**Федеральное Агентство по Образованию**

Государственное образовательное учреждение высшего профессионального образования

**Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И.Ульянова (Ленина)**

**(СПБГЭТУ)**

Кафедра МОЭВМ

Отчет по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Выполнил: Черницын Н.А.

Факультет: КТИ

Группа : 1381

Проверил: Казаков Б.Б.

#### Санкт-Петербург

# 2013

**Формулировка задания:** Топологическая сортировка (ациклического орграфа) на основе поиска в глубину. Предусмотреть проверку входного графа на отсутствие циклов.

Визуализация алгоритма - важен не размер задачи и время счета, а степень понятности и комфортности работы с программой.

**Теоретическая часть:**

Топологическая сортировка— 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.

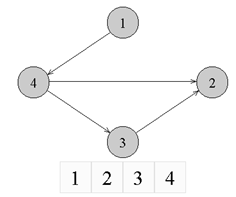


Рисунок 1 – ориентированный граф

Из рисунка 1 видно, что граф не отсортирован, так как ребро из вершины с номером 4 ведет в вершину с меньшим номером (2).

Существует несколько способов топологической сортировки — из наиболее известных:

* Алгоритм Демукрона
* Метод сортировки для представления графа в виде нескольких уровней
* Метод топологической сортировки с помощью обхода в глубину

Будем рассматривать последний метод.

*Поиск в глубину или обход в глубину* (англ. Depth-first search, сокращенно DFS) — один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них.

**Алгоритм работы программы:**

Граф представляется в виде списка смежностей. Список смежности содержит для каждой вершины графа v список смежных ей вершин.

*Основные шаги топологической сортировки с помощью обхода в глубину:*

1. Вызывается обход в глубину от всех вершин. Заканчиваем работу алгоритма, если обнаружен цикл.
2. Создаем правило замены номеров вершин.
3. Если вершина серая, то мы обнаружили цикл. Заканчиваем поиск в глубину.
4. Если вершина черная, то заканчиваем ее обработку.
5. Красим вершину в серый цвет.
6. Обрабатываем список смежных с ней вершин.
7. Кладем вершину в стек, при помощи стека, в дальнейшем, создадим правило замены номеров вершин.
8. Красим вершину в черный цвет

**Листинг**

* *Main.cpp*

#include "graph.h"

int main()

{

Graph\* graph=new Graph;

delete graph;

return 0;

}

* *Graph.cpp*

#include "graph.h"

Graph::Graph()

{

if (!init())

return;

for (unsigned int i=0; i<N; i++)

{

Numbers.push\_back(0);

Color.push\_back(0);

}

std::cout<<"Исходный граф:"<<std::endl;

print();

if (!topological\_sort())

return;

std::cout<<"При анализе стека можно создать правило замены вершин:"<<std::endl;

for (unsigned int i=0; i<N; i++)

std::cout<<i+1<<"="<<Numbers[i]<<std::endl;

std::cout<<std::endl;

for (unsigned int i=0; i<N; i++)

for (unsigned int j=0; j<Edges[i]->size(); j++)

(\*Edges[i])[j]=Numbers[(\*Edges[i])[j]-1];

std::vector <unsigned int>\* temp;

for (unsigned int i=0; i<N; i++)

if ((Numbers[i]-1)!=i)

{

temp=Edges[i];

Edges[i]=Edges[Numbers[i]-1];

Edges[Numbers[i]-1]=temp;

Numbers[Numbers[i]-1]=Numbers[i];

Numbers[i]=i+1;

}

std::cout<<"Произведем замену вершин по созданному правилу\nи сбросим окраску вершин (она нам больше не нужна):"<<std::endl;

for (unsigned int i=0; i<N; i++)

Color[i]=0;

print();

std::cout<<"=================================="<<std::endl;

std::cout<<"Результат топологической сортировки:"<<std::endl;

print();

}

bool Graph::init()

{

std::ifstream ifile ("graph.dat");

if (!ifile.is\_open())

{

std::cerr<<"EEROR! File not open!"<<std::endl;

return false;

}

int node;

bool new\_node=true;

std::vector <unsigned int>\* temp;

while (ifile>>node)

{

//std::cout<<node;

if (new\_node)

{

new\_node=false;

N++;

temp=new std::vector <unsigned int>;

Edges.push\_back(temp);

}

if (node)

{

//Edges[N].push\_back(node);

Edges[N-1]->push\_back(node);

}

else

new\_node=true;

}

//N++;

ifile.close();

return true;

}

bool Graph::topological\_sort()

{

for(unsigned int i=0; i<N; i++)

if (dfs(i+1))

return false;

for(unsigned int i=0; i<N; i++)

{

Numbers[i]=Stack.top();

Stack.pop();

}

return true;

}

bool Graph::dfs(unsigned int v)

{

v--;

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

std::cout<<"Текущая вершина:"<<v+1<<std::endl;

if (Color[v]==1)

{

print();

std::cerr<<"ERROR! Обнаружен цикл в графе!"<<std::endl;

return true;

}

if (Color[v]==2)

{

print();

std::cout<<"Эта вершина уже использована. Возврат к предыдушей вершине."<<std::endl;

std::cout<<std::endl;

return false;

}

Color[v]=1;

std::cout<<"Окрасим ее в серый цвет."<<std::endl;

print();

for (unsigned int i=0; i<Edges[v]->size(); i++)

if (dfs((\*Edges[v])[i]))

return true;

std::cout<<"Из вершины "<<v+1<<" необследованных путей нет."<<std::endl;

std::cout<<"Помещаем в стек вершину с номером:"<<v+1<<std::endl;

std::cout<<"Возврат к предыдушей вершине."<<std::endl;

std::cout<<std::endl;

Stack.push(v+1);

Color[v]=2;

return false;

}

void Graph::print()

{

for (unsigned int i=0; i<N; i++)

{

std::cout<<i+1<<"("<<Color[i]<<")"<<"\t|";

for (unsigned int j=0; j<Edges[i]->size(); j++)

std::cout<<"->"<<(\*Edges[i])[j];

std::cout<<"->X"<<std::endl;

}

}

* *Graph.h*

#ifndef GRAPH\_H

#define GRAPH\_H

#include <vector>

#include <stack>

#include <fstream>

#include <iostream>

class Graph

{

public:

Graph();

bool topological\_sort();

bool dfs(unsigned int v);

bool init();

void print();

private:

std::vector <unsigned int> Color; // — массив, в котором хранятся цвета вершин (0 — белый, 1 — серый, 2 — черный).

unsigned int N; // — количество вершин.

std::vector <std::vector <unsigned int>\* > Edges; // — массив списков смежных вершин.

std::vector <unsigned int> Numbers; // — массив, в котором сохраняются новые номера вершин.

std::stack <unsigned int> Stack; // — стек, в котором складываются вершины после их обработки.

};

#endif // GRAPH\_H

**Результат работы программы:**

1. Зацикленный граф

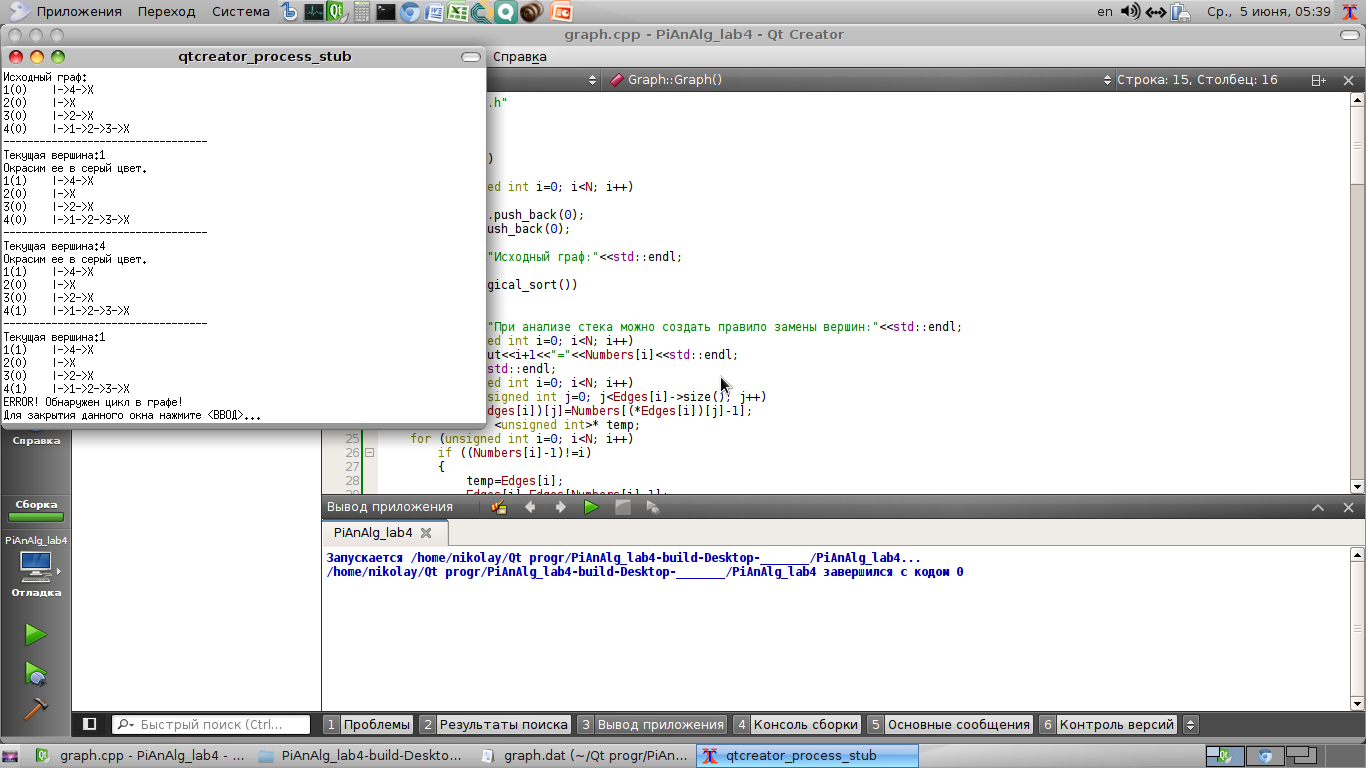


Рисунок 2 – результат работы программы на зацикленной графе

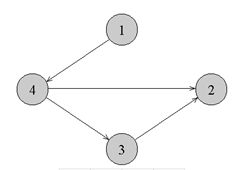


Рисунок 3 – тестируемый граф

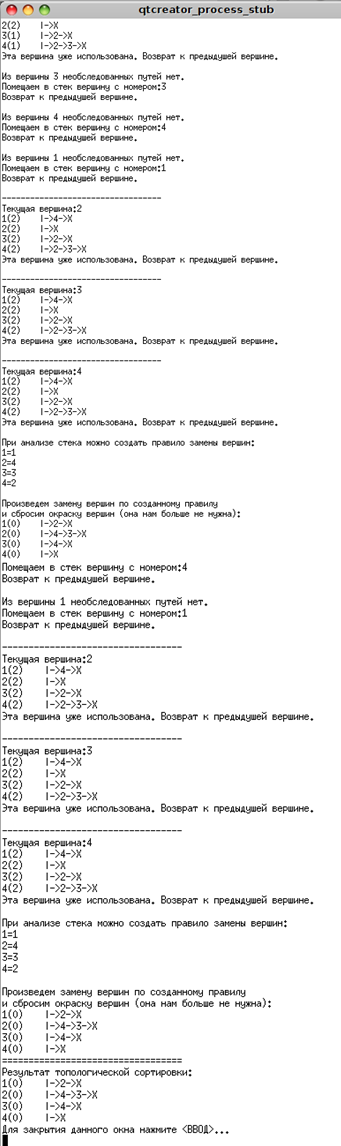


Рисунок 4 – результат работы программы

**Вывод:** в данной лабораторной работе я познакомился с алгоритмом работы топологической сортировки, на практике научился его использовать и отсортировал граф. Научился обрабатывать ошибки, связанные с зацикливанием графа. В качестве алгоритма обхода графа был использован алгоритм обхода в глубину.