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»

УДК 004

ОТЧЁТ

о практическом задании

Тема: «Решение задачи коммивояжера»

Выполнили

Студенты группы РИС-23-1б

Краснов А.М.

Чунарев Д.Н.

Проверил доц. кафедры ИТАС

Полякова О.А.

Пермь 2024

**Содержание**

[Введение 3](#_Toc166673725)

[1 Изучение теоретических основ задачи коммивояжера 4](#_Toc166673726)

[2 Разработка и реализация алгоритмов для решения задачи, создание интерфейса приложения 6](#_Toc166673727)

[3 Практическое применение алгоритмов на конкретных данных и оценка полученных результатов 8](#_Toc166673728)

[Заключение 9](#_Toc166673729)

[Список использованных источников 10](#_Toc166673730)

[Приложение А 11](#_Toc166673731)

[Приложение Б 21](#_Toc166673732)

# Введение

Задача коммивояжера, также известная как задача о рюкзаке и задача о путешественнике, является одной из классических задач комбинаторной оптимизации. Основная цель этой задачи заключается в нахождении самого выгодного маршрута, который проходит через все заданные города ровно один раз и завершается в исходном городе. Решение этой задачи имеет широкое практическое применение в логистике, транспортировке и других областях, где необходимо оптимизировать маршруты.

Задача коммивояжера была сформулирована в 1930-х годах и с тех пор привлекала внимание ученых и исследователей своей сложностью и важностью. Несмотря на то, что задача относится к классу NP-трудных задач комбинаторной оптимизации, существуют различные методы ее приближенного решения, позволяющие найти оптимальные или приближенно оптимальные решения.

В данной лабораторной работе предполагается изучение различных алгоритмов для решения задачи коммивояжера, анализ их эффективности и применение на практике. Среди основных целей работы можно выделить:

1. Изучение теоретических основ задачи коммивояжера.

2. Разработка и реализация алгоритмов для решения задачи, создание интерфейса приложения.

3. Практическое применение алгоритмов на конкретных данных и оценка полученных результатов.

1. **Изучение теоретических основ задачи коммивояжера**

Существуют несколько методов решения задачи коммивояжера.

**1. Полный перебор (Brute Force):**

- Описание: Этот метод предполагает перебор всех возможных комбинаций маршрутов и выбор самого оптимального. Хотя он гарантированно находит оптимальное решение, его вычислительная сложность растет экспоненциально с увеличением числа городов.

- Применение: Подходит для небольших наборов городов, где можно перебрать все возможные варианты маршрутов.

**2. Метод ближайшего соседа (Nearest Neighbor):**

- Описание: Алгоритм выбирает ближайший доступный город в каждом шаге, строя таким образом маршрут. Хотя не гарантирует оптимальное решение, является быстрым и простым в реализации.

- Применение: Эффективен для больших наборов данных, где важна скорость выполнения.

**3. Муравьиный алгоритм (Ant Colony Optimization):**

- Описание: Этот метаэвристический алгоритм моделирует поведение муравьев в поиске оптимального маршрута. Муравьи откладывают феромоны на путях, предпочитая более короткие маршруты с более высокой концентрацией феромонов.

- Применение: Подходит для сложных задач с большим числом городов, где требуется нахождение приближенно оптимального решения.

4. **Метод ветвей и границ (Branch and Bound):**

Описание: Метод ветвей и границ является одним из классических методов точного решения задач комбинаторной оптимизации, включая задачу коммивояжера. Основная идея заключается в разбиении задачи на более мелкие подзадачи (ветвление) и определении верхних и нижних границ для улучшения процесса поиска оптимального решения.

Применение: Метод ветвей и границ эффективен для точного решения задачи коммивояжера, особенно при небольшом числе городов, когда полный перебор становится непрактичным.

Подробнее о методе ветвей и границ можно прочитать по ссылке [1].

Выбор метода решения задачи коммивояжера зависит от размера задачи, доступных ресурсов и требуемой точности результата.

В данной лабораторной работе для решения задачи коммивояжера был выбран метод ветвей и границ.

1. **Разработка и реализация алгоритмов для решения задачи, создание интерфейса приложения**

Для решения задачи необходимо реализовать код по данному алгоритму:

1. **Реализация матрицы смежности.**

Создание матрицы смежности происходит с помощью функций MainWindow::Hider() и MainWindow::Shower(), которые скрывают и показывают матрицу смежности соответственно. Функции используют переменную QString sear, а также встроенный алгоритм findChild для поиска элемента с определенным названием.

1. **Нахождение минимальных значений в строках/столбцах.**

Нахождение минимальных значений происходит с помощью функций FindMindInStr для строк и FindMinInCol для столбцов. Данные функции перебирают элементы с помощью цикла for.

1. **Редукция столбцов/строк.**

Редукция происходит с помощью функций ReductionStr для строк и ReductionCol для столбцов. Данные функции перебирают и переставляют элементы с помощью цикла for.

1. **Реализация поиска оптимального пути.**

Поиск оптимального пути происходит с помощью использования функции MainWindow::NewSearch. Данная функция перебирает возможные варианты пути, а затем выбирает оптимальный. В конце работы функции происходит формирование итогового списка возможных путей и очищение памяти.

1. **Оценивание нулевых элементов.**

Нулевые элементы оцениваются с помощью функции Mark, которая перебирает элементы, а затем копирует значение элементов в массив NewMatr (то есть избавляется от нулевых элементов).

1. **Редукция таблицы.**

Редукция таблицы происходит с помощью функции TableReduction.

1. **Реализация интерфейса с помощью приложения QT.**

Для реализации интерфейса было выбрано приложение QT, так как оно наиболее полно соответствует задаче отрисовки графа и создания понятного и удобного интерфейса

Полный код программы и реализации интерфейса представлен в приложении A.

1. **Практическое применение алгоритмов на конкретных данных и оценка полученных результатов**

Для проверки работы программы были использованы несколько наборов данных, с различным количеством городов и разной длиной пути между ними. Программа выдает корректный результат, а интерфейс понятен в использовании и не требует какой-либо технической документации

Пример работы программы и использования интерфейса представлен в приложении Б.

# Заключение

В ходе данной работы были рассмотрены различные варианты решения задачи коммивояжера, далее был разобран вариант решения задачи с помощью метода ветвей и границ. Был разработан алгоритм, по которому был написан код в среде Visual Studio и разработан интерфейс приложения с уникальным дизайном с помощью QT. Код программы исправно запускается и работает, выдавая верное решение задачи. В дальнейшем программа будет усовершенствоваться, позволяя выбирать методы решения задачи коммивояжера и неограниченное количество городов.

# Список использованных источников

1. Решение задачи коммивояжера, метод ветвей и границ. – URL <https://galyautdinov.ru/post/zadacha-kommivoyazhera>
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. - СПб.: Питер, 2021. - 464 с.
3. QT Creator Manual – URL https://doc.qt.io/qtcreator/

# Приложение А

**Исходный код программы**

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

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 on\_go\_clicked();

void on\_gotab\_clicked();

void Hider();

void Shower();

void on\_build\_clicked();

void on\_gotab\_pressed();

void on\_build\_pressed();

void on\_go\_pressed();

void NewSearch();

private:

Ui::MainWindow \*ui;

protected:

void paintEvent(QPaintEvent\*) override;

};

#endif // MAINWINDOW\_H

**Int main()**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.setWindowTitle("Мой любимый коммивояжер");

w.show();

return a.exec();

}

**Mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <cmath>

#include <string>

#include <QMessageBox> // для вывода окон

#include <QTextStream> // для потокового вывода

using namespace std;

void Mark(int\*\* matr);

int SumMins(int s, int c);

void TableReduction(int\*\* matr, int s, int c);

int r, ris = 0, prirost, xcenter, ycenter, mins = 10000, leng=0, s; // для отрисовки графа

double gradus = 360; // для отрисовки графа

int ways[10][10]; // таблица смежности

bool \*was; // список пунктов, которые были посещены

int \*way, \*minway;

int\*\* matr; // основная матрица смежности

struct Uzel{

int x, y;

};

Uzel\* uzel; // служит для отрисовки графа

int\* minstr, \* mincol; // список минимальных значений в строках и столбцах соответственно

bool\*\* mas; // вспомогательный массив для нахождения оценок нулевых элементов

int sumRoute = 0; // длина кратчайшего пути

int counter = 0; // счетчик

string \*buffer\_str; // список возможных передвижений

QTextStream cout(stdout); // для вывода в консоль (использовалось при тестировании)

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->build->hide();

ui->go->hide();

Hider();

ui->clh1\_2->hide();

}

void MainWindow::Hider()

{

// скрывает элементы матрицы смежности

QString sear; // шаблон элемента

for (int i = 1; i < 10; i++)

{

sear = "clh" + QString::number(i);

this->findChild<QWidget\*>(sear)->hide(); // нахождение элемента с определенным названием и сокрытие его

}

for (int i = 1; i < 10; i++)

{

sear = "clv" + QString::number(i);

this->findChild<QWidget\*>(sear)->hide(); // нахождение элемента с определенным названием и сокрытие его

sear = "d" + QString::number(i);

this->findChild<QWidget\*>(sear)->hide();// нахождение элемента с определенным названием и сокрытие его

for (int e = i+1; e<10; e++)

{

sear = "p" + QString::number(i) + QString::number(e);

this->findChild<QWidget\*>(sear)->hide(); // нахождение элемента с определенным названием и сокрытие его

}

for (int e = 1; e < i; e++)

{

sear = "l" + QString::number(i) + QString::number(e);

this->findChild<QWidget\*>(sear)->hide(); // нахождение элемента с определенным названием и сокрытие его

}

}

}

void MainWindow::Shower()

{

// показывает элементы матрицы смежности

QString sear;

for (int i = 1; i < r + 1; i++)

{

sear = "clh" + QString::number(i);

this->findChild<QWidget\*>(sear)->show(); // нахождение элемента с определенным названием и его отображение

}

for (int i = 1; i < r+1; i++)

{

sear = "clv" + QString::number(i);

this->findChild<QWidget\*>(sear)->show(); // нахождение элемента с определенным названием и его отображение

sear = "d" + QString::number(i);

this->findChild<QWidget\*>(sear)->show(); // нахождение элемента с определенным названием и его отображение

for (int e = i+1; e<r+1;e++)

{

sear = "p" + QString::number(i) + QString::number(e);

this->findChild<QWidget\*>(sear)->show(); // нахождение элемента с определенным названием и его отображение

}

ui->clh1\_2->show();

for (int e = 1; e<i;e++)

{

sear = "l" + QString::number(i) + QString::number(e);

this->findChild<QWidget\*>(sear)->show(); // нахождение элемента с определенным названием и его отображение

}

}

}

MainWindow::~MainWindow()

{

delete ui;

}

// преобразование из типа int в string

string FromIntToString(int a)

{

string c;

c = to\_string(a);

return c;

}

// преобразование из типа int в char

char FromIntToChar(int a)

{

char c;

c = a + '0';

return c;

}

// преобразование из типа char в int

int FromCharToInt(char s)

{

int a = s - '0';

return a;

}

// проверяет, все ли пути найдены

bool NotAtAll(int\*\* matr)

{

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

{

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

{

if (matr[i][j] != -1) // если встретился хотя бы один элемент, не равный -1

return false; // значит он не обработан

}

}

return true;

}

// находит минимальное значение в строках

void FindMinInStr(int \*\*matr)

{

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

{

int min = 1000;

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

{

if (matr[i][j] != -1)

if (matr[i][j] < min)

min = matr[i][j];

}

minstr[i] = min;

if (minstr[i] == 10000)

minstr[i] = 0;

}

}

// находит минимальное значение в столбцах

void FindMinInCol(int\*\* matr)

{

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

{

int min = 1000;

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

{

if (matr[i][j] != -1)

if (matr[i][j] < min)

min = matr[i][j];

}

mincol[j] = min;

if (minstr[j] == 10000)

minstr[j] = 0;

}

}

// производит редукцию строк

void ReductionStr(int\*\* matr)

{

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

{

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

{

if (matr[i][j] != -1)

{

matr[i][j] -= minstr[i];

}

}

}

}

// производит редукцию столбцов

void ReductionCol(int\*\* matr)

{

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

{

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

{

if (matr[j][i] != -1)

{

matr[j][i] -= mincol[i];

}

}

}

}

// алгоритм поиска оптимального пути

void MainWindow::NewSearch()

{

QString qstr = ""; // строка с возможными передвижениями

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

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

mas[i][j] = false;

buffer\_str = new string [r];

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

{

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

{

matr[i][j] = ways[i][j];

}

}

minstr = new int[r];

mincol = new int[r];

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

{

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

{

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

matr[i][j] = -1; // обозначение отсутствия ребер между узлами

}

}

int\*\* copyMatr = new int\* [r]; // копия матрицы смежности

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

copyMatr[i] = new int[r];

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

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

copyMatr[i][j] = 0; // инициализация элементов

while (NotAtAll(matr) == false) // до тех пор, пока все элементы не обработаны

{

FindMinInStr(matr);

ReductionStr(matr); // находим минимумы и производим редукцию строк

FindMinInCol(matr);

ReductionCol(matr); // находим минимумы и производим редукцию столбцов

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

mas[i] = new bool[r];

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

{

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

mas[i][j] = false; // инициализация элементов массива

}

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

{

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

{

copyMatr[i][j] = matr[i][j]; // присваивание значений элементов массива matr массиву copyMatr

}

}

Mark(matr); // нахождение оценок для нулевых элементов

int m = -1;

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

{

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

{

if (mas[i][j] == true)

{

if (m < matr[i][j])

{

m = ways[i][j];

buffer\_str[counter] = FromIntToString(i) + "->" + FromIntToString(j); // формирование списка возможных путей

}

}

}

}

sumRoute += m; // формирование длины кратчайшего пути

string str = buffer\_str[counter];

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

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

matr[i][j] = copyMatr[i][j]; // присваивание значений элементов массива copyMatr массиву matr

TableReduction(matr, FromCharToInt(str[0]), FromCharToInt(str[3])); // редукция таблицы

counter++;

}

string result = "";

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

{

string a = buffer\_str[i];

int first = FromCharToInt(a[0]), last = FromCharToInt(a[3]);

a[0] = FromIntToChar(first + 1);

a[3] = FromIntToChar(last + 1);

buffer\_str[i] = a;

result += "\n" + buffer\_str[i];

}

/\* формирование итогового списка возможных путей \*/

qstr = "Возможные передвижения: " + QString::fromStdString(result);

string res = FromIntToString(sumRoute);

/\* вывод длины кратчайшего пути \*/

qstr += "\nДлина кратчайшего пути = " + QString::fromStdString(res);

QMessageBox::information(this, "Результат", qstr);

/\* очищение памяти \*/

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

delete[] matr[i];

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

delete[] mas[i];

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

delete[] copyMatr[i];

}

int SumMins(int s, int c)

{

// сумма минимальных элементов строки и столбца

int ms = 10000, mc = 10000;

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

{

if (i != c && matr[s][i] != -1)

{

if (ms > matr[s][i])

ms = matr[s][i]; // нахождение минимального значения в строках

}

}

if (ms == 10000) ms = 0;

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

{

if (i != s && matr[i][c] != -1)

{

if (mc > matr[i][c])

mc = matr[i][c]; // нахождение минимального значения в столбцах

}

}

if (mc == 10000) mc = 0;

return ms + mc;

}

void Mark(int\*\* matr)

{

// нахождение оценок для нулевых элементов

int\*\* newMatr = new int\* [r];

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

newMatr[i] = new int[r];

}

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

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

newMatr[i][j] = matr[i][j]; // копирование значений элементов в массив newMatr

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

{

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

{

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

{

mas[i][j] = true;

newMatr[i][j] = SumMins(i, j);

// присваивание элементу суммы минимальных

// значений в соответствующих строке и столбце

}

}

}

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

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

matr[i][j] = newMatr[i][j];

// очищение памяти

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

delete[] newMatr[i];

}

void TableReduction(int\*\* matr, int s, int c)

{

// редукция таблицы

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

{

matr[s][i] = -1; // инициализация всех элементов строки значением -1

}

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

{

matr[i][c] = -1; // инициализация всех элементов столбца значением -1

}

matr[c][s] = -1; // инициализация самого элемента значением -1

}

void MainWindow::on\_go\_clicked()

{

// вычислить

ui->go->setStyleSheet("background-color:rgb(133, 129, 255); border-radius: 10px;");

QString searcher, buffer;

leng = 0;

s = 0;

mins = 10000;

was = new bool[r];

way = new int[r];

minway = new int[r];

mas = new bool\* [r];

matr = new int\*[r];

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

matr[i] = new int[r];

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

mas[i] = new bool[r];

NewSearch();

delete[] was;

delete[] way;

}

void MainWindow::paintEvent(QPaintEvent \*)

{

if (ris)

{

int otstup;

gradus /= r;

prirost = gradus;

gradus = 0;

QPainter paint;

QPen pen;

QBrush brush;

QString stroka;

QFont font;

pen.setColor(Qt::blue);

pen.setWidth(5);

pen.setStyle(Qt::SolidLine);

font.setPointSize(20);

brush.setStyle(Qt::Dense3Pattern);

brush.setColor(Qt::white);

paint.begin(this);

paint.setPen(pen);

paint.setFont(font);

paint.setBrush(brush);

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

{

otstup = -10;

if (gradus > 100)

otstup = 130;

stroka = QString::number(i + 1);

uzel[i].x = xcenter + (200 \* cos(gradus / 57));

uzel[i].y = ycenter + (200 \* sin(gradus / 57));

paint.drawEllipse(uzel[i].x, uzel[i].y, 100, 100);

paint.drawText(uzel[i].x + 48, uzel[i].y + otstup, stroka);

gradus += prirost;

}

if (ris > 1)

{

pen.setStyle(Qt::SolidLine);

pen.setWidth(2);

paint.setPen(pen);

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

{

for (int e = i + 1; e < r; e++)

{

if (ways[i][e])

{

paint.drawLine(uzel[i].x + 50, uzel[i].y + 50, uzel[e].x + 50, uzel[e].y + 50);

}

}

}

}

paint.end();

}

}

void MainWindow::on\_gotab\_clicked()

{

// создать граф (граф и таблица смежности)

ui->gotab->setStyleSheet("background-color:rgb(133, 129, 255); border-radius: 10px;");

ui->go->hide();

if (ui->quantitylabel->text().toInt()<3 || ui->quantitylabel->text().toInt()>9)

{

ui->quantitylabel->setText("");

ui->build->hide();

Hider();

ris=0;

this->repaint();

QMessageBox::warning(this, "Предупреждение", "Количество городов недопустимо");

}

else

{

Hider();

ui->build->show();

r = ui->quantitylabel->text().toInt();

Shower();

uzel= new Uzel[r];

xcenter=800;

ycenter=300;

ris=1;

this->repaint();

}

}

void MainWindow::on\_build\_clicked()

{

// построить пути на графе

ui->build->setStyleSheet("background-color:rgb(133, 129, 255); border-radius: 10px;");

ui->go->hide();

QString searcher, buffer;

for (int i=1;i<r;i++)

{

for (int e=i+1; e<r+1;e++)

{

searcher = "p"+QString::number(i) + QString::number(e);

buffer = this->findChild<QLineEdit\*>(searcher)->text();

if (!buffer.length()) buffer = '0';

searcher = "l"+QString::number(e) + QString::number(i);

this->findChild<QLabel\*>(searcher)->setText(buffer);

}

}

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

{

for (int e=i+1;e<r;e++)

{

searcher="l"+QString::number(e+1) + QString::number(i+1);

buffer = this->findChild<QLabel\*>(searcher)->text();

if (!buffer.length()) buffer='0';

ways[i][e]=buffer.toInt();

ways[e][i]=ways[i][e];

}

}

ui->go->show();

ris=2;

this->repaint();

}

// описание поведения кнопок при их нажатии

void MainWindow::on\_gotab\_pressed()

{

ui->gotab->setStyleSheet("color: rgb(250, 0, 0); background-color: rgb(51,51,204); border: 1px solid rgb(0, 0, 0); border-radius: 10px;");

}

void MainWindow::on\_build\_pressed()

{

ui->build->setStyleSheet("color: rgb(250, 0, 0); background-color: rgb(51,51,204); border: 1px solid rgb(0, 0, 0); border-radius: 10px;");

}

void MainWindow::on\_go\_pressed()

{

ui->go->setStyleSheet("color: rgb(250, 0, 0); background-color: rgb(51,51,204); border: 1px solid rgb(0, 0, 0); border-radius: 10px;");

}

# 

# Приложение Б

**Результаты работы программы**

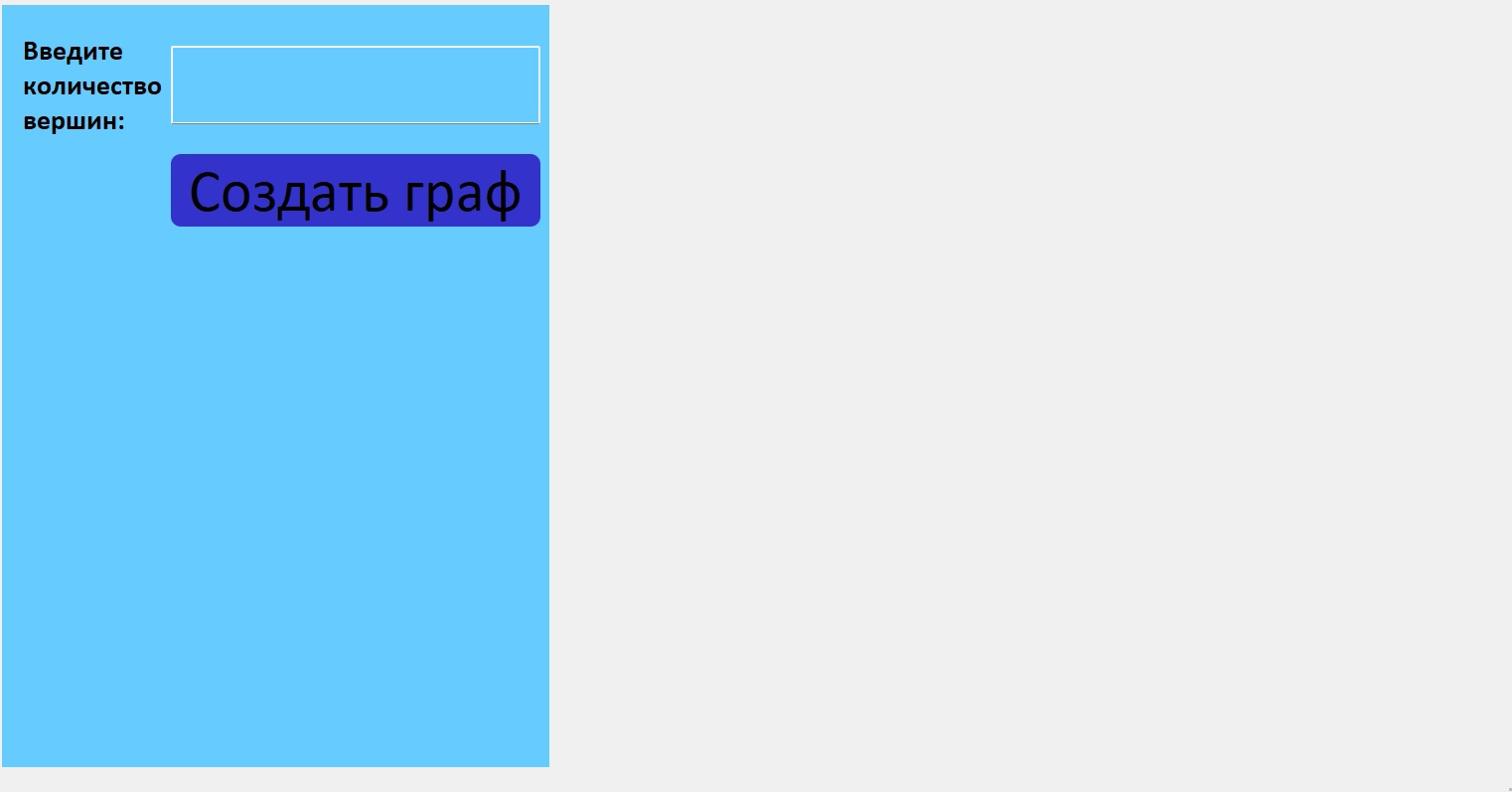


Рис. 1 – Окно создания графа

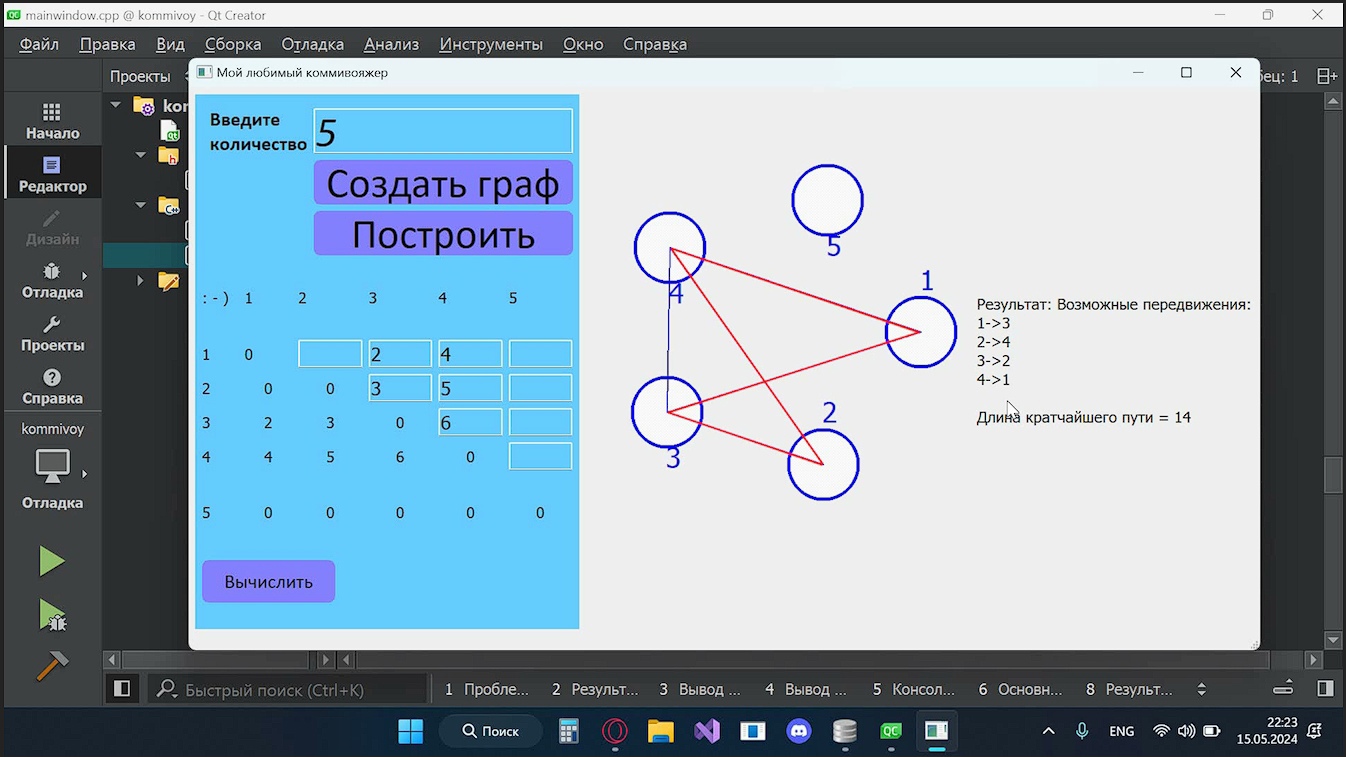


Рис. 2 – Построенный граф с выделенным путем