**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:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
| (0) | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| (1) | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| (2) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| (3) | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| (4) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| (5) | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| (6) | 0 | 0 | 0 | 0 | 1 | 1 | 0 |



e1

e3

e2

e4

e6

e5

e7

e8

Матрица инцидентности –

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | e1 | e2 | e3 | e4 | e5 | e6 | e7 | e8 |
| (0) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| (1) | -1 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |
| (2) | 1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 |
| (3) | 0 | 0 | -1 | 1 | 1 | 0 | 0 | 0 |
| (4) | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 |
| (5) | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |
| (6) | 0 | 0 | 0 | -1 | 0 | 1 | 1 | 0 |

Список смежных вершин:

|  |  |
| --- | --- |
| 0 → | 1 |
| 1 → | 3,4 |
| 2 → |  |
| 3 → | 2,6 |
| 4 → |  |
| 5 → | 2 |
| 6 → | 4,5 |





**Выполнение:**

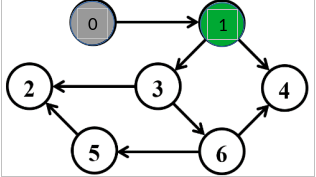
**Поиск в ширину**

Выбираем стартовую вершину 0. Добавляем ее в очередь.

**Очередь: [0]**

**Посещенная очередь: []**

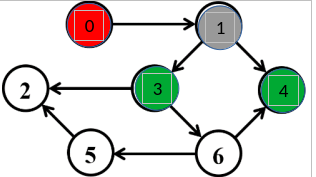
Посещаем вершину 0 и записываем в очередь новые вершины: 1.



**Очередь: [1]**

**Посещенная очередь: [0]**

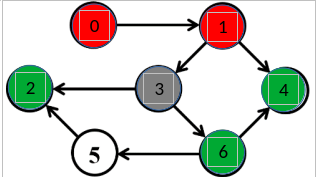
Посещаем вершину 3 и записываем в очередь новые вершины: 3, 4.



**Очередь: [3,4]**

**Посещенная очередь: [0,1]**

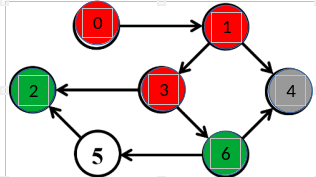
Посещаем вершину 3 и записываем в очередь новые вершины:2,6.



**Очередь: [4, 2, 6]**

**Посещенная очередь: [0,1,3]**

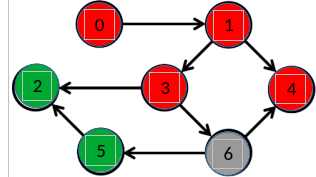
Посещаем вершину 4 и записываем в очередь новые вершины:ничего.



**Очередь: [2, 6]**

**Посещенная очередь: [0,1,3,4]**

Посещаем вершину 6 и записываем в очередь новые вершины: 5.



**Очередь: [6]**

**Посещенная очередь: [0,1,3,4,2]**

Посещаем вершину 6 и записываем в очередь новые вершины:5.

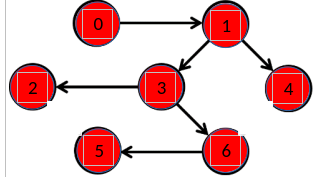
**Очередь: [5]**

**Посещенная очередь: [0,1,3,4,6,2]**

Посещаем вершину 5 и записываем в стек новые вершины:ничего.

**Очередь: []**

**Посещенная очередь: [0,1,3,4,2,6,5]**



**Поиск в глубину**

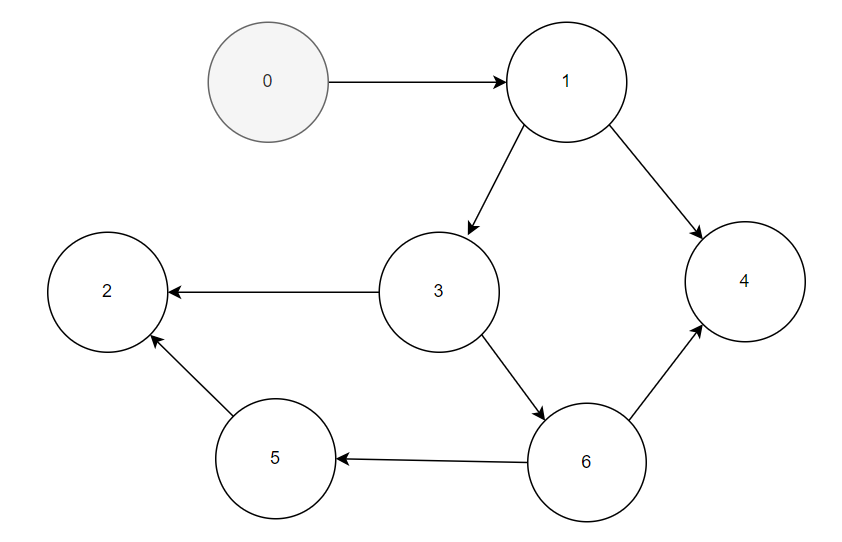


Выбираем стартовую вершину 0

**Стек: [0]**

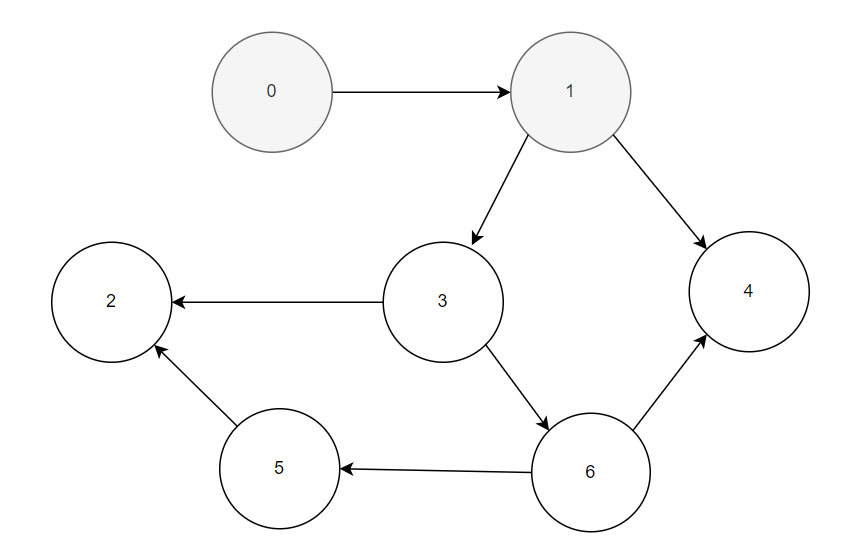
**Посещенные вершины: []**

Посещаем 0. Добавляем в стек:1.



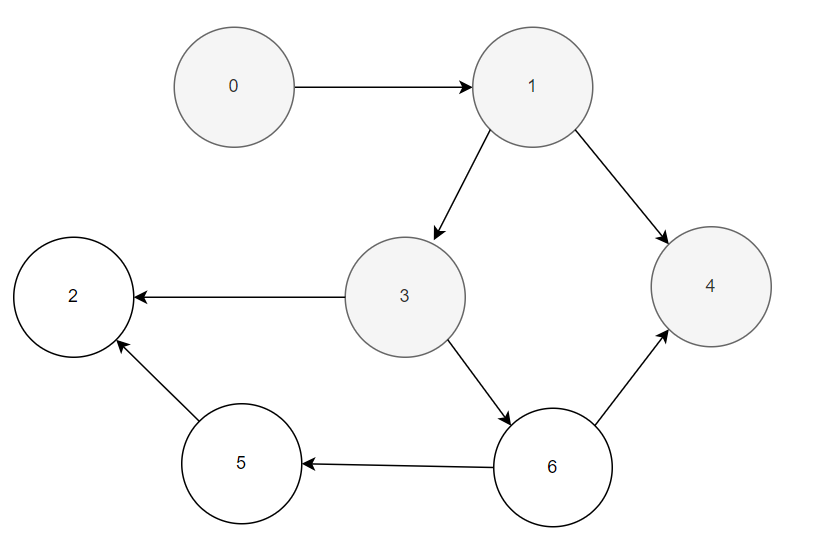
**Стек: [0, 1]**

Посещаем 1. Добавляем в стек: 3, 4.



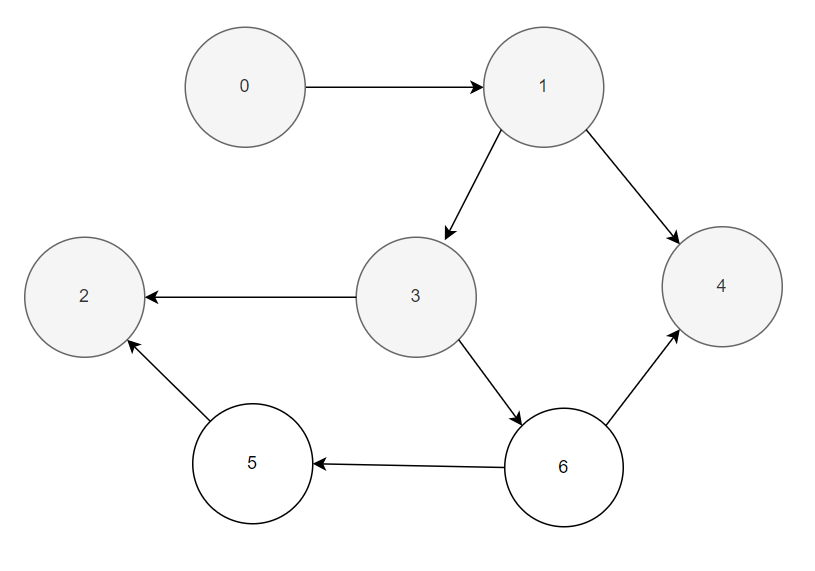
**Стек: [0,1, 4, 3]**

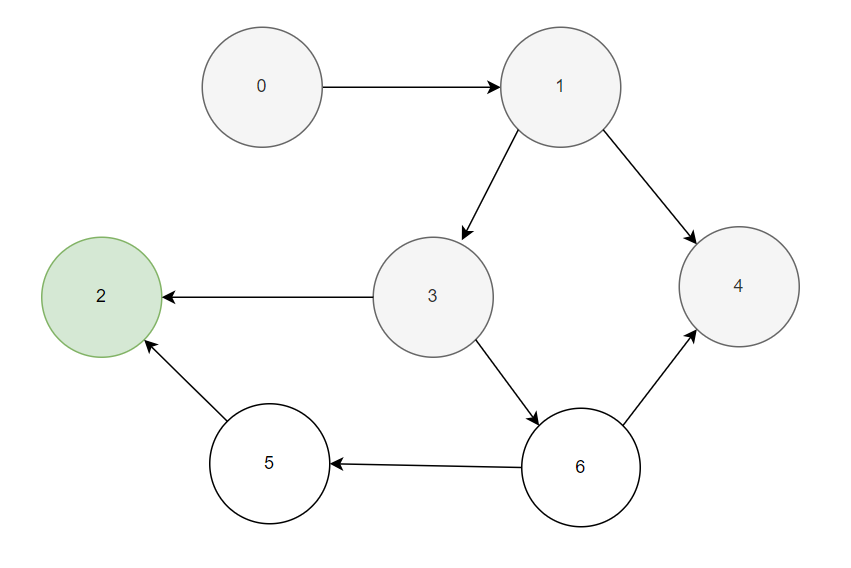
Посещаем 4. Добавляем в стек: ничего.



**Стек: [0,1,3,4,2]**

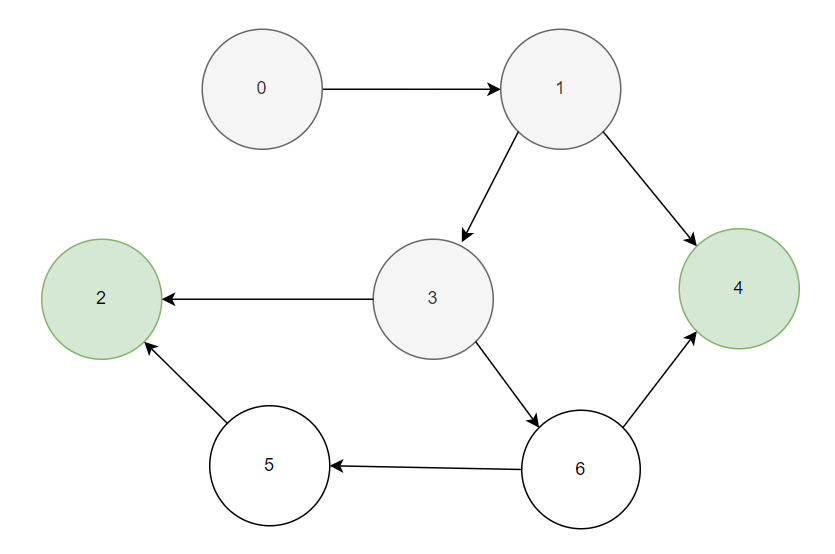
Посещаем 3. Добавляем в стек: 2.





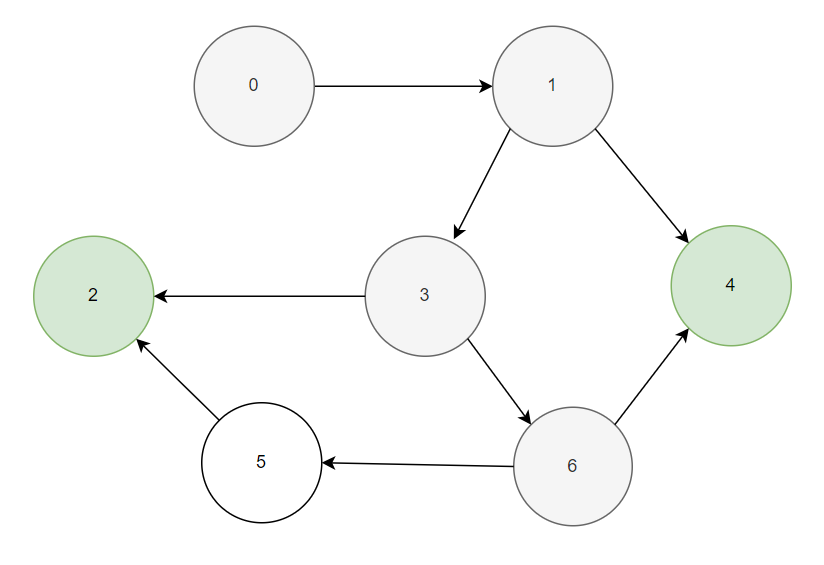
**Стек: [0,1,3,4]**

**Посещенные вершины: [2]**



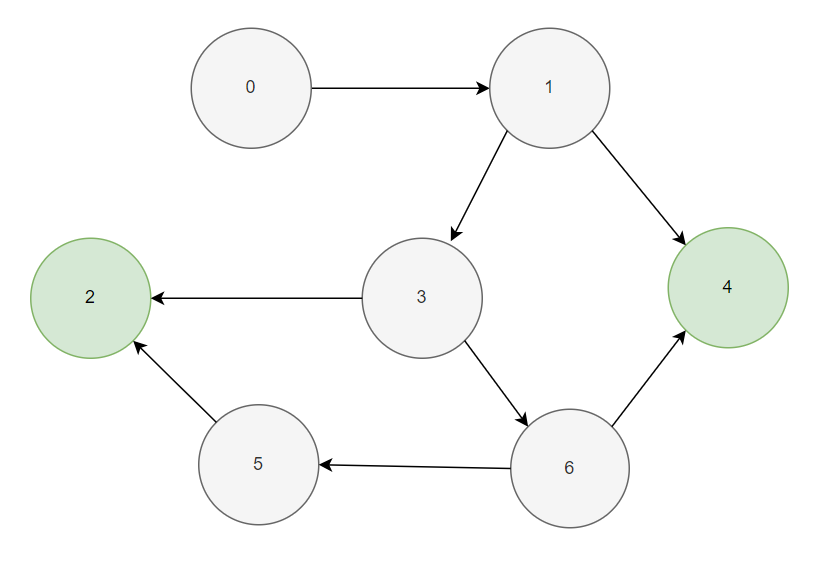
**Стек: [0,1,3]**

**Посещенные вершины: [2,4]**



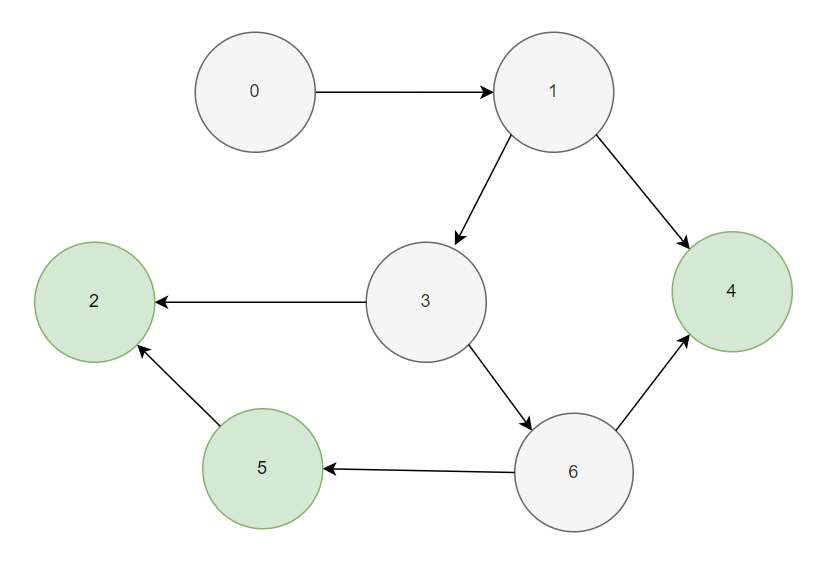
**Стек: [0,1,3,6]**

**Посещенные вершины: [2,4]**



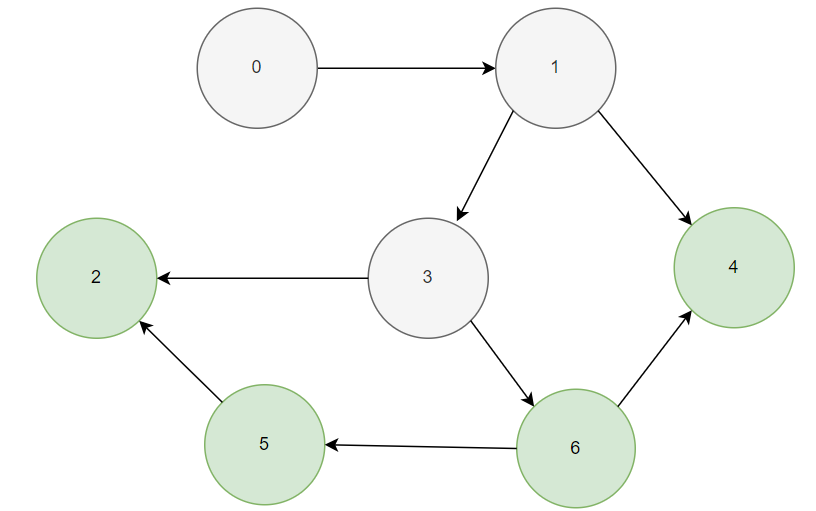
**Стек: [0,1,3,6,5]**

**Посещенные вершины: [2,4]**

****

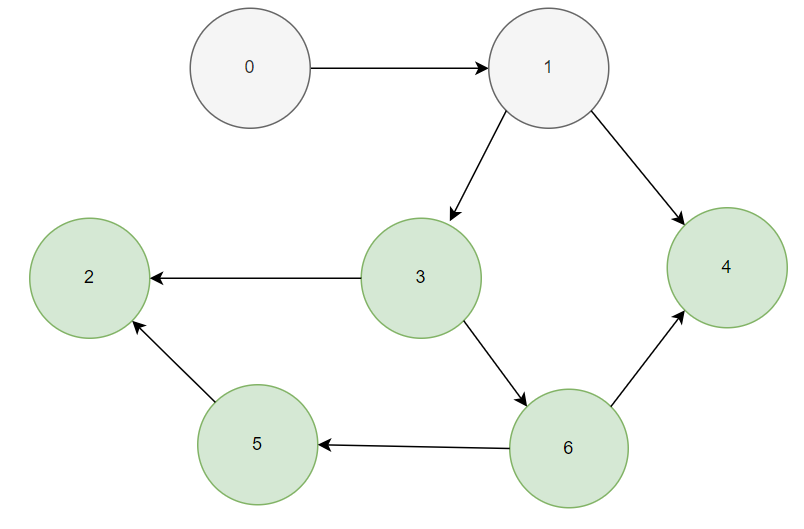
**Стек: [0,1,3,6]**

**Посещенные вершины: [2,4,5]**

****

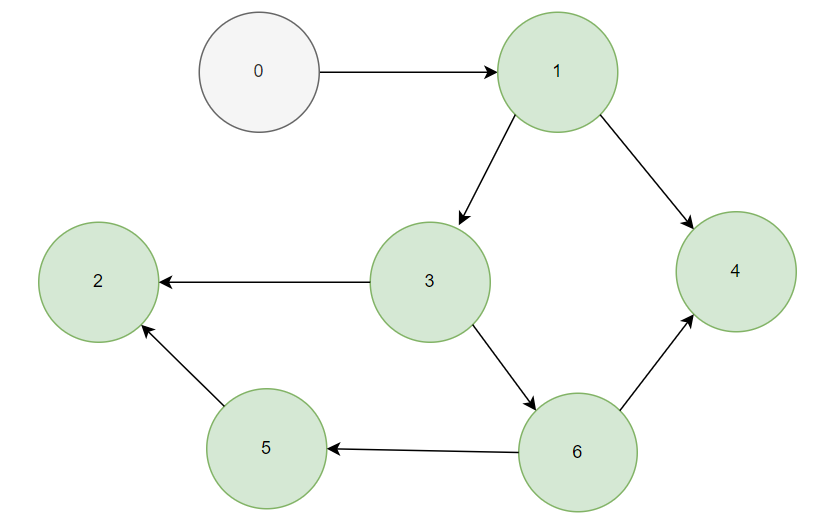
**Стек: [0,1,3]**

**Посещенные вершины: [2,4,5,6]**

****

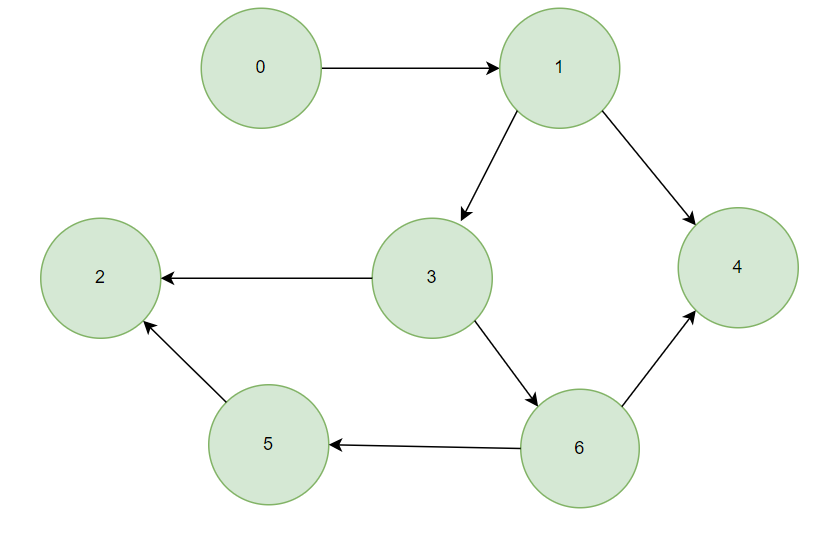
**Стек: [0,1]**

**Посещенные вершины: [2,4,5,6,3]**

****

**Стек: [0]**

**Посещенные вершины: [2,4,5,6,3,1]**

****

**Стек: []**

**Посещенные вершины: [2,4,5,6,3,1,0]**

**Алгоритм топологической сортировки.**

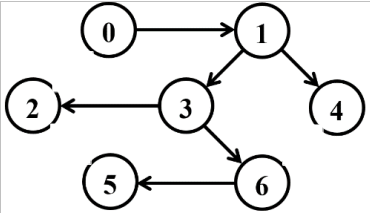


Топологическая сортировка (Topological sort) — один из основных алгоритмов на графах, который применяется для решения множества более сложных задач.

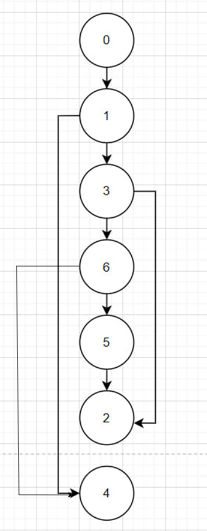
Задача топологической сортировки графа состоит в следующем: указать такой линейный порядок на его вершинах, чтобы любое ребро вело от вершины с меньшим номером к вершине с большим номером. Очевидно, что если в графе есть циклы, то такого порядка не существует.

Топологическая сортировка сводится к обходу графа в глубину, так что

Отсортированный граф будет выглядеть так:



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



***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 на рисунках 1 и 2. Содержимое файла BFS.cpp представлено в листинге 1.1.

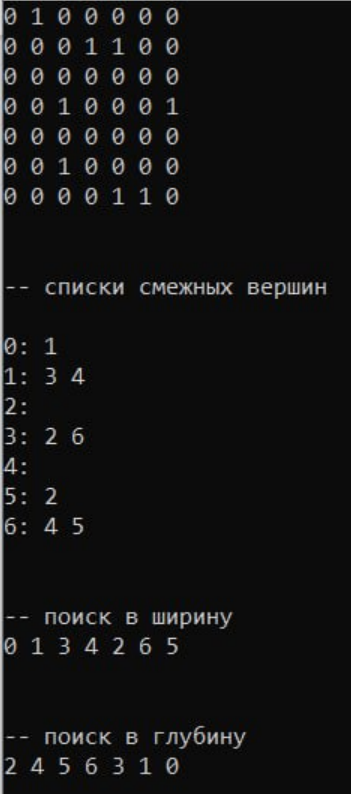
|  |
| --- |
| #include "BFS.h"  void BFS::init(const graph::AList& al, int s)  {  this->al = &al;  this->c = new Color[this->al->n\_vertex];  this->d = new int[this->al->n\_vertex];  this->p = new int[this->al->n\_vertex];  for (int i = 0; i < this->al->n\_vertex; i++)  {  this->c[i] = WHITE;  this->d[i] = 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;} |

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

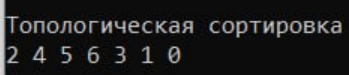
Содержимое файла DFS.cpp представлено в листинге 1.2.

|  |
| --- |
| #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 1.2 – Содержимое DFS.cpp



***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. Начинаем с вершины 0.

8

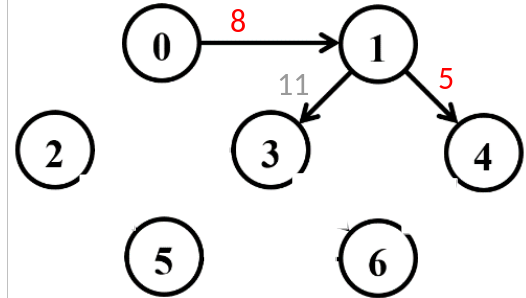
8

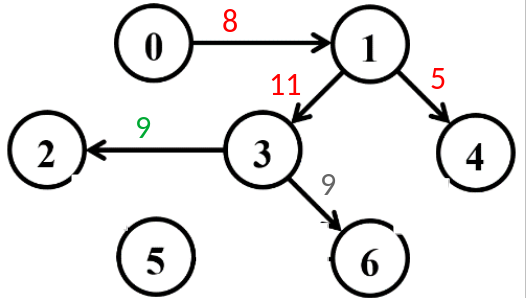
5

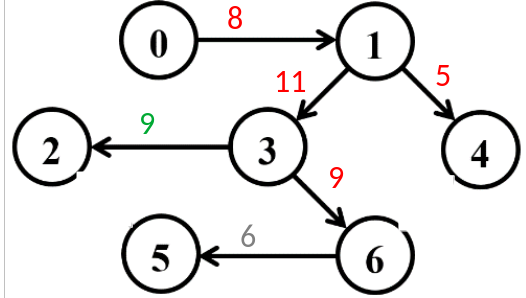
11

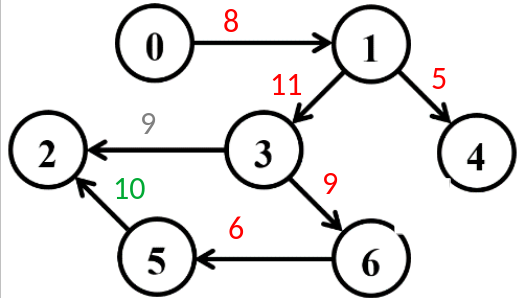


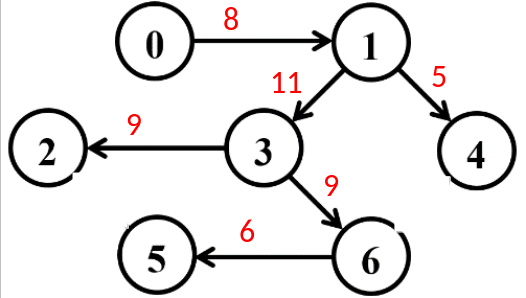












Вес будет равен 48

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



W(e0,1)=8;

W(e1,4)=5;

W(e1,3)=11;

W(e3,2)=9;

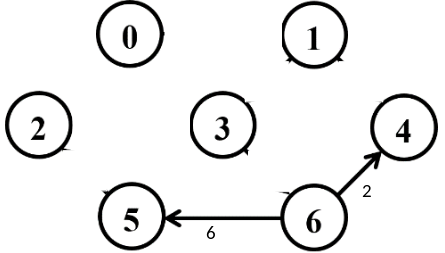
W(e3,6)=9;

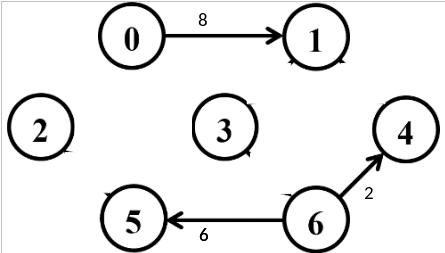
W(e6,5)=6;

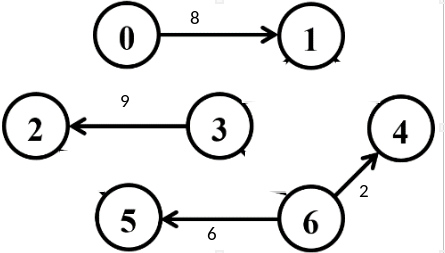
W(e5,2)=10;

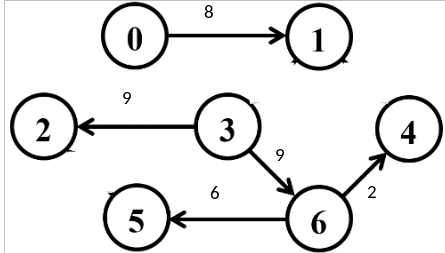
W(e6,4)=2;

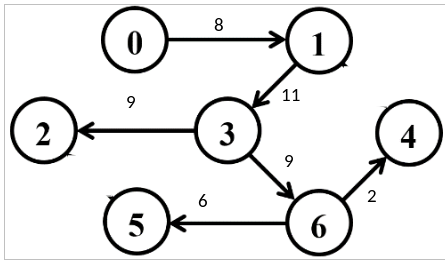












Вес будет равен 48

**Вывод**: в ходе лабораторной работы была освоена программная реализация: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов и разобраны алгоритм Прима и алгоритм Крускала