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

о лабораторной работе

Тема: «Реализация решения задачи о коммивояжере»

Выполнил

Студент группы РИС-23-1б

Валинуров Р.Н.

Проверил доц. кафедры ИТАС

Полякова О.А.

Пермь, 2024

**Содержание**

Введение**3**

1 Постановка задачи**4**

2 Анализ задачи**5-6**

3 Разработка программы**7-8**

Заключение**9**

Использованные источники**10**

Приложение А**11-43**

**Введение**

Задача состоит в том, чтобы коммивояжер (торговец) обошел все намеченные города единожды и в таком порядке, чтобы его путь был наименьшим.

Эта задача заинтересовала меня потому, что её решение интересно с точки зрения программирования и составления алгоритма. Важно нахождение такого алгоритма, который позволит наиболее оптимально решить задачу.

Сейчас решение данной задачи необходимо во многих областях связанных с замкнутыми и при этом жестко связанными по времени системами, такими как: конвейерное производство, многооперационные обрабатывающие комплексы, судовые и железнодорожные погрузочные системы, перевозки грузов по замкнутому маршруту, расчет авиационных линий.

**1 Постановка задачи**

Имеется N городов, которые должен обойти коммивояжер с минимальными затратами. При этом на его маршрут накладывается два ограничения:

* маршрут должен быть замкнутым, то есть коммивояжер должен вернуться в тот город, из которого он начал движение;
* в каждом из городов коммивояжер должен побывать точно один раз, то есть надо обязательно обойти все города, при этом не побывав ни в одном городе дважды.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого (в матрице диагональ заполнена нулями).

Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

**2 Анализ задачи**

Для начала следует сказать, что в основе любого метода решения данной задачи лежит полный перебор всевозможных вариантов путей.

Мы проходимся по каждому маршруту: одни отбрасываем, другие сравниваем с минимальным путем. В конце перебора мы получаем кратчайший путь.

Особенностью этой задачи является то, что с увеличением количества городов растет общее число различных комбинаций прохождения пути. А вместе с тем растет и время расчета результата. Поэтому главным решением оптимизации алгоритма можно свести к тому, чтобы во время вычислений отбрасывать заведомо не минимальные пути. Необходимо задать такой критерий, который отсекал бы лишние ветви в дереве поиска кратчайшего пути.

Для пояснения моего варианта решения задачи следует ввести несколько понятий. Промежуточную длину пути можно определить следующим образом: представим, что торговец выбрал какой-либо путь; он вышел из первого города и сейчас находится в каком-то городе i . Тогда все пройденное расстояние из начала в город i будем называть промежуточная длина пути. Если исходить из того, что торговец в каждый момент времени будет находиться в каком-то i -ом городе, то всегда можно подсчитать какое расстояние он прошел из начала до этого города, то есть промежуточную длину пути.

Минимальным путем будем называть маршрут, проходящий по всем городам и имеющий минимальную длину.

Мой критерий строится на одном простом утверждении: если промежуточная длина пути больше минимального пути, тогда очевидно следующее:

* промежуточная длина будет расти, когда торговец будет двигаться к конечному городу,
* а значит длина всего пути будет больше длины минимального маршрута.

следовательно такой маршрут можно отбросить.

В данной программе используется следующий критерий: при переходе от одного города к другому рассчитывается промежуточная длина пути, и если она больше текущего минимального пути, то вычисления по данной ветви прекращаются. Таким образом, отсекаются лишние ветви.

Решение данной задачи приводит к перебору возможных вариантов пути, но критерии такого рода могут значительно сократить вычисление и уменьшить время работы программы.

**3 Разработка программы**

Разработка програмы будет происходить на языке программировании С++ в среде разработки Qt Creator 5.0.2 (Community)

Для разработки программы, которая будет решать задачу о Коммивояжере, необходимо написать класс Graph[1], который будет формировать графы, на основании которых будет решаться данная задача.

Класс Graph предоставляет функциональность для создания матрицы смежности, представляющей граф, а также методы для добавления и удаления вершин и ребер. В классе Graph будет написан метод SolveTSP, который будет выводить решение в консоль.

Для создания класса Graph первоначально необходимо написать его конструктор по умолчанию, который будет создавать матрицу смежности 1x1 и инициализировать единственный элемент этой матрицы значением 0. Затем создаются методы для удаления вершин DelNode, а также методы для формирования ребер между вершинами AddEdge и DelEdge. Методом PrintMatrix будет выводиться сама матрицу смежности.

Формируется метод SolveTSP, который представляет собой решение задачи коммивояжера методом полного перебора с использованием рекурсии. Переменные min\_cost и min\_route инициализируются для хранения минимальной стоимости и лучшего маршрута соответственно. Создаются векторы visited и path для отслеживания посещенных вершин и хранения текущего пути. Функция dfs рекурсивно ищет все возможные маршруты, обновляя min\_cost и min\_route при нахождении более выгодного маршрута. Затем вызывается функция dfs с начальной вершиной, и в конце проверяется, было ли найдено решение, и выводится результат.

Далее необходимо создать интерфейс и визуализацию программы, для выполнения задачи будет использован Qt Creator[3].

Для визуализации и интерфейса будет сформирован класс mainwindow, который будет включать в себя методы. Конструктор данного класса инициализирует интерфейс и подключает сигналы и слоты для кнопок и функций приложения. Методы AddNode и DelNode будут добавлять или удалять вершину на графической сцене, а также обновлять матрицу смежности. Методы AddEdge и DelEdge будут при помощи диалогового окна удалять либо добавлять ребро, а также обновлять матрицу смежности. Для методов DelNode, AddEdge и DelEdge были сформированы классы DelEdgeWindow, NewEdgeWindow, DelNodeWindow для корректной их работы в окне программы. Метод UpdateArrows обновляет графические стрелки между вершинами на основе матрицы смежности. Деструктор класса будет освобождать память, выделенную для графической сцены и интерфейса. Метод UpdateNumber будет обновлять порядковый номер вершин в окне программы, т.е. чтобы при удалении вершины ее порядковый номер присваивался новой вершине. Деструктор класса mainwindow удаляет сцены и пользовательский интерфейс, тем самым освобождая память.

Программа работает корректно, интерфейс успешно позволяет добавлять и удалять вершины с ребрами, при вызове функции SolveTSP в консоль выводится верное решение, т.е. минимальный вес пути.

**Заключение**

В результате работы было разработано программное решение, способное находить оптимальный маршрут коммивояжера с использованием метода полного перебора, с учетом оптимизации алгоритма. Полученное решение может быть применено в различных областях, требующих нахождения оптимальных маршрутов, таких как логистика, транспортное планирование, и производственное управление.

**Список использованных источников**

1. Технологии разработки объектно-ориентированных программ на языке С++ : учеб. пособие : в 3 ч. / О.А. Полякова, О.Л. Викентьева.
2. Жасмин Бланшет, Марк Саммерфилд Qt 4: программирование GUI на C++
3. Документация Qt Creator [Электронный ресурс]: <https://doc.qt.io/qtcreator/>

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

// graph.h : header file

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QGraphicsEllipseItem>

#include <QGraphicsItemGroup>

#include <QGraphicsLineItem>

class Graph

{

private:

int SizeMatrix;

int\*\* Matrix;

int CurNumber;

QGraphicsEllipseItem\*\* EllipseMatrix;

QGraphicsItemGroup\*\* GroupMatrix;

vector<QGraphicsItemGroup\*> VectorArrows;

public:

Graph();

int GetSize();

int GetNumber();

void IncreaseNumber();

void NewSizeMatrix(int);

void AddLastNode(QGraphicsEllipseItem\*, QGraphicsItemGroup\*);

void DelNode(int);

void SetDelNumber(int);

int GetLastDelNumber();

int\*\* GetMatrix();

QGraphicsItemGroup\*\* GetGroupMatrix() const;

void NewGroupMatrix(int, QGraphicsItemGroup\*);

void NewEllipseMatrix(int, QGraphicsEllipseItem\*);

void AddEdge(int, int, int) const;

void DelEdge(int, int) const;

void PrintMatrix();

QGraphicsEllipseItem\*\* GetEllipseMatrix();

vector<QGraphicsItemGroup\*> &GetVectorArrows();

void NewVector(vector<QGraphicsItemGroup\*>&);

void SolveTSP();

~Graph();

};

#endif // GRAPH\_H

//graph.cpp

#include "graph.h"

#include <QDebug>

#include <algorithm>

#include <limits>

#include <vector>

Graph::Graph()

{

SizeMatrix = 1;

CurNumber = 1;

Matrix = new int\*[SizeMatrix];

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

{

Matrix[i] = new int [SizeMatrix];

}

Matrix[0][0] = 0;

EllipseMatrix = new QGraphicsEllipseItem\* [SizeMatrix];

EllipseMatrix[0] = nullptr;

GroupMatrix = new QGraphicsItemGroup\* [SizeMatrix];

EllipseMatrix[0] = nullptr;

}

int Graph::GetSize()

{

return SizeMatrix;

}

int Graph::GetNumber()

{

return CurNumber;

}

void Graph::IncreaseNumber()

{

CurNumber += 1;

}

void Graph::NewSizeMatrix(int dif)

{

// Создаем новую матрицу и заполняем значениями из текущей матрицы

int\*\* NewMatrix = new int\*[SizeMatrix + dif]();

for (int i = 0; i < SizeMatrix + dif; i++)

{

NewMatrix[i] = new int[SizeMatrix + dif]();

if (i < SizeMatrix)

{

copy(Matrix[i], Matrix[i] + SizeMatrix, NewMatrix[i]);

// Заполняем нулями новые строки и столбцы

fill(NewMatrix[i] + SizeMatrix, NewMatrix[i] + SizeMatrix + dif, 0);

fill(NewMatrix[i] + SizeMatrix, NewMatrix[i] + SizeMatrix + dif, 0);

}

}

// Удаляем старую матрицу

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

{

delete[] Matrix[i];

}

delete[] Matrix;

// Обновляем матрицу

Matrix = NewMatrix;

// Обновляем массив узлов QGraphicsEllipseItem

QGraphicsEllipseItem\*\* NewEllipseMatrix = new QGraphicsEllipseItem\*[SizeMatrix + dif]();

copy(EllipseMatrix, EllipseMatrix + SizeMatrix, NewEllipseMatrix);

delete[] EllipseMatrix;

EllipseMatrix = NewEllipseMatrix;

// Обновляем массив групп элементов QGraphicsItemGroup

QGraphicsItemGroup\*\* NewGroupMatrix = new QGraphicsItemGroup\*[SizeMatrix + dif]();

copy(GroupMatrix, GroupMatrix + SizeMatrix, NewGroupMatrix);

delete[] GroupMatrix;

GroupMatrix = NewGroupMatrix;

SizeMatrix += dif;

}

void Graph::AddLastNode(QGraphicsEllipseItem\* Node, QGraphicsItemGroup\* group)

{

Matrix[0][SizeMatrix-1] = CurNumber;

Matrix[SizeMatrix-1][0] = CurNumber;

EllipseMatrix[SizeMatrix-1] = Node;

GroupMatrix[SizeMatrix-1] = group;

}

void Graph::DelNode(int num)

{

int CurSize = GetSize();

for (int i = num; i < CurSize-1; ++i)

{

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

{

if (i < num || j < num)

Matrix[i][j] = Matrix[i][j];

else

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

}

}

int\*\* NewMatrix = new int\*[CurSize-1];

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

{

NewMatrix[i] = new int[CurSize-1];

}

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

{

for (int j = 0; j < CurSize - 1; ++j)

{

NewMatrix[i][j] = Matrix[i][j];

}

}

delete[] Matrix;

Matrix = NewMatrix;

QGraphicsEllipseItem\*\* NewEllipseMatrix = new QGraphicsEllipseItem\*[CurSize-1];

QGraphicsItemGroup\*\* NewGroupMatrix = new QGraphicsItemGroup\*[CurSize-1];

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

{

NewEllipseMatrix[i] = EllipseMatrix[i];

NewGroupMatrix[i] = GroupMatrix[i];

}

for (int i = num + 1; i < CurSize; ++i)

{

NewEllipseMatrix[i - 1] = EllipseMatrix[i];

NewGroupMatrix[i - 1] = GroupMatrix[i];

}

delete[] EllipseMatrix;

delete[] GroupMatrix;

EllipseMatrix = NewEllipseMatrix;

GroupMatrix = NewGroupMatrix;

SizeMatrix--;

CurNumber--;

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

{

Matrix[0][i] = i;

Matrix[i][0] = i;

}

}

void Graph::SolveTSP() {

int n = GetSize() - 1;

if (n < 2) {

qDebug() << "Граф должен содержать минимум 2 вершины.";

return;

}

int min\_cost = numeric\_limits<int>::max(); // Инициализация минимальной стоимости

vector<int> min\_route; // Для хранения лучшего маршрута

vector<bool> visited(n + 1, false); // Посещенные вершины

vector<int> path; // Текущий путь

path.reserve(n);

// Функция для рекурсивного поиска маршрута

function<void(int, int)> dfs = [&](int current, int cost) {

path.push\_back(current);

if (path.size() == n && Matrix[current][1] > 0) { // Замыкание цикла

cost += Matrix[current][1];

if (cost < min\_cost) {

min\_cost = cost;

min\_route = path;

}

path.pop\_back();

return;

}

visited[current] = true;

for (int next = 1; next <= n; ++next) {

if (!visited[next] && Matrix[current][next] > 0) {

dfs(next, cost + Matrix[current][next]);

}

}

visited[current] = false;

path.pop\_back();

};

dfs(1, 0);

if (min\_cost == numeric\_limits<int>::max()) {

qDebug() << "Решение не найдено";

} else {

QString result = "Минимальная стоимость: " + QString::number(min\_cost) + "; Маршрут: ";

for (size\_t i = 0; i < min\_route.size(); ++i) {

result += QString::number(min\_route[i]);

if (i < min\_route.size() - 1) result += " -> ";

}

result += " -> 1";

qDebug() << result;

}

}

QGraphicsEllipseItem\*\* Graph::GetEllipseMatrix()

{

return EllipseMatrix;

}

int\*\* Graph::GetMatrix()

{

return Matrix;

}

QGraphicsItemGroup\*\* Graph::GetGroupMatrix() const

{

return GroupMatrix;

}

void Graph::NewGroupMatrix(int index, QGraphicsItemGroup\* group)

{

GroupMatrix[index] = group;

}

void Graph::NewEllipseMatrix(int index, QGraphicsEllipseItem\* node)

{

EllipseMatrix[index] = node;

}

vector<QGraphicsItemGroup\*> &Graph::GetVectorArrows()

{

return VectorArrows;

}

void Graph::NewVector(vector<QGraphicsItemGroup\*>& vect)

{

VectorArrows = vect;

}

void Graph::AddEdge(int from, int to, int weight) const

{

Matrix[to][from] = weight;

Matrix[from][to] = weight;

}

void Graph::DelEdge(int from, int to) const

{

Matrix[to][from] = 0;

Matrix[from][to] = 0;

}

void Graph::PrintMatrix()

{

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

{

QString RowString;

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

{

RowString += QString::number(Matrix[i][j]) + " ";

}

qDebug() << RowString;

}

qDebug() << endl;

}

Graph::~Graph()

{

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

{

delete[] Matrix[i];

delete[] EllipseMatrix[i];

delete[] GroupMatrix[i];

}

delete[] Matrix;

delete[] EllipseMatrix;

delete[] GroupMatrix;

}

//mainwindow.h : header file

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <graph.h>

#include <NewEdgeWindow.h>

#include "ui\_NewEdgeWindow.h"

#include <QMainWindow>

#include <QGraphicsScene>

#include <QMouseEvent>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

public slots:

void AddNode();

void DelNode();

void UpdateNumber();

void AddEdge();

void DelEdge();

void UpdateArrows();

void TspSolve();

void Matrix();

private:

Graph graph;

NewEdgeWindow AddEdgeWindow;

QGraphicsScene \*scene;

QGraphicsView \*PrintGraph;

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

//mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "DelEdgeWindow.h"

#include "ui\_DelEdgeWindow.h"

#include "DelNodeWindow.h"

#include "ui\_DelNodeWindow.h"

#include "graph.h"

#include "NewEdgeWindow.h"

#include "ui\_NewEdgeWindow.h"

#include <QDebug>

#include <QGraphicsEllipseItem>

#include <QGraphicsTextItem>

#include <QGraphicsSceneMouseEvent>

#include <QMouseEvent>

#include <QGraphicsItemGroup>

#include <QtMath>

#include <QPolygonF>

#include <QPlainTextEdit>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, AddEdgeWindow(graph)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

PrintGraph = ui->PrintGraph;

scene = new QGraphicsScene;

PrintGraph -> setScene(scene);

connect(ui->NodeBtn, SIGNAL(clicked()), this, SLOT(AddNode()));

connect(ui->EdgeBtn, SIGNAL(clicked()), this, SLOT(AddEdge()));

connect(ui->DelNodeBtn, SIGNAL(clicked()), this, SLOT(DelNode()));

connect(ui->DelEdgeBtn, SIGNAL(clicked()), this, SLOT(DelEdge()));

connect(ui->TspBtn, SIGNAL(clicked()),this, SLOT(TspSolve()));

connect(ui->MatrixBtn, SIGNAL(clicked()), this, SLOT(Matrix()));

}

void MainWindow::TspSolve()

{

graph.SolveTSP();

}

void MainWindow::Matrix()

{

graph.PrintMatrix();

}

void MainWindow::AddNode()

{

QGraphicsEllipseItem \*Node = scene->addEllipse(0,0,75,75, QPen(Qt::black), QBrush(Qt::darkCyan));

QString NodeNumber = QString::number(graph.GetNumber());

QGraphicsTextItem \*TextItem = scene->addText(NodeNumber);

TextItem->setPos(Node->boundingRect().center().x() - TextItem->boundingRect().width() / 2,

Node->boundingRect().center().y() - TextItem->boundingRect().height() / 2);

QList<QGraphicsItem\*> items;

items << Node << TextItem;

QGraphicsItemGroup \*group = scene->createItemGroup(items);

group->setFlag(QGraphicsItem::ItemIsMovable, true);

graph.NewSizeMatrix(1);

graph.AddLastNode(Node, group);

graph.IncreaseNumber();

graph.PrintMatrix();

}

void MainWindow::DelNode()

{

DelNodeWindow DelNodeWindow(graph, scene, PrintGraph, graph.GetGroupMatrix());

DelNodeWindow.setModal(true);

DelNodeWindow.exec();

UpdateNumber();

UpdateArrows();

graph.PrintMatrix();

}

void MainWindow::UpdateNumber()

{

int size = graph.GetSize();

QGraphicsItemGroup\*\* groups = graph.GetGroupMatrix();

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

{

foreach (QGraphicsItem \*item, groups[i]->childItems())

{

QGraphicsTextItem \*textItem = qgraphicsitem\_cast<QGraphicsTextItem\*>(item);

if (textItem)

{

textItem->setPlainText(QString::number(i));

}

}

}

scene->update();

}

void MainWindow::AddEdge()

{

NewEdgeWindow EdgeWindow(graph);

EdgeWindow.setModal(true);

EdgeWindow.exec();

UpdateArrows();

graph.PrintMatrix();

}

void MainWindow::DelEdge()

{

DelEdgeWindow DelEdge(graph);

DelEdge.setModal(true);

DelEdge.exec();

UpdateArrows();

graph.PrintMatrix();

}

void MainWindow::UpdateArrows()

{

int\*\* matrix = graph.GetMatrix();

QGraphicsItemGroup\*\* matrixOfGroups = graph.GetGroupMatrix();

std::vector<QGraphicsItemGroup\*> vectorOfArrows = graph.GetVectorArrows();

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

{

scene->removeItem(vectorOfArrows[i]); // Удаляем группы со сцены

delete vectorOfArrows[i]; // Освобождаем память

}

vectorOfArrows.clear();

for (int i = 1; i < graph.GetSize(); i++)

{

for (int j = i + 1; j < graph.GetSize(); j++)

{

if (matrix[i][j] > 0 && matrix[i][j] == matrix[j][i])

{

QPointF center1 = matrixOfGroups[i]->mapToScene(matrixOfGroups[i]->boundingRect().center());

QPointF center2 = matrixOfGroups[j]->mapToScene(matrixOfGroups[j]->boundingRect().center());

qDebug() << center1;

qDebug() << center2;

qreal angle = qAtan2(center2.y() - center1.y(), center2.x() - center1.x());

QPointF newStart(center1.x() + 32 \* qCos(angle), center1.y() + 32 \* qSin(angle));

QPointF newEnd(center2.x() - 32 \* qCos(angle), center2.y() - 32 \* qSin(angle));

QGraphicsLineItem \*line = new QGraphicsLineItem();

line->setLine(QLineF(newStart, newEnd));

QPointF textPos((center1.x() + center2.x()) / 2, (center1.y() + center2.y()) / 2);

QGraphicsTextItem\* textItem = scene->addText(QString::number(matrix[i][j]));

textItem->setPos(textPos);

QList<QGraphicsItem\*> items;

items << line << textItem;

QGraphicsItemGroup \*group = scene->createItemGroup(items);

vectorOfArrows.push\_back(group);

}

}

}

graph.NewVector(vectorOfArrows);

}

MainWindow::~MainWindow()

{

delete scene;

delete ui;

}

//DelEdgeWindow.h : header file

#ifndef DELEDGEWINDOW\_H

#define DELEDGEWINDOW\_H

#include <QDialog>

#include "graph.h"

namespace Ui

{

class DelEdgeWindow;

}

class DelEdgeWindow : public QDialog

{

Q\_OBJECT

public:

explicit DelEdgeWindow(const Graph&graph, QWidget \*parent = nullptr);

~DelEdgeWindow();

public slots:

void DelEdge();

private:

const Graph&m\_graph;

Ui::DelEdgeWindow \*ui;

};

#endif // DELEDGEWINDOW\_H

//DelEdgeWindow.cpp

#include "DelEdgeWindow.h"

#include "ui\_DelEdgeWindow.h"

DelEdgeWindow::DelEdgeWindow(const Graph&graph, QWidget \*parent)

: QDialog(parent), m\_graph(graph), ui(new Ui::DelEdgeWindow)

{

ui->setupUi(this);

connect(ui->DelEdgeBtn, SIGNAL(clicked()), this, SLOT(DelEdge()));

}

DelEdgeWindow::~DelEdgeWindow()

{

delete ui;

}

void DelEdgeWindow::DelEdge()

{

QString FromLine = ui->FromLine->text();

QString ToLine = ui->ToLine->text();

int from = FromLine.toInt();

int to = ToLine.toInt();

m\_graph.DelEdge(from,to);

close();

}

//DelNodeWindow.h : header file

#ifndef DELNODEWINDOW\_H

#define DELNODEWINDOW\_H

#include <QDialog>

#include <QGraphicsView>

#include "graph.h"

namespace Ui

{

class DelNodeWindow;

}

class DelNodeWindow : public QDialog

{

Q\_OBJECT

public:

explicit DelNodeWindow(Graph&graph, QGraphicsScene\* scene, QGraphicsView\* PrintGraph, QGraphicsItemGroup\*\* MatrixOfGroups, QWidget \*parent = nullptr);

~DelNodeWindow();

public slots:

void DelNode();

private:

Graph& m\_graph;

QGraphicsScene\* m\_scene;

QGraphicsItemGroup\*\* m\_MatrixOfGroups;

QGraphicsView\* m\_PrintGraph;

Ui::DelNodeWindow \*ui;

};

#endif // DELNODEWINDOW\_H

//DelNodeWindow.cpp

#include "DelNodeWindow.h"

#include "ui\_DelNodeWindow.h"

DelNodeWindow::DelNodeWindow(Graph& graph, QGraphicsScene\* scene, QGraphicsView\* PrintGraph, QGraphicsItemGroup\*\* MatrixOfGroups, QWidget \*parent)

: QDialog(parent), m\_graph(graph), m\_scene(scene), m\_MatrixOfGroups(MatrixOfGroups), m\_PrintGraph(PrintGraph), ui(new Ui::DelNodeWindow)

{

ui->setupUi(this);

connect(ui->DelNodeBtn, SIGNAL(clicked()), this, SLOT(DelNode()));

}

DelNodeWindow::~DelNodeWindow()

{

delete ui;

}

void DelNodeWindow::DelNode()

{

QString NodeNumber = ui->NumberLine->text();

int number = NodeNumber.toInt();

m\_scene->removeItem(m\_MatrixOfGroups[number]);

delete m\_MatrixOfGroups[number];

m\_MatrixOfGroups[number] = nullptr;

m\_graph.DelNode(number);

close();

}

//NewEdgeWindow.h : header file

#ifndef NEWEDGEWINDOW\_H

#define NEWEDGEWINDOW\_H

#include <QDialog>

#include "graph.h"

namespace Ui

{

class NewEdgeWindow;

}

class NewEdgeWindow : public QDialog

{

Q\_OBJECT

public:

explicit NewEdgeWindow(const Graph&graph, QWidget \*par = nullptr);

~NewEdgeWindow();

private slots:

void NewEdge();

private:

const Graph& m\_graph;

Ui::NewEdgeWindow \*ui;

};

#endif // NEWEDGEWINDOW\_H

//NewEdgeWindow.cpp

#include "NewEdgeWindow.h"

#include "ui\_NewEdgeWindow.h"

#include <QGraphicsScene>

NewEdgeWindow::NewEdgeWindow(const Graph&graph, QWidget \*parent)

: QDialog(parent), m\_graph(graph), ui(new Ui::NewEdgeWindow)

{

ui->setupUi(this);

connect(ui->NewEdgeButton, SIGNAL(clicked()), this, SLOT(NewEdge()));

}

NewEdgeWindow::~NewEdgeWindow()

{

delete ui;

}

void NewEdgeWindow::NewEdge()

{

QString FromLine = ui->FromLine->text();

QString ToLine = ui->ToLine->text();

QString WeightLine = ui->WeightLine->text();

int from = FromLine.toInt();

int to = ToLine.toInt();

int weight = WeightLine.toInt();

m\_graph.AddEdge(from, to, weight);

close();

}

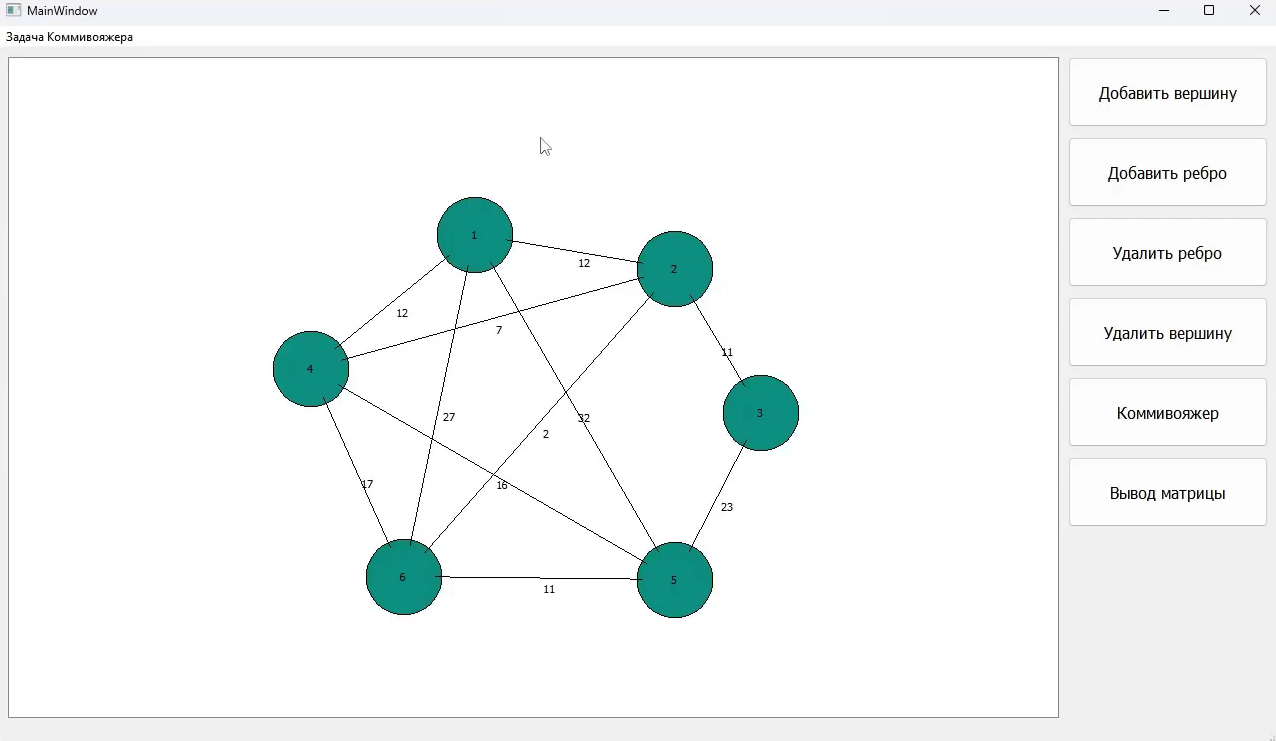
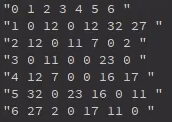
****

Рисунок 1 – Формирование графа для задачи



Рисунк 2 – Вывод матрицы смежности в консоль



Рисунок 3 – Вывод решения для задачи о коммивояжере

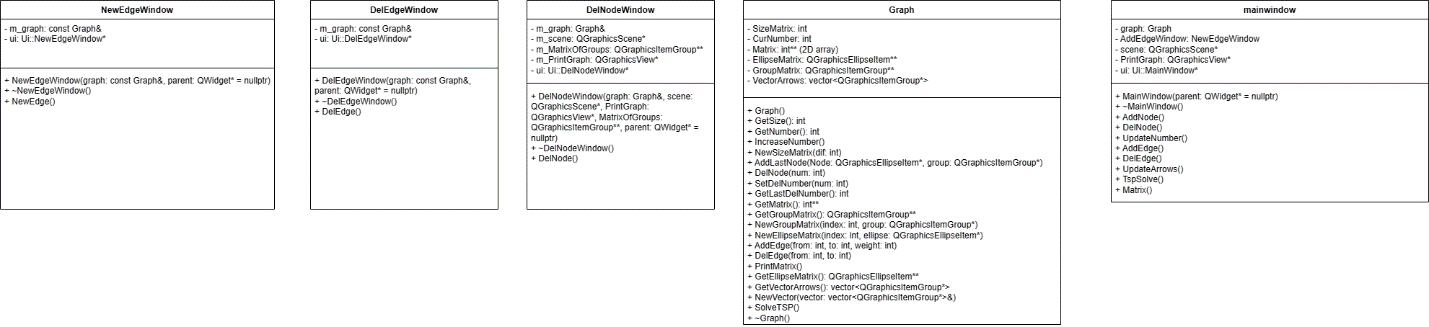


Рисунок 4 – UML диаграмма программы