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 ТВОРЧЕСКОЙ РАБОТЕ №2**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Задача Коммивояжёра

Выполнил работу

Студент группы РИС-22-1Б

Головин М.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Постановка задачи**

1. Реализация приложения на основе QT, для решения задачи Коммивояжёра.
2. Изучить метод ветвей и границ.
3. Визуализация графа и решения задачи.

Метод ветвей и границ заключается в том, что нужно построить дерево потенциальных маршрутов. На каждом шаге нужно рассчитывать оценку перспективности маршрута и сравнивать с базовым циклом, при превышении которого маршрут не является перспективным, и отбрасывать его.

**Код программы**

**mainwindow.h**

**#ifndef MAINWINDOW\_H**

#define MAINWINDOW\_H

#include <QMainWindow>

#include <cityitem.h>

#include <QQueue>

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

struct **index**{

int ii;

int jj;

};

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

int n = -1;

public:

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

~***MainWindow***();

bool **empty**();

bool **chain**();

bool **table**();

void **find\_way**();

QString **depth**(int&, QVector<bool>, QString &);

QString **width**(int &, QVector<bool>&, QString &);

QString **Deixtra**(int &, QString &);

void **fill\_matrix**();

int **get\_vert\_pos**(int);

QVector<int> **GetNbrs**(int);

void **fill\_table**();

bool **AllVisited**(QVector<bool> &);

void **FillLabels**(int &);

int **GetWeight**(int, int);

signals:

void **send\_data\_vert** (int n);

private slots:

void **on\_Answer\_clicked**();

void **on\_Visual\_clicked**();

void **on\_Accept\_clicked**();

void **on\_Accept\_2\_clicked**();

void **on\_depth\_clicked**();

void **on\_width\_clicked**();

void **on\_Deikstra\_clicked**();

void **on\_depth\_2\_clicked**();

private:

Ui::MainWindow \*ui;

cityitem \*graph;

QVector<index> answer;

QVector<QVector<int>> matrix\_graph;

QQueue<int> VertsQueue;

QVector<int> vertexList;

QVector<int> labelList {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

};

#endif // MAINWINDOW\_H

**cityitem.h**

**#ifndef CITYITEM\_H**

#define CITYITEM\_H

#include <QWidget>

#include <QPainter>

#include <QDebug>

namespace **Ui** {

class **cityitem**;

}

class **cityitem** : public QWidget

{

Q\_OBJECT

public:

explicit **cityitem**(QWidget \*parent = 0);

~***cityitem***();

QVector<QString> temp;

QVector<QVector<QString>> matrix\_graph;

void **set\_size**(int);

void **fill\_matrix**(int);

protected:

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

public slots:

void **receiveData** (int x);

signals:

void **firstwindow**();

private slots:

void **on\_back\_clicked**();

private:

QVector<QString> name\_list = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

int n;

Ui::cityitem \*ui;

};

#endif // CITYITEM\_H

**mainwindow.cpp**

**#include "mainwindow.h"**

#include "ui\_mainwindow.h"

#include <QTableWidgetItem>

#include <algorithm>

#include <cmath>

#include <QMessageBox>

#include <ctime>

void MainWindow::**find\_way**()

{

int min = 9999, min\_ = 9999;

QVector<int> mini, mini\_;

QVector<index> answer;

// Минимальное значение

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

{

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

{

if (min >= matrix\_graph[i][j] && i != j && matrix\_graph[i][j] != -1) min = matrix\_graph[i][j];

}

mini.push\_back(min);

min = 9999;

}

// Редукция строк

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

{

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

{

if (i != j && matrix\_graph[i][j] != -1)

{

matrix\_graph[i][j] -= mini[i];

}

}

}

mini.clear();

// Поиск минимального в столбце

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

{

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

{

if (min > matrix\_graph[j][i] && i != j && matrix\_graph[j][i] != -1) min = matrix\_graph[j][i];

}

mini.push\_back(min);

min = 9999;

}

// Удаление по столбцам

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

{

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

{

if (i != j && matrix\_graph[i][j] != -1)

{

matrix\_graph[i][j] -= mini[j];

}

}

}

mini.clear();

// Оценка

// Помечаем 0

// Минимальные

int count\_0 = 0;

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

{

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

{

if (matrix\_graph[i][j] == 0 && i != j) count\_0++;

if (min > matrix\_graph[i][j] && i != j && matrix\_graph[i][j] != 0 && matrix\_graph[i][j] != -1) min = matrix\_graph[i][j];

if (count\_0 > 1) min = 0;

}

mini.push\_back(min);

min = 9999;

count\_0 = 0;

}

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

{

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

{

if (matrix\_graph[j][i] == 0 && i != j) count\_0++;

if (min\_ > matrix\_graph[j][i] && i != j && matrix\_graph[j][i] != 0 && matrix\_graph[j][i] != -1) min\_ = matrix\_graph[j][i];

if (count\_0 > 1) min\_ = 0;

}

mini\_.push\_back(min\_);

min\_ = 9999;

count\_0 = 0;

}

//Добавление к 0

int max = -1;

index ind;

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

{

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

{

if (i != j && matrix\_graph[i][j] == 0)

{

if (max < mini[i] + mini\_[j])

{

ind.ii = i;

ind.jj = j;

max = mini[i] + mini\_[j];

}

}

}

}

max = 0;

this->answer.push\_back(ind);

// Удаление строк и столбцов

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

{

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

{

if (i != j && (i == ind.ii || j == ind.jj))

{

matrix\_graph[i][j] = -1;

}

}

}

}

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

graph = new cityitem();

connect(graph, &cityitem::firstwindow, this, &MainWindow::show);

connect(this, SIGNAL(send\_data\_vert(int)), graph, SLOT(recievData(int)));

}

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

{

delete ui;

}

bool MainWindow::**empty**()

{

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

{

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

{

if(!(ui->LinksTable->item(i,j)))

{

QMessageBox::warning(this, "Ошибка", "Запишите информацию во все ячейки.");

return 0;

}

}

}

return 1;

}

bool MainWindow::**chain**()

{

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

{

if(ui->LinksTable->item(i,i)->text() != "0")

{

QMessageBox::warning(this, "Ошибка", "Не создавайте путь между одним и тем же городом.");

return 0;

}

}

return 1;

}

bool MainWindow::**table**()

{

if (n > 1) return 1;

QMessageBox::warning(this, "Ошибка", "Для использования данной функции создайте таблицу.");

return 0;

}

void MainWindow::**on\_Answer\_clicked**()

{

if(empty() && chain() && table())

{

QVector<int> temp;

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

{

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

{

temp.push\_back(ui->LinksTable->item(i,j)->text().toInt());

}

matrix\_graph.push\_back(temp);

temp.clear();

}

int counter = 0;

while (counter < n)

{

find\_way();

counter++;

}

matrix\_graph.clear();

QString ans;

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

{

if(answer[i].ii != 0)

{

for (int j = i+1; j < n; ++j)

{

if(j != n-1 && answer[j].ii == 0)

{

index swap = answer[i];

answer[i] = answer[j];

answer[j] = swap;

}

}

}

if(answer[i].jj == 0 && i != n -1)

{

for (int j = i+1; j < n; ++j)

{

index swap = answer[i];

answer[i] = answer[n-1];

answer[n-1] = swap;

}

}

if(i != n-1 && answer[i].jj != answer[i+1].ii)

{

for (int j = i+2; j < n; ++j)

{

if(i != n-1 && answer[i].jj == answer[j].ii)

{

index swap = answer[i+1];

answer[i+1] = answer[j];

answer[j] = swap;

}

}

}

}

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

{

ans = ans + QString::number(answer[i].ii + 1) + " -> ";

}

ans += '1';

int lengh = 0;

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

{

if(i != n - 1)

lengh += ui->LinksTable->item(answer[i].ii, answer[i+1].ii)->text().toInt();

}

lengh += ui->LinksTable->item(answer[n-1].ii, 0)->text().toInt();

ans = ans + " Длина = " + QString::number(lengh);

QMessageBox::information(this, "Ответ", ans);

answer.clear();

//

}

}

void MainWindow::**on\_Visual\_clicked**()

{

if (n > 10 || n <= 1)

{

if (n > 10) QMessageBox::warning(this, "Ошибка", "Максимальный размер графа равен 10!");

else QMessageBox::warning(this, "Ошибка", "Размер графа не может быть равен 0!");

}

else

{

if (empty() && chain())

{

graph->set\_size(n);

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

{

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

{

graph->temp.push\_back(ui->LinksTable->item(i,j)->text());

}

graph->matrix\_graph.push\_back(graph->temp);

graph->temp.clear();

}

graph->show();

graph->receiveData(n);

}

}

}

void MainWindow::**on\_Accept\_clicked**()

{

if (n == 10) QMessageBox::warning(this, "Ошибка", "Максимальный размер графа был достигнут!");

else

{

this->vertexList.push\_back(n);

n += 1;

ui->LinksTable->setRowCount(n);

ui->LinksTable->setColumnCount(n);

int temp = 0;

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

{

ui->LinksTable->setRowHeight(i,1);

ui->LinksTable->setColumnWidth(i,1);

ui->LinksTable->setItem(i,i,new QTableWidgetItem*(temp)*);

ui->LinksTable->item(i, i)->setText("0");

}

}

}

void MainWindow::**on\_Accept\_2\_clicked**()

{

this->vertexList.pop\_back();

n -= 1;

ui->LinksTable->setRowCount(n);

ui->LinksTable->setColumnCount(n);

int temp = 0;

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

{

ui->LinksTable->setRowHeight(i,1);

ui->LinksTable->setColumnWidth(i,1);

ui->LinksTable->setItem(i,i,new QTableWidgetItem*(temp)*);

ui->LinksTable->item(i, i)->setText("0");

}

}

void MainWindow::**fill\_table**()

{

srand(time(NULL));

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

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

if (i != j) ui->LinksTable->item(i, j)->setText(QString::number(rand()%10 + 5));

}

void MainWindow::**fill\_matrix**()

{

if(empty() && chain() && table())

{

QVector<int> temp;

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

{

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

{

temp.push\_back(ui->LinksTable->item(i,j)->text().toInt());

}

this->matrix\_graph.push\_back(temp);

temp.clear();

}

}

}

// Обход в глубину

QString MainWindow::**depth**(int& startVertex, QVector<bool> visitedVerts, QString &res)

{

if (visitedVerts[this->get\_vert\_pos(startVertex)] == false)

{

this->VertsQueue.push\_back(startVertex);

res += "Вершина " + QString::number(startVertex + 1) + " обработана\n" ;

visitedVerts[this->get\_vert\_pos(startVertex)] = true;

}

QVector<int> neighbors = this->GetNbrs(startVertex);

this->VertsQueue.pop\_front();

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

{

if (!visitedVerts[this->get\_vert\_pos(neighbors[i])])

{

this->VertsQueue.push\_back(neighbors[i]);

visitedVerts[this->get\_vert\_pos(neighbors[i])] = true;

res += "Вершина " + QString::number(neighbors[i] + 1) + " обработана\n";

}

}

if (!this->VertsQueue.empty()) depth(*VertsQueue.front()*, visitedVerts, *res*);

return res;

}

// Обход в ширину

QString MainWindow::**width**(int &start, QVector<bool>& visitedVerts, QString &res)

{

res+= "Вершина " + QString::number(start+1) + " обработана\n";

visitedVerts[this->get\_vert\_pos(start)] = true;

QVector<int> neighbors = this->GetNbrs(start);

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

if (!visitedVerts[this->get\_vert\_pos(neighbors[i])])

this->width(*neighbors[i]*, *visitedVerts*, *res*); }

return res;

}

bool MainWindow::**AllVisited**(QVector<bool> &visitedVerts)

{

bool flag = true;

for (int i = 0; i < this->vertexList.size(); i++)

if (visitedVerts[i] != true)

flag = false;

return flag;

}

void MainWindow::**FillLabels**(int &start)

{

for (int i = 0, size = vertexList.size(); i < size; ++i)

labelList[i] = 1000000;

int pos = get\_vert\_pos(start);

labelList[pos] = 0;

};

int MainWindow::**GetWeight**(int x, int y)

{

this -> fill\_matrix();

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

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

if(i == x && j == y)

return matrix\_graph[i][j];

return 0;

}

QString MainWindow::**Deixtra**(int &start, QString &res)

{

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

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

if (matrix\_graph[i][j] < 0)

return res;

if (get\_vert\_pos(start) == -1)

return res;

QVector<bool> visitedVerts(vertexList.size());

for (auto it = visitedVerts.begin(); it!=visitedVerts.end(); ++it) \*it = false;

this->FillLabels(*start*);

int curSrc = start;

QVector<int> neighbors;

while (!this->AllVisited(*visitedVerts*))

{

neighbors = this->GetNbrs(curSrc);

int startLabel = labelList[get\_vert\_pos(curSrc)];

int\* minNeighbor\_ptr = nullptr;

int minW = 1000000;

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

{

int weight = this->GetWeight(curSrc, neighbors[i]);

int nIndex = this->get\_vert\_pos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (startLabel + weight < nextLabel) labelList[nIndex] = startLabel + weight;

if (!visitedVerts[nIndex] && minW > labelList[nIndex])

{

minW = labelList[nIndex];

minNeighbor\_ptr = &neighbors[i];

}

}

visitedVerts[get\_vert\_pos(curSrc)] = true;

if (minNeighbor\_ptr != nullptr) curSrc = \*minNeighbor\_ptr;

}

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

{

if (i != get\_vert\_pos(start)){

res+= "Кратчайшее расстояние от вершины " + QString::number(start + 1)

+ " до вершины " + QString::number(vertexList[i] + 1) + " равно "

+ QString::number(labelList[get\_vert\_pos(vertexList[i])]) + "\n";

}

}

return res;

}

int MainWindow::**get\_vert\_pos**(int a)

{

fill\_matrix();

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

{

if (a == i)

return i;

}

return -1;

}

QVector<int> MainWindow::**GetNbrs**(int start)

{

QVector<int> nbrs;

fill\_matrix();

int pos = this->get\_vert\_pos(start);

if (pos != -1)

{

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

{

if (this->matrix\_graph[pos][i] != 0)

{

nbrs.push\_back(this->vertexList[i]);

}

}

}

return nbrs;

}

void MainWindow::**on\_depth\_2\_clicked**()

{

fill\_table();

}

**main.cpp**

**#include "mainwindow.h"**

#include <QApplication>

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

{

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

MainWindow w;

w.setWindowTitle("Коммивояжёр");

w.show();

return a.exec();

}

**cityitem.cpp**

**#include "cityitem.h"**

#include "ui\_cityitem.h"

#include <QtMath>

#include "mainwindow.h"

cityitem::**cityitem**(QWidget \*parent) :

QWidget(*parent*),

ui(new Ui::cityitem)

{

ui->setupUi(this);

}

void cityitem::**set\_size**(int new\_size)

{

this -> n = new\_size;

}

cityitem::~***cityitem***()

{

delete ui;

}

// Отрисовка графа

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

{

Q\_UNUSED(event);

QPainter painter(this);

painter.setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::FlatCap));

int y, x = -150;

int r = 150;

int len = 300/(n/2);

int \*\*citycoords = new int\*[n];

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

{

citycoords[i] = new int[2];

}

int k = 0;

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

{

y = qSqrt(r\*r - x\*x);

citycoords[k][0] = 200 - x;

citycoords[k][1] = 200 - y;

k++;

x += len;

}

x = 150;

if(n%2 == 0)

{

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

{

y = qSqrt(r\*r - x\*x)\*(-1);

citycoords[k][0] = 200 - x;

citycoords[k][1] = 200 - y;

k++;

x -= len;

}

}

else

{

len = 300/(n/2 + 1);

for(int i =0; i < n/2 + 1; i++)

{

y = qSqrt(r\*r - x\*x)\*(-1);

citycoords[k][0] = 200 - x;

citycoords[k][1] = 200 - y;

k++;

x -= len;

}

}

// Отрисовка рёбер

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

{

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

{

if(matrix\_graph[i][j] != "0" && matrix\_graph[j][i] != "0")

{

QPen pen(Qt::black, 2, Qt::SolidLine);

painter.setPen(pen);

painter.drawLine(citycoords[i][0]+25, citycoords[i][1]+25, citycoords[j][0]+25, citycoords[j][1]+25);

}

}

}

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

{

// Контур вершин

QPen pen1(Qt::black, 2, Qt::SolidLine);

painter.setPen(pen1);

// Вершины

painter.setBrush(QBrush(QColor(250,150,100,255), Qt::SolidPattern));

painter.drawEllipse(citycoords[i][0], citycoords[i][1], 50, 50);

painter.setFont(QFont(QString::number(i+1),14));

// Текст внутри вершин графа

QPen pen2(QColor(150,40,0,255), 2, Qt::SolidLine);

painter.setPen(pen2);

painter.drawText(citycoords[i][0]+15, citycoords[i][1]+30, name\_list[i]);

}

}

void cityitem::**receiveData**(int x)

{

n = x;

}

void cityitem::**on\_back\_clicked**()

{

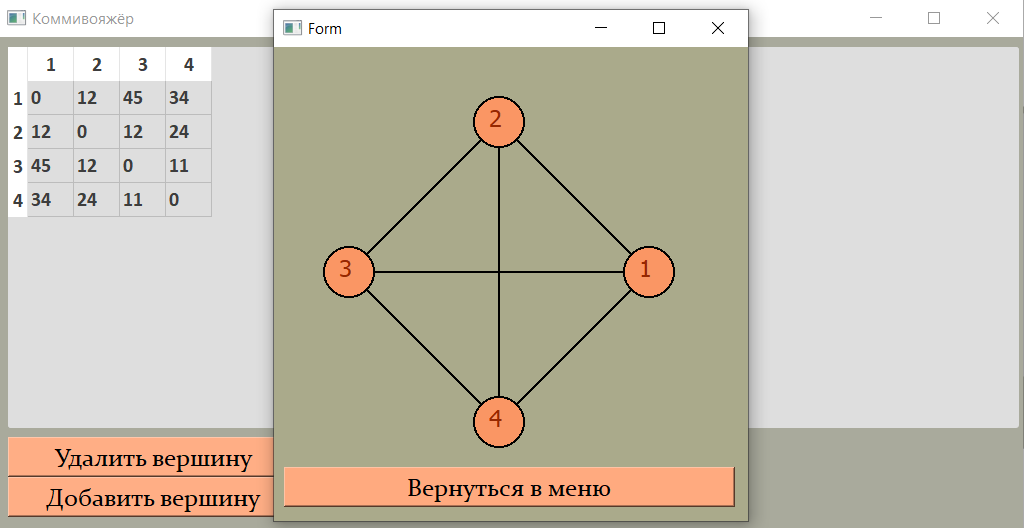
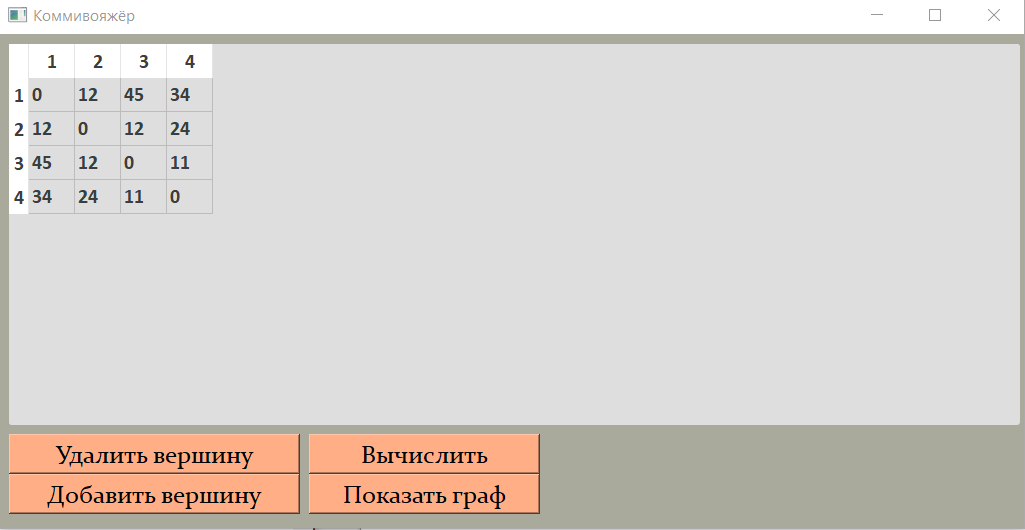
this->close();

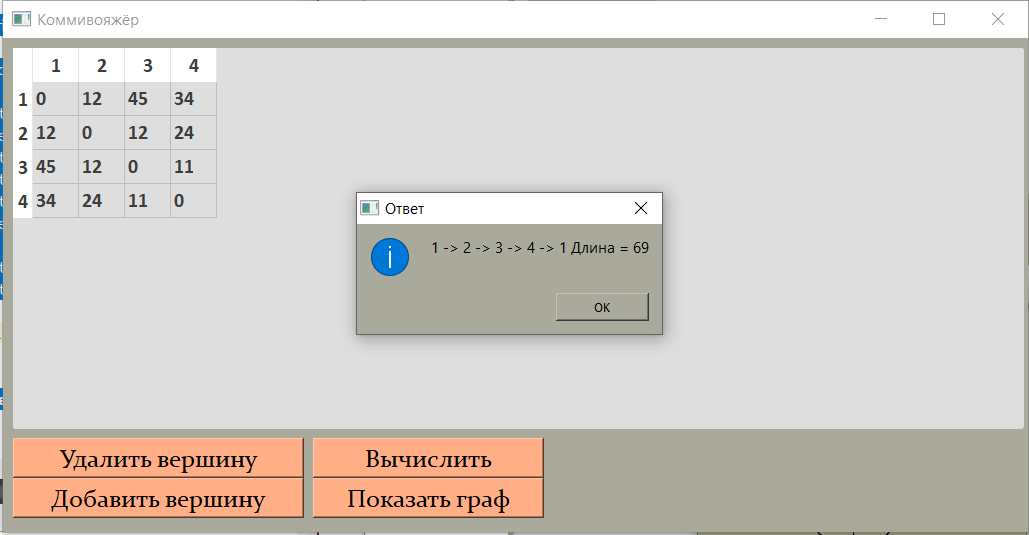
this->matrix\_graph.clear();

emit firstwindow();

}

**Вывод программы**





**Вывод**

Программа успешно справилась ввс поставленной задачей.