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**

**по творческому заданию**

Дисциплина: «Информатика»

Тема: Задача коммивояжера

Выполнил работу:

студентка группы РИС-20-2Б

Морозова Екатерина Максимовна

Проверила:

Доцент кафедры ИТАС

Полякова О.А

**Пермь, 2021**

**Цель работы**

**Постановка задачи**

Создание замкнутого кольцевого маршрута минимальной протяженности с однократным посещением каждого пункта.

**Анализ задачи**

**Суть задачи о коммивояжёре:**

Постановка задачи коммивояжёра гласит следующее: коммивояжёру, требуется найти минимальный путь до нужного города, с условиями:

1. Посетить все города ровно 1 раз;
2. Вернутся в город, из которого он начал обход городов;

В задании используется метод ветвей и границ, чтобы найти оптимальный маршрут коммивояжёра данным методом, необходимо выполнить следующие шаги:

1. Все нули таблицы представить в самое большое число;
2. Найти минимумы строк в таблицы, и выписать в отдельный столбец;
3. Из значений ячеек каждой строки вычесть соответствующий минимум;
4. Найти минимумы столбцов в таблице и выписать в отдельную строку;
5. Из значений ячеек каждого столбца вычесть соответствующий минимум;
6. Выбрать нулевую клетку (ячейку с числом 0) с максимальной оценкой (если таких ячеек несколько, выбирается любая);
7. Получаем пару ветвей;
8. Вычислить нижнюю границу первой ветви: найти минимумы по строкам, произвести редукцию строк, то же со столбцами, вычислить нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов;
9. Вычислить по принципу выше нижнюю границу второй ветви;
10. Если полный маршрут не найден, продолжить решение, перейти к шагу 8;
11. Построить полный маршрут и определить его длину:

#pragma once

#include <vector>

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Edge {

int gorodone ;

int gorodtwo;

};

class way

{

vector<vector<int>> Matrix;

vector<Edge> result;

vector<vector<int>> copy\_matrix;

string interval;

void reduct\_line();

void reduct\_col();

int findMinCol(int, int);

int findMinLine(int, int);

int lenght = 0;

public:

way(int);

~way() {};

void print\_matrix();

void reduct();

Edge FindEdge();

void make\_interval();

string get\_interval();

int get\_intervalLenght();

void make\_files();};

#include "Graph.h"

way::way(int n) {

track = "";

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

vector<int> tmp;

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

if (i == j) {

tmp.push\_back(-1);

}

else {

int a;

cout << "Из " << i + 1 << " в " << j + 1 << " : ";

cin >> a;

tmp.push\_back(a);

}

}

Matrix.push\_back(tmp);

}

copy\_Matrix = Matrix;

}

void way::print\_matrix() {

if (Matrix.size() == 0)

cout << "Matrix is empty" << endl;

else {

cout << "\nМатрица смежности:\n";

for (int i = 1; i <= Matrix.size(); i++) {

cout << "\t" << "(" << i << ")";

}

cout << endl;

for (int i = 0; i < Matrix.size(); i++) {

cout << "(" << i + 1 << ")" << "\t";

for (int j = 0; j < Matrix.size(); j++) {

cout << Matrix[i][j] << "\t";

}

cout << endl;

}

}

}

void way::reduct\_line() {

for (int i = 0; i < Matrix.size(); i++) {

int min = INT\_MAX;

for (int j = 0; j < Matrix[0].size(); j++) {

if (min > Matrix[i][j] && Matrix[i][j] >= 0) {

min = Matrix[i][j];

}

}

if (min != INT\_MAX) {

for (int j = 0; j < Matrix[0].size(); j++) {

if (Matrix[i][j] >= 0)

Matrix[i][j] -= min;

}

}

}

}

void way::reduct\_col() {

for (int i = 0; i < Matrix.size(); i++) {

int min = INT\_MAX;

for (int j = 0; j < Matrix[0].size(); j++) {

if (min > Matrix[j][i] && Matrix[j][i] >= 0) {

min = Matrix[j][i];

}

}

if (min != INT\_MAX) {

for (int j = 0; j < Matrix[0].size(); j++) {

if (Matrix[j][i] >= 0)

Matrix[j][i] -= min;

}

}

}

}

void way::reduct() {

reduct\_line();

reduct\_col();

}

int way::findMinCol(int i, int j) {

int min = INT\_MAX;

for (int l = 0; l < Matrix[0].size(); l++) {

if (l != i && Matrix[l][j] != -1 && min > Matrix[l][j]) {

min = Matrix[l][j];

}

}

return min;

}

int way::findMinLine(int i, int j) {

int min = INT\_MAX;

for (int c = 0; c < Matrix[0].size(); c++) {

if (c != j && Matrix[i][c] != -1 && min > Matrix[i][c]) {

min = Matrix[i][c];

}

}

return min;

}

Edge way::FindEdge() {

int max = -1;

Edge edge{ 0, 0 };

for (int i = 0; i < Matrix[0].size(); i++) {

for (int j = 0; j < Matrix[0].size(); j++) {

if (Matrix[i][j] == 0) {

int min\_sum = findMinCol(i, j) + findMinLine(i, j);

if (min\_sum > max) {

max = min\_sum;

edge.gorodone = i;

edge.gorodtwo = j;

}

}

}

}

Matrix[edge.gorodtwo][edge.gorodone] = -1;

for (int i = 0; i < Matrix[0].size(); i++) {

Matrix[edge.gorodone][i] = -1;

Matrix[i][edge.gorodtwo] = -1;

}

return edge;

}

void way::make\_interval() {

int n = Matrix.size();

for (int i = 0; i < n - 2; i++) {

reduct();

result.push\_back(FindEdge());

}

bool f = 0;

int lenght = INT\_MAX;

Edge edge;

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

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

if (Matrix[i][j] != -1 && Matrix[i][j] < lenght) {

lenght = Matrix[i][j];

edge.gorodone = i;

edge.gorodtwo = j;

f = 1;

}

}

if (f) result.push\_back(edge);

lenght = INT\_MAX;

f = 0;

}

}

string way::get\_interval() {

Edge current = result[0];

track += to\_string(current.gorodone + 1) + "->" + to\_string(current.gorodtwo + 1);

int count = 0;

while (count < result.size())

for (int i = 0; i < result.size(); i++) {

Edge next = result[i];

if (current.gorodtwo == next.gorodone) {

count++;

track += "->" + to\_string(next.gorodtwo + 1);

current = next;

break;

}

}

track = track.substr(0, track.size() - 3);

return track;

}

int way::get\_intervalLenght() {

Edge current = result[0];

lenght += copy\_matrix[current.gorodone][current.gorodtwo];

int count = 0;

while (count < result.size()) {

for (int i = 0; i < result.size(); i++) {

Edge next = result[i];

if (current.gorodtwo == next.gorodone) {

count++;

lenght += copy\_matrix[current.gorodone][current.gorodtwo];

current = next;

break;

}

}

}

lenght -= copy\_matrix[current.gorodone][current.gorodtwo];

return lenght;

}

void way::make\_files() {

fstream stream("Matrix.txt", ios::out | ios::trunc);

if (!stream) return;

for (int i = 0; i < Matrix.size(); i++) {

for (int j = 0; j < Matrix.size(); j++) {

stream << Matrix[i][j] <<" ";

}

stream << endl;

}

stream.close();

fstream stream2("Track.txt", ios::out | ios::trunc);

if (!stream2) return;

stream2 << track << endl << lenght << endl << Matrix[0].size();

stream2.close();

}

#include "Graph.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

int n;

cout << "Введите количество узлов: "; cin >> n;

way graph(n);

graph.print\_matrix();

graph.make\_interval();

cout << "\nПуть: " << graph.get\_interval();

cout << "\nЕго длина: " << graph.get\_intervalLenght();

graph.make\_files();

return 0;

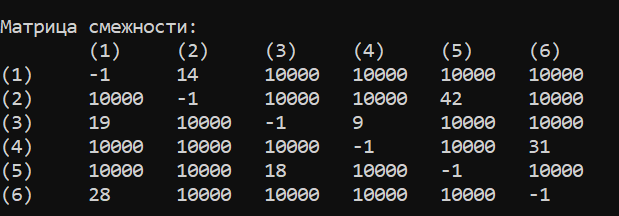
}

Необходимые действия:

1) Для того чтобы решить задачу коммивояжера, мною был придуман вариант решения, идея которого заключалась в том, что при нажатии кнопки пользователь вызывает консоль, в которую вводит все данные и затем выводится решение задачи и при нажатии кнопки, отвечающей за отрисовку изображения, выводится граф со всеми введенными пользователем значениями.

1) Для начала требуется ввести количество узлов, которое будет содержаться в графе.



2) Затем требуется ввести значения числовых промежутков между узлами. Для того чтобы указать, что дороги между двумя узлами нет, вводится значение-10000.

3) В консоли выводится решение нашей задачи и одновременно матрица смежности, затем решение записывается в файл.

4) В классе widgetOP объявлены функции:

1. Функция отвечающая за запуск файла консоли

void widgetOP::**writefile**() {

QProcess::startDetached("C:/mywork/run.bat");

}

2. Функция отвечающая за считывание данных с файла и создание с помощью этих данных матрицы смежности

void widgetOP::**readfile**() {

this->repaint();

QFile myWay("C:/mywork/Track.txt");

myWay.*open*(QFile::ReadOnly | QFile::Text);

Way = myWay.readLine();

qDebug() << Way << endl;

LenghtWay = myWay.readLine().toInt();

qDebug() << LenghtWay << endl;

kol = myWay.readLine().toInt();

qDebug() << kol << endl;

myWay.flush();

myWay.*close*();

QFile myMatrix("C:/mywork/Matrix.txt");

myMatrix.*open*(QFile::ReadOnly | QFile::Text);

QString line, a= "";

QVector<int> tmp;

QVector<QVector<int>> tmp2;

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

line = myMatrix.readLine();

for(int j = 0; j < line.size(); j++) {

if(line[j] != " ") {

a += line[j];

}

else {

tmp.push\_back(a.toInt());

a= "";

}

}

tmp2.push\_back(tmp);

tmp.clear();

a = "";

}

matrix = tmp2;

myMatrix.flush();

myMatrix.*close*();

}

3. Функция отвечающая за перерисовку графа

void widgetOP::**obnovlenie**()

{

this->update();

}

4. Функция отвечающая за перерисовку графа

void widgetOP::***paintEvent***(QPaintEvent \*event)

{

QPainter painter;

painter.begin(this);

mypainter->paint(*&painter*,*event*);

painter.end();

}

В классе MyPaint объявлена функция, с помощью которой рисуется изображение:

class **MyPaint**

{

public:

**MyPaint**();

public:

void **paint**(QPainter \*painter, QPaintEvent \*event);

QBrush front;

private:

QBrush back;

QFont Font;

QPen Pen;

};

Переменная QBrush front создана для фона обьектов, переменная  **Qbrush back создана для того, чтобы задать цвет окну, в котором рисуется сам граф, переменная QFont Font создана для задания размера шрифту и QPen Pen создана для задания цвета контуру узлов и линий между ними.**

**Функция , с помощью которой рисуется изображение:**

**void MyPaint::paint(QPainter \*painter, QPaintEvent \*event)**

{

painter->fillRect(event->rect(),back);

painter->setBrush(front);

painter->setPen(Pen);

painter->setFont(Font);

Pen.setWidth(4);

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

{

double a=pi\*3/2+pi\*2/kol\*i;

int xc=cos(a)\*R+xa;

int yc=sin(a)\*R+ya;

int c1=xc-20; //В данной части кода вычисляются координаты для текста

int c2=yc-Text/2;

int c3=40;

int c4=Text;

QString text; text.setNum(i+1);

painter->drawEllipse(QRectF(xc-r,yc-r,2\*r,2\*r));

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

for (int k=0;k<kol;k++)

{

if (matrix[i][k]!=-1 && matrix[i][k] != 10000&& matrix[i][k]!=0)

{

//Данная часть кода отвечает за рисование линии

double a2=pi\*3/2+pi\*2/kol\*k;

double xc2=cos(a2)\*R+xa;

double yc2=sin(a2)\*R+ya;

double l=sqrt((xc2-xc)\*(xc2-xc)+(yc2-yc)\*(yc2-yc));

int xl1=(xc2-xc)/l\*r+xc;

int yl1=(yc2-yc)/l\*r+yc;

int xl2=xc2-(xc2-xc)/l\*r;

int yl2=yc2-(yc2-yc)/l\*r;

painter->drawLine(xl1,yl1,xl2,yl2);

// Данная часть кода отвечает за рисование стрелки

double b;

if (xc2==xl2 && yl2<yc2) b=pi\*3/2;

else if (yc2==yl2 && xl2<xc2) b=pi;

else if (xc2==xl2 && yl2>yc2) b=pi/2;

else if (yc2==yl2 && xl2>xc2) b=0;

else if (xl2>xc2 && yl2<yc2) b=atan((yc2-yl2)/(xc2-xl2)+pi\*2;

else if (xl2<xc2 && yl2<yc2) b=atan((yc2-yl2)/(xc2-xl2))+pi;

else if (xl2<xc2 && yl2>yc2) b=atan((yc2-yl2)/(xc2-xl2))+pi;

else if (xl2>xc2 && yl2>yc2) b=atan((yc2-yl2)/(xc2-xl2));

int xs1=cos(b+angle)\*str+xl2;

int ys1=sin(b+angle)\*str+yl2;

int xs2=cos(b-angle)\*str+xl2;

int ys2=sin(b-angle)\*str+yl2;

painter->drawLine(xl2,yl2,xs1,ys1);

painter->drawLine(xl2,yl2,xs2,ys2);

// Данная часть кода отвечает за текст

c1=(xl2+xl1)/2-20+qrand()%strasp-strasp/2;

c2=(yl2+yl1)/2-Text/2+qrand()%strasp-strasp/2;

c3=40;

c4=Text;

text.setNum(matrix[i][k]);

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

QString s,s2;

//Данная часть кода отвечает за отображение решения задачи

s="Путь: " +Way;

s2="Длина пути: " +QString::number(LenghtWay);

painter->drawText(QRect(20,40,1000,40),Qt::AlignCenter,s);

painter->drawText(QRect(55,60,1000,40),Qt::AlignCenter,s2);

}

}

}

}

В классе MainWindow объявлена функция, которая отвечает за создание кнопок и окна, и за их месторасположение.

MainWindow::**MainWindow**(QWidget \*parent)

:QMainWindow(*parent*),ui(new Ui::MainWindow)

{

//Данная часть кода отвечает за вставку картинки на фон приложения

QPalette palette;

QPixmap pan("C:/mywork/soul.jpg");

pan = pan.scaled(this->size(), Qt::IgnoreAspectRatio);

palette.setBrush(QPalette::Background, pan);

this->setPalette(palette);

ui->setupUi(this);

// Данная часть кода отвечает за создание кнопок и окна в котором рисуется граф

myGLWidget \*openGLW=new myGLWidget(this);

QGridLayout \*grid=new QGridLayout();

QButtonGroup \*G=new QButtonGroup;

for(int i=0;i<3;i++)

{

QPushButton \*btn=new QPushButton;

btn->setFixedSize(200,100);

grid->addWidget(*btn*,i,1,1,2);

G->addButton(*btn*,i);

}

//Данная часть кода отвечае за местоположение кнопок и текст, который принадлежит определенной кнопке

G->button(0)->setText("обновить");

G->button(1)->setText("Отобразить граф");

G->button(2)->setText("Внести элементы матрицы ");

G->button(0)->setFixedSize(300,100);

G->button(1)->setFixedSize(300,100);

G->button(2)->setFixedSize(300,100);

grid->addWidget(*openGLW*,0,0,9,2);

grid->addWidget(*G->button(*0*)*, 3, 2);

grid->addWidget(*G->button(*1*)*, 2, 2);

grid->addWidget(*G->button(*2*)*, 1, 2);

for (int i=0;i<3;i++)

{

G->button(i)->setStyleSheet("QPushButton { background-color: rgb(0,255,255); }\n");

}

ui→centralwidget→setLayout(grid);

// Данная часть кода отвечает за связывание кнопок и действий которые будут совершаться при нажатии на определенную кнопку

connect(G->button(0),&QPushButton::clicked,openGLW,&myGLWidget::redraw);

connect(G->button(1), &QPushButton::clicked, openGLW,&myGLWidget::read\_from\_file);

connect(G->button(2), &QPushButton::clicked, openGLW, &myGLWidget::make\_file);

QFile inputFileTrack("C:/mywork/Track.txt");

inputFileTrack.*open*(QFile::ReadOnly | QFile::Text);

int s2;

QString s1;

s1 = inputFileTrack.readLine();

qDebug() << s1 << endl;

s2 = inputFileTrack.readLine().toInt();

qDebug() << s2 << endl;

inputFileTrack.flush();

inputFileTrack.*close*();

2. Программный код

**#ifndef MAINWINDOW\_H**

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QButtonGroup>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**#include "mainwindow.h"**

#include "ui\_mainwindow.h"

#include "widgetop.h"

#include <QPainter>

#include <QLineEdit>

#include <QGridLayout>

#include <QPushButton>

#include <mypaint.h>

#include<QButtonGroup>

#include<QFile>

#include<QDebug>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QPalette palette;

QPixmap pan("C:/mywork/soul.jpg");

pan = pan.scaled(this->size(), Qt::IgnoreAspectRatio);

palette.setBrush(QPalette::Background, pan);

this->setPalette(palette);

ui->setupUi(this);

ui->setupUi(this);

widgetOP \*openGLW=new widgetOP(this);

QGridLayout \*grid=new QGridLayout();

QButtonGroup \*G=new QButtonGroup;

for(int i=0;i<3;i++)

{

QPushButton \*btn=new QPushButton;

btn->setFixedSize(200,100);

grid->addWidget(*btn*,i,1,1,2);

G->addButton(*btn*,i);

}

G->button(0)->setText("обновить");

G->button(1)->setText("Отобразить граф");

G->button(2)->setText("Внести элементы матрицы ");

G->button(0)->setFixedSize(300,100);

G->button(1)->setFixedSize(300,100);

G->button(2)->setFixedSize(300,100);

grid->addWidget(*openGLW*,0,0,9,2);

grid->addWidget(*G->button(*0*)*, 3, 2);

grid->addWidget(*G->button(*1*)*, 2, 2);

grid->addWidget(*G->button(*2*)*, 1, 2);

for (int i=0;i<3;i++)

{

G->button(i)->setStyleSheet("QPushButton { background-color: rgb(0,255,255); }\n");

}

ui->centralwidget->setLayout(*grid*);

connect(G->button(0),&QPushButton::clicked,openGLW,&widgetOP::redraw);

connect(G->button(1), &QPushButton::clicked, openGLW,&widgetOP::read\_from\_file);

connect(G->button(2), &QPushButton::clicked, openGLW, &widgetOP::make\_file);

QFile inputFileTrack("C:/mywork/Track.txt");

inputFileTrack.*open*(QFile::ReadOnly | QFile::Text);

int s2;

QString s1;

s1 = inputFileTrack.readLine();

qDebug() << s1 << endl;

s2 = inputFileTrack.readLine().toInt();

qDebug() << s2 << endl;

inputFileTrack.flush();

inputFileTrack.*close*();

}

MainWindow::~***MainWindow***()

{

delete ui;

}

**#ifndef MYPAINT\_H**

#define MYPAINT\_H

#include <QBrush>

#include <QFont>

#include <QPen>

#include <QWidget>

class **MyPaint**

{

public:

**MyPaint**();

public:

void **paint**(QPainter \*painter, QPaintEvent \*event);

QBrush front;

private:

QBrush back;

QFont Font;

QPen Pen;

};

#endif // MYPAINT\_H

**#include "mypaint.h"**

#include "mypaint.h"

#include <QPaintEvent>

#include <QPainter>

#include <QWidget>

#include <QVector>

#include <QFile>

#include <QString>

#include <QTextStream>

#include <QtCore/qmath.h>

#include <widgetop.h>

#include <QDebug>

#include <QProcess>

#include<QLineEdit>

int kol;

int R=225;

int r=40;

double pi=3.1415;

int Text=20;

int xa=370;

int ya=450;

double angle=pi/8;

int str=30;

int strasp=60;

int LenghtWay;

QString Way;

bool find;

QVector<QVector<int>> matrix;

MyPaint::**MyPaint**()

{

Pen=QPen(QColor(150,33,250));

front=QBrush(QColor(0,255,180));

back=QBrush(QColor(255,255,255));

Font.setPixelSize(Text);

}

void widgetOP::**writefile**() {

QProcess::startDetached("C:/mywork/run.bat");

}

void widgetOP::**readfile**() {

this->repaint();

QFile myWay("C:/mywork/Track.txt");

myWay.*open*(QFile::ReadOnly | QFile::Text);

Way = myWay.readLine();

qDebug() << Way << endl;

LenghtWay = myWay.readLine().toInt();

qDebug() << LenghtWay << endl;

kol = myWay.readLine().toInt();

qDebug() << kol << endl;

myWay.flush();

myWay.*close*();

QFile myMatrix("C:/mywork/Matrix.txt");

myMatrix.*open*(QFile::ReadOnly | QFile::Text);

QString line, a= "";

QVector<int> tmp;

QVector<QVector<int>> tmp2;

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

line = myMatrix.readLine();

for(int j = 0; j < line.size(); j++) {

if(line[j] != " ") {

a += line[j];

}

else {

tmp.push\_back(a.toInt());

a= "";

}

}

tmp2.push\_back(tmp);

tmp.clear();

a = "";

}

matrix = tmp2;

myMatrix.flush();

myMatrix.*close*();

}

void MyPaint::**paint**(QPainter \*painter, QPaintEvent \*event)

{

painter->fillRect(event->rect(),back);

painter->setBrush(front);

painter->setPen(Pen);

painter->setFont(Font);

Pen.setWidth(4);

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

{

double a=pi\*3/2+pi\*2/kol\*i;

int xc=cos(a)\*R+xa;

int yc=sin(a)\*R+ya;

int c1=xc-20;

int c2=yc-Text/2;

int c3=40;

int c4=Text;

QString text; text.setNum(i+1);

painter->drawEllipse(QRectF(xc-r,yc-r,2\*r,2\*r));

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

for (int k=0;k<kol;k++)

{

if (matrix[i][k]!=-1 && matrix[i][k] != 10000&& matrix[i][k]!=0)

{

double a2=pi\*3/2+pi\*2/kol\*k;

double xc2=cos(a2)\*R+xa;

double yc2=sin(a2)\*R+ya;

double l=sqrt((xc2-xc)\*(xc2-xc)+(yc2-yc)\*(yc2-yc));

int xl1=(xc2-xc)/l\*r+xc;

int yl1=(yc2-yc)/l\*r+yc;

int xl2=xc2-(xc2-xc)/l\*r;

int yl2=yc2-(yc2-yc)/l\*r;

painter->drawLine(xl1,yl1,xl2,yl2);

double b;

if (xc2==xl2 && yl2<yc2) b=pi\*3/2;

else if (yc2==yl2 && xl2<xc2) b=pi;

else if (xc2==xl2 && yl2>yc2) b=pi/2;

else if (yc2==yl2 && xl2>xc2) b=0;

else if (xl2>xc2 && yl2<yc2) b=atan((yc2-yl2)/(xc2-xl2))+pi\*2;

else if (xl2<xc2 && yl2<yc2) b=atan((yc2-yl2)/(xc2-xl2))+pi;

else if (xl2<xc2 && yl2>yc2) b=atan((yc2-yl2)/(xc2-xl2))+pi;

else if (xl2>xc2 && yl2>yc2) b=atan((yc2-yl2)/(xc2-xl2));

int xs1=cos(b+angle)\*str+xl2;

int ys1=sin(b+angle)\*str+yl2;

int xs2=cos(b-angle)\*str+xl2;

int ys2=sin(b-angle)\*str+yl2;

painter->drawLine(xl2,yl2,xs1,ys1);

painter->drawLine(xl2,yl2,xs2,ys2);

c1=(xl2+xl1)/2-20+qrand()%strasp-strasp/2;

c2=(yl2+yl1)/2-Text/2+qrand()%strasp-strasp/2;

c3=40;

c4=Text;

text.setNum(matrix[i][k]);

painter->drawText(QRect(c1,c2,c3,c4),Qt::AlignCenter,text);

QString s,s2;

s="Путь: " +Way;

s2="Длина пути: " +QString::number(LenghtWay);

painter->drawText(QRect(20,40,1000,40),Qt::AlignCenter,s);

painter->drawText(QRect(55,60,1000,40),Qt::AlignCenter,s2);

}

}

}

}

**#ifndef WIDGETOP\_H**

#define WIDGETOP\_H

#include "mypaint.h"

#include <QOpenGLWidget>

class **widgetOP** : public QOpenGLWidget

{

public:

**widgetOP**(QWidget \*parent);

public slots:

void **obnovlenie**();

void **writefile**();

void **readfile**();

protected:

void ***paintEvent***(QPaintEvent \*event) override;

private:

MyPaint \*mypainter;

};

#endif // WIDGETOP\_H

**#include "widgetop.h"**

#include "mypaint.h"

#include <QPainter>

#include <QFile>

#include <QTextStream>

widgetOP::**widgetOP**(QWidget \*parent)

:QOpenGLWidget(*parent*)

{

mypainter=new MyPaint;

}

void widgetOP::***paintEvent***(QPaintEvent \*event)

{

QPainter painter;

painter.begin(this);

mypainter->paint(*&painter*,*event*);

painter.end();

}

void widgetOP::**obnovlenie**()

{

this->update();

}

**#include "mainwindow.h"**

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

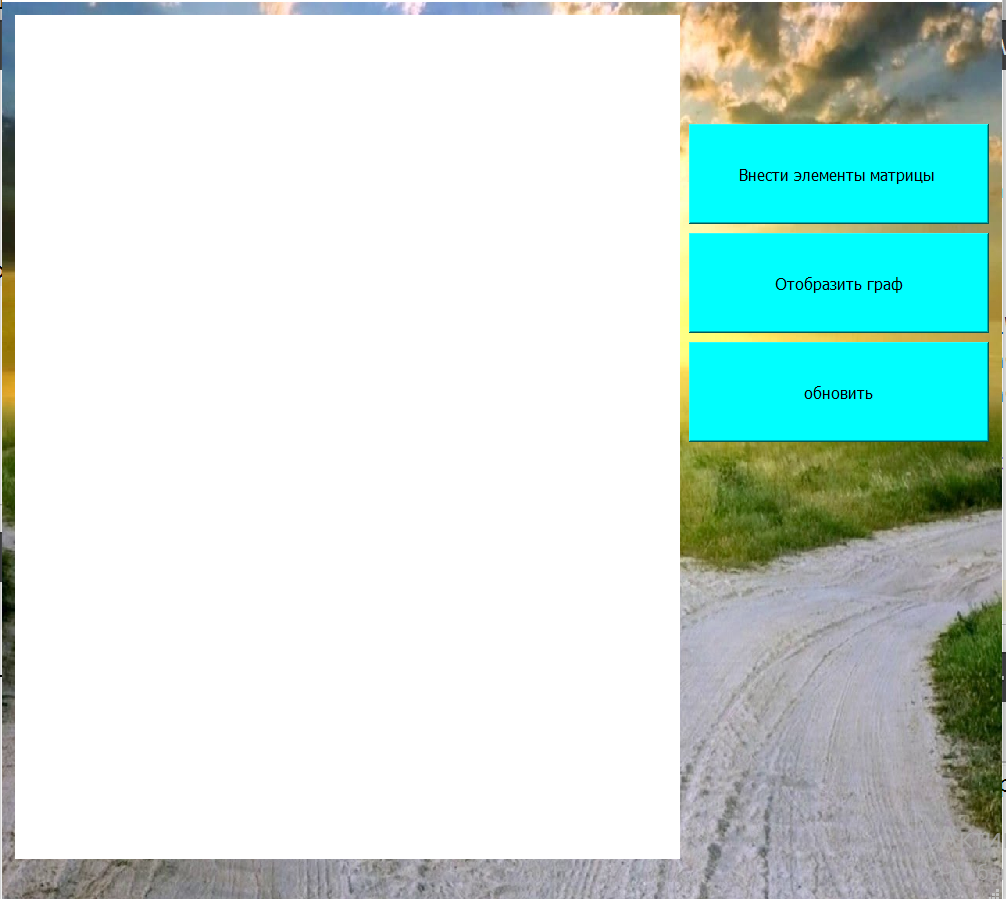
MainWindow w;

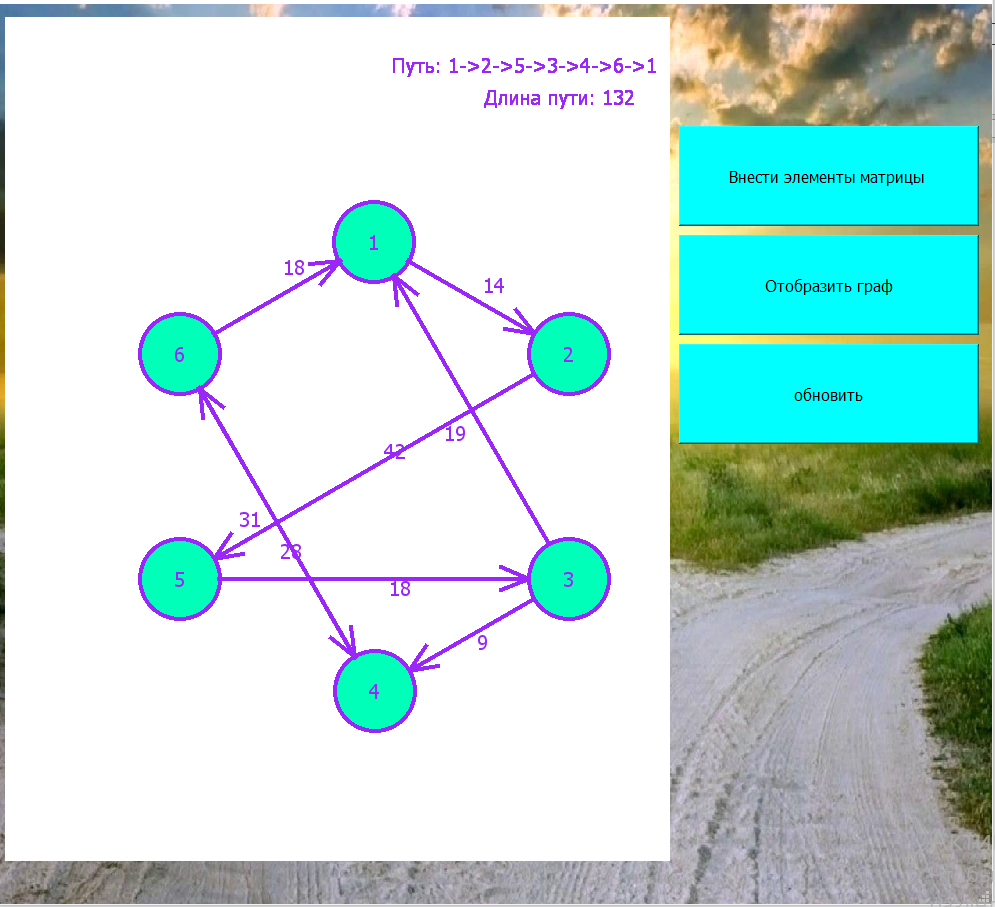
w.show();

return a.exec();

}

**Результат**

****

**UML-диаграмма**

