Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №5**

**Тема АЛГОРИТМЫ НА ГРАФАХ**

**(алгоритмы поиска в ширину и глубину, топологическая сортировка)**

Выполнила:

Студентка 2 курса 2 группы ФИТ

Максимова Вера Владимировна

Минск 2022

**Цель:** освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов.

**Задание 1.** Ориентированный граф G взять в соответствии с вариантом. 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.

Вариант 3.



1. **Алгоритм поиска в ширину (BFS).**

**Ход работы:**

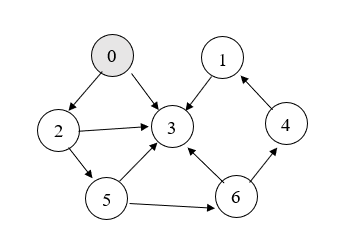
Исходный граф: (0,2), (0,3), (1,3), (2,3), (2,5), (4,1), (5,3), (5,6), (6,4)

1. Алгоритм поиска в ширину (BFS)

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

Шаг 1.

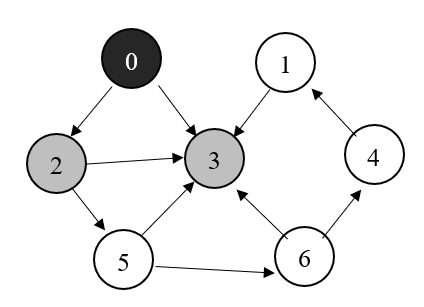
В качестве стартовой вершины выбираем вершину с номером 0.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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.

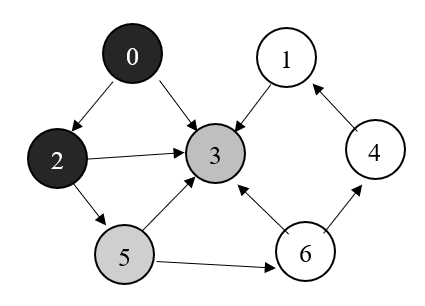
У вершины 0 две смежные вершины. Для последующего пути выбираем вершину с наименьшим весом – 2, а 3-ю добавляем в начало очереди. Закрашиваем 0 в черный цвет как пройденную. Смежные в серый.



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

Шаг 3.

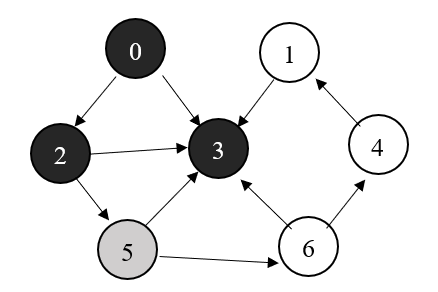
Переходим в смежную вершину, вершину номер 2. С ней смежные вершины номер 3 и 5. Закрашиваем 3 и 5 в серый цвет, а вершину номер 2 в черный, как пройденную.



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

Шаг 4.

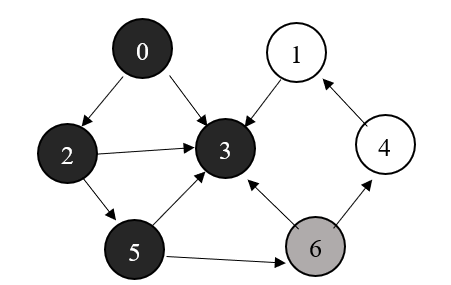
Переходим в смежную вершину, вершину номер 3. Вершина номер 3 не имеет смежных вершин поэтому просто окрашиваем ее в черный, как пройденную.



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

Шаг 5.

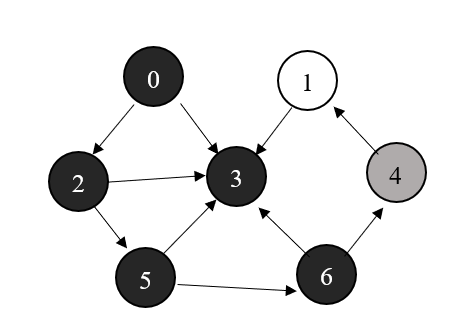
Переходим в смежную вершину, вершину номер номер 5. Вершина номер 5 имеет одну смежную вершину, номер 6, закраиваем ее в серый цвет, а вершину 5 в черный.



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

Шаг 6.

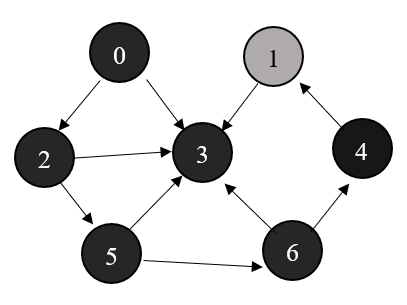
Переходим смежную вершину 6, закрашиваем ее черным цветом, как пройденную, а смежную вершину 4 в серый как смежную



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 4 |  |  |  |  |  |  |
| C | B | W | B | B | G | B | B |
| D | 0 | I | 1 | 1 | 4 | 2 | 3 |
| P | N | N | 0 | 0 | 5 | 3 | 4 |

Шаг 7.

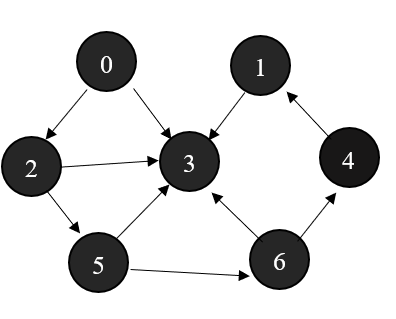
Закрашиваем вершину номер 4 в черный, как пройденную, а вершину номер 1 в серый как смежную.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 1 |  |  |  |  |  |  |
| C | B | G | B | B | B | B | B |
| D | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| P | 0 | 6 | 0 | 0 | 5 | 3 | 4 |

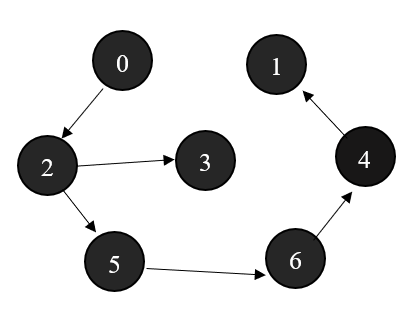
Шаг 8.

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



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q |  |  |  |  |  |  |  |
| C | B | B | B | B | B | B | B |
| D | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| P | N | 6 | 0 | 0 | 5 | 3 | 4 |

В результате получили BFS-дерево:



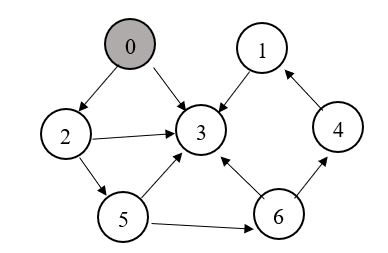
Порядок обхода:

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

1. **Алгоритм поиска в глубину (DFS).**

Шаг 1.

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

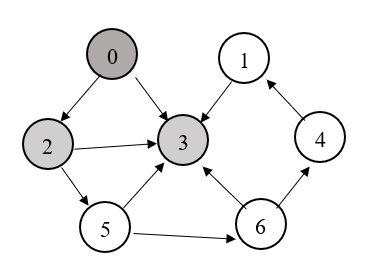


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W | W | W |
| D | 1 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 2.

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

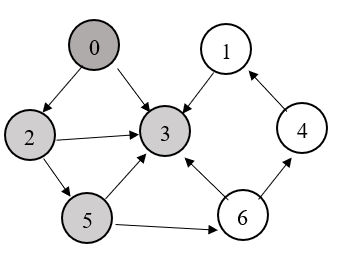
И выбираем т что с наименьшим весом.



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

Шаг 3.

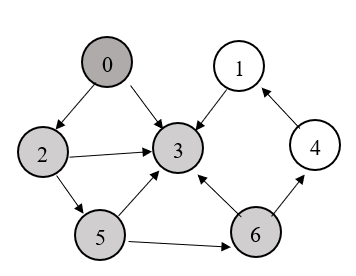
Вершина 2 имеет одну не отмеченную смежную вершину номер 5, закрашиваем ее в серый цвет.



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

Шаг 4.

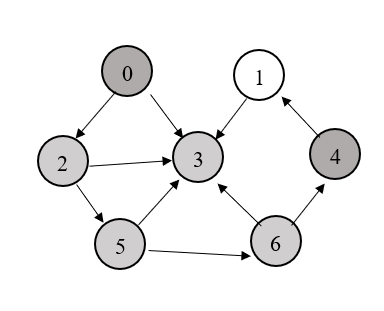
Вершина номер 5 имеет, одну смежную неокрашенную вершину, это 6. Закрашиваем ее в серый цвет.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | G | G |
| D | 1 | I | 2 | 2 | I | 3 | 4 |
| P | N | N | 0 | 0 | N | 3 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 5.

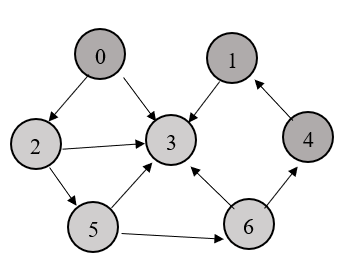
Вершина номер 6 имеет одну смежную не закрашенную вершину, это номер 4, закрашиваем ее, в серый цвет.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | G | G | G |
| D | 1 | I | 2 | 2 | 5 | 3 | 4 |
| P | N | N | 0 | 0 | 5 | 3 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

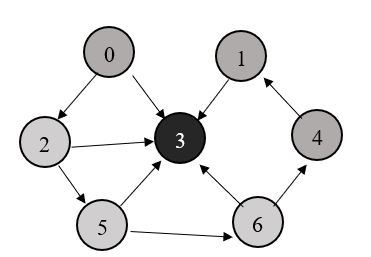
Шаг 6.

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



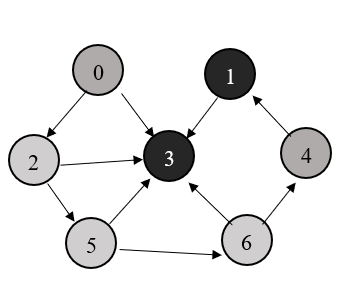
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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.



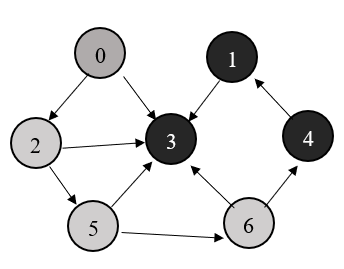
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | B | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 0 | 0 | 7 | 0 | 0 | 0 |

Шаг 8.



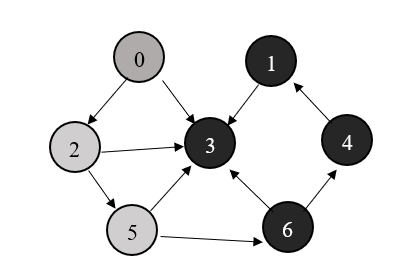
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 0 | 0 | 0 |

Шаг 9.



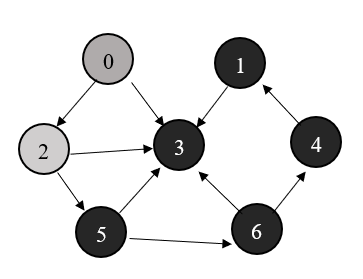
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 0 | 0 |

Шаг 10.



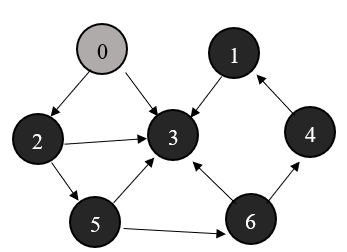
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | G | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 0 | 10 |

Шаг 11.



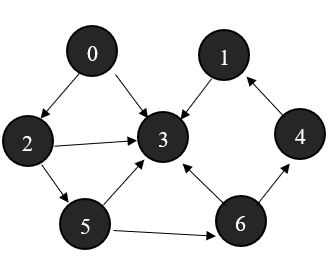
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 11 | 10 |

Шаг 12.



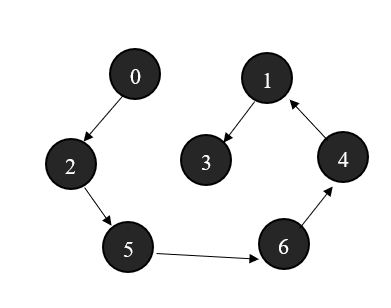
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 12 | 7 | 9 | 11 | 10 |

Шаг 13.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 13 | 8 | 12 | 7 | 9 | 11 | 10 |

В результате получили DFS-дерево:



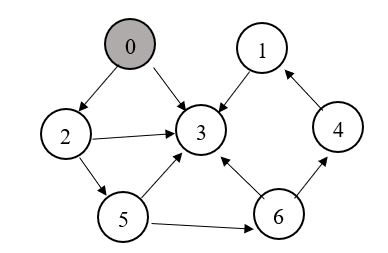
Порядок обхода: 3,1,4,6,5,2,0.

**3. Алгоритм топологической сортировки.**

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

Шаг 1.

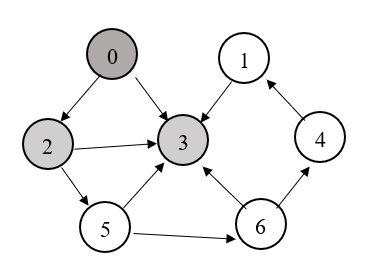
В качестве стартовой вершины выбираем вершину с номером 0. Окрашиваем ее в серый цвет.



Шаг 2.

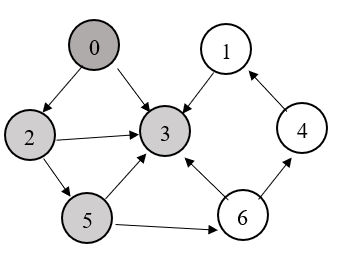
Вершина 0 имеет 2 смежные вершины 2 и 3. Закрашиваем их серым цветом.

И выбираем т что с наименьшим весом.



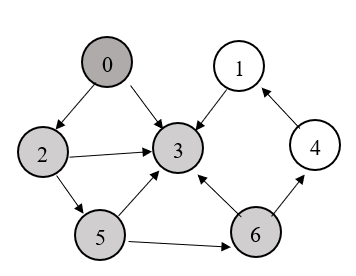
Шаг 3.

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



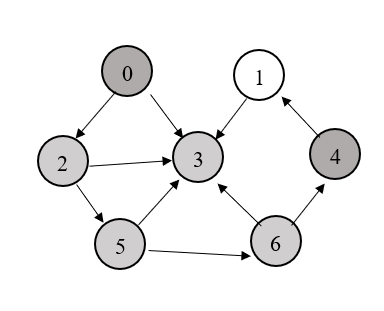
Шаг 4.

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



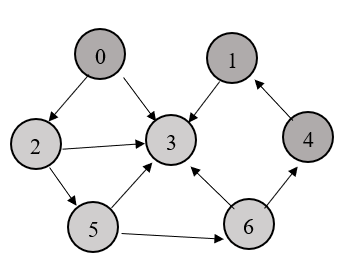
Шаг 5.

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

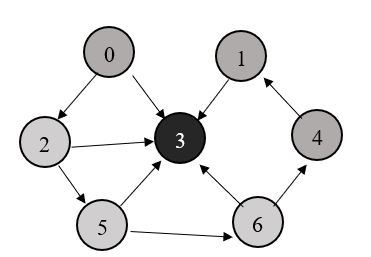


Шаг 6.

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

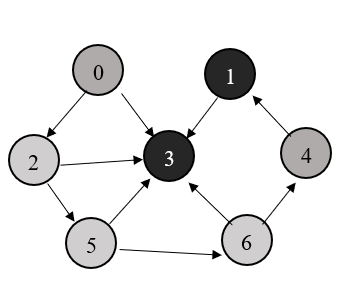


Шаг 7. Вершина 1 имеет смежную вершину номер 3, но она уже закрашена в серый цвет и не имеет смежную вершин следовательно закрашиваем вершину номер 3 в черный цвет и таким образом двигаемся по обратному пути. Добавляем вершину 3 в стек.



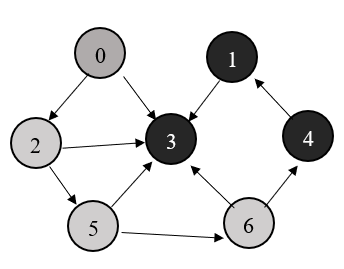
Шаг 8.

Добавляем вершину 1 в стек.



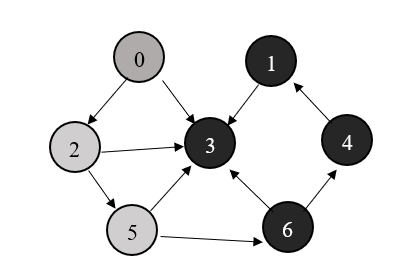
Шаг 9.

Добавляем вершину 4 в стек.



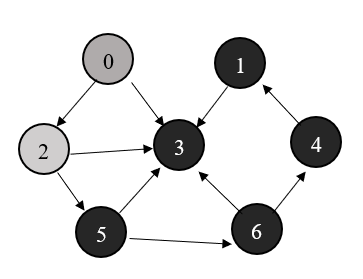
Шаг 10.

Добавляем вершину 6 в стек.



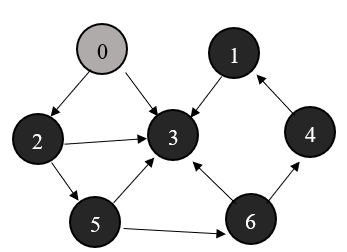
Шаг 11.

Добавляем вершину 5 в стек.



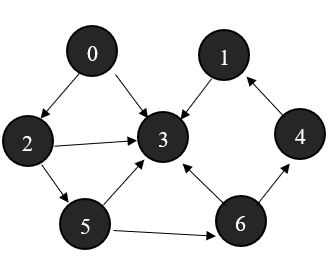
Шаг 12.

Добавляем вершину 2 в стек.



Шаг 13.

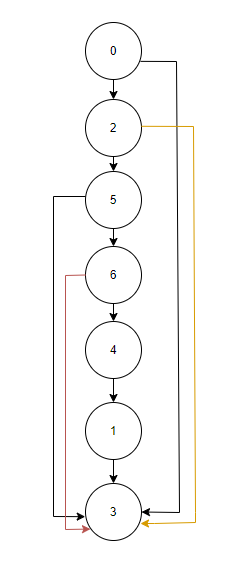
Добавляем вершину 0 в стек.



Таким образом имеем сортировку:

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

Схема:



**Задание 2.** 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.

Код:

// --- Graph.h

//

#pragma once

#include <list>

#include "Graph.h"

namespace graph

{

struct AList;

struct AMatrix // матрица смежности

{

int n\_vertex; // количество вершин

int \*mr; // матрица

AMatrix(int n); // создать нулевую матрицу n\*n

AMatrix(int n, int mr[]); // создать матрицу n\*n и

AMatrix(const AMatrix& am); // создать подобную матрицу

AMatrix(const AList& al); // создать матрицу из спискового

void set(int i, int j, int r); // записать mr[i,j] = r

int get(int i, int j)const; // элемент mr[i,j]

};

struct AList // списки смежности

{

int n\_vertex; // количество вершин

std::list<int> \*mr; // массив списков

void create(int n); // создать массив пустых списков

AList(int n); // создать массив пустых списков

AList(int n, int mr[]); // создать списковое представление

AList(const AMatrix& am); // создать списковое представление

AList(const AList& al); // создать подобную структуру

void add(int i, int j); // добавить в i-ый список

int size(int i) const; // размер i-го списка

int get(int i, int j)const; // j-ый элемент i-го списка

};

};

// --- Graph.cpp

//

#include "stdafx.h"

#include "Graph.h"

namespace graph

{

AMatrix::AMatrix(int n)

{

this->n\_vertex = n;

this->mr = new int [this->n\_vertex\*this->n\_vertex];

for(int i = 0; i < n\*n; i++)mr[i] = 0;

};

AMatrix::AMatrix(int n, int mr[])

{

this->n\_vertex = n;

this->mr = mr;

};

AMatrix::AMatrix(const AMatrix& am)

{

this->n\_vertex = am.n\_vertex;

this->mr = new int [this->n\_vertex\*this->n\_vertex];

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

this->set(i,j, am.get(i,j));

};

AMatrix::AMatrix(const AList& al)

{

this->n\_vertex = al.n\_vertex;

this->mr = new int [this->n\_vertex \* this->n\_vertex];

for(int k = 0; k < this->n\_vertex\*this->n\_vertex; k++)mr[k] = 0;

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->set(i,al.get(i,j),1);

};

void AMatrix::set(int i, int j, int r) {this->mr[i\*this->n\_vertex+j] = r;};

int AMatrix::get(int i, int j)const

{return this->mr[i\*this->n\_vertex+j];};

void AList::create(int n)

{this->mr = new std::list<int>[this->n\_vertex = n];};

AList::AList(int n) {create(n);}

AList::AList(const AMatrix& am)

{

this->create(am.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (am.get(i,j)!=0) this->add(i,j);

};

AList::AList(const AList& al)

{

this->create(al.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->add(i, al.get(i,j));

};

AList::AList(int n, int mr[])

{

this->create(n);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (mr[i\*this->n\_vertex+j]!= 0) this->add(i,j);

};

void AList::add(int i, int j){this->mr[i].push\_back(j);};

int AList::size(int i) const {return (int)this->mr[i].size();};

int AList::get(int i, int j)const

{

std::list<int>::iterator rc = this->mr[i].begin();

for (int k = 0; k < j; k++) rc++;

return (int)\*rc;

};

};

// ---BFS.h

//

#pragma once

#include "Graph.h"

#include <queue>

struct BFS // breadth-first search поиск в ширину (связный граф)

{

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

};

// ---BFS.cpp

//

#include "stdafx.h"

#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();

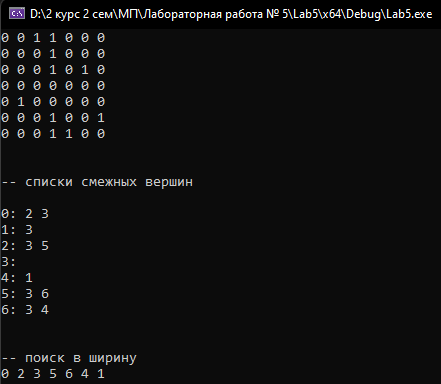
this->c[rc] = BLACK;

};

return rc;

}

Результат:



**Задание 3.** Разработать функцию DFS обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// ---DFS.h

//

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

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

};

// ---DFS.cpp

//

#include "stdafx.h"

#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++) // iая статистика

{

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;

};

// Graphs\_Optimization.cpp: определяет точку входа для консольного приложения.

// --- main

// алгоритмы BFS и DFS

#include "stdafx.h"

#include <iostream>

#include "Graph.h"

#include "BFS.h"

#include "DFS.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

int m[7][7] ={

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

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

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

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

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

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

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

};

setlocale(LC\_ALL, "rus");

graph::AMatrix g1(7,(int\*) m);

std::cout<<std::endl;

std::cout<<std::endl<< "-- матрица смежности "<< std::endl;

for (int i = 0; i < g1.nV; i++)

{

std::cout<<std::endl;

for (int j = 0; j < g1.nV; j++)

std::cout<< g1.get(i,j) << " ";

};

std::cout<<std::endl;

graph::AList g2(g1);

std::cout<<std::endl;

std::cout<<std::endl<< "-- списки смежных вершин "<< std::endl;

for (int i = 0; i < g1.nV; i++)

{

std::cout<<std::endl<< i << ": ";

for (int j = 0; j < g2.size(i); j++)

std::cout<<g2.get(i,j)<< " ";

}

std::cout<<std::endl;

BFS b1(g2,0);

std::cout<<std::endl;

std::cout<<std::endl<< "-- поиск в ширину "<< std::endl;

int k1;

while ((k1 = b1.get())!= BFS::NIL)

std::cout<< k1 << " ";

std::cout<<std::endl;

DFS b2(g2);

std::cout<<std::endl;

std::cout<<std::endl<< "-- поиск в глубину "<< std::endl;

for (int i = 0; i < g2.nV; i++)

std::cout<< b2.get(i) << " ";

std::cout<<std::endl;

std::cout << std::endl << "Топологическая сортировка" << std::endl;

for (std::vector <int>::iterator i (b2.topological\_sort.begin ()); i != b2.topological\_sort.end (); ++i)

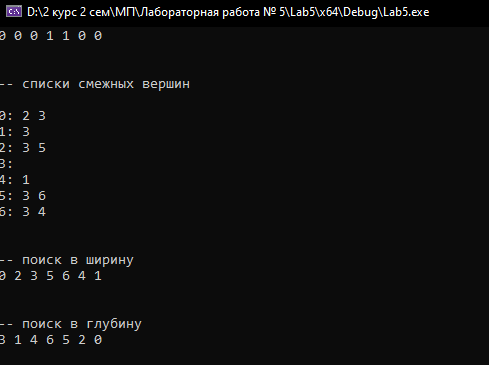
std::cout << \*i << ' ';

std::cout << std::endl;

system("pause");

return 0;

}



**Задание 4.** Доработайте функцию DFS, для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Код:

сpp

#include "stdafx.h"

#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->nV];

this->d = new int[this->al->nV];

this->f = new int[this->al->nV];

this->p = new int[this->al->nV];

this->t = 0;

for (int i = 0; i < this->al->nV; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->nV; 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->nV; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{

min1 = this->f[k];

ntx = k;

};

min2 = min1; min1 = INF;

};

return ntx;

};

dfs.h

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

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

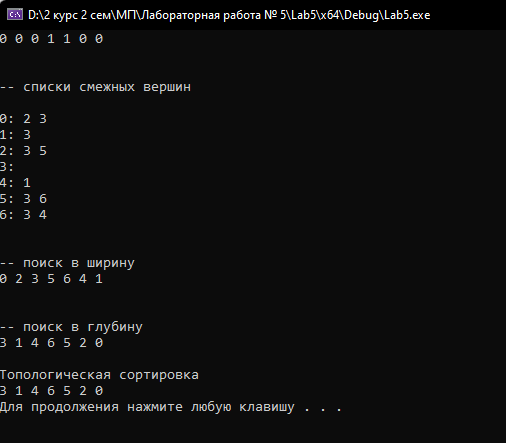
void visit(int v);

void init(const graph::AList& al);

int get(int i); // получить вершину

};

Результат:



**Вывод.**

В результате выполненной 5 лабораторной работы, была освоена сущность и программная реализация способов представления графов, алгоритмов поиска в ширину и глубину, алгоритма топологической сортировки графа.