Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе

Тема: «Графы»

Выполнил

Студент группы РИС-22-2б

Искендеров Р.С.

Проверил доц. Кафедры ИТАС

Полякова О.А.

Пермь 2023

# Постановка задачи

# Постановка задачи

Реализовать алгоритмы для собственного варианта графа, имеющего не менее 6 вершин.

Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

Требования:

1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)

2. Визуализация графа с использованием любой доступной графической библиотеки (SFML, SDL, OpenGL и подобных)

3. Реализованные алгоритмы должны справляться как с графом, представленным в задании варианта, так и с другими на усмотрение проверяющего.

4. Необходимо реализовать функции для редактирования графа:

- Создание новой вершины.

- Удаление вершины.

- Добавление и удаление ребра.

- Редактирование весов ребер.

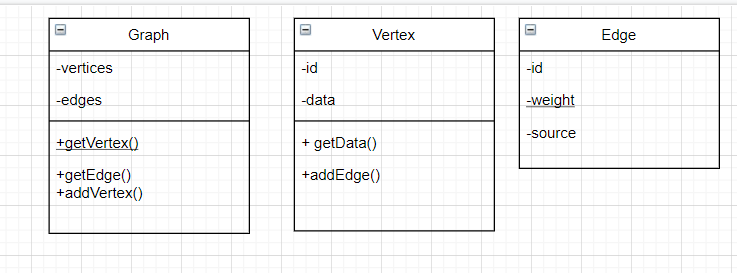
- Редактирование матрицы смежности (или инцидентности в зависимости от реализации).

Реализовать граф, построив все необходимые матрицы.

Выполнение начать с вершины 5.

UML – диаграмма

UML – диаграмма



Код программы

**graph.cpp**

#include "graph.h"

#include <QMessageBox>

#include <queue>

#include <stack>

using namespace std;

Graph::Graph(int vertices) {

numVertices = vertices;

// Инициализация матрицы смежности

adjacencyMatrix.resize(numVertices, vector<int>(numVertices, 0));

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

adjacencyMatrix[i][i] = 0; // Вес петли равен 0

}

}

// Возвращение количества вершин в графе

int Graph::getNumVertices() const {

return numVertices;

}

//Возвращение веса ребра между вершинами

int Graph::getEdgeWeight(int v1, int v2) const {

return adjacencyMatrix[v1][v2];

}

//Добавление нового ребра в граф.

void Graph::addEdge(int v1, int v2, int weight) {

adjacencyMatrix[v1][v2] = weight;

adjacencyMatrix[v2][v1] = weight;

}

//Возвращает вектор, содержащий все вершины графа

vector<int> Graph::getVertices() const {

vector<int> vertices(numVertices);

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

vertices[i] = i;

}

return vertices;

}

//Добавление новой вершины в граф

void Graph::addVertex() {

numVertices++;

adjacencyMatrix.resize(numVertices, vector<int>(numVertices, 0));

for (int i = 0; i < numVertices - 1; i++) {

adjacencyMatrix[i].resize(numVertices);

}

}

//Удаление вершины из графа

void Graph::removeVertex(int vertex) {

if (vertex < 0 || vertex >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверный номер вершины");

return;

}

numVertices--;

// Удаляем ребра, связанные с данной вершиной

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

adjacencyMatrix[i].erase(adjacencyMatrix[i].begin() + vertex);

}

// Удаляем вершину из матрицы смежности

adjacencyMatrix.erase(adjacencyMatrix.begin() + vertex);

}

//Возвращение матрицы смежности графа

vector<std::vector<int>> Graph::getAdjacencyMatrix() const {

return adjacencyMatrix;

}

//Удаление ребра из графа

void Graph::removeEdge(int v1, int v2) {

if (v1 < 0 || v1 >= numVertices || v2 < 0 || v2 >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверные номера вершин");

return;

}

adjacencyMatrix[v1][v2] = 0;

adjacencyMatrix[v2][v1] = 0;

}

//Изменение веса ребра

void Graph::editEdgeWeight(int v1, int v2, int weight) {

if (v1 < 0 || v1 >= numVertices || v2 < 0 || v2 >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверные номера вершин");

return;

}

adjacencyMatrix[v1][v2] = weight;

adjacencyMatrix[v2][v1] = weight;

}

void Graph::breadthFirstSearch(int startVertex) const {

if (startVertex < 0 || startVertex >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверный номер вершины");

return;

}

vector<bool> visited(numVertices, false);

queue<int> queue;

visited[startVertex] = true;

queue.push(startVertex);

QString result;

while (!queue.empty()) {

int currentVertex = queue.front();

queue.pop();

result += QString::number(currentVertex) + " ";

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

if (adjacencyMatrix[currentVertex][i] > 0 && !visited[i]) {

visited[i] = true;

queue.push(i);

}

}

}

QMessageBox::information(nullptr, "Результат", result);

}

void Graph::depthFirstSearch(int startVertex) const {

if (startVertex < 0 || startVertex >= numVertices) {

QMessageBox::critical(nullptr, "Ошибка", "Неверный номер вершины");

return;

}

vector<bool> visited(numVertices, false);

stack<int> stack;

visited[startVertex] = true;

stack.push(startVertex);

QString output;

while (!stack.empty()) {

int currentVertex = stack.top();

stack.pop();

output += QString::number(currentVertex) + " ";

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

if (adjacencyMatrix[currentVertex][i] > 0 && !visited[i]) {

visited[i] = true;

stack.push(i);

}

}

}

QMessageBox::information(nullptr, "Результат", output);

}

void Graph::Dijkstra(int startVertex, Graph graph) const {

vector<int> shortestDistances(numVertices, INT\_MAX);

vector<bool> visited(numVertices, false);

shortestDistances[startVertex] = 0;

for (int i = 0; i < numVertices - 1; i++) {

int minDistance = INT\_MAX;

int minIndex = -1;

for (int j = 0; j < numVertices; j++) {

if (!visited[j] && shortestDistances[j] <= minDistance) {

minDistance = shortestDistances[j];

minIndex = j;

}

}

visited[minIndex] = true;

for (int j = 0; j < numVertices; j++) {

if (!visited[j] && graph.getEdgeWeight(minIndex, j) > 0 && shortestDistances[minIndex] != INT\_MAX &&

shortestDistances[minIndex] + graph.getEdgeWeight(minIndex, j) < shortestDistances[j]) {

shortestDistances[j] = shortestDistances[minIndex] + graph.getEdgeWeight(minIndex, j);

}

}

}

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

QString message = "Кратчайшее расстояние от вершины " + QString::number(startVertex) + " до вершины " + QString::number(i) + ": ";

if (shortestDistances[i] == INT\_MAX) {

message += "нет пути";

} else {

message += QString::number(shortestDistances[i]);

}

QMessageBox::information(nullptr, "Результат", message);

}

}

**graph.h**

#ifndef GRAPH\_H

#define GRAPH\_H

#include <vector>

class Graph {

private:

int numVertices;

std::vector<std::vector<int>> adjacencyMatrix;

public:

Graph(int vertices);

void addVertex();

void removeVertex(int vertex);

void removeEdge(int v1, int v2);

void editEdgeWeight(int v1, int v2, int weight);

int getNumVertices() const;

int getEdgeWeight(int v1, int v2) const;

void addEdge(int v1, int v2, int weight);

void breadthFirstSearch(int startVertex) const;

void depthFirstSearch(int startVertex) const;

void Dijkstra(int startVertex, Graph graph) const;

std::vector<int> getVertices() const;

std::vector<std::vector<int>> getAdjacencyMatrix() const;

};

#endif // GRAPH\_H

**graphwidget.h**

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QMainWindow>

#include <QObject>

#include <QWidget>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QGraphicsLineItem>

#include <QPen>

#include <QBrush>

#include <QColor>

#include <QPointF>

#include <QRectF>

#include <QGraphicsTextItem>

#include "graph.h"

using namespace std;

// Define the PathInfo struct

struct PathInfo {

std::vector<int> path;

int cost;

PathInfo(const std::vector<int>& \_path, int \_cost) : path(\_path), cost(\_cost) {}

};

class GraphWidget : public QGraphicsView

{

Q\_OBJECT

public:

GraphWidget(QWidget\* parent = nullptr) : QGraphicsView(parent) {

setRenderHint(QPainter::Antialiasing);

setScene(new QGraphicsScene(this));

}

void reshGraph(const Graph& graph, const PathInfo& optimalPath);

void visGraph(const Graph& graph);

private:

void drawEdges(const Graph& graph, const vector<QPointF>& vertexPositions);

void drawVertices(const vector<QPointF>& vertexPositions);

void drawPath(const vector<int>& path, const vector<QPointF>& vertexPositions);

const int vertexLabelOffset = 5;

};

#endif // GRAPHWIDGET\_H

**graphwidget.cpp**

#include "graph.h"

#include "graphwidget.h"

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QGraphicsLineItem>

#include <QPen>

#include <QBrush>

#include <QColor>

#include <QPointF>

#include <QRectF>

void GraphWidget::reshGraph(const Graph& graph, const PathInfo& optimalPath) {

// Очистить сцену

scene()->clear();

// Получить количество вершин в графе

int numVertices = graph.getNumVertices();

// Вычислить положения вершин в круге для визуализации

const int sceneWidth = 1000;

const int sceneHeight = 1000;

const QPointF center(sceneWidth / 2, sceneHeight / 2);

const qreal radius = qMin(sceneWidth, sceneHeight) \* 0.4;

const qreal angleIncrement = 2 \* M\_PI / numVertices;

std::vector<QPointF> vertexPositions;

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

qreal angle = i \* angleIncrement;

qreal x = center.x() + radius \* qCos(angle);

qreal y = center.y() + radius \* qSin(angle);

vertexPositions.emplace\_back(x, y);

}

// Нарисовать рёбра в графе

drawEdges(graph, vertexPositions);

// Нарисовать вершины в графе

drawVertices(vertexPositions);

// Нарисовать оптимальный путь

drawPath(optimalPath.path, vertexPositions);

}

void GraphWidget::visGraph(const Graph& graph)

{

// Очистить сцену

scene()->clear();

// Получить количество вершин в графе

int numVertices = graph.getNumVertices();

// Вычислить положения вершин в круге для визуализации

const int sceneWidth = 1000;

const int sceneHeight = 1000;

const QPointF center(sceneWidth / 2, sceneHeight / 2);

const qreal radius = qMin(sceneWidth, sceneHeight) \* 0.4;

const qreal angleIncrement = 2 \* M\_PI / numVertices;

std::vector<QPointF> vertexPositions;

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

qreal angle = i \* angleIncrement;

qreal x = center.x() + radius \* qCos(angle);

qreal y = center.y() + radius \* qSin(angle);

vertexPositions.emplace\_back(x, y);

}

// Нарисовать рёбра в графе

drawEdges(graph, vertexPositions);

// Нарисовать вершины в графе

drawVertices(vertexPositions);

}

void GraphWidget::drawEdges(const Graph& graph, const std::vector<QPointF>& vertexPositions) {

const int numVertices = graph.getNumVertices();

const QPen edgePen(Qt::black);

for (int v1 = 0; v1 < numVertices; v1++) {

for (int v2 = 0; v2 < numVertices; v2++) {

int weight = graph.getEdgeWeight(v1, v2);

if (weight > 0) {

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), edgePen);

}

}

}

}

void GraphWidget::drawVertices(const std::vector<QPointF>& vertexPositions) {

const int numVertices = vertexPositions.size();

const int vertexRadius = 10;

const QPen vertexPen(Qt::black);

const QBrush vertexBrush(Qt::blue);

for (int v = 0; v < numVertices; v++) {

QPointF position = vertexPositions[v];

QRectF rect(position.x() - vertexRadius, position.y() - vertexRadius,

2 \* vertexRadius, 2 \* vertexRadius);

scene()->addEllipse(rect, vertexPen, vertexBrush);

// Добавьте метки к вершинам

QGraphicsTextItem\* label = scene()->addText(QString::number(v));

label->setPos(position.x() - vertexLabelOffset, position.y() - vertexLabelOffset);

}

}

void GraphWidget::drawPath(const std::vector<int>& path, const std::vector<QPointF>& vertexPositions) {

if (path.empty())

return;

const int pathPenWidth = 2;

const QColor pathColor = Qt::red;

const QPen pathPen(pathColor, pathPenWidth);

const int numVertices = path.size();

for (int i = 0; i < numVertices - 1; i++) {

int v1 = path[i];

int v2 = path[i + 1];

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), pathPen);

}

// Соедините последнюю и первую вершины пути

int v1 = path.back();

int v2 = path.front();

QPointF p1 = vertexPositions[v1];

QPointF p2 = vertexPositions[v2];

scene()->addLine(p1.x(), p1.y(), p2.x(), p2.y(), pathPen);

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "graphwidget.h"

#include "graph.h"

#include "qpushbutton.h"

#include <QLineEdit>

#include <QMessageBox>

#include <QString>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void onVertexCountChanged(const QString& text);

void updateGraph(int vertexCount);

void showAdjacencyMatrix();

void addVertex();

void addEdge();

void removeEdge();

void removeVertex();

void editWeight();

void breadth();

void depth();

void Dijkstra();

private:

Ui::MainWindow \*ui;

GraphWidget \*graphWidget; // Указатель на виджет графа

Graph graph; // Граф

int startVertex; // Начальная вершина для задачи Коммивояжера

QPushButton\* breadthButton;

QPushButton\* depthButton;

QPushButton\* DijkstraButton;

QLineEdit\* vertexCountLineEdit;

QPushButton\* adjacencyMatrixButton;

QPushButton\* addVertexButton;

QPushButton\* addEdgeButton;

QPushButton\* removeEdgeButton;

QPushButton\* removeVertexButton;

QPushButton\* editEdgeWeightButton;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <iostream>

#include <queue>

#include <QMessageBox>

#include <QLineEdit>

#include <QIntValidator>

#include <QInputDialog>

#include <QVBoxLayout>

using namespace std;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow), graph(0)

{

ui->setupUi(this);

// Создание графического виджета и добавление его на главное окно

graphWidget = new GraphWidget(this);

// Создание кнопки и установка текста

breadthButton = new QPushButton("Обход в ширину", this);

// Подключение сигнала нажатия кнопки к слоту

connect(breadthButton, &QPushButton::clicked, this, &MainWindow::breadth);

depthButton = new QPushButton("Обход в глубину", this);

// Подключение сигнала нажатия кнопки к слоту

connect(depthButton, &QPushButton::clicked, this, &MainWindow::depth);

DijkstraButton = new QPushButton("Алгоритм Дейкстры", this);

// Подключение сигнала нажатия кнопки к слоту

connect(DijkstraButton, &QPushButton::clicked, this, &MainWindow::Dijkstra);

// Создание поля для ввода числа вершин

vertexCountLineEdit = new QLineEdit(this);

vertexCountLineEdit->setGeometry(100,100,300,100);

vertexCountLineEdit->setPlaceholderText("Число вершин");

vertexCountLineEdit->setValidator(new QIntValidator(2, 100, this));

// Подключение сигнала изменения текста в поле для ввода

connect(vertexCountLineEdit, &QLineEdit::textChanged, this, &MainWindow::onVertexCountChanged);

adjacencyMatrixButton = new QPushButton("Матрица смежности",this);

connect(adjacencyMatrixButton, &QPushButton::clicked, this, &MainWindow::showAdjacencyMatrix);

addVertexButton = new QPushButton("Добавить вершину", this);

connect(addVertexButton, &QPushButton::clicked, this, &MainWindow::addVertex);

addEdgeButton = new QPushButton("Добавить ребро", this);

connect(addEdgeButton, &QPushButton::clicked, this, &MainWindow::addEdge);

removeEdgeButton = new QPushButton("Удалить ребро", this);

connect(removeEdgeButton, &QPushButton::clicked, this, &MainWindow::removeEdge);

removeVertexButton = new QPushButton("Удалить вершину", this);

connect(removeVertexButton, &QPushButton::clicked, this, &MainWindow::removeVertex);

editEdgeWeightButton = new QPushButton("Изменить вес ребра", this);

connect(editEdgeWeightButton, &QPushButton::clicked, this, &MainWindow::editWeight);

QVBoxLayout \*layout = new QVBoxLayout;

layout->addWidget(vertexCountLineEdit);

layout->addWidget(addVertexButton);

layout->addWidget(removeVertexButton);

layout->addWidget(addEdgeButton);

layout->addWidget(removeEdgeButton);

layout->addWidget(editEdgeWeightButton);

layout->addWidget(graphWidget);

layout->addWidget(breadthButton);

layout->addWidget(depthButton);

layout->addWidget(DijkstraButton);

layout->addWidget(adjacencyMatrixButton);

QWidget \*widget = new QWidget(this);

widget->setLayout(layout);

setCentralWidget(widget);

}

MainWindow::~MainWindow()

{

delete ui;

}

// Функция, которая вызывается при изменении числа вершин

void MainWindow::onVertexCountChanged(const QString& text)

{

bool ok;

int vertexCount = text.toInt(&ok);

if (ok) {

// Изменение графа

updateGraph(vertexCount);

} else {

// Ввод некорректного значения

QMessageBox::warning(this, "Ошибка", "Пожалуйста, введите целое число вершин.");

}

}

// Функция, которая обновляет граф при изменении числа вершин

void MainWindow::updateGraph(int vertexCount)

{

graph = Graph(vertexCount);

if (vertexCount < 2) {

QMessageBox::warning(this, "Ошибка", "Число вершин должно быть не менее 2.");

return;

}

bool ok = true;

while(ok){

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, vertexCount - 1, 1, &ok);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, vertexCount - 1, 1, &ok);

int weight = QInputDialog::getInt(this, "Вес ребра", "Введите вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1, &ok);

graph.addEdge(startikVertex, endVertex, weight);

QMessageBox::StandardButton reply = QMessageBox::question(this, "Создание ребра", "Хотите создать еще одно ребро?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::No) {

ok = false;

}

}

graphWidget->visGraph(graph);

}

// Функция, которая выводит матрицу смежности на экран

void MainWindow::showAdjacencyMatrix()

{

int numVertices = graph.getNumVertices();

std::vector<std::vector<int>> adjacencyMatrix = graph.getAdjacencyMatrix();

QString matrixString;

// Формирование строки с матрицей смежности

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

for (int j = 0; j < numVertices; j++) {

matrixString += QString::number(adjacencyMatrix[i][j]) + "\t";

}

matrixString += "\n";

}

// Отображение матрицы смежности в всплывающем окне

QMessageBox::information(this, "Матрица смежности", matrixString);

}

// Функция, которая позволяет добавить вершину в граф

void MainWindow::addVertex() {

graph.addVertex();

updateGraph(graph.getNumVertices());

graphWidget->visGraph(graph);

}

// Функция, которая позволяет добавить ребро в граф

void MainWindow::addEdge()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1);

int weight = QInputDialog::getInt(this, "Вес ребра", "Введите вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1);

graph.addEdge(startikVertex, endVertex, weight);

graphWidget->visGraph(graph);

}

// Функция, которая позволяет удалить ребро из графа

void MainWindow::removeEdge()

{

bool ok = true;

int numVertices = graph.getNumVertices();

while(ok){

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1, &ok);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1, &ok);

graph.removeEdge(startikVertex, endVertex);

QMessageBox::StandardButton reply = QMessageBox::question(this, "Удаление ребра", "Хотите удалить еще одно ребро?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::No) {

ok = false;

}

}

graphWidget->visGraph(graph);

}

// Функция, которая позволяет удалить вершину из графа

void MainWindow::removeVertex()

{

int numVertices = graph.getNumVertices();

int Vertex = QInputDialog::getInt(this, "Вершина", "Введите номер удаляемой вершины", 0, 0, numVertices - 1, 1);

graph.removeVertex(Vertex);

graphWidget->visGraph(graph);

}

// Функция, которая позволяет изменить вес ребра

void MainWindow::editWeight()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины для ребра", 0, 0, numVertices - 1, 1);

int endVertex = QInputDialog::getInt(this, "Конечная вершина", "Введите номер конечной вершины для ребра", 0, 0, numVertices - 1, 1);

int weight = QInputDialog::getInt(this, "Новый вес ребра", "Введите новый вес ребра", 0, 0, std::numeric\_limits<int>::max(), 1);

graph.editEdgeWeight(startikVertex, endVertex, weight);

graphWidget->visGraph(graph);

}

// Функция, которая позволяет вывести на экран результат обхода в ширину

void MainWindow::breadth()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины", 0, 0, numVertices - 1, 1);

graph.breadthFirstSearch(startikVertex);

graphWidget->visGraph(graph);

}

// Функция, которая позволяет вывести на экран результат обхода в глубину

void MainWindow::depth()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины", 0, 0, numVertices - 1, 1);

graph.depthFirstSearch(startikVertex);

graphWidget->visGraph(graph);

}

// Функция, которая позволяет вывести на экран результат алгоритма Дейкстры

void MainWindow::Dijkstra()

{

int numVertices = graph.getNumVertices();

int startikVertex = QInputDialog::getInt(this, "Начальная вершина", "Введите номер начальной вершины", 0, 0, numVertices - 1, 1);

graph.Dijkstra(startikVertex, graph);

graphWidget->visGraph(graph);

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

#include <algorithm>

#include <functional>

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**Результат**

