**Пермский Национальный Исследовательский Политехнический Университет**

Творческая работа:

«Задача Коммивояжера»

*Выполнил:*

*Ившин Максим Сергеевич РИС-23- 2б*

*Коротаев Александр Дмитриевич РИС-23-3Б*

*Проверила:*

*доцент кафедры ИТАС О.А. Полякова*

2024

**Цель работы**

Получить практические навыки работы с графами

**Постановка задачи**

1) Реализовать алгоритмы для собственного двунаправленного

графа, имеющего не менее 6 вершин. Алгоритмы:

1. Обход в ширину.

2. Обход в глубину.

3. Алгоритм Дейкстры.

2) Разработать пользовательский интерфейс

3) Визуализация графа с использованием любой доступной

графической библиотеки (SFML, SDL, OpenGL и подобных)

4) Необходимо реализовать функции для редактирования графа:

- Создание новой вершины.

- Удаление вершины.

- Добавление и удаление ребра.

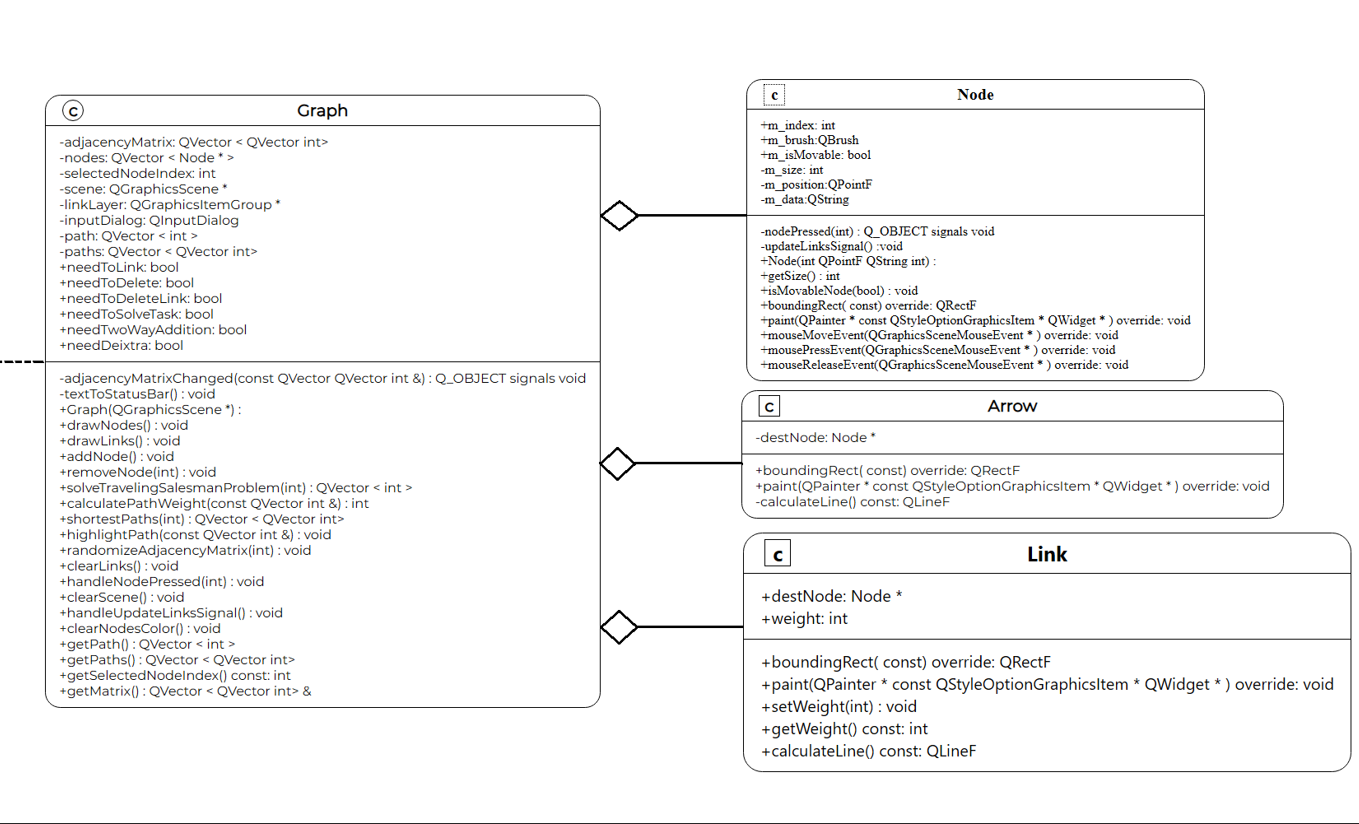
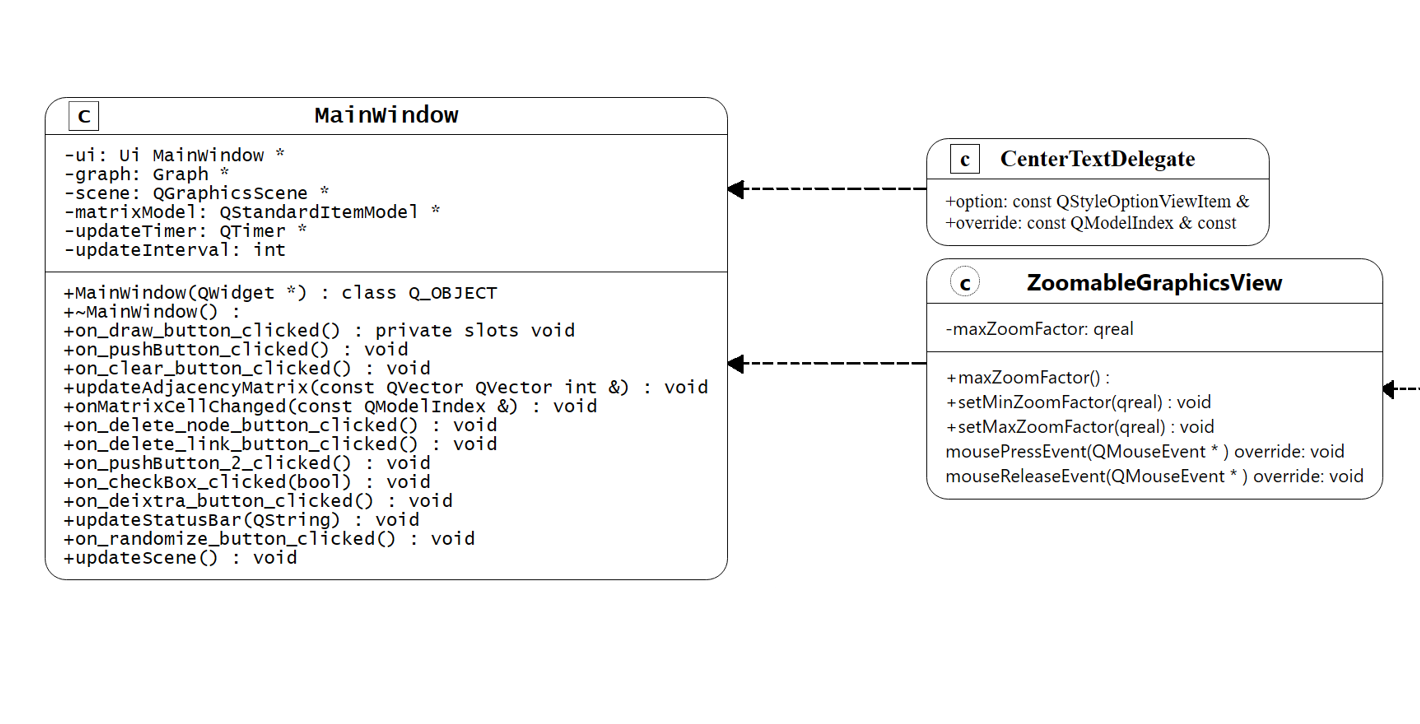
- Редактирование весов ребер.

- Редактирование матрицы смежности (или инцидентности в

зависимости от реализации).

- Реализовать вывод графа.

**UML Диаграммы**



**Код на C++**

Arrow.h

#ifndef ARROW\_H

#define ARROW\_H

#include <QGraphicsItem>

#include <QPainter>

#include <math.h>

#include "Node.h"

*class* **Arrow** : *public* QGraphicsItem

{

*public*:

**Arrow**(Node\* sourceNode, Node\* destNode)

{

*this*->sourceNode = sourceNode;

*this*->destNode = destNode;

}

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

{

*return* QRectF(QPointF(0, 0), QSizeF(100, 20)).normalized();

}

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

{

Q\_UNUSED(option)

Q\_UNUSED(widget)

QPen pen;

pen.setWidth(2);

painter->setPen(pen);

*//* *получение* *начальных* *координат*

QLineF line = calculateLine();

double angle = atan2(line.dy(), line.dx());

double arrowSize = 10;

QPointF arrowP1 = line.pointAt(0);

QPointF arrowP2 = line.pointAt(0);

int radius = destNode->getSize() / 2;

qreal delta\_y = sin(angle) \* radius;

qreal delta\_x = cos(angle) \* radius;

arrowP1 += QPointF{delta\_x, delta\_y};

arrowP2 += QPointF{delta\_x\*1.5, delta\_y\*1.5};

*//* *расчет* *координат* *для* *стрелки*

QLineF arrowLine1(arrowP1, arrowP2 + QPointF(arrowSize \* 1 \* cos(angle + M\_PI\_4), arrowSize \* 1 \* sin(angle + M\_PI\_4)));

QLineF arrowLine2(arrowP1, arrowP2 + QPointF(arrowSize \* 1 \* cos(angle - M\_PI\_4), arrowSize \* 1 \* sin(angle - M\_PI\_4)));

painter->drawLine(arrowLine1);

painter->drawLine(arrowLine2);

}

*private*:

Node\* sourceNode;

Node\* destNode;

QLineF **calculateLine**() *const*

{

QPointF sourcePos = sourceNode->mapToScene(sourceNode->*boundingRect*().center());

QPointF destPos = destNode->mapToScene(destNode->*boundingRect*().center());

*return* QLineF(sourcePos, destPos);

}

};

#endif *//* *ARROW\_H*

Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QVector>

#include <QQueue>

#include <QGraphicsScene>

#include <QMessageBox>

#include <QTimer>

#include <QThread>

#include <QInputDialog>

#include <QString>

#include <QDebug>

#include "Node.h"

#include "arrow.h"

#include "link.h"

*class* **Graph** : *public* QObject

{

Q\_OBJECT

signals:

void **adjacencyMatrixChanged**(*const* QVector<QVector<int>>& adjacencyMatrix);

void **textToStatusBar**(QString);

*private*:

*//* *Создание* *матрицы* *смежности*

QVector<QVector<int>> adjacencyMatrix;

*//* *вектор* *всех* *узлов*

QVector<Node\*> nodes;

*//* *индекс* *выделенного* *узла*

int selectedNodeIndex = -1;

*//* *сцена* *для* *отрисовки*

QGraphicsScene \*scene;

*//* *Новый* *слой* *для* *связей*

QGraphicsItemGroup\* linkLayer;

*//* *диалоговое* *окно*

QInputDialog inputDialog;

*//* *Ответ* *на* *задачу* *коммивояжера*

QVector<int> path;

*//* *результат* *алгоритмов* *обхода*

QVector<QVector<int>> paths;

*public*:

*//* *флаг* *для* *добавления* *новых* *связей* *между* *узлами*

bool needToLink = *false*;

*//* *флаг* *для* *удаления* *узла*

bool needToDelete = *false*;

*//* *удаление* *связи*

bool needToDeleteLink = *false*;

*//* *комивояжер*

bool needToSolveTask = *false*;

*//* *Двусвязное* *добавление*

bool needTwoWayAddition = *false*;

*//* *Алгоритм* *дейкстры*

bool needDeixtra = *false*;

*//* *Алгоритм* *Флойда*

bool needFloid = *false*;

*//* *Обход* *в* *глубину*

bool needInDeep = *false*;

*//* *Обход* *в* *ширину*

bool needInWidth = *false*;

*public*:

**Graph**(QGraphicsScene \*new\_scene)

{

scene = new\_scene;

*//* *Создаем* *новый* *слой* *для* *связей*

linkLayer = *new* QGraphicsItemGroup();

*//* *Добавляем* *его* *на* *сцену*

scene->addItem(*linkLayer*);

}

QVector<int> **getPath**(){*return* path;}

QVector<QVector<int>> **getPaths**(){*return* paths;}

int **getSelectedNodeIndex**() *const* {*return* selectedNodeIndex;}

QVector<QVector<int>>& **getMatrix**() {*return* adjacencyMatrix;}

*public*:

void **turn\_off\_buttons**()

{

QVector<bool\*> all\_buttons = { &needToLink, &needToDelete, &needToDeleteLink, &needToSolveTask, &needTwoWayAddition,

&needDeixtra, &needInDeep, &needInWidth, &needFloid };

*for* (bool\* flag : all\_buttons)

{

\*flag = *false*;

}

}

*//* *отрисовка* *узлов*

void **drawNodes**()

{

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

{

*if* (nodes[i]->scene() != scene)

{

scene->addItem(*nodes[i]*);

}

nodes[i]->update();

}

}

*//* *отрисовка* *связей*

void **drawLinks**()

{

*//* *Удаляем* *все* *связи* *из* *слоя*

clearLinks();

drawNodes();

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

{

*for* (int j = 0; j < adjacencyMatrix[i].size(); ++j)

{

*if* (adjacencyMatrix[i][j] > 0)

{

*//* *Добавление* *связи* *с* *весом* *на* *слой*

Link\* weightLink = *new* Link(*nodes[i]*, *nodes[j]*, adjacencyMatrix[i][j]);

linkLayer->addToGroup(*weightLink*);

*//* *Добавление* *стрелки* *на* *слой*

Arrow\* arrow = *new* Arrow(*nodes[i]*, *nodes[j]*);

linkLayer->addToGroup(*arrow*);

}

}

}

scene->update();

}

*//* *добавление* *узла* *(рандомно)*

void **addNode**()

{

*//* *Ввод* *данных*

bool confirm;

QString text = inputDialog.getText(*nullptr*, "Введите данные в узел",

"Данные:", QLineEdit::*Normal*,

"", *&confirm*);

*//* *При* *отмене* *\* *пустой* *строке* *\* *не* *число*

*if* ( !confirm */\*||* *text* *==* *""* *||* *!text.toInt()* *\*/*)

{

*return*;

}

*//* *создание* *узла* *и* *рандомизация* *его* *расположения*

QPointF position(qrand() % 400, qrand() % 400);

Node\* node = *new* Node(nodes.size(), position, text, 60);

node->setPos(position);

nodes.append(node);

*//* *нажатие* *на* *узел*

connect(node, &Node::nodePressed, *this*, &Graph::handleNodePressed);

*//* *перемещение* *узла*

connect(node, &Node::updateLinksSignal, *this*, &Graph::handleUpdateLinksSignal);

*//* *Добавляем* *новую* *строку* *и* *столбец* *в* *матрицу* *смежности*

*for* (int i = 0; i < nodes.size() - 1; ++i)

{

adjacencyMatrix[i].append(0);

}

QVector<int> newRow(nodes.size(), 0);

adjacencyMatrix.append(newRow);

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

emit adjacencyMatrixChanged(adjacencyMatrix);

}

*//* *Метод* *для* *удаления* *узла*

void **removeNode**(int index)

{

*//* *Проверка* *на* *корректность* *индекса*

*if* (index < 0 || index >= nodes.size())

{

*return*;

}

*//* *Удаление* *узла* *из* *сцены* *и* *вектора* *узлов*

scene->removeItem(*nodes[index]*);

*delete* nodes[index];

nodes.erase(nodes.begin() + index);

*//* *Обновление* *индексов* *узлов*

*for* (int i = index; i < nodes.size(); ++i)

{

nodes[i]->m\_index = i;

}

*//* *Обновление* *матрицы* *смежности*

adjacencyMatrix.erase(adjacencyMatrix.begin() + index); *//* *Удаление* *строки*

*for* (*auto*& row : adjacencyMatrix)

{

row.erase(row.begin() + index); *//* *Удаление* *столбца*

}

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

emit adjacencyMatrixChanged(adjacencyMatrix);

}

*//* *Решение* *задачи* *коммивояжера*

QVector<int> **solveTravelingSalesmanProblem**(int startIndex)

{

*//* *Возвращаем* *пустой* *вектор,* *если* *граф* *пуст* *или* *индекс* *некорректен*

*if* (nodes.isEmpty() || startIndex < 0 || startIndex >= nodes.size())

*return* QVector<int>();

QVector<bool> visited(nodes.size(), *false*);

QVector<int> path;

path.reserve(nodes.size());

*//* *Начинаем* *с* *указанного* *узла*

int currentNode = startIndex;

visited[currentNode] = *true*;

path.append(currentNode);

*//* *Пока* *не* *посетим* *все* *узлы*

*while* (path.size() < nodes.size())

{

int nextNode = -1;

int minDistance = INT\_MAX;

*//* *Ищем* *ближайший* *непосещенный* *узел*

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

{

*if* (!visited[i] && adjacencyMatrix[currentNode][i] > 0 && adjacencyMatrix[currentNode][i] < minDistance)

{

minDistance = adjacencyMatrix[currentNode][i];

nextNode = i;

}

}

*if* (nextNode == -1)

*break*; *//* *Если* *не* *удалось* *найти* *следующий* *узел,* *выходим* *из* *цикла*

*//* *Добавляем* *следующий* *узел* *в* *путь* *и* *отмечаем* *его* *как* *посещенный*

path.append(nextNode);

visited[nextNode] = *true*;

currentNode = nextNode;

}

*//* *Возвращаемся* *в* *начальный* *узел*

path.append(startIndex);

*return* path;

}

int **calculatePathWeight**(*const* QVector<int>& path)

{

int totalWeight = 0;

*for* (int i = 0; i < path.size() - 1; ++i)

{

int fromNode = path[i];

int toNode = path[i + 1];

totalWeight += adjacencyMatrix[toNode][fromNode];

}

*return* totalWeight;

}

QVector<QVector<int>> **shortestPaths**(int startIndex)

{

int size = nodes.size();

QVector<QVector<int>> paths(size, QVector<int>());

QVector<int> distances(size, std::numeric\_limits<int>::max());

QVector<bool> visited(size, *false*);

distances[startIndex] = 0;

*for* (int count = 0; count < size - 1; ++count)

{

int minDistance = std::numeric\_limits<int>::max();

int minIndex = -1;

*//* *Выбираем* *узел* *с* *минимальным* *расстоянием*

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

{

*if* (!visited[i] && distances[i] <= minDistance)

{

minDistance = distances[i];

minIndex = i;

}

}

*//* *узел* *с* *минимальным* *расстоянием* *среди* *непосещенных* *узлов.*

int u = minIndex;

visited[u] = *true*;

*//* *Обновляем* *расстояния* *до* *соседей* *выбранного* *узла*

*for* (int v = 0; v < size; ++v)

{

*if* (!visited[v] && adjacencyMatrix[v][u] && distances[u] != std::numeric\_limits<int>::max() &&

distances[u] + adjacencyMatrix[v][u] < distances[v])

{

distances[v] = distances[u] + adjacencyMatrix[v][u];

paths[v] = paths[u]; *//* *Копируем* *путь* *до* *u*

paths[v].append(v); *//* *Добавляем* *узел* *v* *в* *путь*

}

}

}

*//* *Устанавливаем* *пустой* *путь* *для* *начального* *узла*

paths[startIndex].clear();

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

{

paths[i].push\_front(startIndex);

}

*return* paths;

}

*//* *Визуализация* *обхода*

void **highlightPath**(*const* QVector<int>& path)

{

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

{

QTimer::singleShot( i\*1000, [*this*, path, i]()

{

int prev\_index = -1;

*if* ((i - 1) >= 0)

{

prev\_index = path[i-1];

}

int index = path[i];

*//* *Перекрашивание* *узла*

nodes[index]->m\_brush = Qt::*green*;

*//* *Обновление* *узла* *на* *сцене*

nodes[index]->update();

*//* *закрашивание* *предыдущий* *серым*

*if* (prev\_index != -1)

{

nodes[prev\_index]->m\_brush = Qt::*gray*;

nodes[prev\_index]->update();

}

});

}

*//* *очистка* *цвета* *узлов*

QTimer::singleShot(path.size() \* 1000 + 500, *this*, [=]() { clearNodesColor(); });

}

*//* *Обход* *в* *глубину*

void **depthFirstSearch**(int startIndex, QVector<bool>& visited, QVector<int>& path)

{

*if* (startIndex < 0 || startIndex >= nodes.size() || visited[startIndex])

*return*;

*//* *Отмечаем* *узел* *как* *посещенный*

visited[startIndex] = *true*;

*//* *Добавляем* *узел* *в* *путь*

path.append(startIndex);

*//* *Рекурсивно* *обходим* *всех* *непосещенных* *соседей*

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

{

*if* (adjacencyMatrix[i][startIndex] > 0 && !visited[i])

{

depthFirstSearch(i, *visited*, *path*);

}

}

}

*//* *Обход* *в* *ширину*

void **widthFirstCrawl**(int startIndex, QVector<int>& path)

{

QVector<bool> visited(nodes.size(), *false*);

QQueue<int> queue;

visited[startIndex] = *true*;

queue.enqueue(startIndex);

*while* (!queue.isEmpty())

{

int currentNode = queue.dequeue();

path.append(currentNode);

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

{

*if* (adjacencyMatrix[i][currentNode] > 0 && !visited[i])

{

visited[i] = *true*;

queue.enqueue(i);

}

}

}

}

*//* *Рандомизация* *матрицы* *и* *создания* *узлов*

void **randomizeAdjacencyMatrix**(int size)

{

*//* *Очистка* *текущих* *узлов* *и* *матрицы* *смежности*

clearScene();

*//* *Создание* *новой* *матрицы* *смежности*

adjacencyMatrix.clear();

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

{

QVector<int> row(size, 0);

adjacencyMatrix.append(row);

}

*//* *Заполнение* *матрицы* *смежности* *случайными* *значениями*

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

{

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

{

*//* *Вес* *от* *1* *до* *50*

int weight = qrand() % 50 + 1;

adjacencyMatrix[i][j] = weight;

adjacencyMatrix[j][i] = weight;

}

}

*//* *Создание* *узлов*

nodes.clear();

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

{

QPointF position(qrand() % 400, qrand() % 400); *//* *Случайная* *позиция* *на* *сцене*

QString data = QString::number(i + 1); *//* *Данные* *узла*

Node\* node = *new* Node(i, position, data, 60); *//* *Создание* *узла*

node->setPos(position);

nodes.append(node);

*//* *Подключение* *сигналов* *узла*

connect(node, &Node::nodePressed, *this*, &Graph::handleNodePressed);

connect(node, &Node::updateLinksSignal, *this*, &Graph::handleUpdateLinksSignal);

}

*//* *Отрисовка* *узлов*

drawLinks();

*//* *Отрисовка* *связей*

drawNodes();

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

emit adjacencyMatrixChanged(adjacencyMatrix);

}

*//* *очистка* *всех* *связей*

void **clearLinks**()

{

QList<QGraphicsItem\*> items = linkLayer->childItems();

foreach (QGraphicsItem \*item, items)

{

*if* (item != *nullptr*)

linkLayer->removeFromGroup(*item*);

*delete* item;

}

}

*public* slots:

*//* *обработка* *сигнала* *нажатия* *на* *узел*

void **handleNodePressed**(int index)

{

*//* *--------------------------Добавление* *связи-----------------------------------------------------------------*

*if* (!needToLink && !needToDelete && !needToDeleteLink && !needToSolveTask && !needDeixtra && !needInDeep && !needInWidth && !needFloid)

{

*//* *Очистка* *цвета* *узлов*

clearNodesColor();

*return*;

}

*//* *Проверяем,* *есть* *ли* *уже* *выделенный* *узел*

*if* ((needToLink || needToDeleteLink) && selectedNodeIndex != -1)

{

bool confirm = *true*;

*if* (needToLink)

{

*//* *Предложение* *пользователю* *ввести* *вес* *связи*

QString text = QInputDialog::getText(*nullptr*, "Введите вес связи",

"Вес связи:", QLineEdit::*Normal*,

"", *&confirm*);

*if* (confirm && !text.isEmpty())

{

int weight = text.toInt(); *//* *Преобразование* *строки* *в* *целое* *число*

adjacencyMatrix[index][selectedNodeIndex] = weight;

*if* (needTwoWayAddition) adjacencyMatrix[selectedNodeIndex][index] = weight;

}

*//* *Убираем* *флаг* *на* *добавление* *узла*

needToLink = *false*;

}

*else* *if* (needToDeleteLink)

{

adjacencyMatrix[index][selectedNodeIndex] = 0;

*if* (needTwoWayAddition) adjacencyMatrix[selectedNodeIndex][index] = 0;

needToDeleteLink = *false*;

}

*//* *Очистка* *цвета* *узлов*

clearNodesColor();

*//* *Обновляем* *состояние* *выделенного* *узла*

selectedNodeIndex = -1;

}

*else* *if* (needToLink || needToDeleteLink)

{

*//* *Сохраняем* *индекс* *нажатого* *узла*

selectedNodeIndex = index;

}

*//* *------------------------------Удаление* *узла-----------------------------------------------------------------*

*else* *if* (needToDelete)

{

removeNode(index);

needToDelete = *false*;

}

*//* *--------------------------------Задача* *Коммивояжера------------------------------------------------------------------*

*else* *if* (needToSolveTask)

{

needToSolveTask = *false*;

*//* *вызов* *метода* *решения* *задачи* *Комивояжера,* *принимающий* *индекс* *узла* *в* *матрице* *смежности*

qDebug() << "Вы решили задачу коммивояжера!";

*//* *Здесь* *можно* *обработать* *результат,* *например,* *вывести* *путь* *или* *обновить* *интерфейс*

QVector<int> path = solveTravelingSalesmanProblem(index);

qDebug() << path;

*//* *Формирование* *строки* *для* *вывода* *в* *statusbar*

QString \*path\_text = *new* QString;

\*path\_text = "Ответ на задачу коммивояжера: " + QString::number(path[0] + 1);

*for* (int i = 1; i < path.size(); ++i)

{

\*path\_text += " -> " + QString::number(path[i] + 1);

}

*//* *Вычисление* *веса* *всего* *пути* *и* *запись* *в* *statusbar*

\*path\_text += ". Длина вычисленного пути: " + QString::number(calculatePathWeight(path));

emit textToStatusBar(\*path\_text);

*delete* path\_text;

*//* *Отрисовка* *найденного* *пути*

highlightPath(path);

}

*//* *--------------------------------Алгоритм* *Дейкстры---------------------------------------------------------------*

*else* *if* (needDeixtra)

{

needDeixtra = *false*;

qDebug() << "Алгоритм Дейкстры!";

QVector<QVector<int>> paths = shortestPaths(index);

qDebug() << paths;

QString \*text = *new* QString;

\*text = "Кратчайшие пути из узла во все другие.\n";

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

{

*if* (paths[i].size() != 1)

{

\*text += "\nИз " + QString::number(index + 1) + " в " + QString::number(i+1) + ":\n";

\*text += QString::number(paths[i][0] + 1);

*for* (int j = 1; j < paths[i].size(); ++j)

{

\*text += " -> " + QString::number(paths[i][j] + 1);

}

\*text += ".\nСтоимость пути: " + QString::number(calculatePathWeight(paths[i])) + "\n";

}

}

QMessageBox msgBox;

msgBox.setWindowTitle("Алгоритм Дейкстры");

msgBox.setInformativeText(\*text);

msgBox.adjustSize();

msgBox.*exec*();

*delete* text;

}

*//* *--------------------------------Алгоритм* *Флойда---------------------------------------------------------------*

*else* *if* (needFloid)

{

needFloid = *false*;

QStringList all\_paths;

*for* (int index = 0; index < nodes.size(); index++)

{

QVector<QVector<int>> paths = shortestPaths(index);

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

{

*if* (paths[i].size() != 1)

{

QString text = "\nИз " + QString::number(index + 1) + " в " + QString::number(i+1) + ":\n";

text += QString::number(paths[i][0] + 1);

*for* (int j = 1; j < paths[i].size(); ++j)

{

text += " -> " + QString::number(paths[i][j] + 1);

}

text += ".\nСтоимость пути: " + QString::number(calculatePathWeight(paths[i])) + "\n";

all\_paths.append(text);

}

}

}

QMessageBox msgBox;

msgBox.setWindowTitle("Алгоритм Флойда");

QString text\_for\_msg = "";

*for* (int i = 0; i < all\_paths.size(); i++)

{

text\_for\_msg += all\_paths[i];

}

msgBox.setInformativeText(text\_for\_msg);

msgBox.adjustSize();

msgBox.*exec*();

}

*//* *----------------------------------------Обход* *в* *глубину---------------------------------------------------*

*else* *if* (needInDeep)

{

needInDeep = *false*;

qDebug() << "Вы круто обошли граф в глубину";

QVector<bool> visited(nodes.size(), *false*);

depthFirstSearch(index, *visited*, *path*);

qDebug() << path;

highlightPath(path);

path.clear();

}

*//* *----------------------------------------Обход* *в* *ширину---------------------------------------------------*

*else* *if* (needInWidth)

{

needInWidth = *false*;

qDebug() << "Вы круто обошли граф в ширину";

widthFirstCrawl(index, *path*);

qDebug() << path;

highlightPath(path);

path.clear();

}

*//* *----------------------------------------------------------------------------------------------------------*

*//* *отрисовка* *связей*

drawNodes();

drawLinks();

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

emit adjacencyMatrixChanged(adjacencyMatrix);

}

*//* *очистка* *сцены*

void **clearScene**()

{

*//* *Удаляем* *все* *узлы* *из* *вектора* *nodes*

*for* (Node\* node : nodes)

{

scene->removeItem(*node*); *//* *Удаляем* *узел* *из* *сцены*

*delete* node; *//* *Освобождаем* *память,* *выделенную* *под* *узел*

}

*//* *Очищаем* *матрицу*

adjacencyMatrix.clear();

*//* *Очищаем* *вектор* *nodes*

nodes.clear();

*//* *Очистка* *узлов*

clearLinks();

*//* *обновление* *сцены*

drawNodes();

drawLinks();

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

emit adjacencyMatrixChanged(adjacencyMatrix);

}

*//* *слот* *для* *обновления* *связей*

void **handleUpdateLinksSignal**()

{

drawNodes();

drawLinks();

}

*//* *Очистка* *цвета* *узлов*

void **clearNodesColor**()

{

*for* (Node \*node : nodes)

{

QBrush brush(Qt::*white*);

node->m\_brush = brush;

node->update();

}

}

};

#endif *//* *GRAPH\_H*

Link.h

#ifndef LINK\_H

#define LINK\_H

#include <QGraphicsItem>

#include <QPainter>

#include <QFont>

#include "Node.h"

*class* **Link** : *public* QGraphicsItem

{

*public*:

**Link**(Node\* sourceNode, Node\* destNode, int weight)

{

*this*->sourceNode = sourceNode;

*this*->destNode = destNode;

*this*->weight = weight;

}

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

{

*return* QRectF(QPointF(0, 0), QSizeF(100, 20)).normalized();

}

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

{

Q\_UNUSED(option)

Q\_UNUSED(widget)

QPen pen;

pen.setWidth(2);

painter->setPen(pen);

*//* *Получение* *начальных* *координат*

QLineF line = calculateLine();

*//------------------------------*

double angle = atan2(line.dy(), line.dx());

QPointF dest = line.pointAt(0);

int radius = destNode->getSize() / 2;

qreal delta\_y = sin(angle) \* radius;

qreal delta\_x = cos(angle) \* radius;

dest = QPointF{delta\_x, delta\_y};

line.setP1(line.p1() + dest);

line.setP2(line.p2() - dest);

*//------------------------------*

*//* *Отрисовка* *линии*

painter->drawLine(line);

*//* *Отрисовка* *прямоугольника* *с* *весом* *связи*

QPointF center = line.pointAt(0.5);

QRectF rect(center.x() - 20, center.y() - 10, 40, 20);

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

painter->drawRect(rect);

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

painter->setFont(QFont("Arial", 10));

painter->drawText(rect, Qt::*AlignCenter*, QString::number(weight));

}

void **setWeight**(int newWeight) {

weight = newWeight;

update(); *//* *Обновляем* *отображение* *веса*

}

int **getWeight**() *const* {

*return* weight;

}

*public*:

Node\* sourceNode;

Node\* destNode;

int weight;

QLineF **calculateLine**() *const*

{

QPointF sourcePos = sourceNode->mapToScene(sourceNode->*boundingRect*().center());

QPointF destPos = destNode->mapToScene(destNode->*boundingRect*().center());

*return* QLineF(sourcePos, destPos);

}

};

#endif *//* *LINK\_H*

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QItemDelegate>

#include <QPainter>

#include <QTime>

#include <QDebug>

#include "Node.h"

#include "graph.h"

#include "zoomablegraphicsview.h"

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QGraphicsItem>

#include <QStandardItemModel>

#include <QTableView>

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\_draw\_button\_clicked**();

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

void **on\_clear\_button\_clicked**();

*//* *отрисовка* *матрицы*

void **updateAdjacencyMatrix**(*const* QVector<QVector<int>>& adjacencyMatrix);

*//* *отрисовка* *изменений* *в* *матрице*

void **onMatrixCellChanged**(*const* QModelIndex &index);

void **on\_delete\_node\_button\_clicked**();

void **on\_delete\_link\_button\_clicked**();

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

void **on\_checkBox\_clicked**(bool checked);

void **on\_deixtra\_button\_clicked**();

void **updateStatusBar**(QString text);

void **on\_randomize\_button\_clicked**();

void **updateScene**();

void **on\_pushButton\_3\_clicked**();

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

void **on\_button\_floid\_clicked**();

*private*:

Ui::MainWindow \*ui;

Graph \*graph;

*//* *Сцена* *для* *отрисовки*

QGraphicsScene \*scene;

*//* *Модель* *для* *отображения* *матрицы* *смежности*

QStandardItemModel \*matrixModel;

QTimer \*updateTimer;

int updateInterval = 1000; *//* *интервал* *обновления* *в* *миллисекундах*

};

*class* **CenterTextDelegate** : *public* QItemDelegate

{

*public*:

void ***paint***(QPainter \*painter, *const* QStyleOptionViewItem &option, *const* QModelIndex &index) *const* *override*

{

QStyleOptionViewItem opt = option;

opt.displayAlignment = Qt::*AlignCenter*;

QItemDelegate::paint(*painter*, opt, index);

}

};

#endif *//* *MAINWINDOW\_H*

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

{

ui->setupUi(*this*);

*//* *создание* *сцены*

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

*//* *создание* *графа*

graph = *new* Graph(scene);

ui->graphicsView->setScene(scene);

*//* *коннект* *кнопки* *"очистить"*

connect(ui->clear\_button, &QPushButton::clicked, graph, &Graph::clearScene);

*//* *коннект* *обновления* *статусбара* *из* *Graph*

connect(graph, &Graph::textToStatusBar, *this*, &MainWindow::updateStatusBar);

*//* *----------------------------------------------Матрица* *смежности----------------------------------------------------------*

*//* *модель* *для* *матрицы* *смежности*

matrixModel = *new* QStandardItemModel(*this*);

*//* *установка* *модели* *к* *обьекту* *QTableView*

ui->matrixView->setModel(matrixModel);

*//* *минимальный* *размер* *колонок* *и* *столбцов* *QTableView*

ui->matrixView->verticalHeader()->setMinimumSectionSize(5);

ui->matrixView->horizontalHeader()->setMinimumSectionSize(5);

*//* *коннеут* *для* *изменения* *графа* *при* *изменении* *матрицы*

connect(matrixModel, &QStandardItemModel::dataChanged, *this*, &MainWindow::onMatrixCellChanged);

*//* *коннект* *матрицы* *смежности*

connect(graph, &Graph::adjacencyMatrixChanged, *this*, &MainWindow::updateAdjacencyMatrix);

*//* *Отправляем* *сигнал* *об* *изменении* *матрицы* *смежности*

updateAdjacencyMatrix(graph->getMatrix());

*//* *--------------------------------------------------------------------------------------------------------------------------*

*//* *Стиль* *кнопок*

ui->delete\_link\_button->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

ui->delete\_node\_button->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

ui->draw\_button->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

ui->pushButton->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

ui->pushButton\_2->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

ui->clear\_button->setStyleSheet("QPushButton::pressed { background-color: lightgray; border: 1px solid black; }");

}

MainWindow::~MainWindow()

{

*delete* ui;

*//* *очистка* *таймера*

updateTimer->stop();

*delete* updateTimer;

}

void MainWindow::on\_draw\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

qDebug() << "Добавление узла.";

graph->addNode();

*//* *отрисовка* *узлов*

updateScene();

}

void MainWindow::on\_pushButton\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

graph->needToLink = *true*;

}

void MainWindow::on\_clear\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

qDebug() << "Очистка.";

}

void MainWindow::updateAdjacencyMatrix(*const* QVector<QVector<int>>& adjacencyMatrix)

{

*//* *Центрирование* *полей*

CenterTextDelegate \*delegate = *new* CenterTextDelegate;

ui->matrixView->setItemDelegate(delegate);

*//* *Очистка*

matrixModel->clear();

matrixModel->setColumnCount(adjacencyMatrix.size());

matrixModel->setRowCount(adjacencyMatrix.size());

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

{

*for* (int j = 0; j < adjacencyMatrix[i].size(); ++j)

{

QStandardItem \*item = *new* QStandardItem(QString::number(adjacencyMatrix[i][j]));

matrixModel->setItem(i, j, item);

}

}

*//* *Установка* *ширины* *колонок* *и* *высоты* *строк* *на* *всю* *ширину* *и* *высоту* *окна* *QTableView*

ui->matrixView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch); *//* *Растягиваем* *все* *колонки*

ui->matrixView->verticalHeader()->setSectionResizeMode(QHeaderView::Stretch); *//* *Растягиваем* *все* *строки*

}

void MainWindow::onMatrixCellChanged(*const* QModelIndex &index)

{

int row = index.row();

int column = index.column();

int value = matrixModel->data(index).toInt();

*//* *Обновляем* *матрицу* *смежности*

graph->getMatrix()[row][column] = value;

*//* *Перерисовываем* *граф*

updateScene();

}

void MainWindow::updateScene()

{

graph->drawLinks();

graph->drawNodes();

}

void MainWindow::on\_delete\_node\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needToDelete;

*if* (value)

{

value = *false*;

qDebug() << "Удаление узла. off";

}

*else*

{

value = *true*;

qDebug() << "Удаление узла. on";

}

}

void MainWindow::on\_delete\_link\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needToDeleteLink;

*if* (value)

{

value = *false*;

qDebug() << "Удаление связи. off";

}

*else*

{

value = *true*;

qDebug() << "Удаление связи. on";

}

}

void MainWindow::on\_pushButton\_2\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needToSolveTask;

*if* (value)

{

value = *false*;

qDebug() << "Коммивояжер. off";

}

*else*

{

value = *true*;

qDebug() << "Коммивояжер. on";

}

}

void MainWindow::on\_checkBox\_clicked(bool checked)

{

graph->needTwoWayAddition = checked;

}

void MainWindow::on\_deixtra\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needDeixtra;

*if* (value)

{

value = *false*;

qDebug() << "Дейкстра. off";

}

*else*

{

value = *true*;

qDebug() << "Дейкстра. on";

}

}

void MainWindow::updateStatusBar(QString text)

{

ui->statusbar->showMessage(text);

}

void MainWindow::on\_randomize\_button\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

qDebug() << "Рандомизация.";

graph->randomizeAdjacencyMatrix(6);

}

void MainWindow::on\_pushButton\_3\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needInDeep;

*if* (value)

{

value = *false*;

qDebug() << "Обход в глубину. off";

}

*else*

{

value = *true*;

qDebug() << "Обход в глубину. on";

}

}

void MainWindow::on\_butti\_in\_width\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needInWidth;

*if* (value)

{

value = *false*;

qDebug() << "Обход в ширину. off";

}

*else*

{

value = *true*;

qDebug() << "Обход в ширину. on";

}

}

void MainWindow::on\_button\_floid\_clicked()

{

*//* *Выключение* *других* *функций*

graph->turn\_off\_buttons();

*//* *переключение*

bool &value = graph->needFloid;

*if* (value)

{

value = *false*;

qDebug() << "Алгоритм Флойда. off";

}

*else*

{

value = *true*;

qDebug() << "Алгоритм Флойда. on";

}

}

Node.h

#ifndef NODE\_H

#define NODE\_H

#include <QGraphicsItem>

#include <QPainter>

#include <QDebug>

#include <cmath>

#include <QGraphicsItem>

#include <QGraphicsSceneMouseEvent>

*class* **Node** : *public* QObject, *public* QGraphicsItem

{

Q\_OBJECT

signals:

void **nodePressed**(int index); *//* *Объявление* *сигнала* *нажатия* *на* *узел*

void **updateLinksSignal**(); *//* *сигнал* *на* *обновление* *связи*

*public*:

**Node**(int index, QPointF position, QString data, int size)

{

m\_index = index;

m\_position = position;

m\_data = data;

m\_size = size;

m\_brush = Qt::*white*;

m\_isMovable = *true*;

setFlag(*ItemIsMovable*, *true*); *//* *Устанавливаем* *флаг,* *позволяющий* *перемещать* *узлы*

}

int **getSize**() {*return* m\_size;}

void **isMovableNode**(bool flag)

{

m\_isMovable = flag;

}

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

{

*return* QRectF(-m\_size/2, -m\_size/2, m\_size, m\_size); *//* *Размеры* *узла*

}

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

{

Q\_UNUSED(option)

Q\_UNUSED(widget)

QPen pen;

pen.setWidth(2);

painter->setPen(pen);

*//* *основной* *элемент* *узла*

painter->setBrush(m\_brush);

painter->drawEllipse(-m\_size/2, -m\_size/2, m\_size, m\_size);

painter->drawText(-10, 10, m\_data);

*//* *прямоугольник* *с* *номером* *узла*

QRectF rect(*boundingRect*().topLeft(), QSizeF(15, 15));

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

painter->drawRect(rect);

painter->setFont(QFont("Arial", 9));

painter->drawText(rect, Qt::*AlignCenter*, QString::number(m\_index+1));

}

*//* *перемещение* *узла* *по* *зажатию* *ЛКМ*

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event) *override*

{

*if* (m\_isMovable)

{

QGraphicsItem::mouseMoveEvent(*event*);

QPointF move = event->scenePos() - event->lastScenePos();

m\_position += move;

*//* *Обновляем* *связи* *при* *перемещении* *узла*

emit updateLinksSignal();

}

}

*//* *нажатие* *ЛКМ*

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

{

QGraphicsItem::mousePressEvent(*event*);

*//* *Выделяем* *узел* *синим* *цветом*

QBrush brush(Qt::*cyan*);

*//* *Сохраняем* *кисть* *для* *использования* *в* *методе* *paint*

m\_brush = brush;

}

*//* *отжатие* *ЛКМ*

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

{

QGraphicsItem::mouseReleaseEvent(*event*);

*//* *Отправляем* *сигнал* *с* *индексом* *нажатого* *узла*

emit nodePressed(m\_index);

}

int m\_index; *//* *индекс* *узла* *(для* *матрицы* *смежности)*

QBrush m\_brush;

bool m\_isMovable;

*private*:

int m\_size; *//* *размер* *окружности* *узла*

QPointF m\_position; *//* *позиция* *узла*

QString m\_data;

};

#endif *//NODE\_H*

Zoomablegraphicsview.h

#ifndef ZOOMABLEGRAPHICSVIEW\_H

#define ZOOMABLEGRAPHICSVIEW\_H

#include <QGraphicsView>

#include <QWheelEvent>

#include <QMouseEvent>

#include <QApplication>

*class* **ZoomableGraphicsView** : *public* QGraphicsView

{

*public*:

**ZoomableGraphicsView**(QWidget \*parent = *nullptr*) : QGraphicsView(*parent*), minZoomFactor(0.5), maxZoomFactor(2.0) {}

void **setMinZoomFactor**(qreal factor) { minZoomFactor = factor; }

void **setMaxZoomFactor**(qreal factor) { maxZoomFactor = factor; }

*protected*:

void ***wheelEvent***(QWheelEvent \*event) *override*

{

*if* (QApplication::keyboardModifiers() == Qt::*ControlModifier*)

{

qreal factor = transform().scale(1.2, 1.2).mapRect(QRectF(0, 0, 1, 1)).width();

*//* *Увеличение* *масштаба*

*if* (event->angleDelta().y() > 0 && factor < maxZoomFactor)

{

scale(1.2, 1.2);

}

*//* *Уменьшение* *масштаба*

*else* *if* (event->angleDelta().y() < 0 && factor > minZoomFactor)

{

scale(1 / 1.2, 1 / 1.2);

}

}

*else*

{

QGraphicsView::wheelEvent(*event*);

}

}

void ***mousePressEvent***(QMouseEvent \*event) *override*

{

*if* (event->button() == Qt::*LeftButton* && QApplication::keyboardModifiers() == Qt::*ControlModifier*)

{

setDragMode(QGraphicsView::*ScrollHandDrag*);

}

QGraphicsView::mousePressEvent(*event*);

}

void ***mouseReleaseEvent***(QMouseEvent \*event) *override*

{

*if* (event->button() == Qt::*LeftButton*)

{

setDragMode(QGraphicsView::*NoDrag*);

}

QGraphicsView::mouseReleaseEvent(*event*);

}

*private*:

qreal minZoomFactor;

qreal maxZoomFactor;

};

#endif *//* *ZOOMABLEGRAPHICSVIEW\_H*

Main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

*//* *рандомизация*

qsrand(QTime::currentTime().msec());

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

MainWindow w;

w.show();

*return* a.exec();

}

**Вывод**

В ходе работы мы применили знания о работе с классами, и

интерфейсами. По ходу работы был разработан граф, операции с которым

выполняются посредством работы с интерфейсом, разработанным с

помощью средств QT. Были разработаны обходы графа в ширину и

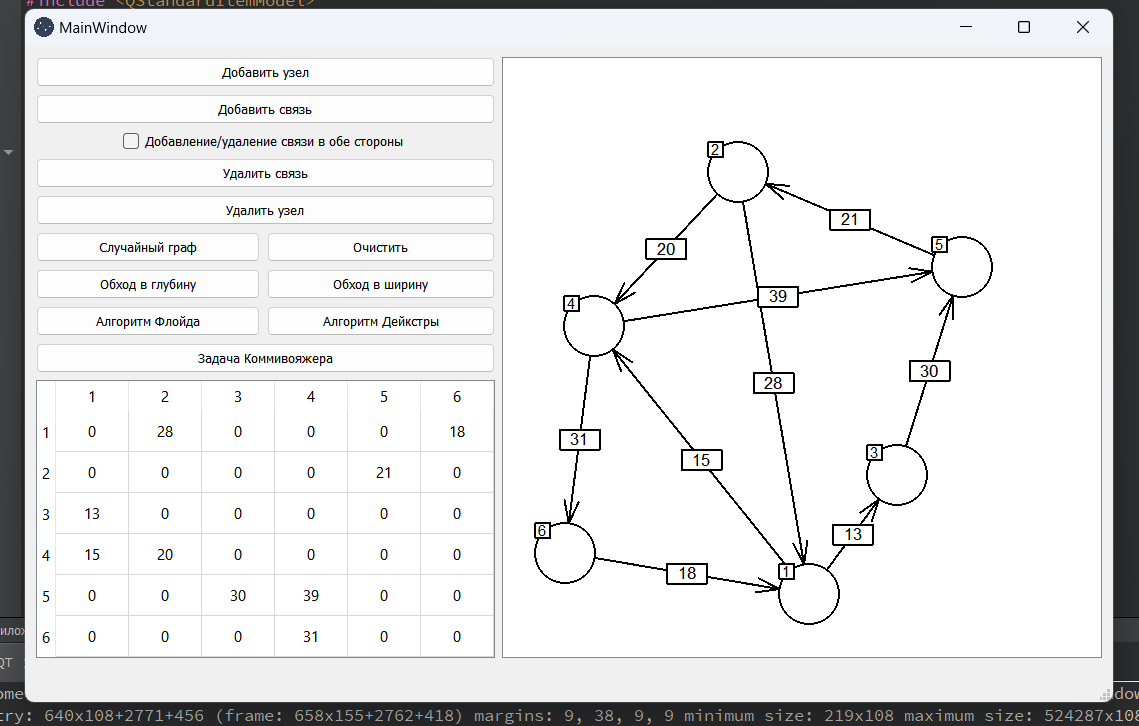
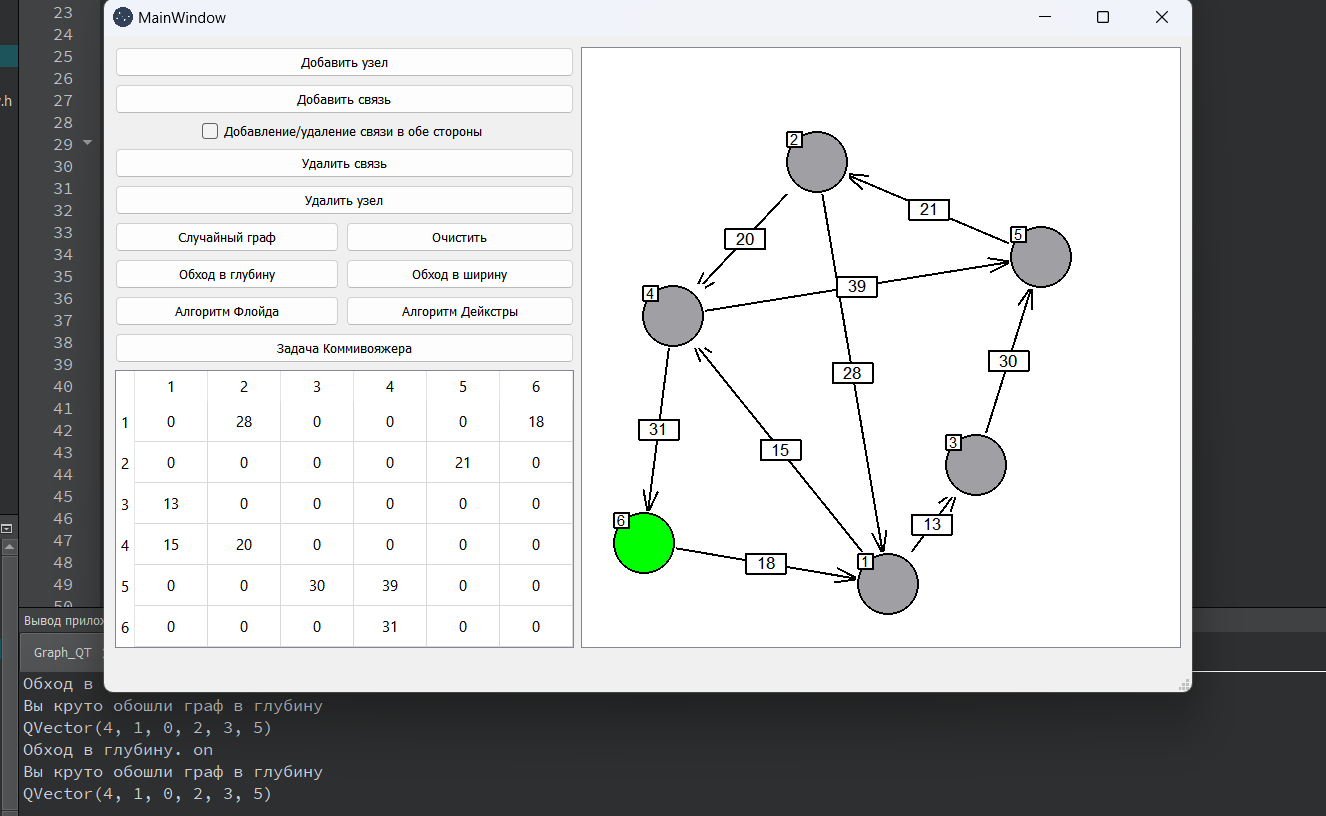
глубину, реализованы алгоритмы Дейкстера и Флойда, функции вывода

графа. Также, были разработаны функции редактирования графа: добавление

и удаление вершины, добавление и удаление ребра. По итогу работы был реализован граф, с меню, которое позволяет управлять им.

**GitHub**: [Graph\_QT](https://github.com/geroineee/Graph_QT)

**Результаты работы**

**** **** 