Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

**“Задача коммивояжёра”**

Выполнили:   
студенты группы РИС-23-1б   
Головизнин А. В.

Карпов И. В.

Проверила:   
доцент кафедры ИТАС   
Полякова О. А.

Пермь, 2024 г.

1) Постановка задачи:

На плоскости расположены N городов, заданы расстояния между каждым из них. Требуется найти маршрут минимальной длины с посещением каждого города ровно один раз и с возвращением в исходный город. Задачу необходимо решить методом ветвей и границ, реализовать интерфейс

2) Анализ задачи:

* Создадим класс Graph, в котором будут выполняться все основные действия для решения задачи: вывод матрицы смежности, вставка и удаление вершин, добавление и изменение веса рёбер, изменение индексов вершин, непосредственно нахождения решения задачи коммивояжёра
* В функции, которая находит решение задачи коммивояжёра происходят следующие действия:
  + - В начале функции выполняется получение матрицы смежности графа и проверка на возможность выполнения функции (проверяется, что количество вершин больше 2 и каждая вершина имеет как минимум два ребра)
    - Затем исходная матрица смежности обрабатывается для подготовки к дальнейшему выполнению алгоритма
    - Далее происходит процесс выбора ребер графа на основе минимальных значений в матрице смежности с последующим их исключением из рассмотрения
    - Выбор оптимальных ребер осуществляется на основе вычисления суммы минимальных значений для каждой вершины и выбора пары вершин с максимальной суммой
    - После построения оптимального маршрута формируется строка, содержащая последовательность вершин маршрута и общую сумму расстояний
    - Выводится результат в виде строки с последовательностью вершин маршрута и общей суммой расстояний или сообщение о невозможности решения задачи, если условия не выполнены

3) Код на C++

Заголовочные файлы

(edge.h)

#ifndef EDGE\_H

#define EDGE\_H

#include "graph.h"

#include "vertex.h"

#include <QMainWindow>

#include <QGraphicsItem>

#include <QVector>

#include <QPointF>

#include <QGraphicsView>

#include <QWidget>

#include <QStyleOptionGraphicsItem>

#include <QKeyEvent>

*class* **Vertex**;

*class* **Edge**;

*class* **Graph**;

*class* **Edge** : *public* QGraphicsItem {

*public*:

**Edge**(Vertex \*source, Vertex \*dest, double weight);

~***Edge***();

void **adjust**();

Vertex \***sourceVertex**() *const*;

Vertex \***destVertex**() *const*;

double **getWeight**();

void **setWeight**(double weight);

QString **getColor**();

void **setColor**(QString color);

*protected*:

QRectF ***boundingRect***() *const* *override*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget) *override*;

*private*:

Vertex \*source, \*dest;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize = 7;

QString color;

double weight;

};

#endif *//* *EDGE\_H*

(graph.h)

#ifndef GRAPH\_H

#define GRAPH\_H

#include <vertex.h>

#include <edge.h>

#include <QMainWindow>

#include <QGraphicsItem>

#include <QVector>

#include <QStack>

#include <QQueue>

#include <QPointF>

#include <QGraphicsView>

#include <QWidget>

#include <QStyleOptionGraphicsItem>

#include <QKeyEvent>

#include <QPushButton>

#include <QHBoxLayout>

#include <QStyle>

#include <QDialog>

#include <QLineEdit>

#include <QLabel>

#include <QWidget>

#include <QTableWidget>

#include <QApplication>

#include <QGroupBox>

#include <QToolButton>

#include <QMenu>

#include <QDir>

#include <QSaveFile>

#include <QFileDialog>

#include <windows.h>

#include <conio.h>

#include <process.h>

#include <TlHelp32.h>

*using* *namespace* std;

*class* **vertex**;

*class* **Edge**;

*class* **Graph**;

*class* Graph : *public* QGraphicsView {

Q\_OBJECT

*public*:

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

void **itemMoved**();

int **getVertexNum**();

void **runDFS**(int temp, bool\* visited);

void **runBFS**(int index, bool\* visited, QQueue<int>\* bfsQueue);

void **runDijkstra**(int vertexIndex);

void **createDFSWindow**();

int \*\* **GetAdjMatrix**();

void **RemoveAdjMatrix**(int \*\* AdjMatrix);

void **runTSP**();

*protected*:

void ***timerEvent***(QTimerEvent \*event) *override*;

void ***drawBackground***(QPainter \*painter, *const* QRectF &rect) *override*;

void **createTabWidget**(*const* QRectF &rect);

void **createMenuBar**();

void **scaleView**(qreal scaleFactor);

void **updateIndex**();

void **insertVertex**();

void **setVertexPos**(int vertexIndex, double x, double y);

void **eraseVertex**(int vertexIndex);

void **insertEdge**(int source, int dest, double weight);

void **updateWeight**(int source, int dest, double weight);

void **eraseEdge**(int source, int dest);

bool **checkAdjacent**(int source, int dest);

int **getDegree**(int vertexIndex);

void **clear**();

void **createEraseVertexWindow**();

void **createAddEdgeWindow**();

void **createEraseEdgeWindow**();

void **createUpdateWeightWindow**();

void **createBFSWindow**();

void **createDijkstraWindow**();

void **createAdjMatrixWindow**();

void **createFunctionWindow**();

void **createTSPWindow**();

*private* slots:

void **eraseVertexSlot**();

void **addEdgeSlot**();

void **eraseEdgeSlot**();

void **updateWeightSlot**();

void **StartSelectedFunction**();

*private*:

unsigned int edgeNum=0;

int vertexNum;

bool connected;

QVector<Vertex \*> vertexList;

QGraphicsScene \*scene;

int timerId = 0;

QWidget \*window;

QLineEdit \*input1;

QLineEdit \*input2;

QLineEdit \*input3;

QLineEdit \*input5;

QString curFile;

};

#endif *//* *GRAPH\_H*

#endif // GRAPH\_H

(mainwindow.h)

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

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*

(vertex.h)

#ifndef VERTEX\_H

#define VERTEX\_H

#include "graph.h"

#include <QMainWindow>

#include <QGraphicsItem>

#include <QVector>

#include <QPointF>

#include <QGraphicsView>

#include <QWidget>

#include <QStyleOptionGraphicsItem>

#include <QKeyEvent>

*class* **vertex**;

*class* **Edge**;

*class* **Graph**;

*class* Vertex : *public* QGraphicsItem {

*public*:

**Vertex**(Graph \*graph, int index);

~***Vertex***();

void **addEdge**(Edge\* edge);

QVector<Edge \*> **getEdges**() *const*;

void **calculateForces**();

bool **advancePosition**();

QString **getColor**();

void **setColor**(QString color);

int **getDegree**();

int **getIndex**();

void **setIndex**(int index);

void **setPosition**(double xvel, double yvel);

void **insertEdge**(int dest, double weight);

bool **pathExist**(int dest);

void **updateWeight**(int dest, double weight);

void **eraseEdge**(int dest);

void **clearEdge**();

void **changeEdgeColor**(int dest);

QPointF **getPos**();

QRectF ***boundingRect***() *const* *override*;

QPainterPath ***shape***() *const* *override*;

void ***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*widget) *override*;

*protected*:

QVariant ***itemChange***(GraphicsItemChange change, *const* QVariant &value) *override*;

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event) *override*;

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event) *override*;

*private*:

int vertexIndex;

int degree;

QVector<Edge \*> edgeList;

QPointF newPos;

Graph \*graph;

QString color;

};

#endif *//* *VERTEX\_H*

Основные файлы

(edge.cpp)

#include "vertex.h"

#include "edge.h"

#include "graph.h"

#include <QtMath>

#define M\_PI 3.1415926

Edge::**Edge**(Vertex \*source, Vertex \*dest, double weight) {

*this*->source = source;*//начало* *дуги*

*this*->dest = dest;*//конец* *дуги*

setAcceptedMouseButtons(Qt::*NoButton*);

source->addEdge(*this*);*//добавление* *дуги*

*this*->weight = weight;*//устанавливаем* *вес*

color = "white";

}

Edge::~***Edge***(){

}

Vertex \*Edge::**sourceVertex**() *const*{

*return* source;*//начало*

}

Vertex \*Edge::**destVertex**() *const*{

*return* dest;*//конец*

}

void Edge::**adjust**() {*//регулировка* *параметров* *для* *дуги*

*if*(!source || !dest) {

*return*;

}

QLineF line(mapFromItem(source, 0, 0), mapFromItem(dest, 0, 0));*//задание* *объекта* *для* *дуги*

qreal length = line.length();*//длина*

prepareGeometryChange();

*//установка* *нужных* *размеров* *для* *линии*

*if* (length > qreal(20.)) {

QPointF edgeOffset((line.dx() \* 10) / length, (line.dy() \* 10) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

}

*else* {

sourcePoint = destPoint = line.p1();

}

}

QRectF Edge::***boundingRect***() *const* {

*if*(!source || !dest){

*return* QRectF();

}

qreal penWidth = 1;

qreal extra = (penWidth + arrowSize) / 2.0;

*return* QRectF(sourcePoint, QSizeF(destPoint.x() - sourcePoint.x(), destPoint.y() - sourcePoint.y())).normalized().adjusted(-extra, -extra, extra, extra);

}

void Edge::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*, QWidget \*){

*if*(!source || !dest){

*return*;

}

QLineF line(sourcePoint, destPoint);

*if*(qFuzzyCompare(line.length(), qreal(0.))) {

*return*;

}

painter->setBrush(Qt::*black*);

painter->setPen(QPen(Qt::*black*, 1, Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*));

painter->drawLine(line);*//рисование* *линии*

painter->setBrush(Qt::*red*);

painter->setPen(QPen(Qt::*red*, 1, Qt::*SolidLine*, Qt::*RoundCap*, Qt::*RoundJoin*));

painter->drawText(QRectF(QPointF(sourcePoint.x() - 15, sourcePoint.y() - 15), QPointF(destPoint.x() + 15, destPoint.y() + 15)), Qt::*AlignCenter*, QString::number(weight));*//вывод* *веса*

}

double Edge::**getWeight**(){

*return* weight;

}

void Edge::**setWeight**(double weight){

*this*->weight = weight;

}

QString Edge::**getColor**(){

*return* color;

}

void Edge::**setColor**(QString color){

*this* -> color = color;

qApp->processEvents();

update();

}

(graph.cpp)

#include "graph.h"

#include <QTextStream>

#include <QMessageBox>

#include <map>

QString str;*//для* *функций*

Graph::**Graph**(QWidget \*parent) : QGraphicsView(*parent*) {

scene = *new* QGraphicsScene(*this*);

scene->setItemIndexMethod(QGraphicsScene::*NoIndex*);

scene->setSceneRect(-500, -450, 900, 900);

setScene(scene);

setCacheMode(*CacheBackground*);

setViewportUpdateMode(*BoundingRectViewportUpdate*);

setRenderHint(QPainter::*Antialiasing*);

setTransformationAnchor(*AnchorUnderMouse*);

scale(qreal(0.8), qreal(0.8));

setMinimumSize(800, 900);

}

void Graph::**itemMoved**() {

*if*(!timerId) {

timerId = startTimer(1000 / 25);

}

}

void Graph::***timerEvent***(QTimerEvent \*event) {

Q\_UNUSED(event);

*const* QList<QGraphicsItem \*> items = scene -> items();

*for*(Vertex \*temp : as\_const(vertexList)){

temp->calculateForces();

}

bool itemsMoved = *false*;

*for*(Vertex \*temp : as\_const(vertexList)){

*if*(temp->advancePosition()){

itemsMoved = *true*;

}

}

*if*(!itemsMoved){

killTimer(timerId);

timerId = 0;

}

}

void Graph::***drawBackground***(QPainter \*painter, *const* QRectF &rect){

Q\_UNUSED(rect);

QRectF sceneRect = *this*->sceneRect();

QRectF rightShadow(sceneRect.right(), sceneRect.top() + 5, 5, sceneRect.height());

QRectF bottomShadow(sceneRect.left() + 5, sceneRect.bottom(), sceneRect.width(), 5);

*if* (rightShadow.intersects(rect) || rightShadow.contains(rect)){

painter->fillRect(rightShadow, Qt::*darkGray*);

}

*if* (bottomShadow.intersects(rect) || bottomShadow.contains(rect)){

painter->fillRect(bottomShadow, Qt::*darkGray*);

}

QLinearGradient gradient(sceneRect.topLeft(), sceneRect.bottomRight());

gradient.setColorAt(0, Qt::*white*);

gradient.setColorAt(1, Qt::*lightGray*);

painter->fillRect(rect.intersected(sceneRect), gradient);

painter->setBrush(Qt::*NoBrush*);

painter->drawRect(sceneRect);

QRectF textRect(sceneRect.left() + 4, sceneRect.top() + sceneRect.height()/9 + 35, sceneRect.width() - 4, 20);

QFont font = painter->font();

font.setBold(*true*);

font.setPointSize(14);

painter->setFont(font);

painter->setPen(Qt::*lightGray*);

createTabWidget(rect);

}

void Graph::**createTabWidget**(*const* QRectF &rect) {

Q\_UNUSED(rect);

QRectF sceneRect = *this*->sceneRect();

QWidget \*CommandsWidget = *new* QWidget;

CommandsWidget->setGeometry(sceneRect.left() + 1, sceneRect.top() + 3, sceneRect.width()/6, sceneRect.height() - 2);

QVBoxLayout \*vertexTabLayout = *new* QVBoxLayout(*CommandsWidget*);

QFont textfont = QFont();

textfont.setPointSize(12);

*//кнопка* *добавления* *вершины*

QPushButton \*addVertexButton = *new* QPushButton();

addVertexButton->setText("Добавить вершину");

addVertexButton->setFont(textfont);

addVertexButton->setMaximumSize(QSize(300, 120));

vertexTabLayout->addWidget(addVertexButton);

connect(addVertexButton, &QPushButton::clicked, *this*, &Graph::insertVertex);

*//кнопка* *удаления* *вершины*

QPushButton \*eraseVertexButton = *new* QPushButton();

eraseVertexButton->setText("Удалить вершину");

eraseVertexButton->setFont(textfont);

textfont.setPointSize(12);

eraseVertexButton->setMaximumSize(QSize(300, 120));

vertexTabLayout->addWidget(eraseVertexButton);

connect(eraseVertexButton, &QPushButton::clicked, *this*, &Graph::createEraseVertexWindow);

*//кнопка* *добавления* *ребра*

QPushButton \*addEdgeButton = *new* QPushButton();

addEdgeButton->setText("Добавить ребро");

addEdgeButton->setFont(textfont);

addEdgeButton->setMaximumSize(QSize(300, 120));

connect(addEdgeButton, &QPushButton::clicked, *this*, &Graph::createAddEdgeWindow);

vertexTabLayout->addWidget(addEdgeButton);

*//кнопка* *для* *обновления* *веса* *вершины*

QPushButton \*updateWeightButton = *new* QPushButton();

updateWeightButton->setText("Изменить вес");

updateWeightButton->setFont(textfont);

updateWeightButton->setMaximumSize(QSize(300, 120));

connect(updateWeightButton, &QPushButton::clicked, *this*, &Graph::createUpdateWeightWindow);

vertexTabLayout->addWidget(updateWeightButton);

*//удаление* *вершины*

QPushButton \*eraseEdgeButton = *new* QPushButton();

eraseEdgeButton->setText("Удалить ребро");

eraseEdgeButton->setFont(textfont);

eraseEdgeButton->setMaximumSize(QSize(300, 120));

connect(eraseEdgeButton, &QPushButton::clicked, *this*, &Graph::createEraseEdgeWindow);

vertexTabLayout->addWidget(eraseEdgeButton);

*//кнопка* *функции* *графа*

QPushButton \*FuncButton = *new* QPushButton();

FuncButton->setText("Функции");

FuncButton->setFont(textfont);

FuncButton->setMaximumSize(QSize(300, 120));

connect(FuncButton, &QPushButton::clicked, *this*, &Graph::createFunctionWindow);

vertexTabLayout->addWidget(FuncButton);

*//кнопка* *удаления* *графа*

QPushButton \*clearButton = *new* QPushButton();

clearButton->setText("Удалить граф");

clearButton->setFont(textfont);

clearButton->setMaximumSize(QSize(300, 120));

connect(clearButton, &QPushButton::clicked, *this*, &Graph::clear);

vertexTabLayout->addWidget(clearButton);

*//кнопка* *вывода* *матрицы* *смежности*

QPushButton \*AdjMatrixButton = *new* QPushButton();

AdjMatrixButton->setText("Матрица смежности");

AdjMatrixButton->setFont(textfont);

AdjMatrixButton->setMaximumSize(QSize(300, 120));

connect(AdjMatrixButton, &QPushButton::clicked, *this*, &Graph::createAdjMatrixWindow);

vertexTabLayout->addWidget(AdjMatrixButton);

scene->addWidget(*CommandsWidget*);

}

void Graph::**createAdjMatrixWindow**() {*//вывод* *матрицы* *смежности*

int rows = vertexList.size() + 1;

int cols = vertexList.size() + 1;

QTableWidget \*AdjMatrixWindow = *new* QTableWidget(rows, cols);

AdjMatrixWindow->setColumnWidth(0, 30);

AdjMatrixWindow->setRowHeight(0, 30);

*for* (int i = 0; i < getVertexNum(); i++) {

AdjMatrixWindow->setRowHeight(1+i, 30);

AdjMatrixWindow->setItem(0, i+1, *new* QTableWidgetItem*(*QString*::number(i))*);

AdjMatrixWindow->item(0, i+1)->setTextAlignment(Qt::*AlignRight* | Qt::*AlignVCenter*);

}

*for*(int i = 0; i < getVertexNum(); i++) {

AdjMatrixWindow->setItem(1+i, 0, *new* QTableWidgetItem*(*QString*::number(i))*);

AdjMatrixWindow->item(1+i, 0)->setTextAlignment(Qt::*AlignRight* | Qt::*AlignVCenter*);

*for* (Edge \*edge : vertexList[i]->getEdges()) {

int pos = edge->destVertex()->getIndex();

AdjMatrixWindow->setItem(1+i, pos+1, *new* QTableWidgetItem(QString::number(edge->getWeight())));

AdjMatrixWindow->item(1+i, pos+1)->setTextAlignment(Qt::AlignRight | Qt::AlignVCenter);

}

AdjMatrixWindow->setColumnWidth(i+1, 30);

}

AdjMatrixWindow->setWindowTitle("Матрица смежности");

AdjMatrixWindow->show();

}

int Graph::**getVertexNum**() {*//получение* *размера* *списка* *вершин*

*return* vertexList.size();

}

void Graph::**scaleView**(qreal scaleFactor) {

qreal factor = transform().scale(scaleFactor, scaleFactor).mapRect(QRectF(0, 0, 1, 1)).width();

*if*(factor < 0.07 || factor > 100){

*return*;

}

scale(scaleFactor, scaleFactor);

}

void Graph::**insertVertex**() {*//вставка* *вершины*

int vertexIndex = vertexList.size();

vertexList.append(*new* Vertex(*this*, vertexIndex));

vertexList[vertexIndex]->setPos(0, 0);

scene->addItem(vertexList[vertexIndex]);

vertexNum++;

}

void Graph::**updateIndex**() { *//* *Изменение* *индексов* *вершин*

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

vertexList[i]->setIndex(i);

}

}

void Graph::**eraseVertex**(int vertexIndex) {*//удаление* *вершины*

Vertex \*temp = vertexList[vertexIndex];

*for*(Edge \*edge : temp->getEdges()) {*//удаление* *всех* *дуг,* *связанных* *с* *вершиной*

eraseEdge(vertexIndex, edge->destVertex()->getIndex());

}

temp->clearEdge();

scene->removeItem(temp);

edgeNum -= vertexList[vertexIndex]->getEdges().size()\*2;*//изменяем* *количество* *дуг*

vertexList.erase(vertexList.begin() + vertexIndex);

updateIndex();

vertexNum--;*//изменяем* *количество* *вершин*

}

void Graph::**insertEdge**(int source, int dest, double weight){*//Вставка* *дуги*

*if*(checkAdjacent(source, dest)) { *//если* *уже* *соединены* *-* *выход*

*return*;

}

*//создаём* *двунаправленную* *стрелку*

scene->addItem(*new* Edge(vertexList[source], vertexList[dest], weight));*//убрать* *вес*

scene->addItem(*new* Edge(vertexList[dest], vertexList[source], weight));

edgeNum += 2;

}

void Graph::**updateWeight**(int source, int dest, double weight){*//изменение* *веса*

*if* (!checkAdjacent(source, dest)){*//если* *не* *соединены* *-* *выход*

*return*;

}

vertexList[source]->updateWeight(dest, weight);

vertexList[dest]->updateWeight(source, weight);

}

void Graph::**eraseEdge**(int source, int dest) {*//удаление* *дуги*

*if* (!checkAdjacent(source, dest)) {*//если* *вершины* *не* *соединены* *-* *выход*

*return*;

}

*//удаление* *для* *начальной* *вершины*

*for* (Edge \*edge : vertexList[source]->getEdges()) {

*if*(edge->destVertex()->getIndex() == dest) {

vertexList[source]->eraseEdge(dest);

scene->removeItem(edge);

}

}

*//удаление* *для* *конечной* *вершины*

*for* (Edge \*edge : vertexList[dest]->getEdges()){

*if*(edge->destVertex()->getIndex() == source){

vertexList[dest]->eraseEdge(source);

scene->removeItem(edge);

}

}

edgeNum -= 2;

}

bool Graph::**checkAdjacent**(int source, int dest) {*//Проверка,* *соединены* *ли* *вершины*

*return* vertexList[source]->pathExist(dest);

}

int Graph::**getDegree**(int vertexIndex) {

*return* vertexList[vertexIndex]->getDegree();

}

void Graph::**clear**() {*//полное* *удаление* *графа*

*while* (!vertexList.empty()) {

eraseVertex(0);

}

}

int \*\* Graph::**GetAdjMatrix**() {

int \*\*AdjMatrix = *new* int\*[getVertexNum()] {};

*for* (int i = 0; i < getVertexNum(); i++) {

AdjMatrix[i] = *new* int[getVertexNum()]{};

}

*for* (Vertex\* i : vertexList) {

*for* (Edge\* j: i->getEdges()) {

AdjMatrix[i->getIndex()][j->destVertex()->getIndex()] = j->getWeight();

}

}

*return* AdjMatrix;

}

void Graph::**RemoveAdjMatrix**(int \*\* AdjMatrix) {

*for* (int i = 0; i < getVertexNum(); i++) {

*delete*[] AdjMatrix[i];

}

*delete*[] AdjMatrix;

}

void Graph::**createEraseVertexWindow**() {*//Окно* *для* *удаления* *выбранной* *вершины*

window = *new* QWidget;

input1 = *new* QLineEdit;

QLabel \*label = *new* QLabel;

label->setFrameStyle(QFrame::*Box* | QFrame::*Plain*);

QPushButton \*okButton = *new* QPushButton(tr("OK"));

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(tr(*"Введите номер вершины:"*))*, 0, 0);

layout->addWidget(*input1*, 0, 1);

layout->addWidget(okButton, 1, 1, Qt::*AlignRight*);

layout->setSizeConstraint(QLayout::*SetFixedSize*);

window->setLayout(layout);

window->setWindowTitle("Удаление вершины");

connect(okButton, &QPushButton::clicked, *this*, &Graph::eraseVertexSlot);

window->show();

}

void Graph::**eraseVertexSlot**() {*//удаление* *выбранной* *вершины*

int srcIndex = input1->text().toInt();

window->close();

*if* (srcIndex >= vertexList.size()){

*return*;

}

eraseVertex(srcIndex);

}

void Graph::**createAddEdgeWindow**() {*//окно* *для* *добавления* *дуги*

window = *new* QWidget;

input1 = *new* QLineEdit;

input2 = *new* QLineEdit;

input3 = *new* QLineEdit;

QLabel \*label = *new* QLabel;

label->setFrameStyle(QFrame::*Box* | QFrame::*Plain*);

QPushButton \*okButton = *new* QPushButton(tr("OK"));

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(tr(*"Введите начальную вершину:"*))*, 0, 0);

layout->addWidget(*input1*, 0, 1);

layout->addWidget(*new* QLabel*(tr(*"Введите конечную вершину:"*))*, 1, 0);

layout->addWidget(*input2*, 1, 1);

layout->addWidget(*new* QLabel*(tr(*"Введите вес дуги:"*))*, 2, 0);

layout->addWidget(*input3*, 2, 1);

layout->addWidget(okButton, 3, 1, Qt::*AlignRight*);

layout->setSizeConstraint(QLayout::*SetFixedSize*);

window->setLayout(layout);

window->setWindowTitle("Добавление дуги");

connect(okButton, &QPushButton::clicked, *this*, &Graph::addEdgeSlot);

window->show();

}

void Graph::**addEdgeSlot**() {*//добавление* *дуги*

int srcIndex = input1->text().toInt();

int destIndex = input2->text().toInt();

double edgeWeight = input3->text().toDouble();

*if* (srcIndex >= vertexList.size() || destIndex >= vertexList.size() || edgeWeight <= 0){

*return*;

}

insertEdge(srcIndex, destIndex, edgeWeight);

}

void Graph::**createEraseEdgeWindow**() {*//окно* *для* *удаления* *выбранной* *дуги*

window = *new* QWidget;

input1 = *new* QLineEdit;

input2 = *new* QLineEdit;

QLabel \*label = *new* QLabel;

label->setFrameStyle(QFrame::*Box* | QFrame::*Plain*);

QPushButton \*okButton = *new* QPushButton(tr("OK"));

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(tr(*"Номер начальной вершины:"*))*, 0, 0);

layout->addWidget(*input1*, 0, 1);

layout->addWidget(*new* QLabel*(tr(*"Номер конечной вершины:"*))*, 1, 0);

layout->addWidget(*input2*, 1, 1);

layout->addWidget(okButton, 2, 1, Qt::*AlignRight*);

layout->setSizeConstraint(QLayout::*SetFixedSize*);

window->setLayout(layout);

window->setWindowTitle("Удаление дуги");

connect(okButton, &QPushButton::clicked, *this*, &Graph::eraseEdgeSlot);

window->show();

}

void Graph::**eraseEdgeSlot**() {*//удаление* *выбранной* *дуги*

int srcIndex = input1->text().toInt();

int destIndex = input2->text().toInt();

window->close();

*if* (srcIndex >= vertexList.size() || destIndex >= vertexList.size()){

*return*;

}

eraseEdge(srcIndex, destIndex);

}

void Graph::**createUpdateWeightWindow**() { *//окно* *для* *изменения* *веса*

window = *new* QWidget;

input1 = *new* QLineEdit;

input2 = *new* QLineEdit;

input3 = *new* QLineEdit;

QLabel \*label = *new* QLabel;

label->setFrameStyle(QFrame::*Box* | QFrame::*Plain*);

QPushButton \*okButton = *new* QPushButton(tr("OK"));

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(tr(*"Введите номер начальной вершины:"*))*, 0, 0);

layout->addWidget(*input1*, 0, 1);

layout->addWidget(*new* QLabel*(tr(*"Введите номер конечной вершины:"*))*, 1, 0);

layout->addWidget(*input2*, 1, 1);

layout->addWidget(*new* QLabel*(tr(*"Новый вес:"*))*, 2, 0);

layout->addWidget(*input3*, 2, 1);

layout->addWidget(okButton, 3, 1, Qt::*AlignRight*);

layout->setSizeConstraint(QLayout::*SetFixedSize*);

window->setLayout(layout);

window->setWindowTitle("Изменение веса дуги");

connect(okButton, &QPushButton::clicked, *this*, &Graph::updateWeightSlot);

window->show();

}

void Graph::**updateWeightSlot**() {*//изменение* *веса*

int srcIndex = input1->text().toInt();

int destIndex = input2->text().toInt();

double edgeWeight = input3->text().toDouble();

window->close();

*if*(srcIndex >= vertexList.size() || destIndex >= vertexList.size() || edgeWeight <= 0){

*return*;

}

updateWeight(srcIndex, destIndex, edgeWeight);

}

void Graph::**createFunctionWindow**() { *//окно* *функций*

window = *new* QWidget;

input1 = *new* QLineEdit;

input5 = *new* QLineEdit;

QLabel \*label = *new* QLabel;

label->setText("1. DFS\n2. BFS\n3. Алгоритм Дейкстры\n4. Задача Коммивояжёра\n");

QPushButton \*okButton = *new* QPushButton(tr("OK"));

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*label*, 0, 0);

layout->addWidget(*new* QLabel*(tr(*"Номер функции:"*))*, 1, 0);

layout->addWidget(*input5*, 1, 1);

layout->addWidget(*new* QLabel*(tr(*"Номер начальной вершины:"*))*, 2, 0);

layout->addWidget(*input1*, 2, 1);

layout->addWidget(okButton, 3, 0, Qt::*AlignRight*);

layout->setSizeConstraint(QLayout::*SetFixedSize*);

window->setLayout(layout);

window->setWindowTitle("Выбор функции");

connect(okButton, &QPushButton::clicked, *this*, &Graph::StartSelectedFunction);

window->show();

}

void Graph::**runDFS**(int temp, bool\* visited) {

*if* (!visited[temp]) {

visited[temp] = *true*;

str = str + "Вершина " + QString::number(temp) + " посещена\n";

qDebug() << str;

}

Vertex \*tempVertex = vertexList[temp];

*for* (Edge \*tempEdge : tempVertex->getEdges()) {*//проход* *по* *соседям*

*if* (!visited[tempEdge->destVertex()->getIndex()]) {

runDFS(tempEdge->destVertex()->getIndex(), visited);

}

}

}

void Graph::**runBFS**(int index, bool\* visited, QQueue<int>\* bfsQueue) {

*if* (visited[index] == *false*) {

bfsQueue->push\_back(index);

str += "Вершина " + QString::number(index) + " обработана\n";

visited[index] = *true*;

}

Vertex \*tempVertex = vertexList[index];

bfsQueue->pop\_front();

*for* (Edge \*tempEdge : tempVertex->getEdges()) {

*if*(!visited[tempEdge->destVertex()->getIndex()]) {

str += "Вершина " + QString::number(tempEdge ->destVertex() ->getIndex()) + " обработана\n";

visited[tempEdge->destVertex()->getIndex()] = *true*;

bfsQueue->push\_back(tempEdge->destVertex()->getIndex());

}

}

*if* (!bfsQueue->empty()) {

runBFS(bfsQueue->front(), *visited*, *bfsQueue*);

}

}

void Graph::**runDijkstra**(int vertexIndex) {

int\*\* AdjMatrix = GetAdjMatrix();

QVector <int> distance(getVertexNum(), INT\_MAX);

distance[vertexIndex] = 0;

map<int, int> m;

m[0] = vertexIndex;

*while*(!m.empty()) {

int m\_lenght = (\*m.begin()).first;

int m\_vertex = (\*m.begin()).second;

m.erase(m.begin());

*if* (m\_lenght <= distance[m\_vertex]) {

distance[m\_vertex] = m\_lenght;

*for* (int i = 0; i < getVertexNum(); i++) {

*if* (distance[i] > distance[m\_vertex] + AdjMatrix[m\_vertex][i] && AdjMatrix[m\_vertex][i] != 0) {

distance[i] = distance[m\_vertex] + AdjMatrix[m\_vertex][i];

m[distance[i]] = i;

}

}

}

}

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

*if* (distance[i] != INT\_MAX && i != vertexIndex) {

str += "Расстояние от вершины " + QString::number(vertexIndex) + " до вершины " + QString::number(i) + " cоставляет " + QString::number(distance[i]) + '\n';

}

*else* *if* (i != vertexIndex) {

str += "Вершины " + QString::number(vertexIndex) + " и " + QString::number(i) + " не соединены\n";

}

}

RemoveAdjMatrix(*AdjMatrix*);

}

void Graph::**runTSP**() {

int \*\*AdjMatrix = GetAdjMatrix();*//получение* *матрицы* *смежности*

*//проверка* *на* *возможность* *выполнения* *функции*

bool FlagToContinue = (getVertexNum() > 2);

*for* (int i = 0; i < getVertexNum() && FlagToContinue; i++) {

FlagToContinue = (vertexList[i]->getEdges().size() > 1);

}

*if* (FlagToContinue) {

int \*\* OrigianalAdjMatrix = GetAdjMatrix();

int m = INT\_MAX, Sum = 0;

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

*if* (AdjMatrix[i][j] == 0) {

AdjMatrix[i][j] = m;

}

}

}

QMap<int,int> roads = {};

*for* (int z = 0; z < getVertexNum(); z++) {

int maxi = 0, maxj = 0;

int\* di = *new* int[getVertexNum()];

int\* dj = *new* int[getVertexNum()];

int maxScore = 0;

*for* (int i = 0; i < getVertexNum(); i++) {

di[i] = dj[i] = m;

}

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

*if* (AdjMatrix[i][j] < di[i]) {

di[i] = AdjMatrix[i][j];

}

}

}

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

*if* (AdjMatrix[i][j] != m) {

AdjMatrix[i][j] -= di[i];

}

}

}

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

*if* (AdjMatrix[j][i] < dj[i]) {

dj[i] = AdjMatrix[j][i];

}

}

}

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

*if* (AdjMatrix[j][i] != m) {

AdjMatrix[j][i] -= dj[i];

}

}

}

*for* (int i = 0; i < getVertexNum(); i++) {

*for* (int j = 0; j < getVertexNum(); j++) {

int imin = m, jmin = m;

*if* (AdjMatrix[i][j] == 0) {

*for* (int k = 0; k < getVertexNum(); k++) {

*if* (AdjMatrix[k][j] < imin && i != k) {

imin = AdjMatrix[k][j];

}

*if* (AdjMatrix[i][k] < jmin) {

jmin = AdjMatrix[i][k];

}

}

*if* (maxScore < imin + jmin) {

maxScore = imin + jmin;

maxi = i;

maxj = j;

}

}

}

}

roads[maxj] = maxi;

AdjMatrix[maxj][maxi] = m;

Sum += OrigianalAdjMatrix[maxi][maxj];

*for* (int i = 0; i < getVertexNum(); i++) {

AdjMatrix[i][maxj] = m;

AdjMatrix[maxi][i] = m;

}

}

int i = 0;

str += QString::number(i);

*while*(FlagToContinue) {

str += "-> " + QString::number(roads[i]);

i = roads[i];

FlagToContinue = (i != 0);

}

str += " Сумма = " + QString::number(Sum);

RemoveAdjMatrix(*OrigianalAdjMatrix*);

}

*else* {

str += "Решение задачи Коммивояжёра невозможно!\nКаждая вершина графа должна иметь как минимум два ребра!";

}

RemoveAdjMatrix(*AdjMatrix*);

}

void Graph::**StartSelectedFunction**() {

int index = input1->text().toInt();

int NumOfFunction = input5->text().toInt();

*if* (index >= 0 && index < vertexList.size() && NumOfFunction >= 0 && NumOfFunction < 5) {

*enum* **Functions** {

*DFS* = 1, *BFS* = 2, *Dijkstra* = 3, *Travel* = 4

};

*switch* (NumOfFunction) {

*case*(Functions::*DFS*): {

bool\* visited = *new* bool[vertexList.size()];

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

visited[i] = 0;

}

runDFS(index, *visited*);

createDFSWindow();

*delete*[] visited;

*break*;

}

*case*(Functions::*BFS*): {

bool\* visited = *new* bool[vertexList.size()];

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

visited[i] = 0;

}

runBFS(index, *visited*, *new* QQueue*<*int*>*);

createBFSWindow();

*delete*[] visited;

*break*;

}

*case*(Functions::*Dijkstra*): {

runDijkstra(index);

createDijkstraWindow();

*break*;

}

*case*(Functions::*Travel*): {

runTSP();

createTSPWindow();

*break*;

}

}

}

}

void Graph::**createDFSWindow**() { *//Окно* *DFS* *функции*

window = *new* QWidget;

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(str)*, 0, 0);

str = "";

window->setLayout(layout);

window->setWindowTitle("DFS");

window->show();

}

void Graph::**createBFSWindow**() { *//Окно* *BFS* *функции*

window = *new* QWidget;

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(str)*, 0, 0);

str = "";

window->setLayout(layout);

window->setWindowTitle("BFS");

window->show();

}

void Graph::**createDijkstraWindow**() { *//Окно* *Dijkstra* *функции*

window = *new* QWidget;

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(str)*, 0, 0);

str = "";

window->setLayout(layout);

window->setWindowTitle("Алгоритм Дейкстры");

window->show();

}

void Graph::**createTSPWindow**() { *//Окно* *TSP* *функции*

window = *new* QWidget;

QGridLayout \*layout = *new* QGridLayout;

layout->addWidget(*new* QLabel*(str)*, 0, 0);

str = "";

window->setLayout(layout);

window->setWindowTitle("Решение задачи Коммивояжёра");

window->show();

}

(main.cpp)

#include "mainwindow.h"

#include "graph.h"

#include <QApplication>

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

{

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

Graph w;

w.show();

*return* a.exec();

}

(mainwindow.cpp)

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

}

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

{

*delete* ui;

}

(vertex.cpp)

#include "vertex.h"

#include "edge.h"

#include "graph.h"

#include <QVector>

Vertex::**Vertex**(Graph \*graph, int index) : graph(graph) {

setFlag(*ItemIsMovable*);

setFlag(*ItemSendsGeometryChanges*);

setCacheMode(*DeviceCoordinateCache*);

setZValue(-1);

vertexIndex = index;

degree = 0;

color = "white";

}

Vertex::~***Vertex***(){

clearEdge();

degree = 0;

}

QString Vertex::**getColor**(){

*return* *this*->color;

}

void Vertex::**setColor**(QString color){

*this*->color = color;

qApp->processEvents();

update();

}

void Vertex::**addEdge**(Edge \*edge){

edgeList << edge;

edge->adjust();

degree++;

}

QVector<Edge \*> Vertex::getEdges() *const*{

*return* edgeList;

}

void Vertex::**calculateForces**(){

*if*(!scene() || scene()->mouseGrabberItem() == *this*) {

newPos = pos();

*return*;

}

qreal xvel = 0;

qreal yvel = 0;

*const* QList<QGraphicsItem \*> items = scene()->items();

*for*(QGraphicsItem \*item : items) {

Vertex \*temp = qgraphicsitem\_cast<Vertex \*>(item);

*if*(!temp){

*continue*;

}

QPointF vec = mapToItem(temp, 0, 0);

qreal dx = vec.x();

qreal dy = vec.y();

double l = 2.0 \* (dx \* dx + dy \* dy);

*if*(l > 0 && l < 600){

xvel += (dx \* 150.0) / l;

yvel += (dy \* 150.0) / l;

}

}

*if*(qAbs(xvel) < 0.1 && qAbs(yvel) < 0.1){

xvel = yvel = 0;

}

QRectF sceneRect = scene()->sceneRect();

newPos = pos() + QPointF(xvel, yvel);

newPos.setX(qMin(qMax(newPos.x(), sceneRect.left() + 10), sceneRect.right() - 10));

newPos.setY(qMin(qMax(newPos.y(), sceneRect.top() + 145), sceneRect.bottom() - 10));

}

bool Vertex::**advancePosition**() {

*if*(newPos == pos()){

*return* *false*;

}

setPos(newPos);

*return* *true*;

}

int Vertex::**getDegree**(){

*return* degree;

}

int Vertex::**getIndex**() {

*return* vertexIndex;

}

void Vertex::**setIndex**(int index) {

vertexIndex = index;

qApp->processEvents();

update();

}

bool Vertex::**pathExist**(int dest) {*//проверка,* *является* *ли* *вершина* *конечной* *точкой*

*for* (Edge \*temp : edgeList) {

*if* (temp->destVertex()->getIndex() == dest) {

*return* *true*;

}

}

*return* *false*;

}

void Vertex::**setPosition**(double xvel, double yvel) {

QRectF sceneRect = scene()->sceneRect();

newPos = pos() + QPointF(xvel, yvel);

newPos.setX(qMin(qMax(newPos.x(), sceneRect.left() + 10), sceneRect.right() - 10));

newPos.setY(qMin(qMax(newPos.y(), sceneRect.top() + 145), sceneRect.bottom() - 10));

}

void Vertex::**insertEdge**(int dest, double weight) {*//добавление* *дуги*

*const* QList<QGraphicsItem \*> items = scene()->items();

*for*(QGraphicsItem \*item : items){

Vertex \*temp = qgraphicsitem\_cast<Vertex \*>(*item*);

*if*(!temp){

*continue*;

}

*if*(temp->getIndex() == dest) {

Edge \*edge = *new* Edge(*this*, *temp*, weight);

edgeList.append(edge);

*return*;

}

}

}

void Vertex::**updateWeight**(int dest, double weight) {*//изменение* *веса*

*for*(Edge \*temp : edgeList){

*if*(temp->destVertex()->getIndex() == dest) {

temp->setWeight(weight);

}

}

}

void Vertex::**eraseEdge**(int dest) {*//удаление* *конечной* *точки* *для* *дуг*

*if* (!pathExist(dest)) {*//вершина* *не* *является* *конечной* *точкой* *-* *выход*

*return*;

}

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

*if* (edgeList[i]->destVertex()->getIndex() == dest) {

edgeList.erase(edgeList.begin() + i);

}

}

degree--;

}

void Vertex::**clearEdge**() {

edgeList.clear();

degree = 0;

}

void Vertex::**changeEdgeColor**(int dest) { *//изменение* *цвета* *дуги*

*for* (Edge \*edge : edgeList){

*if* (edge->destVertex()->getIndex() == dest) {

edge->setColor("black");

}

}

}

QPointF Vertex::**getPos**() {

*return* newPos;

}

QRectF Vertex::***boundingRect***() *const* {

qreal adjust = 2;

*return* QRectF(-10 - adjust, -10 - adjust, 23 + adjust, 23 + adjust);

}

QPainterPath Vertex::***shape***() *const* {

QPainterPath path;

path.addEllipse(-10, -10, 20, 20);

*return* path;

}

void Vertex::***paint***(QPainter \*painter, *const* QStyleOptionGraphicsItem \*option, QWidget \*) {

painter->setPen(Qt::*NoPen*);

painter->setBrush(Qt::*darkGray*);

painter->drawEllipse(-7, -7, 20, 20);

QRadialGradient gradient(-3, -3, 10);

prepareGeometryChange();

painter->setBrush(QBrush(Qt::*white*, Qt::*SolidPattern*));

painter->setPen(QPen(Qt::*black*, 0));

painter->drawEllipse(-10, -10, 20, 20);*//рисование* *круга*

prepareGeometryChange();

QString index = QString::number(vertexIndex);

painter->drawText(QRect(-10, -10, 20, 20), Qt::*AlignCenter*, index);*//рисование* *номера* *вершины*

}

QVariant Vertex::***itemChange***(GraphicsItemChange change, *const* QVariant &value) {

*switch*(change){

*case* ItemPositionHasChanged:

*for*(Edge \*edge : qAsConst(edgeList)){

graph->itemMoved();

edge->adjust();

*for* (Edge \*reverseEdge : edge->destVertex()->getEdges()) {

reverseEdge->adjust();

}

}

graph->itemMoved();

*break*;

*default*:

*break*;

};

*return* QGraphicsItem::itemChange(change, value);

}

void Vertex::***mousePressEvent***(QGraphicsSceneMouseEvent \*event){

update();

QGraphicsItem::mousePressEvent(event);

}

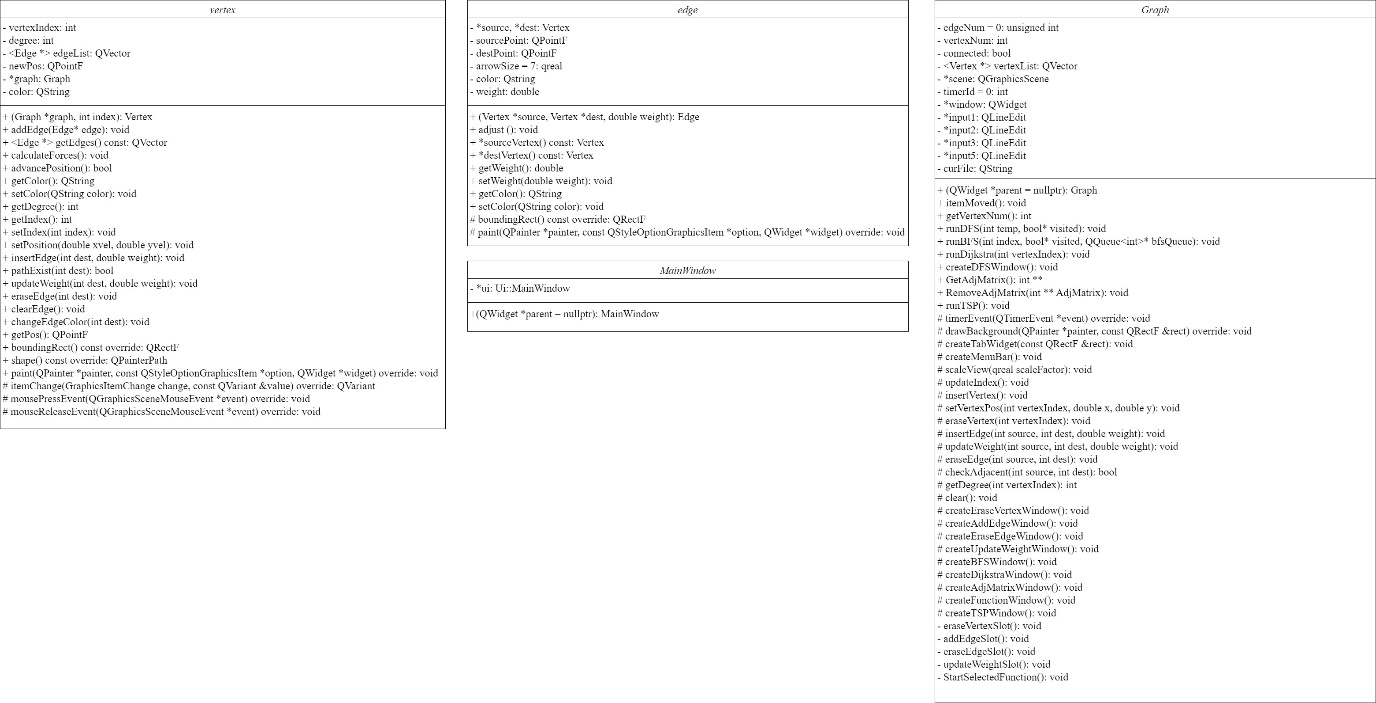
void Vertex::***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event){

update();

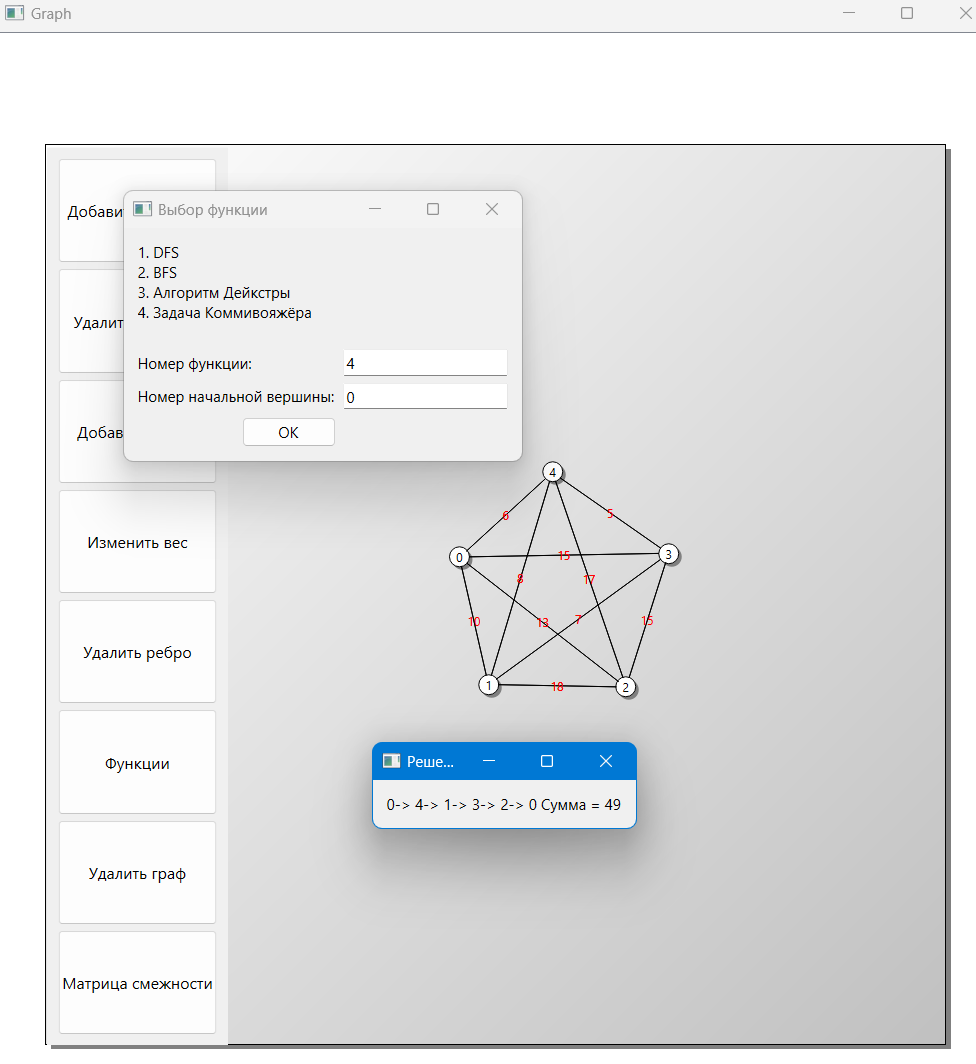
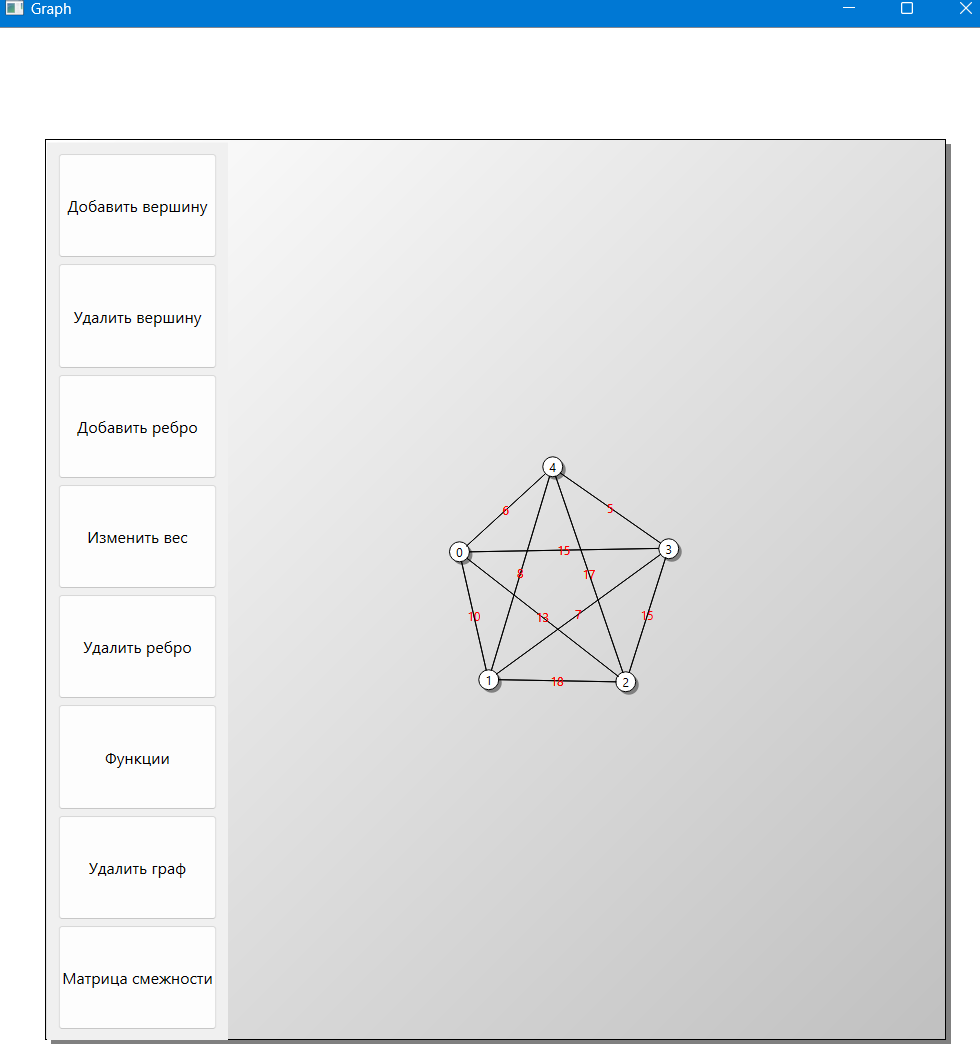
QGraphicsItem::mouseReleaseEvent(event);

}

4) UML – диаграмма



5) Работа программы



6) github

<https://github.com/Oklikcream/PNRPU/tree/main/Лабораторны%202%20семестр%202%20половина/GraphMaster>