Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Лабораторная работа: Графы

Выполнил работу

Студент группы РИС 22-2б

Вахрушева А.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

Реализовать граф и методы: добавление ребра, добавление вершины, удаление ребра, удаление вершины, обходы (в ширину, в глубину), алгоритм Дейкстры, вывод графа с помощью SFML.

**Диаграмма класса**



Рисунок 1 – диаграмма класса Graph

**Код программы**

Libraries.h

#pragma once

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include <set>

#include <map>

#include <cmath>

#include <string>

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

#define LEN 250

#define PI 3.14159

Graph.h

#pragma once

#include "Libraries.h"

class Graph

{

int numVertices;

int size;

vector<vector<int>> matrix;

vector<int> existVertex;

vector<int> visited;

map<int, int> way;

vector<Vector2f> button\_coords;

public:

Graph(int vertices);

void addEdge(int FV, int SV, int weight);

void addVertex();

void delEdge(int FV, int SV);

void delVertex(int vertex);

void BFS(int vertex);

void DFS(int vertex);

int getWeight(int FV, int SV);

int getSize() { return size; }

vector<Vector2f> getButtonCoords() { return button\_coords; }

void DKR(int vertex);

void Salesman(int &sum, vector<vector<int>> temp, String &wayS);

void printGraph();

void visualGraph(RenderWindow& w);

void EventMouseButtonPressed(Vector2f, RenderWindow& w);

};

Graph.cpp

#include "Graph.h"

Graph::Graph(int vertices)

{

numVertices = vertices;

size = vertices;

existVertex.resize(numVertices);

matrix.resize(numVertices);

for (size\_t i = 0; i < numVertices; i++)

matrix[i].resize(numVertices, 100000);

visited.resize(numVertices);

button\_coords.resize(5);

}

void Graph::addEdge(int FV, int SV, int weight)

{

if (FV == SV)

{

cout << "Нельзя проложить путь от вершины к самой себе!\n\n";

return;

}

else if (weight < 1)

{

cout << "Вес ребра должен быть положительным!\n\n";

return;

}

else if (FV >= 0 && FV < numVertices && SV >= 0 && SV < numVertices)

{

matrix[FV][SV] = weight;

matrix[SV][FV] = weight;

return;

}

else

cout << "Вершина(ы) введена(ы) некорректно!\n\n";

}

void Graph::addVertex()

{

++size;

for (size\_t i = 0; i < numVertices; i++)

if (existVertex[i] == -1)

{

existVertex[i] = 0;

return;

}

matrix.resize(++numVertices);

for (size\_t i = 0; i < numVertices; i++)

matrix[i].resize(numVertices, 100000);

existVertex.resize(numVertices);

}

void Graph::delEdge(int FV, int SV)

{

if (FV >= 0 && FV < numVertices && SV >= 0 && SV < numVertices && matrix[FV][SV] != 100000)

matrix[FV][SV] = 100000;

else

cout << "Вершина(ы) введена(ы) некорректно!\n";

}

void Graph::delVertex(int vertex)

{

if (vertex >= 0 && vertex < numVertices && size != 0)

{

for (size\_t i = 0; i < matrix[vertex].size(); ++i)

{

matrix[vertex][i] = 100000;

matrix[i][vertex] = 100000;

}

existVertex[vertex] = -1;

--size;

}

else

cout << "Вершина введена некорректно!\n";

}

void Graph::BFS(int vertex)

{

if (vertex >= 0 && vertex < numVertices)

{

for\_each(visited.begin(), visited.end(), [](int& el) { el = 0; });

visited[vertex] = 1;

queue<int> q;

q.push(vertex);

while (!q.empty())

{

int v = q.front();

q.pop();

cout << "Посещена вершина: " << v << endl;

for (size\_t i = 0; i < numVertices; ++i)

if (!visited[i] && matrix[v][i] != 100000)

{

visited[i] = 1;

q.push(i);

}

}

}

else

cout << "Вершина введена некорректно!\n";

}

void Graph::DFS(int vertex)

{

if (vertex >= 0 && vertex < numVertices)

{

visited[vertex] = 1;

cout << "Посещена вершина: " << vertex << endl;

for (size\_t i = 0; i < numVertices; i++)

{

if (matrix[vertex][i] != 100000 && !visited[i])

DFS(i);

}

}

else

cout << "Вершина введена некорректно!\n";

}

int Graph::getWeight(int FV, int SV)

{

if (FV >= 0 && FV < numVertices && SV >= 0 && SV < numVertices && matrix[FV][SV] != 100000)

return matrix[FV][SV];

return 100000;

}

void Graph::DKR(int vertex)

{

if (vertex >= 0 && vertex < numVertices)

{

set<int> s;

int w = 0, minEl;

vector<int> D(numVertices, 100000);

for (size\_t i = 0; i < D.size(); i++)

{

if (i != vertex)

D[i] = getWeight(vertex, i);

}

s.insert(vertex);

while (s.size() != numVertices)

{

minEl = INT\_MAX;

for (size\_t i = 0; i < D.size(); i++)

{

if (D[i] < minEl && s.find(i) == s.end())

{

minEl = D[i];

w = i;

}

}

s.insert(w);

for (size\_t i = 0; i < D.size() && s.size() != numVertices; i++)

if (s.find(i) == s.end())

D[i] = min(D[i], D[w] + getWeight(w, i));

}

for (size\_t i = 0; i < D.size(); i++)

if (i != vertex && D[i] != 100000)

cout << "Кратчайший путь от " << vertex << "-й вершины до " << i << "-й: " << D[i] << endl;

else if (i != vertex && existVertex[i] != -1)

cout << "Нет пути от " << vertex << "-й вершины до " << i << "-й" << endl;

}

else

cout << "Вершина введена некорректно!\n";

}

void Graph::Salesman(int &sum, vector<vector<int>> temp, String &wayS)

{

//------------------------------------------------

for (size\_t i = 0; i < numVertices; ++i)

{

int min\_el = 100000;

for (size\_t j = 0; j < numVertices; ++j)

if (temp[i][j] < min\_el)

min\_el = temp[i][j];

for (size\_t j = 0; j < numVertices; ++j)

if (temp[i][j] != 100000)

temp[i][j] -= min\_el;

}

//----------------------------------------------------

for (size\_t i = 0; i < numVertices; ++i)

{

int min\_el = 100000;

for (size\_t j = 0; j < numVertices; ++j)

if (temp[j][i] < min\_el)

min\_el = temp[j][i];

for (size\_t j = 0; j < numVertices; ++j)

if (temp[j][i] != 100000)

temp[j][i] -= min\_el;

}

//----------------------------------------------------

set<pair<int, int>> vertex\_number;

for (size\_t i = 0; i < numVertices; ++i)

for (size\_t j = 0; j < numVertices; ++j)

if (temp[i][j] == 0)

vertex\_number.insert(make\_pair(i, j));

//----------------------------------------------------

map<int, pair<int, int>, greater<int>> graduate;

for (auto& i : vertex\_number)

{

pair<int, int> min(100000, 100000);

for (size\_t j = 0; j < numVertices; ++j)

{

if (temp[i.first][j] < min.first && j != i.second)

min.first = temp[i.first][j];

if (temp[j][i.second] < min.second && j != i.first)

min.second = temp[j][i.second];

}

graduate.insert(make\_pair(min.first + min.second, make\_pair(i.first, i.second)));

}

//----------------------------------------------------

for (auto& i : graduate)

{

if (sum != 0)

return;

for (size\_t j = 0; j < numVertices; ++j)

{

temp[i.second.first][j] = 100000;

temp[j][i.second.second] = 100000;

}

temp[i.second.second][i.second.first] = 100000;

way[i.second.first] = i.second.second;

if (way.size() == size)

{

for (auto& i : way)

sum += matrix[i.first][i.second];

cout << "\nДлина пути: " << sum;

return;

}

Salesman(sum, temp, wayS);

}

//----------------------------------------------------

}

void Graph::printGraph()

{

for (int i = 0; i < numVertices; ++i)

if (existVertex[i] != -1)

{

cout << "----- [Смежные вершины с вершиной " << i << "] -----\n\n";

for (size\_t j = 0; j < matrix[i].size(); ++j)

if (matrix[i][j] != 100000)

cout << "Имя Вершины: " << j << "\nВес ребра между вершинами (" << i << ":" << j << "): " << matrix[i][j] << endl;

cout << endl;

}

}

**Результаты работы программы**

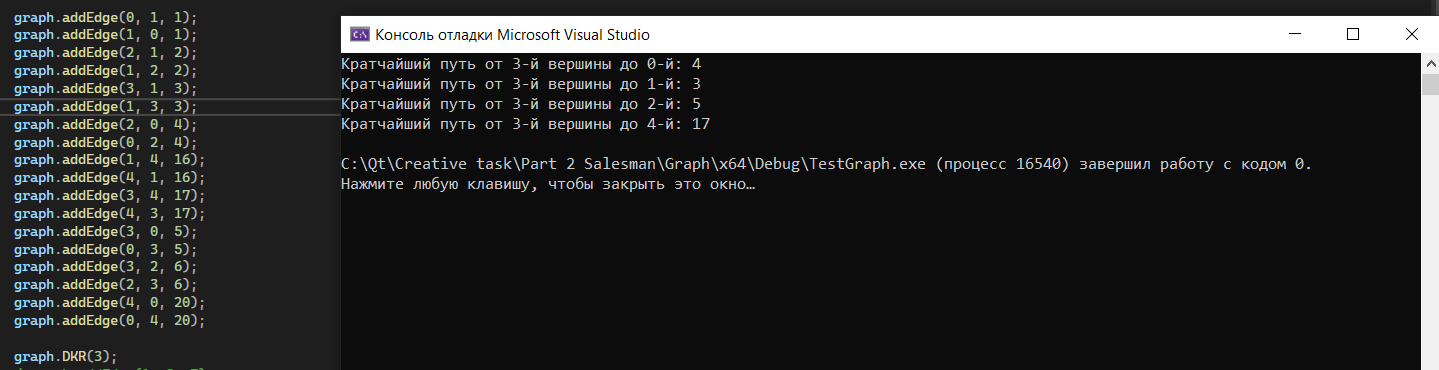


Рисунок 2 – алгоритм Дейкстры

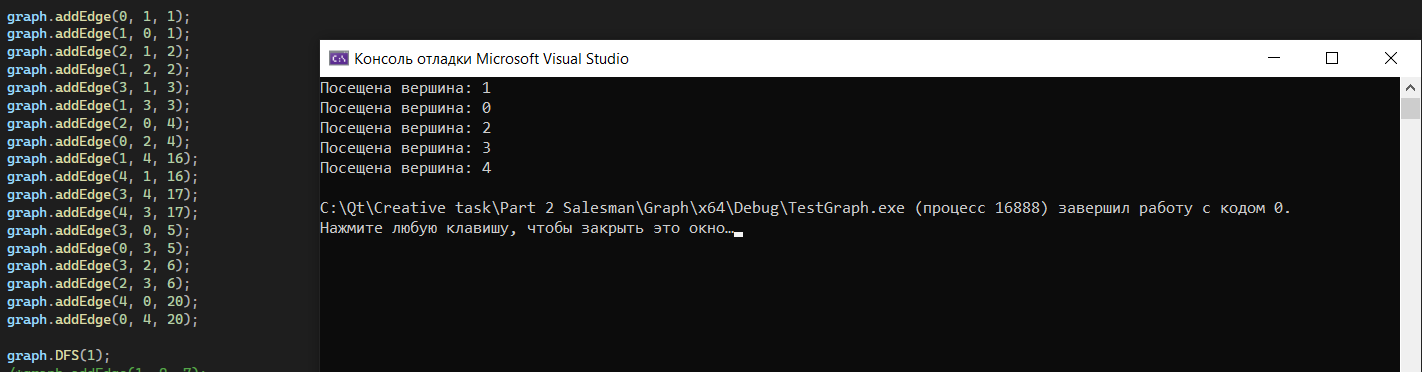


Рисунок 3 – обход в ширину

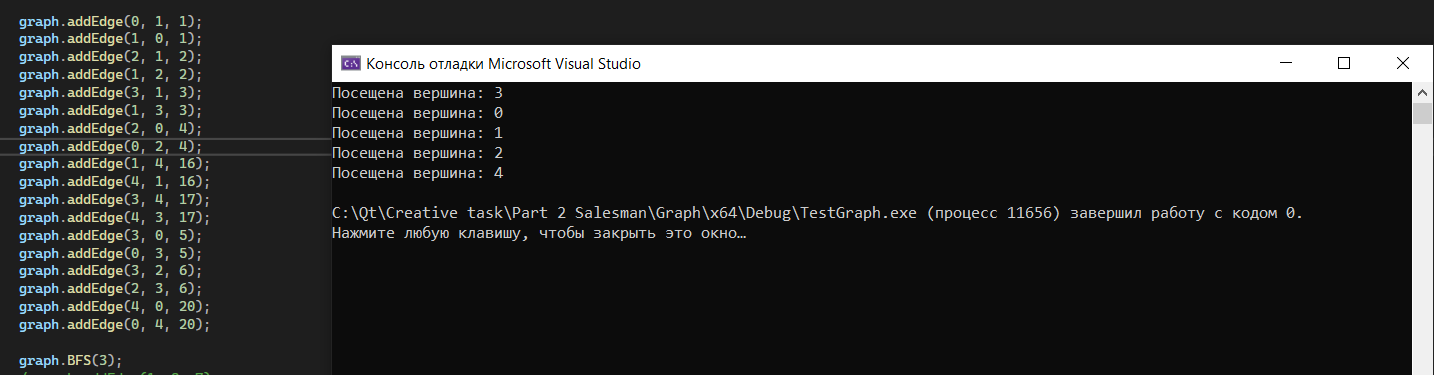


Рисунок 4 – обход в глубину

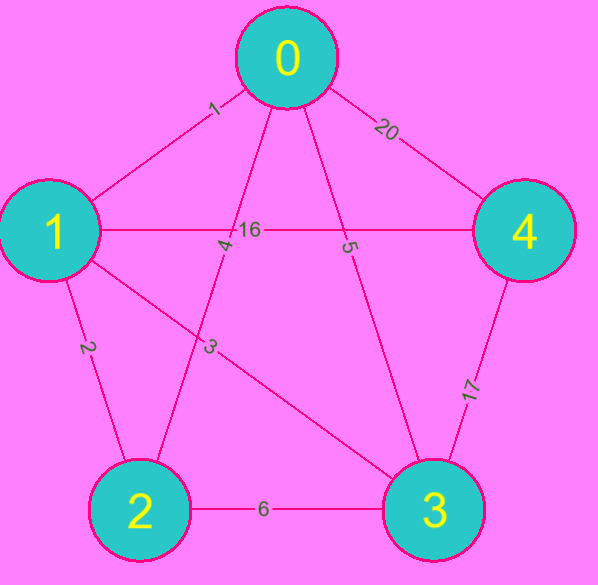


Рисунок 5 – Граф использованный для всех методов