Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

По теме: «Графы»

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Бреднев Максим Павлович

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2022

**Постановка задачи**

Реализовать граф со следующими методами:

- Обход в ширину.

- Обход в глубину.

- Алгоритм Дейкстры.

- Создание новой вершины.

- Удаление вершины.

- Добавление и удаление ребра.

- Редактирование весов ребер.

- Редактирование матрицы смежности (или инцидентности в зависимости от реализации).

**Анализ задачи**

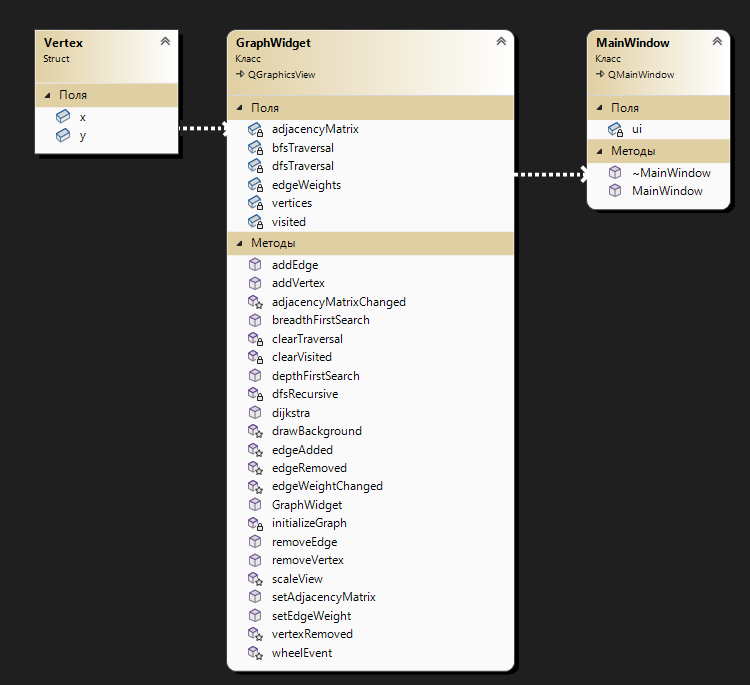
- Графы рисуются с помощью базовых библиотек QPainter и QtMath, а также созданных методов addEdge и addVertex.

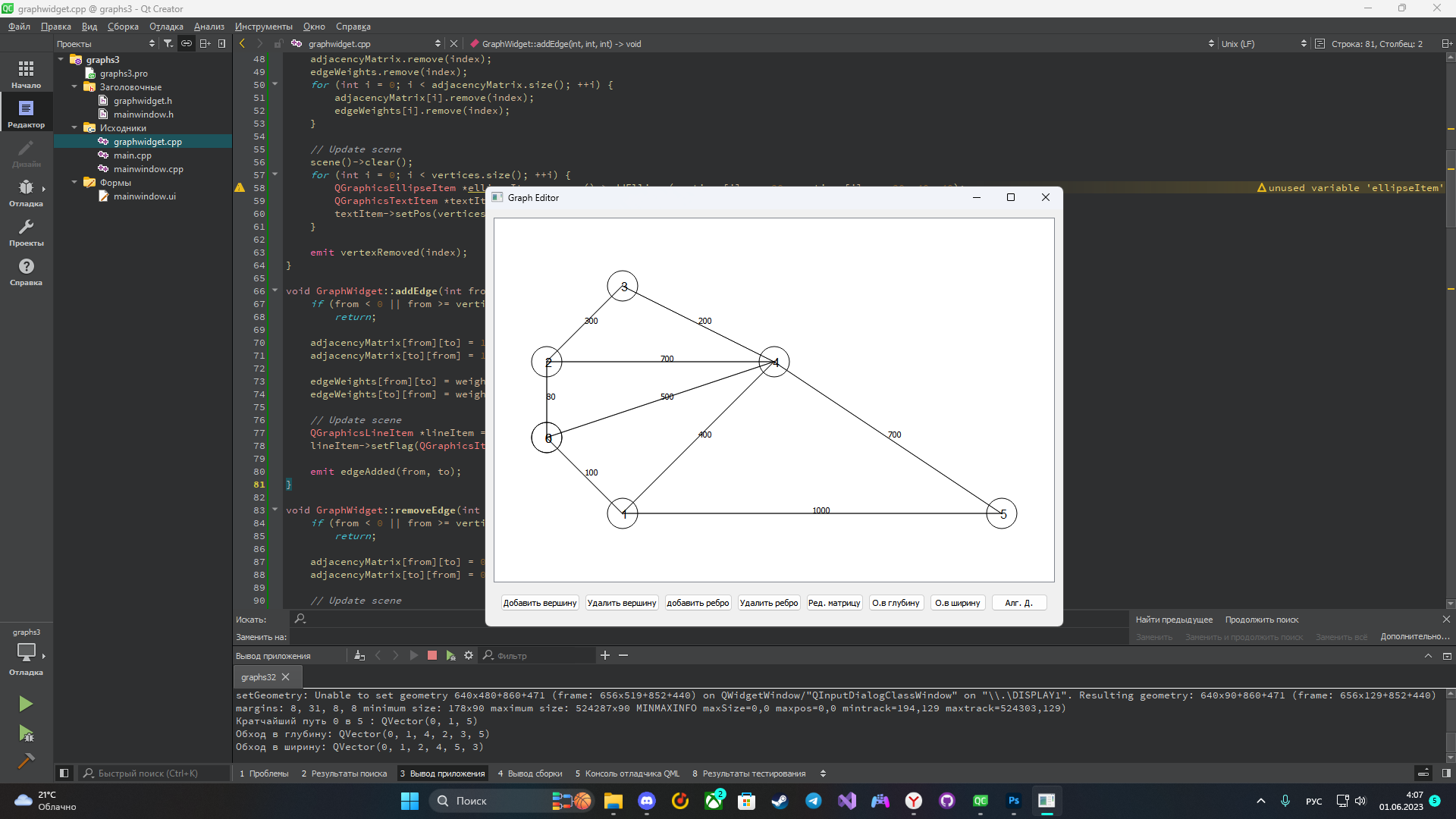
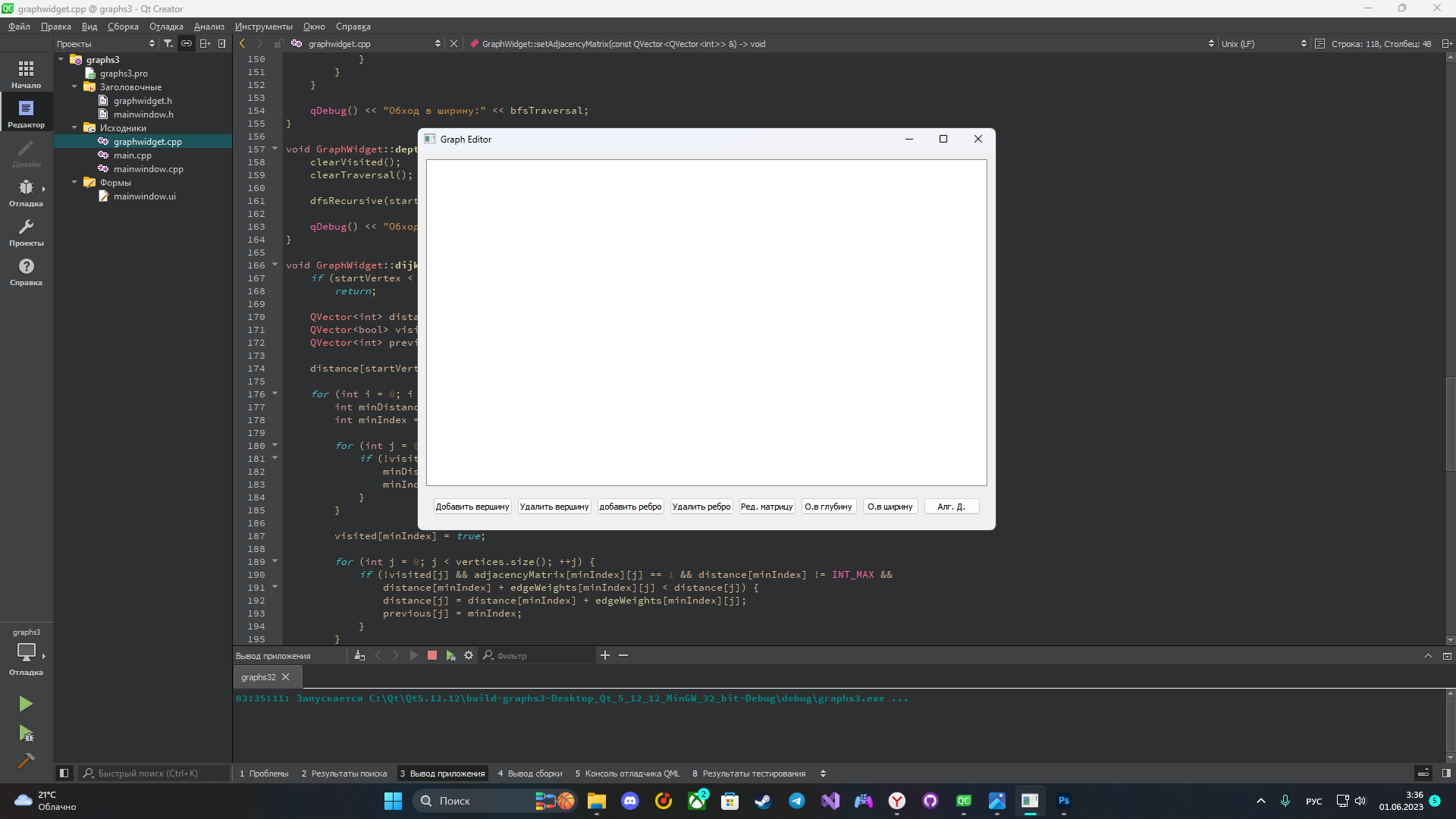
- Для реализации методов был создан класс GraphWidget.

- Для визуализации использовался фреймворк QT, вершины графа рисуются по координатам.

- Для реализации обходов в глубину, в ширину и алгоритма Дейкстры были созданы методы breadthFirstSearch, depthFirstSearch и dijkstra, результат выводится в консоли QT.

**Выполнение программы**

****



**Код программы**

graphwidget.h

#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QVector>

*struct* **Vertex** {

int x;

int y;

};

*class* **GraphWidget** : *public* QGraphicsView

{

Q\_OBJECT

*public*:

*explicit* **GraphWidget**(QWidget \*parent = *nullptr*);

void **addVertex**(int x, int y);

void **removeVertex**(int index);

void **addEdge**(int from, int to, int weight);

void **removeEdge**(int from, int to);

void **setEdgeWeight**(int from, int to, int weight);

void **setAdjacencyMatrix**(*const* QVector<QVector<int>> &matrix);

void **breadthFirstSearch**(int startVertex);

void **depthFirstSearch**(int startVertex);

void **dijkstra**(int startVertex, int endVertex);

*protected*:

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

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

void **scaleView**(qreal scaleFactor);

signals:

void **vertexAdded**(int index);

void **vertexRemoved**(int index);

void **edgeAdded**(int from, int to);

void **edgeRemoved**(int from, int to);

void **edgeWeightChanged**(int from, int to, int weight);

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

*private*:

void **initializeGraph**();

void **clearVisited**();

void **clearTraversal**();

void **dfsRecursive**(int vertex);

QVector<Vertex> vertices;

QVector<QVector<int>> adjacencyMatrix;

QVector<QVector<int>> edgeWeights;

QVector<bool> visited;

QVector<int> bfsTraversal;

QVector<int> dfsTraversal;

};

#endif *//* *GRAPHWIDGET\_H*

*graphwidget.cpp*

#include "graphwidget.h"

#include <QKeyEvent>

#include <QRandomGenerator>

#include <QGraphicsScene>

#include <QGraphicsEllipseItem>

#include <QGraphicsTextItem>

#include <QGraphicsLineItem>

#include <QPainter>

#include <QDebug>

#include <climits>

GraphWidget::**GraphWidget**(QWidget \*parent)

: QGraphicsView(*parent*)

{

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

setScene(*scene*);

setRenderHint(QPainter::*Antialiasing*, *true*);

setRenderHint(QPainter::*TextAntialiasing*, *true*);

initializeGraph();

}

void GraphWidget::**addVertex**(int x, int y) {

Vertex vertex;

vertex.x = x;

vertex.y = y;

vertices.append(vertex);

adjacencyMatrix.resize(vertices.size());

edgeWeights.resize(vertices.size());

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

adjacencyMatrix[i].resize(vertices.size());

edgeWeights[i].resize(vertices.size());

}

*//* *Update* *scene*

QGraphicsEllipseItem \*ellipseItem = scene()->addEllipse(x - 20, y - 20, 40, 40);

QGraphicsTextItem \*textItem = scene()->addText(QString::number(vertices.size() - 1), QFont("Arial", 12));

textItem->setPos(x - 6, y - 12);

edgeWeights[vertices.size() - 1].resize(vertices.size());

emit vertexAdded(vertices.size() - 1);

}

void GraphWidget::**removeVertex**(int index) {

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

*return*;

vertices.remove(index);

adjacencyMatrix.remove(index);

edgeWeights.remove(index);

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

adjacencyMatrix[i].remove(index);

edgeWeights[i].remove(index);

}

*//* *Update* *scene*

scene()->clear();

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

QGraphicsEllipseItem \*ellipseItem = scene()->addEllipse(vertices[i].x - 20, vertices[i].y - 20, 40, 40);

QGraphicsTextItem \*textItem = scene()->addText(QString::number(i), QFont("Arial", 12));

textItem->setPos(vertices[i].x - 6, vertices[i].y - 12);

}

emit vertexRemoved(index);

}

void GraphWidget::**addEdge**(int from, int to, int weight) {

*if* (from < 0 || from >= vertices.size() || to < 0 || to >= vertices.size())

*return*;

adjacencyMatrix[from][to] = 1;

adjacencyMatrix[to][from] = 1;

edgeWeights[from][to] = weight;

edgeWeights[to][from] = weight;

*//* *Update* *scene*

QGraphicsLineItem \*lineItem = scene()->addLine(vertices[from].x, vertices[from].y, vertices[to].x, vertices[to].y);

lineItem->setFlag(QGraphicsItem::*ItemIsSelectable*, *true*);

emit edgeAdded(from, to);

}

void GraphWidget::**removeEdge**(int from, int to) {

*if* (from < 0 || from >= vertices.size() || to < 0 || to >= vertices.size())

*return*;

adjacencyMatrix[from][to] = 0;

adjacencyMatrix[to][from] = 0;

*//* *Update* *scene*

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

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

QGraphicsLineItem \*lineItem = qgraphicsitem\_cast<QGraphicsLineItem \*>(*item*);

*if* (lineItem && lineItem->line().p1().x() == vertices[from].x && lineItem->line().p1().y() == vertices[from].y &&

lineItem->line().p2().x() == vertices[to].x && lineItem->line().p2().y() == vertices[to].y) {

scene()->removeItem(*lineItem*);

*delete* lineItem;

*break*;

}

}

emit edgeRemoved(from, to);

}

void GraphWidget::**setEdgeWeight**(int from, int to, int weight) {

*if* (from < 0 || from >= vertices.size() || to < 0 || to >= vertices.size())

*return*;

edgeWeights[from][to] = weight;

edgeWeights[to][from] = weight;

emit edgeWeightChanged(from, to, weight);

}

void GraphWidget::**setAdjacencyMatrix**(*const* QVector<QVector<int>> &matrix) {

*if* (matrix.size() != vertices.size())

*return*;

adjacencyMatrix = matrix;

*//* *Update* *scene*

scene()->clear();

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

QGraphicsEllipseItem \*ellipseItem = scene()->addEllipse(vertices[i].x - 20, vertices[i].y - 20, 40, 40);

QGraphicsTextItem \*textItem = scene()->addText(QString::number(i), QFont("Arial", 12));

textItem->setPos(vertices[i].x - 6, vertices[i].y - 12);

}

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

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

*if* (adjacencyMatrix[i][j] == 1) {

QGraphicsLineItem \*lineItem = scene()->addLine(vertices[i].x, vertices[i].y, vertices[j].x, vertices[j].y);

lineItem->setFlag(QGraphicsItem::*ItemIsSelectable*, *true*);

}

}

}

emit adjacencyMatrixChanged(adjacencyMatrix);

}

void GraphWidget::**breadthFirstSearch**(int startVertex) {

clearVisited();

clearTraversal();

QList<int> queue;

visited[startVertex] = *true*;

queue.append(startVertex);

bfsTraversal.append(startVertex);

*while* (!queue.isEmpty()) {

int currentVertex = queue.takeFirst();

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

*if* (adjacencyMatrix[currentVertex][i] == 1 && !visited[i]) {

visited[i] = *true*;

queue.append(i);

bfsTraversal.append(i);

}

}

}

qDebug() << "Обход в ширину:" << bfsTraversal;

}

void GraphWidget::**depthFirstSearch**(int startVertex) {

clearVisited();

clearTraversal();

dfsRecursive(startVertex);

qDebug() << "Обход в глубину:" << dfsTraversal;

}

void GraphWidget::**dijkstra**(int startVertex, int endVertex) {

*if* (startVertex < 0 || startVertex >= vertices.size() || endVertex < 0 || endVertex >= vertices.size())

*return*;

QVector<int> distance(vertices.size(), INT\_MAX);

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

QVector<int> previous(vertices.size(), -1);

distance[startVertex] = 0;

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

int minDistance = INT\_MAX;

int minIndex = -1;

*for* (int j = 0; j < vertices.size(); ++j) {

*if* (!visited[j] && distance[j] < minDistance) {

minDistance = distance[j];

minIndex = j;

}

}

visited[minIndex] = *true*;

*for* (int j = 0; j < vertices.size(); ++j) {

*if* (!visited[j] && adjacencyMatrix[minIndex][j] == 1 && distance[minIndex] != INT\_MAX &&

distance[minIndex] + edgeWeights[minIndex][j] < distance[j]) {

distance[j] = distance[minIndex] + edgeWeights[minIndex][j];

previous[j] = minIndex;

}

}

}

QVector<int> shortestPath;

int currentVertex = endVertex;

*while* (currentVertex != -1) {

shortestPath.prepend(currentVertex);

currentVertex = previous[currentVertex];

}

qDebug() << "Кратчайший путь" << startVertex << "в" << endVertex << ":" << shortestPath;

}

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

Q\_UNUSED(rect);

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

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

painter->drawRect(-2000, -2000, 4000, 4000);

painter->setRenderHint(QPainter::*Antialiasing*, *true*);

painter->setPen(QPen(Qt::*black*, 0.5));

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

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

*if* (adjacencyMatrix[i][j] == 1) {

painter->drawLine(vertices[i].x, vertices[i].y, vertices[j].x, vertices[j].y);

}

}

}

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

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

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

*if* (adjacencyMatrix[i][j] == 1) {

QPoint p1(vertices[i].x, vertices[i].y);

QPoint p2(vertices[j].x, vertices[j].y);

QPointF center = (p1 + p2) / 2.0;

painter->drawText(center, QString::number(edgeWeights[i][j]));

}

}

}

}

void GraphWidget::***wheelEvent***(QWheelEvent \*event) {

scaleView(pow((double)2, -event->delta() / 240.0));

}

void GraphWidget::**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 GraphWidget::**initializeGraph**() {

vertices.clear();

adjacencyMatrix.clear();

edgeWeights.clear();

visited.clear();

bfsTraversal.clear();

dfsTraversal.clear();

clearTraversal();

scene()->clear();

}

void GraphWidget::**clearVisited**() {

visited.clear();

visited.resize(vertices.size());

}

void GraphWidget::**clearTraversal**() {

bfsTraversal.clear();

dfsTraversal.clear();

}

void GraphWidget::**dfsRecursive**(int vertex) {

visited[vertex] = *true*;

dfsTraversal.append(vertex);

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

*if* (adjacencyMatrix[vertex][i] == 1 && !visited[i]) {

dfsRecursive(i);

}

}

}

main.cpp

#include <QApplication>

#include <QPushButton>

#include <QHBoxLayout>

#include "graphwidget.h"

#include <QInputDialog>

#include <QDebug>

#include <QPainter>

#include <QLabel>

*using* *namespace* std;

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

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

*//kommivoyajer*

GraphWidget graphWidget;

*//* *Create* *buttons* *for* *graph* *editing*

QPushButton addButton("Добавить вершину");

QPushButton removeButton("Удалить вершину");

QPushButton addEdgeButton("добавить ребро");

QPushButton removeEdgeButton("Удалить ребро");

QPushButton editMatrixButton("Ред. матрицу");

QPushButton dfsButton("О.в глубину");

QPushButton bfsButton("О.в ширину");

QPushButton dijkstraButton("Алг. Д.");

*//* *Connect* *button* *signals* *to* *appropriate* *slots*

QObject::connect(&addButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Prompt* *the* *user* *for* *vertex* *coordinates*

int x = QInputDialog::getInt(*nullptr*, "Добавить вершину", "Введите X коорд-ту:");

int y = QInputDialog::getInt(*nullptr*, "Добавить вершину", "Введите Y коорд-ту:");

*//* *Add* *the* *vertex* *to* *the* *graph*

graphWidget.addVertex(x, y);

});

QObject::connect(&removeButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Prompt* *the* *user* *for* *the* *vertex* *index* *to* *remove*

int index = QInputDialog::getInt(*nullptr*, "Удалить вершину", "Введите индекс удаляемой вершины:");

*//* *Remove* *the* *vertex* *from* *the* *graph*

graphWidget.removeVertex(index);

});

QObject::connect(&addEdgeButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Prompt* *the* *user* *for* *the* *source* *and* *destination* *vertices* *for* *the* *edge*

int from = QInputDialog::getInt(*nullptr*, "Добавить ребро", "Из вершины:");

int to = QInputDialog::getInt(*nullptr*, "Добавить ребро", "В вершину:");

int weight = QInputDialog::getInt(*nullptr*, "Добавить вес ребра", "Вес ");

*//* *Add* *the* *edge* *to* *the* *graph*

graphWidget.addEdge(from, to, weight);

});

QObject::connect(&removeEdgeButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Prompt* *the* *user* *for* *the* *source* *and* *destination* *vertices* *of* *the* *edge* *to* *remove*

int from = QInputDialog::getInt(*nullptr*, "Удалить ребро", "Из вершины:");

int to = QInputDialog::getInt(*nullptr*, "Удалить ребро", "В вершину:");

*//* *Remove* *the* *edge* *from* *the* *graph*

graphWidget.removeEdge(from, to);

});

QObject::connect(&editMatrixButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Prompt* *the* *user* *for* *the* *new* *adjacency* *matrix*

QVector<QVector<int>> matrix;

*//* *Assuming* *a* *6x6* *matrix* *for* *this* *example*

*for* (int i = 0; i < 6; ++i) {

QVector<int> row;

*for* (int j = 0; j < 6; ++j) {

int value = QInputDialog::getInt(*nullptr*, "Ред. матрицу",

QString("Введите значение для элемента (%1,%2):").arg(i).arg(j));

row.append(value);

}

matrix.append(row);

}

*//* *Set* *the* *new* *adjacency* *matrix* *for* *the* *graph*

graphWidget.setAdjacencyMatrix(matrix);

});

QObject::connect(&dfsButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Perform* *depth-first* *search* *on* *the* *graph*

graphWidget.depthFirstSearch(0); *//* *Start* *from* *vertex* *0*

});

QObject::connect(&bfsButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Perform* *breadth-first* *search* *on* *the* *graph*

graphWidget.breadthFirstSearch(0); *//* *Start* *from* *vertex* *0*

});

QObject::connect(&dijkstraButton, &QPushButton::clicked, [&graphWidget]() {

*//* *Perform* *Dijkstra's* *algorithm* *on* *the* *graph*

graphWidget.dijkstra(0, 5); *//* *Start* *from* *vertex* *0*

});

*//* *Create* *a* *layout* *for* *the* *buttons*

QHBoxLayout layout;

layout.addWidget(*&addButton*);

layout.addWidget(*&removeButton*);

layout.addWidget(*&addEdgeButton*);

layout.addWidget(*&removeEdgeButton*);

layout.addWidget(*&editMatrixButton*);

layout.addWidget(*&dfsButton*);

layout.addWidget(*&bfsButton*);

layout.addWidget(*&dijkstraButton*);

*//* *Create* *a* *main* *widget* *and* *set* *the* *layout*

QWidget mainWidget;

mainWidget.setLayout(*&layout*);

*//* *Create* *a* *main* *layout* *for* *the* *main* *widget*

QVBoxLayout mainLayout;

mainLayout.addWidget(*&graphWidget*);

mainLayout.addWidget(*&mainWidget*);

*//* *Set* *the* *main* *layout* *for* *the* *application* *window*

QWidget window;

window.setLayout(*&mainLayout*);

window.setWindowTitle("Graph Editor");

window.show();

*return* app.exec();

}