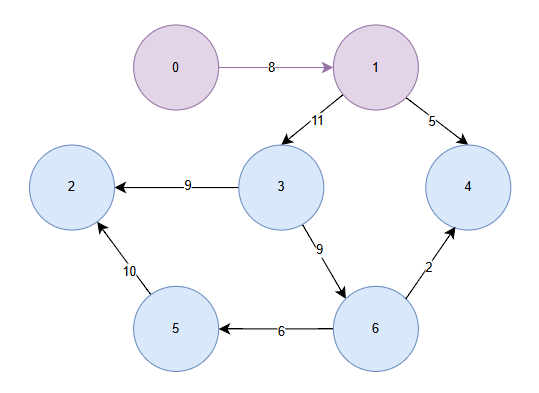
Шаг 1:

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



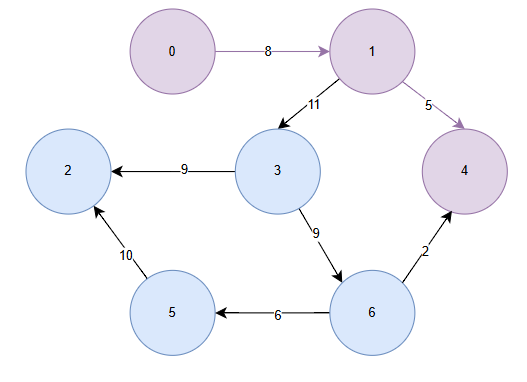
Шаг 2:

Среди всех ребер, инцидентных стартовой вершине, отыскивается ребро, имеющее наименьшую длину (на рисунке – ребро (0, 1)). Вторая (неокрашенная) вершина ребра окрашивается, а само ребро вместе с концевыми вершинами включается в будущее минимальное остовное дерево.



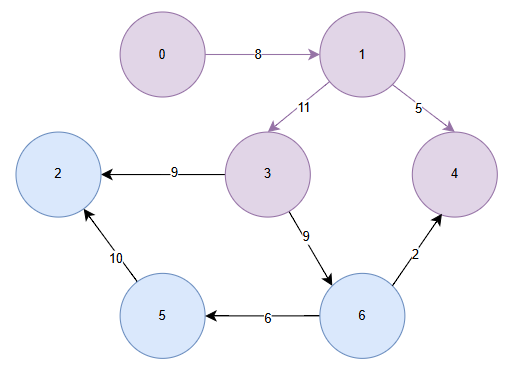
Шаг 3:

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



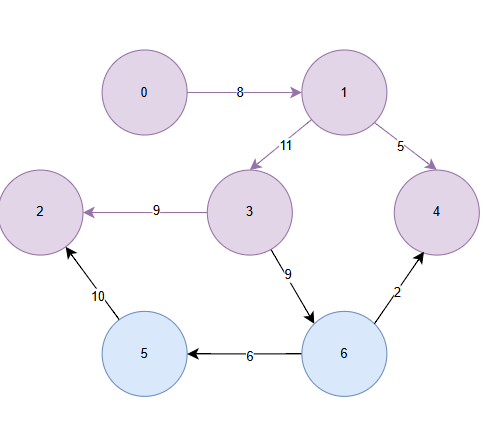
Шаг 4:

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (1, 3) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.



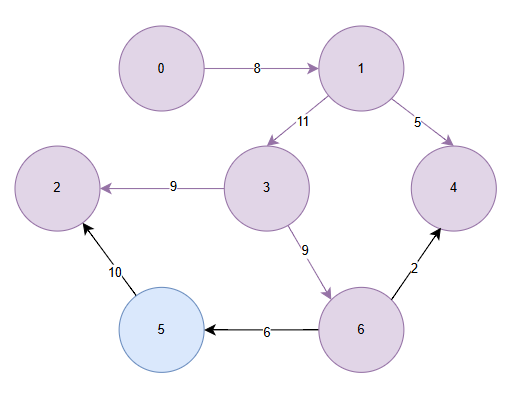
Шаг 5:

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (3, 2) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.



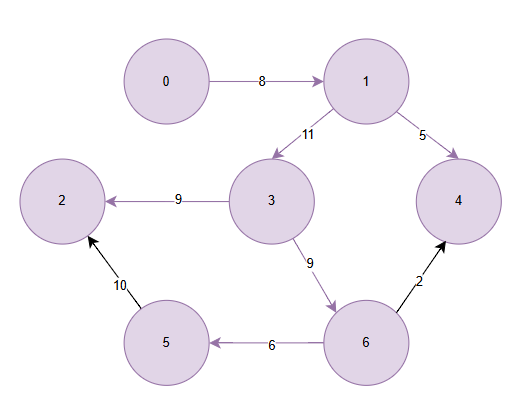
Шаг 6:

Выбирается ребро с минимальной длиной и одной неокрашенной концевой вершиной. Это ребро (3, 6) Неокрашенная вершина окрашивается, выбранное ребра пополняет строящееся минимальное остовное дерево.

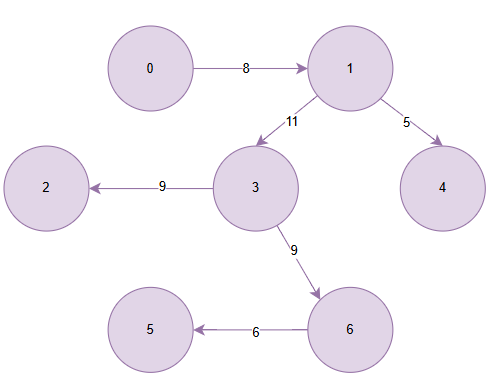


Шаг 7:

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

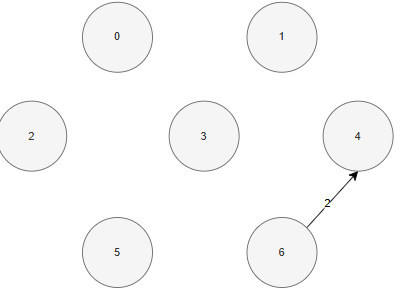


**Результат:**

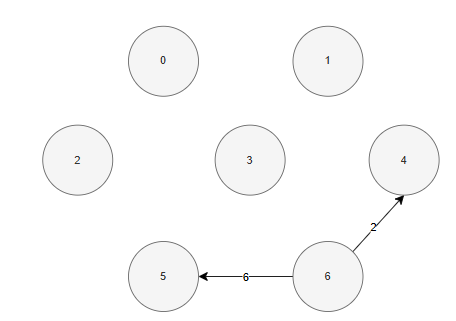


Вес минимального остовного дерева :48

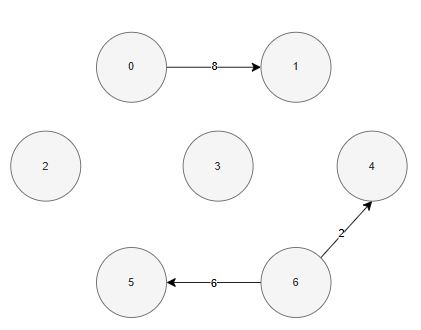
***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете.



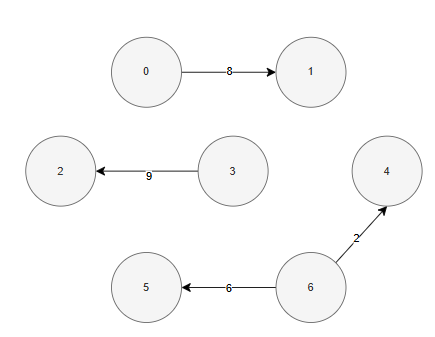
Шаг2:



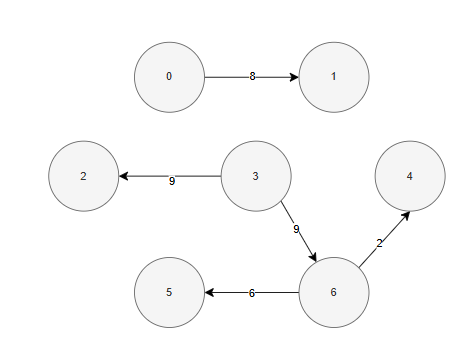
Шаг3:



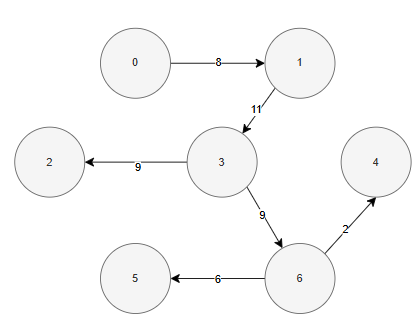
Шаг4:



Шаг5:



Шаг6:



Вес минимального остовного дерева: 45

Вывод: Были освоены сущность и программная реализация способов представления графов; алгоритмов поиска в ширину и глубину; алгоритма топологической сортировки графов. Разобраны алгоритм Прима и алгоритм Крускала.