**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёт по лабораторной работе №6

По дисциплине «Математическое программирование»

На тему «Оптимизационные задачи на графах»

Выполнил:

Студент 2 курса 8 группы

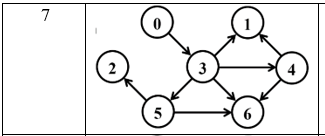
Федорович Вадим Геннадьевич

Преподаватель: асс. Ромыш А.С.

2025, Минск

**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.

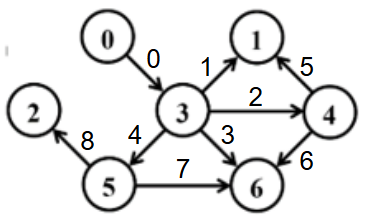
***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.



**Матрица смежности:**

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

Для того, чтобы построить матрицу инцидентности, нам нужно определить количество ребер в нашем ориентированном графе. В нашем графе их количество равно 9. Теперь отметим эти ребра на нашем рисунке:



**Матрица инцидентности:**

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

**Список смежных вершин:**  
S0 = {3}

S1 = ∅

S2 = ∅

S3 = {1, 4, 5, 6}

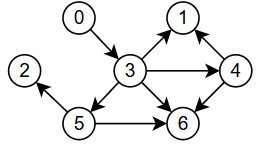
S4 = {1, 6}

S5 = {2, 6}

S6 = ∅

***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.

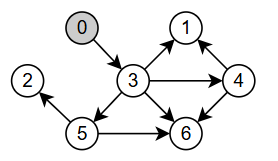
**Поиск в ширину**



Шаг 1

Очередь: 0

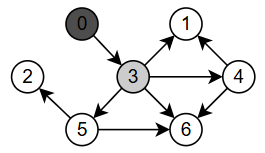
Последовательность: ∅



Шаг 2

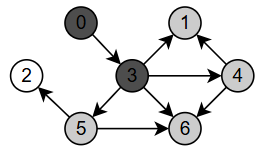
Очередь: 3

Последовательность: 0



Шаг 3

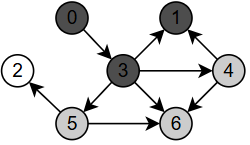
Очередь: 1, 4, 5, 6  
Последовательность: 0, 3



Шаг 4

Очередь: 4, 5, 6

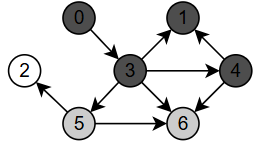
Последовательность: 0, 3, 1



Шаг 5

Очередь: 5, 6

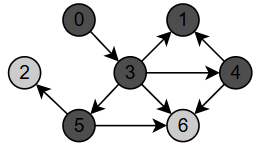
Последовательность: 0, 3, 1, 4



Шаг 6

Очередь: 6, 2

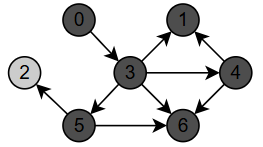
Последовательность: 0, 3, 1, 4, 5



Шаг 7

Очередь: 2

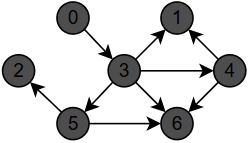
Последовательность: 0, 3, 1, 4, 5, 6



Шаг 8

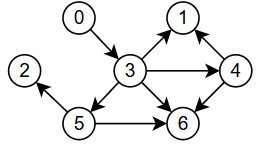
Очередь: ∅

Последовательность: 0, 3, 1, 4, 5, 6, 2



Обход в ширину: 0, 3, 1, 4, 5, 6, 2

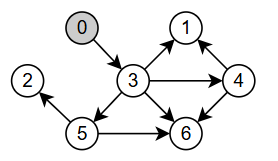
**Поиск в глубину**



Шаг 1

Стек: 0

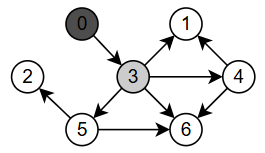
Последовательность: ∅



Шаг 2

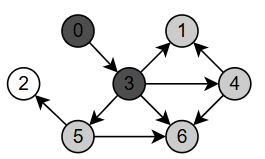
Стек: 3

Последовательность: 0



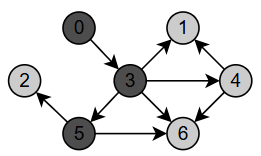
Шаг 3

Стек: 5,4,6,1  
Последовательность: 0, 3



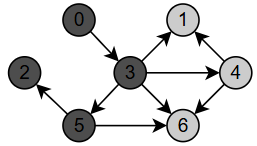
Шаг 4

Стек: 2,4,6,1  
Последовательность: 0, 3, 5



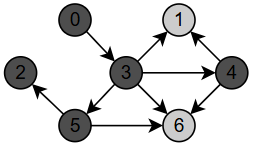
Шаг 5

Стек: 4,6,1  
Последовательность: 0, 3, 5, 2



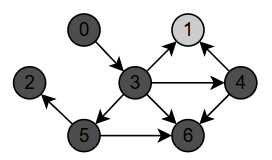
Шаг 6

Стек: 6,1  
Последовательность: 0, 3, 5, 2, 4



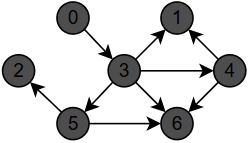
Шаг 7

Стек: 1  
Последовательность: 0, 3, 5, 2, 4, 6



Шаг 8

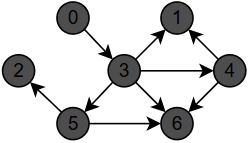
Стек: ∅  
Последовательность: 0, 3, 5, 2, 4, 6, 1



Обход в глубину: 0, 3, 5, 2, 4, 6, 1

**Топологическая сортировка**

Топологическая сортировка — это упорядочивание вершин ориентированного графа, при котором для каждого ребра (u→v) вершина u идет перед вершиной vв этом упорядочивании. Она возможна только для ациклических ориентированных графов (DAG). В нашем случае результат топологической сортировки будет совпадать с результатом DFS, так как топологическая сортировка использует тот же механизм, что и DFS: вершины добавляются в стек в порядке их завершения. Так как в графе нет циклов, и DFS обходит все вершины в порядке, который уже соответствует топологическому упорядочиванию, то результат топологической сортировки совпадает с порядком завершения вершин в DFS.



Результат топологической сортировки: 0, 3, 5, 2, 4, 6, 1

***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.

Данные 3 задания будут размещены в одном программном файле и результат данной программы будет прикреплен ниже снимком экрана:

|  |
| --- |
| #include <iostream>  #include <stack>  #include <queue>  #include <vector>  #include <algorithm>  using namespace std;  const int V = 7;  int matrix[V][V] =  {  {0, 0, 0, 1, 0, 0, 0},  {0, 0, 0, 0, 0, 0, 0},  {0, 0, 0, 0, 0, 0, 0},  {0, 1, 0, 0, 1, 1, 1},  {0, 1, 0, 0, 0, 0, 1},  {0, 0, 1, 0, 0, 0, 1},  {0, 0, 0, 0, 0, 0, 0}  };  void BFS\_by\_matrix(int start) {  queue<int> q;  bool visited[V] = { false };  q.push(start);  visited[start] = true;  while (!q.empty()) {  int current = q.front();  q.pop();  cout << current << " ";  for (int i = 0; i < V; i++) {  if (matrix[current][i] == 1 && !visited[i]) {  q.push(i);  visited[i] = true;  }  }  }  }  void DFS\_by\_matrix(int start, vector<bool>& visited, stack<int>& stack) {  visited[start] = true;  for (int i = 0; i < V; i++) {  if (matrix[start][i] == 1 && !visited[i]) {  DFS\_by\_matrix(i, visited, stack);  }  }  stack.push(start);  }  void topologicalSort() {  stack<int> stack;  vector<bool> visited(V, false);  for (int i = 0; i < V; i++) {  if (!visited[i]) {  DFS\_by\_matrix(i, visited, stack);  }  }  cout << "Топологическая сортировка: ";  while (!stack.empty()) {  cout << stack.top() << " ";  stack.pop();  }  cout << endl;  }  void printAdjacencyList() {  cout << "Список смежных вершин:" << endl;  for (int i = 0; i < V; i++) {  cout << i << ": ";  for (int j = 0; j < V; j++) {  if (matrix[i][j] == 1) {  cout << j << " ";  }  }  cout << endl;  }  }  void printEdgeList() {  cout << "Список ребер:" << endl;  for (int i = 0; i < V; i++) {  for (int j = 0; j < V; j++) {  if (matrix[i][j] == 1) {  cout << "(" << i << " -> " << j << ") ";  }  }  }  cout << endl;  }  int main() {  setlocale(LC\_ALL, "RU");  int start = 0;  cout << "Обход в ширину (BFS): ";  BFS\_by\_matrix(start);  cout << endl;  cout << "Обход в глубину (DFS): ";  vector<bool> visited(V, false);  stack<int> stack;  DFS\_by\_matrix(start, visited, stack);  while (!stack.empty()) {  cout << stack.top() << " ";  stack.pop();  }  cout << endl;  topologicalSort();  printAdjacencyList();  printEdgeList();  return 0;  } |

Листинг 1 – Main.cpp

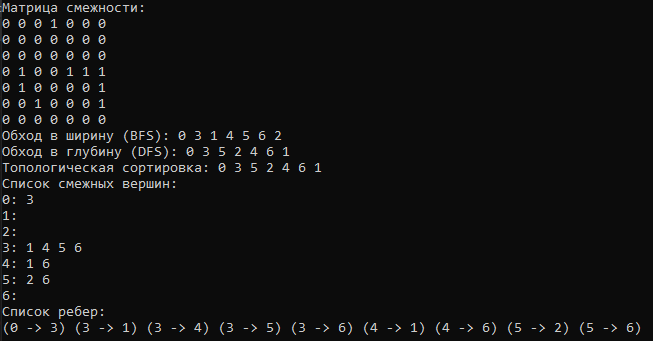


Рисунок 1 – Результат программы

***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;

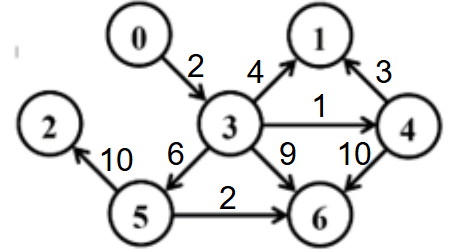
W(e4,3)=4; W(e3,4)=1;

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;

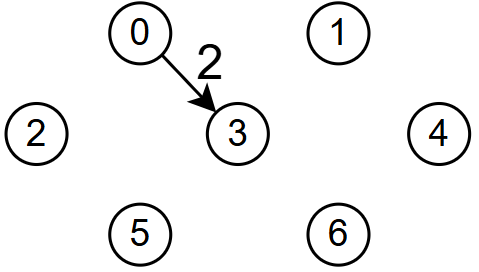


Шаг 1:

Предположим, что в множество U пока только входит вершина 0, нужно найти ребро с минимальной стоимостью, которое соединяет вершину 0 с вершиной, которая в множество U не входит.

U = {0}

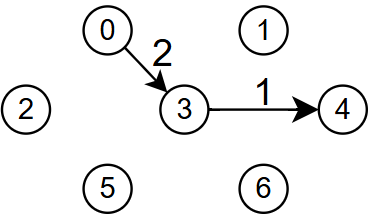
Вершина 0 соединена только с вершиной 3. Вес данной вершины равен 2, включаем данную вершину в множество U.



Шаг 2:

U = {0, 3}

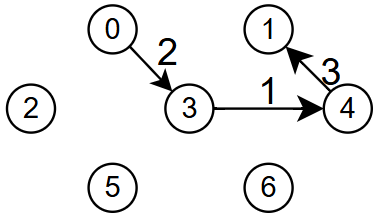
Вершина 3 соединена с вершинами 1, 4, 5, 6. Веса этих рёбер равны 4, 1, 9, 6. Из всех этих рёбер минимальной стоимостью обладает ребро 3-4 (вес = 1). Включаем вершину 4 в множество U.



Шаг 3:

U = {0, 3, 4}

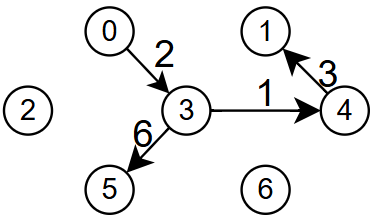
Вершины 3 соединена с вершинами 1, 5, 6, а вершина 4 с вершинами 1 и 6. Среди данных рёбер наименьшую стоимость имеет ребро 4-1 (вес = 3). Вершину 1 включаем в множество U.



Шаг 4:

U = {0, 3, 4, 1}

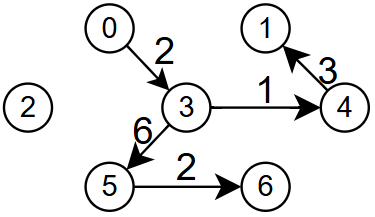
Вершина 3 соединена с вершинами 1, 5, 6, но так как мы вершину 1 уже посетили, ребро 3-1 мы не рассматриваем. Вершина 4 соединена с вершиной 6. Среди данных рёбер наименьшую стоимость имеет ребро 3-5 (вес = 6). Включаем вершину 5 в множество U.



Шаг 5:

U = {0, 3, 4, 1, 5}

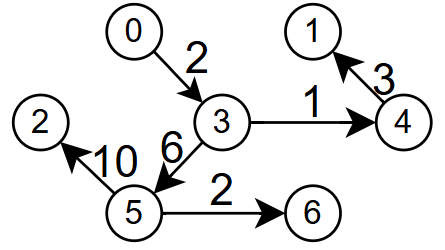
Вершина 3 соединена с вершиной 6, а вершина 5 с вершиной 2 и 6. Среди данных рёбер наименьшую стоимость имеет ребро 5-6 (вес = 2). Добавляем вершину 6 в множество U.



Шаг 6:

U = {0, 3, 4, 1, 5, 6}

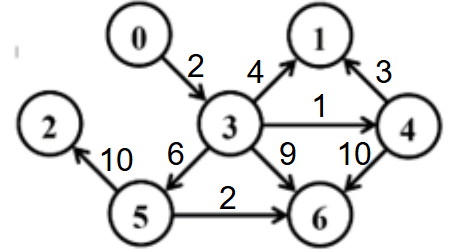
Оставшаяся вершина 2 соединена с вершиной 5, вес данного ребра равен 10. Так что просто добавляем вершину 2 в множество U.



U = {0, 3, 4, 1, 5, 6, 2}

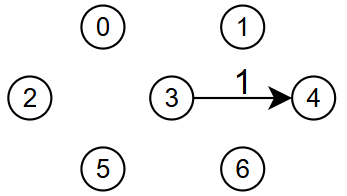
Вес минимального остовного дерева: 24

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

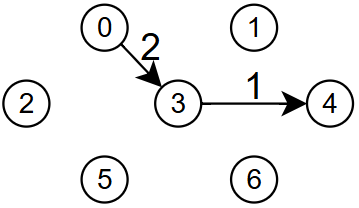


Шаг 1:

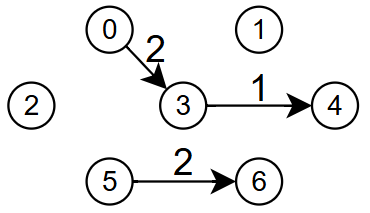
Из всех ребер выбираем ребро с минимальной стоимостью (весом) и включаем это ребро в остовное дерево. Повторяем так далее, однако смотрим на то, чтобы следующее ребро соединяло вершины из разных компонент.



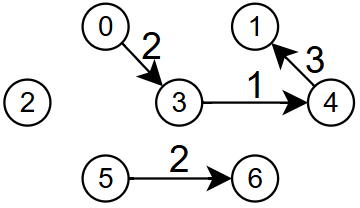
Шаг 2:



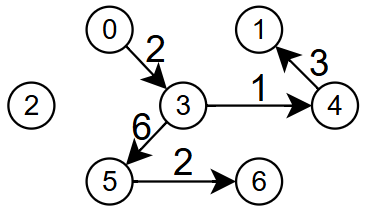
Шаг 3:



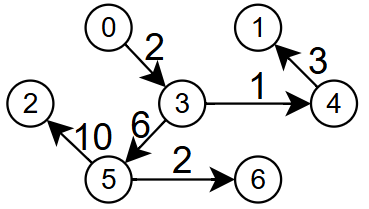
Шаг 4:



Шаг 5:



Шаг 6:



Вес минимального остовного дерева: 24