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

Хайбиев Константин

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).

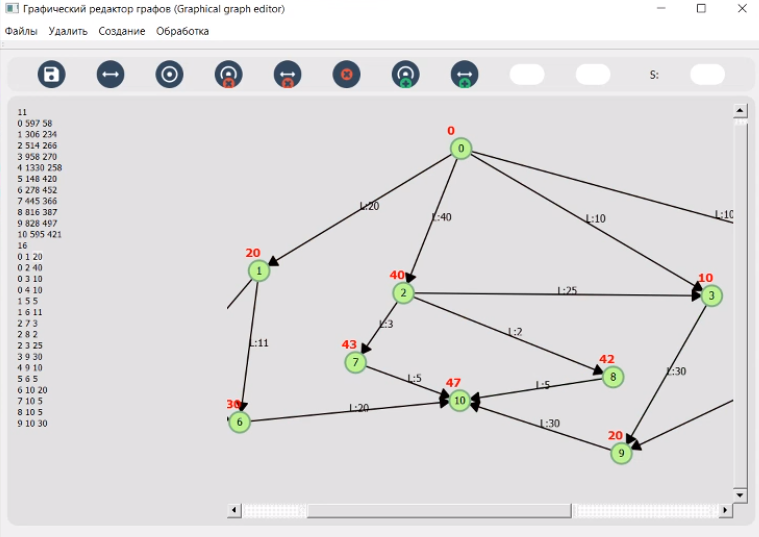
**Анализ задачи**

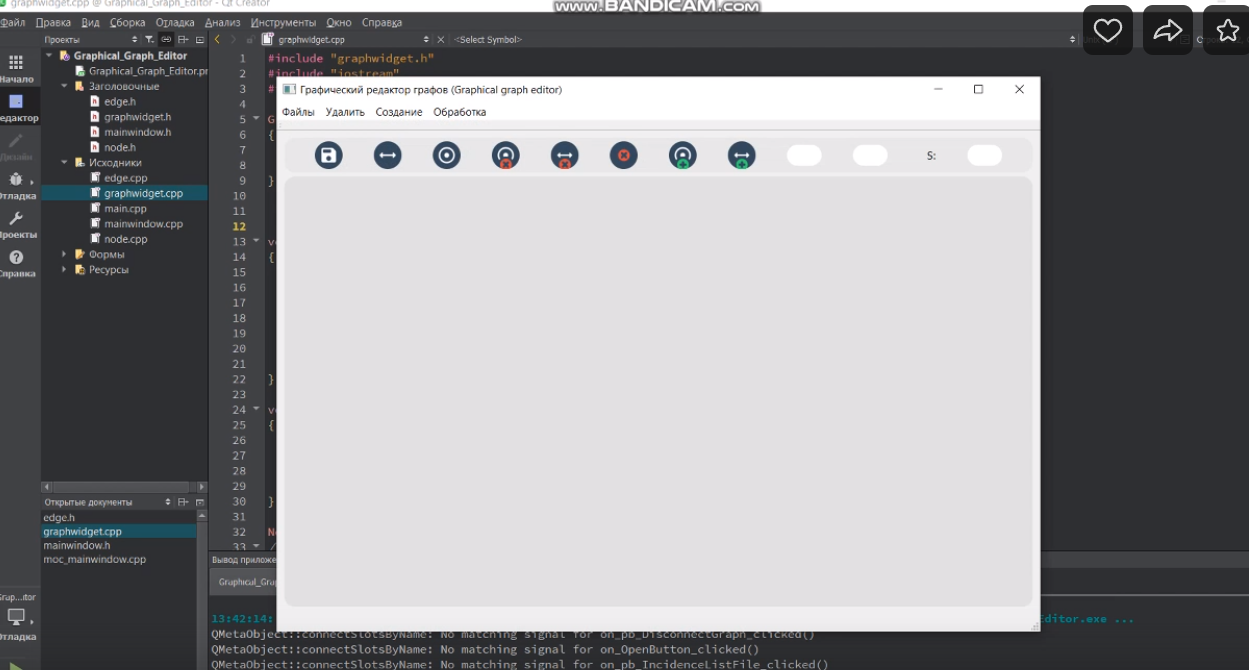
- Графы рисуются с помощью базовых библиотек QPainter и QtMath, а также созданных методов Edge и Vertex.

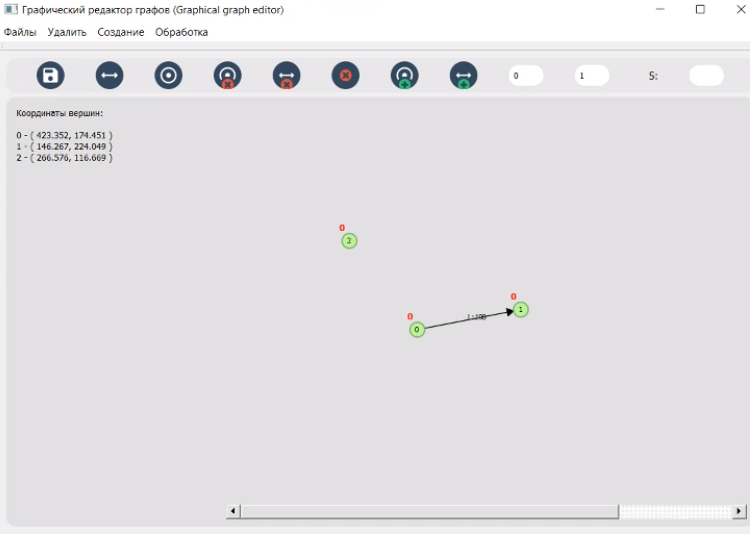
- Для реализации методов был создан класс GraphWidget.

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

**Выполнение программы**

****

****

****

**Код**

Mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "iostream"

#include "graphwidget.h"

#include "QFile"

#include "QTextStream"

#include "QFileDialog"

#include "QMessageBox"

#include <QSettings> //Для сохранения настроек

#include <limits>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

;

GraphWidget \*graph;

Ui::MainWindow \*ui;

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

void writeSettings();//Для сохранения настроек

void readSettings ();//Для восстановления настроек

std::pair<bool,std::vector<int> > DFS(int from, int to, std::vector<int> marked, unsigned timer);

signals:

void changeOpenFile();

private:

bool stop;

QSettings m\_settings;//Для сохранения настроек

QString InputPath ="";

std::vector<int> shortPath;

private slots:

void deleteNode();// Удаляет выбранные вершины графа

void on\_pb\_CoordinatesOfAllNode\_clicked();// Выписывает координаты всех вершин графа

void on\_pb\_LengthOfEdges\_clicked();// Выписывает длину всех рёбер графа

//void on\_pb\_ConnectGraph\_clicked();// Сделать граф полным

void on\_pb\_DisconnectGraph\_clicked();// Удалить все рёбра в графе

void on\_OpenButton\_clicked(); //Открыть форматированный файл с графом

void on\_saveButton\_clicked(); // Сохранить форматированный граф в файл

void on\_pb\_RemoveEdges\_clicked();//Удалить рёбра delete

void on\_pb\_DeleteSelectedNodes\_clicked();//delete

void on\_pb\_CreateNode\_clicked();//создаем вершину

void on\_pb\_CreateEdge\_clicked();//Создаём дугу

void on\_pb\_ClearAll\_clicked();//Очистить всё

void on\_pb\_IncidenceListFile\_clicked(); // Создание графа из списка инцидентности

void on\_pushAlgorithm\_clicked();

void on\_pushStop\_clicked();

void on\_action\_Open\_file\_triggered();

void on\_action\_Save\_File\_triggered();

void on\_action\_createNode\_triggered();

void on\_action\_createEdge\_triggered();

void on\_action\_Algoritm\_triggered();

void on\_action\_delEdge\_triggered();

void on\_action\_clear\_all\_triggered();

void on\_action\_triggered();

void on\_pushFind\_clicked();

void on\_pushButton\_clicked();

void on\_lineEdit\_AlgSource\_cursorPositionChanged(int arg1, int arg2);

};

#endif // MAINWINDOW\_H

Mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "graphwidget.h"

#include <QCoreApplication>

#include <QTextStream>

#include <cstring>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

, m\_settings("VEB","MyProgram") //Для сохранения настроек

{

ui->setupUi(this);

readSettings();//Для восстановления настроек

// Создаётся сигнал для управления удалением вершин

ui->actionDeleteItem->setShortcut(tr("Delete"));

ui->actionDeleteItem->setStatusTip(tr("Удалить выбранные вершины из графа"));

connect(ui->actionDeleteItem, SIGNAL(triggered(bool)), this, SLOT(deleteNode()));

connect(this,SIGNAL(changeOpenFile()),

SLOT(on\_OpenButton\_clicked())); //удалить

// QAction\* pactOpen = new QAction("Open File", 0);

ui->action\_Open\_file->setText("&Open...");

ui->action\_Open\_file->setShortcut(QKeySequence("CTRL+O"));

ui->action\_Open\_file->setToolTip("Open Document");

ui->action\_Open\_file->setStatusTip("Open an existing file");

ui->action\_Open\_file->setWhatsThis("Open an existing file");

ui->action\_Open\_file->setIcon(QPixmap(":/cars/fileopen.png"));

//QAction\* pactSave = new QAction("Save File", 0);

ui->action\_Save\_File->setText("&Save");

ui->action\_Save\_File->setShortcut(QKeySequence("CTRL+S"));

ui->action\_Save\_File->setToolTip("Save Document");

ui->action\_Save\_File->setStatusTip("Save the file to disk");

ui->action\_Save\_File->setWhatsThis("Save the file to disk");

ui->action\_Save\_File->setIcon(QPixmap(":/cars/filesave.png"));

//connect(pactSave, SIGNAL(triggered()), SLOT(slotSave()));

QToolBar\* ptbFile = new QToolBar("File Operations");

ptbFile->addAction(ui->action\_Open\_file);

ptbFile->addAction(ui->action\_Save\_File);

ptbFile->addAction(ui->action\_clear\_all);

ptbFile->addAction(ui->action);

ui->actionMenuFileExit->setShortcuts(QKeySequence::Quit);

ui->actionMenuFileExit->setStatusTip(tr("Выйти из программы"));

connect(ui->actionMenuFileExit, SIGNAL(triggered()), this, SLOT(close()));

graph = ui->graphicsView;

graph->setDragMode(QGraphicsView::RubberBandDrag);

graph->setOptimizationFlags(QGraphicsView::DontSavePainterState);

graph->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);

graph->setTransformationAnchor(QGraphicsView::AnchorUnderMouse);

graph->setInteractive(true);

graph->setRenderHint(QPainter::Antialiasing);

}

MainWindow::~MainWindow()

{

delete ui;

writeSettings();

}

void MainWindow::readSettings()

{

// Проверить

m\_settings.beginGroup("/Settings");

int nWidth = m\_settings.value("/width", width()).toInt();

int nHeight = m\_settings.value("/height", height()).toInt();

QString nInputPath = m\_settings.value("/fileData", "").toString();

InputPath = nInputPath;

resize(nWidth, nHeight);

m\_settings.endGroup();

}

void MainWindow::writeSettings()

{

m\_settings.beginGroup("/Settings");

m\_settings.setValue("/width", width());

m\_settings.setValue("/height", height());

m\_settings.setValue("/fileData", InputPath);

m\_settings.endGroup();

}

void MainWindow::deleteNode()

// Удаляет выбранные вершины графа

{

graph->deleteSelectedItems();

graph->changeIndecesOfAllVerteces();

}

void MainWindow::on\_pb\_CoordinatesOfAllNode\_clicked()

// Выписывает координаты всех вершин графа

{

QString s = "Координаты вершин: \n\n";

for (int i = 0; i < graph->getListOfNodeSize(); i++)

{

s += QString("%1 - ( %2, %3 )\n").arg(i).arg(graph->getPosOfNode(i).x()).arg(graph->getPosOfNode(i).y());

}

s.chop(1);

ui->textBrowser->setText(s);

}

void MainWindow::on\_pb\_LengthOfEdges\_clicked()

// Выписывает длину всех рёбер графа

{

QString s = "Длины рёбер:\n\n";

int i = 0;

foreach (Edges \*item, graph->getEdges())

{

s += QString("%1. ( %2, %3 ) - %4\n").arg(++i).arg(item->source).arg(item->dest).arg(item->length);

}

s.chop(1);

ui->textBrowser->setText(s);

}

void MainWindow::on\_pb\_DisconnectGraph\_clicked()

// Удалить все рёбра в графе

{

foreach (Node\* Node, graph->listOfNode)

// Для каждой вершины -

{

foreach (Edge \*edge, Node->edges())

// - смотрим все, связанные с ней, рёбра

{

graph->scene->removeItem(edge);

//Убрать ребро из окна

edge->destNode()->removeEdge(edge);

//Убрать ребро из списка рёбер первой вершины

edge->sourceNode()->removeEdge(edge);

//Убрать ребро из списка рёбер второй вершины

delete edge;

}

//delete Node;

}

}

void MainWindow::on\_saveButton\_clicked()

{

QString fileName = QFileDialog::getSaveFileName(this, "Сохранить граф", InputPath, "Graph file (\*.graph)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::WriteOnly))

{

QTextStream out(&file);

int n = graph->getListOfNodeSize();

out << n << endl;

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

out << i << " " << graph->getPosOfNode(i).x() << " " << graph->getPosOfNode(i).y() << endl;

int i = 0; foreach (Edges \*item, graph->getEdges()) i=i+1;

out << i << endl;

foreach (Edges \*item, graph->getEdges())

out << item->source << " " << item->dest << " " << item->length << endl;

file.close();

}

else {

QMessageBox::information(this, "Невозможно открыть файл", file.errorString());

return;

}

} else return;

}

void MainWindow::on\_pb\_RemoveEdges\_clicked() //Удалить рёбра

{

on\_pb\_DisconnectGraph\_clicked();

}

void MainWindow::on\_pb\_DeleteSelectedNodes\_clicked() //Удалить вершины

{

deleteNode();

}

void MainWindow::on\_pb\_CreateNode\_clicked()//создаем вершину

{

graph->addNode();

}

void MainWindow::on\_pb\_CreateEdge\_clicked()

{//Создаём ребро

Node \*source;// = graph->listOfNode.first();

Node \*dest;// = graph->listOfNode.last();

int Source1 = (ui->lineEdit\_Source->text()).toInt();//

int Dest1 = (ui->lineEdit\_Dest->text()).toInt();//

//по номеру надо найти вершину сделать функцию

source = graph ->findNode(Source1);

dest = graph->findNode(Dest1);

graph->addEdge(source,dest);

}

void MainWindow::on\_pb\_ClearAll\_clicked()

{

graph->deleteAllItems();

}

void MainWindow::on\_OpenButton\_clicked()

{

QString fileName = QFileDialog::getOpenFileName(this, "Открыть граф",

/\*path\*/InputPath, "Graph file (\*.graph)");

if (!fileName.isEmpty())

{

QFile file(fileName);

if (file.open(QIODevice::ReadOnly))

{

QTextStream in(&file);

on\_pb\_ClearAll\_clicked();

//graph->getEdges().clear();

graph->listOfNode.clear();

QFileInfo fi(file);

QString fn4 = fi.absolutePath();

InputPath = fn4;

int vertices\_count, edges\_count;

in >> vertices\_count;

for (int i = 0; i < vertices\_count; i++)

{

int id, x, y;

in >> id >> x >> y;

Node \*Node1 = new Node(graph); // Создаём новую вершину

Node1->setIndex(id);

Node1->setPos(x,y);

graph->scene->addItem(Node1); // Добавляем её на экран

graph->listOfNode << Node1; // Добавить эту вершину в список вершин

}

graph->update();// Обновяем экран\*/

bool accept = true;

in >> edges\_count;

for (int i = 0; i < edges\_count; i++)

{

accept = true;

int uid, vid;

unsigned length;

in >> uid >> vid >> length;

Node \*source, \*dest;

//по номеру надо найти вершину

source = graph->findNode(uid);

dest = graph->findNode(vid);

foreach (Edges \*edge, graph->getEdges())

{

if (edge->source == uid && edge->dest == vid)

accept = false;

}

if (accept)

graph->scene->addItem(new Edge(source, dest, length));

}

graph->update();// Обновяем экран

file.close();

//пытаюсь открыть файл ещё раз с начала

if(file.open(QIODevice::ReadOnly | QIODevice::Text))

{

QTextStream in(&file);

ui->textBrowser->setText(in.readAll());

}

file.close();

}

else {

QMessageBox::information(this, "Невозможно открыть файл", file.errorString());

return;

}

}

else return;

}

void MainWindow::on\_pushFind\_clicked()

{

for (auto i : graph->getEdges())

{

i->edge->setColor(QColor(0,0,0));

i->edge->update();

}

// unsigned timer = static\_cast<unsigned>(ui->spinBox->value());

int source = ui->lineEdit\_AlgSource->text().toInt();

for (int i=0; i<graph->getListOfNodeSize(); i++)

{

if (graph->findNode(i)->isSelected())

{

int p = i;

//creating path

while(p != source)

{

for (auto i : graph->getEdges())

{

if (i->dest == p && i->source == shortPath[p])

{

for (int j=0; j<255; j+=10)

{

i->edge->setColor(QColor(j, 0, 0));

i->edge->update();

// Sleep(timer/40);

QCoreApplication::processEvents();

}

}

}

//Sleep(timer/2);

p = shortPath[p];

}

}

}

}

void MainWindow::on\_action\_Open\_file\_triggered()

{

on\_OpenButton\_clicked();

}

void MainWindow::on\_action\_Save\_File\_triggered()

{

on\_saveButton\_clicked();

}

void MainWindow::on\_action\_createNode\_triggered()

{

on\_pb\_CreateNode\_clicked();

}

void MainWindow::on\_action\_createEdge\_triggered()

{

on\_pb\_CreateEdge\_clicked();

}

void MainWindow::on\_action\_delEdge\_triggered()

{

on\_pb\_DisconnectGraph\_clicked();

}

void MainWindow::on\_action\_clear\_all\_triggered()

{

on\_pb\_ClearAll\_clicked();

}

// Создание графа из списка инцидентности

void MainWindow::on\_pb\_IncidenceListFile\_clicked()

{

QString FileName = QFileDialog::getOpenFileName(this, "Открыть файл...", InputPath, "Text files (\*.txt)");

if (FileName.isEmpty())

return;

QFile File(FileName);

if(File.open(QIODevice::ReadOnly | QIODevice::Text))

{

on\_pb\_ClearAll\_clicked();

if (!graph->createGraphFromInc(File.readAll())) {

QMessageBox error;

error.setText("Ошибка в файле! Вершины должны быть пронумерованы исключительно числами!");

error.exec();

}

File.close();

//пытаюсь открыть файл eщё раз с начала

if(File.open(QIODevice::ReadOnly | QIODevice::Text))

{

QTextStream in(&File);

ui->textBrowser->setText(in.readAll());

}

File.close();

}

}

void MainWindow::on\_pushAlgorithm\_clicked()

{

std::vector<int> VParent(static\_cast<unsigned>(graph->getListOfNodeSize()), 0);

std::vector<int> SDist(static\_cast<unsigned>(graph->getListOfNodeSize()), INT\_MAX);

std::vector<bool> marked(static\_cast<unsigned>(graph->getListOfNodeSize()), false);

unsigned source = static\_cast<unsigned>(ui->lineEdit\_AlgSource->text().toInt());

SDist[source] = 0;

int dMin;

unsigned vMin = 0, u = 0;

for (int p=0; p<graph->getListOfNodeSize(); p++)

{

if (stop)

{

stop = false;

return;

}

dMin = INT\_MAX;

for (unsigned i=0; i<SDist.size(); i++)

{

if (SDist[i] < dMin && marked[i] == false)

{

dMin = SDist[i];

vMin = i;

}

}

if (dMin == INT\_MAX)

return;

marked[vMin] = true;

graph->findNode(static\_cast<int>(vMin))->setSelected(true);

for (auto i : graph->findNode(static\_cast<int>(vMin))->edges())

{

u = static\_cast<unsigned>(i->destNode()->getIndex());

for (int j=10; j<255; j+=10)

{

i->setColor(QColor(j, 0, 0));

i->update();

//Sleep(timer/40);

QCoreApplication::processEvents();

}

if (SDist[vMin] + i->getLength() < SDist[u])

{

for (int j=10; j<255; j+=10)

{

i->setColor(QColor(255-j, j, 0));

i->update();

// Sleep(timer/40);

QCoreApplication::processEvents();

}

SDist[u] = SDist[vMin] + i->getLength();

graph->findNode(static\_cast<int>(u))->setDist(SDist[u]);

VParent[u] = static\_cast<int>(vMin);

QCoreApplication::processEvents();

// Sleep(timer/2);

for (int j=0; j<255; j+=10)

{

i->setColor(QColor(0, 255-j, 0));

i->update();

// Sleep(timer/40);

QCoreApplication::processEvents();

}

}

else {

// Sleep(timer/2);

for (int j=0; j<255; j+=10)

{

i->setColor(QColor(255-j, 0, 0));

i->update();

// Sleep(timer/40);

QCoreApplication::processEvents();

}

}

}

graph->findNode(static\_cast<int>(vMin))->setSelected(false);

}

shortPath = VParent;

}

void MainWindow::on\_action\_Algoritm\_triggered()

{

on\_pushAlgorithm\_clicked();

}

Edge.cpp:

﻿#include "edge.h"

static const double Pi = 3.14159265358979323846264338327950288419717;

static double TwoPi = 2.0 \* Pi;

Edge::Edge(Node \*sourceNode, Node \*destNode): arrowSize(15)

{

setAcceptedMouseButtons(nullptr);

setCacheMode(DeviceCoordinateCache);

source = sourceNode;

dest = destNode;

source->addEdge(this);

dest->addEdge(this);

adjust();

length = 100;

flow = 0;

color = Qt::black;

timer = new QTimeLine; //создание таймера

timer->setFrameRange(0,500);

timer->setLoopCount(1);

posAnim= new QGraphicsItemAnimation; //создание анимации

posAnim->setTimeLine(timer);

it = new QGraphicsPixmapItem;

}

Edge::Edge(Node \*sourceNode, Node \*destNode, unsigned length) : Edge(sourceNode, destNode)

{

setLength(length);

}

void Edge::setColor(QBrush newColor)

{

color = newColor;

setZValue(999);

}

void Edge::setStyle(Qt::PenStyle st)

{

style = st;

update();

}

Node \*Edge::sourceNode() const

{

return source;

}

Node \*Edge::destNode() const

{

return dest;

}

void Edge::adjust()

{

if (!source || !dest) return;

QLineF line(mapFromItem(source, 0, 0), mapFromItem(dest, 0, 0));

qreal length = line.length();

prepareGeometryChange();

QPointF edgeOffset((line.dx() \* 15) / length, (line.dy() \* 15) / length);

// Рисование простого ребра

if (length > qreal(30.))

{

QPointF edgeOffset((line.dx() \* 15) / length, (line.dy() \* 15) / length);

sourcePoint = line.p1() + edgeOffset;

destPoint = line.p2() - edgeOffset;

}

else

{

sourcePoint = destPoint = line.p1();

}

}

qreal Edge::getLength()

{

return length;

}

int Edge::getFlow()

{

return flow;

}

void Edge::setLength(unsigned value)

{

length = value;

update();

}

void Edge::setFlow(int value)

{

flow = value;

update();

}

void Edge::addFlow(int value)

{

flow += value;

update();

}

QRectF Edge::boundingRect() const

{

if (!source || !dest) return QRectF();

qreal penWidth = 2;

qreal extra = (penWidth + arrowSize)+60;

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;

painter->setRenderHint(QPainter::Antialiasing);

QLineF line(sourcePoint, destPoint);

if (qFuzzyCompare(line.length(), qreal(0.))) return;

// Draw the line

painter->setPen(QPen(color, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->drawLine(line);

QFont font;

font.setPixelSize(15);

painter->setFont(font);

// Draw the arrows

double angle = ::acos(line.dx() / line.length());

if (line.dy() >= 0)

angle = TwoPi - angle;

QPointF destArrowP1 = destPoint + QPointF(sin(angle - Pi / 3) \* arrowSize,

cos(angle - Pi / 3) \* arrowSize);

QPointF destArrowP2 = destPoint + QPointF(sin(angle - Pi + Pi / 3) \* arrowSize,

cos(angle - Pi + Pi / 3) \* arrowSize);

int weight = static\_cast<int>(length);

QString output;

output += "L:" + QString::number(weight);

if (flow != 0)

output += " F:" + QString::number(flow);

if (flow < 0)

{

QPointF bottom = {(sourcePoint.rx()+destPoint.rx())/2, (sourcePoint.ry()+destPoint.ry())/2-20};

painter->drawText(bottom, output);

}

else

painter->drawText((sourcePoint+destPoint)/2, output);

painter->setBrush(Qt::black);

painter->drawPolygon(QPolygonF() << line.p2() << destArrowP1 << destArrowP2);

}

void Edge::animate(QGraphicsScene \*scene, bool auto1)

{

if(sourcePoint == destPoint)

return;

QLineF line(sourcePoint, destPoint);

/\*\*/ double angle = ::acos(line.dx() / line.length());

if (line.dy() >= 0)

angle = TwoPi - angle;

//Пусть объект движется по прямой, а не по дуге

QPointF c = 0.5\*line.p1() + 0.5\*line.p2();// +QPointF(-sin(angle)\*40,cos(Pi - angle)\*40);

// QPointF c = 0.5\*line.p1() + 0.5\*line.p2() +QPointF(-sin(angle)\*40,cos(Pi - angle)\*40);

QPainterPath a;

a.moveTo(sourcePoint);

a.quadTo(c,destPoint);

timer->setDuration(static\_cast<int>(a.length()/0.1));

if (auto1)

{

QPixmap p(":/cars/1.png");

it->setPixmap(p); //установить изображение на объект

it->setOffset(-p.size().width()/2.0,-p.size().height()/2.0); //установка центра

it->setZValue(0.5);

it->setPos(sourceNode()->scenePos());

it->setTransformationMode(Qt::SmoothTransformation);

posAnim->setItem(it);

}

/\*\*/

// эта часть кода обеспечивает движение

bool ce = true;

for(int i = 0; i <= 10; i++)

{

qreal p = i/10.0;

qreal p\_pred =0;

if(i>0)

p\_pred =(i-1)/10.0;

if(a.angleAtPercent(p) > a.angleAtPercent(p\_pred))

ce = false;

if(i==0)

posAnim->setPosAt(p,sourceNode()->scenePos());

else if(i==10)

posAnim->setPosAt(p,destNode()->scenePos());

else

posAnim->setPosAt(p,a.pointAtPercent(p));

/\*\*/ posAnim->setRotationAt(p,

ce

? -a.angleAtPercent(p)

: 360 - a.angleAtPercent(p)

);/\*\*/

}/\*\*/

if (!auto1)

{

QGraphicsEllipseItem\* a1 = scene->addEllipse(0, 0, 10, 10);

QColor col = Qt::black;

a1->setParentItem(this);

a1->setPos(sourcePoint);

a1->setPen(QPen(col, 0));

a1->setBrush(QBrush(col));

a1->update();

posAnim->setItem(a1);

}

scene->addItem(it);

timer->start();

// Соединение завершения таймера и удаление всех созданных объектов

// connect(timer,&QTimeLine::finished,posAnim,&QGraphicsItemAnimation::deleteLater);

}

Edge.h: ﻿#ifndef EDGE\_H

#define EDGE\_H

#include <QGraphicsItem>

#include <math.h>

#include <QtMath>

#include <QPainter>

#include "node.h"

#include <QtWidgets>

#include <iostream>

#include <QRegion>

class Edge : public QGraphicsItem

{

Q\_INTERFACES(QGraphicsItem)

private:

Node \*source, \*dest;

QPointF sourcePoint;

QPointF destPoint;

qreal arrowSize;

QBrush color;

Qt::PenStyle style;

QGraphicsItemAnimation \*posAnim;

QTimeLine \*timer;

QGraphicsScene\* scene;

unsigned length;

int flow;

public:

Edge(Node \*sourceNode, Node \*destNode);

Edge(Node \*sourceNode, Node \*destNode, unsigned length);

QRectF boundingRect() const;

Node \*sourceNode() const;

Node \*destNode() const;

void adjust();

void setColor(QBrush newColor);

void setStyle(Qt::PenStyle st);

qreal getLength(); //Возвращает длину ребра

int getFlow();

void setLength(unsigned value);

void setFlow(int value);

void addFlow(int value);

QGraphicsPixmapItem \*it; //объект на ребре

void animate(QGraphicsScene \*scene, bool auto1);

protected:

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

};

#endif // EDGE\_H

Graphwidget.h:

﻿#ifndef GRAPHWIDGET\_H

#define GRAPHWIDGET\_H

#include <QGraphicsView>

#include <QList>

#include "edge.h"

#include "node.h"

struct Edges

{

Edge \*edge;

int source;

int dest;

int length;

};

class GraphWidget : public QGraphicsView // Виджет для рисования вершин и рёбер

{

Q\_OBJECT

public:

QGraphicsScene \*scene; // Окно, на котором рисуются все элементы

QList<Node \*> listOfNode; // Список вершин, отображающихся на окне

Node\* findNode(int val/\*,QList<Node \*> listOfNode\*/);//по номеру надо найти вершину

explicit GraphWidget(QWidget \*parent = 0);//Конструктор

QPointF getPosOfNode(int index);// Возвращает местоположение вершины с номером index

int getListOfNodeSize();//Возвращает количество вершин

void changeIndecesOfAllVerteces();//Переименовывет вершины после удаления некоторых вершин

QVector<Edges \*> getEdges();

void addNode();

void deleteEdge(Edge \*edge);

Node\* addNode1(QPointF position,int t);

Edge \*addEdge(Node \*source, Node \*dest, unsigned length);

void addEdge(Node \*source, Node \*dest);

void deleteAllItems();//Удаляет все вершины

bool createGraphFromInc(QString temp);

QString add\_spaces\_and\_simplifie(QString str\_for\_work);

public slots:

void deleteSelectedItems();//Удаляет выбранные вершины

protected:

#ifndef QT\_NO\_WHEELEVENT

void wheelEvent(QWheelEvent \*event) Q\_DECL\_OVERRIDE;

#endif

void scaleView(qreal scaleFactor);

};

#endif // GRAPHWIDGET\_H

Graphwidget.cpp:

#include "graphwidget.h"

#include "iostream"

#include <QKeyEvent>

GraphWidget::GraphWidget(QWidget \*parent): QGraphicsView(parent)//Конструктор

{

scene = new QGraphicsScene(this);

setScene(scene);

}

void GraphWidget::addNode()//Добавляем вершину

{

Node \*node = new Node(this); // Создаём новую вершину

scene->addItem(node); // Добавляем её на экран

listOfNode << node; // Добавить эту вершину в список вершин

changeIndecesOfAllVerteces(); // Поставить новый индекс вершине

int ii = node->getIndex();

node->setPos(QPointF(50+ii\*10, 50+ii\*10));

update(); // Обновяем экран

}

void GraphWidget::deleteEdge(Edge \*edge)

{

scene->removeItem(edge);

edge->destNode()->removeEdge(edge);

edge->sourceNode()->removeEdge(edge);

delete edge;

};

Node\* GraphWidget::addNode1(QPointF position,int t)

//Добавляем вершину с заданной позицией и номером

{

Node \*node = new Node(this); // Создаём новую вершину

node->setIndex(t);

scene->addItem(node); // Добавляем её на экран

listOfNode << node; // Добавить эту вершину в список вершин

node->setPos(position);

update(); // Обновяем экран

return node;

};

Edge \* GraphWidget::addEdge(Node \*source,Node \*dest, unsigned length)

{

Edge \*edge = new Edge(source, dest, length);

scene->addItem(edge);

return edge;

}

void GraphWidget::addEdge(Node \*source, Node \*dest)

{

scene->addItem(new Edge(source, dest));

};

QPointF GraphWidget::getPosOfNode(int index)

// Возвращает местоположение вершины с номером index

{

return listOfNode.at(index)->pos();

}

int GraphWidget::getListOfNodeSize()

//Возвращает количество вершин

{

return listOfNode.size();

}

void GraphWidget::changeIndecesOfAllVerteces()

//Переименовывет вершины после удаления некоторых вершин

{

foreach (Node \*itemNode, listOfNode)

{

itemNode->setIndex(listOfNode.indexOf(itemNode));

itemNode->update();

}

}

QVector<Edges \*> GraphWidget::getEdges()

// Возвращает список всех рёбер графа

{

QVector<Edges \*> EdgesList;

foreach (Node \*Node, listOfNode)

{

foreach (Edge \*edge, Node->edges())

{

if (Node->getIndex() == edge->destNode()->getIndex()) continue;

Edges \*curr = new Edges;

curr->source = Node->getIndex(); // В каждую запись включает первую вершину ребра

curr->dest = edge->destNode()->getIndex(); // Вторую вершину ребра

curr->length = (int)edge->getLength(); // Длину ребра

curr->edge = edge; // Ссылку на само ребро

EdgesList << curr;

}

}

return EdgesList;

}

void GraphWidget::deleteSelectedItems()

//Удаляет выбранные вершины

{

foreach (QGraphicsItem \*itemNode, scene->selectedItems())//Для каждой выбранной вершины

{

scene->removeItem(itemNode);//Убрать её из окна

listOfNode.removeAt(listOfNode.indexOf((Node \*)itemNode)); //Убрать из списка вершин

foreach (Edge \*itemEdge, ((Node \*)itemNode)->edges()) //Для каждого ребра, связанного с данной вершиной

{

// ???? пытаюсь удалить автомобиль

if(!itemEdge->it->pixmap().isNull()) // Удалить со сцены аним. объект

scene->removeItem(itemEdge->it);

scene->removeItem(itemEdge);//Убрать ребро из окна

itemEdge->destNode()->removeEdge(itemEdge);//Убрать ребро из списка рёбер первой вершины

itemEdge->sourceNode()->removeEdge(itemEdge);//Убрать ребро из списка рёбер второй вершины

delete itemEdge;//Удалить само ребро

}

delete itemNode;//Удалить саму вершину

}

}

void GraphWidget::deleteAllItems() //Удаляет все вершины

{

foreach (Node \*itemNode, listOfNode)

{

scene->removeItem(itemNode);//Убрать её из окна

foreach (Edge \*itemEdge, itemNode->edges())

//Для каждого ребра, связанного с данной вершиной

{

scene->removeItem(itemEdge);//Убрать ребро из окна

itemEdge->destNode()->removeEdge(itemEdge);

//Убрать ребро из списка рёбер первой вершины

itemEdge->sourceNode()->removeEdge(itemEdge);

//Убрать ребро из списка рёбер второй вершины

delete itemEdge;//Удалить само ребро

}

itemNode->edges().clear();

delete itemNode;//Удалить саму вершину

}

listOfNode.clear();

scene->clear();

}

#ifndef QT\_NO\_WHEELEVENT

//! [5]

void GraphWidget::wheelEvent(QWheelEvent \*event)

{

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

}

//! [5]

#endif

//! [7]

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);

}

//! [7]

bool GraphWidget::createGraphFromInc(QString temp) // Создать граф из списка инц.

{

QList<QString> nodes;

QList<QStringList> child\_of\_nodes;

temp = temp.trimmed();

QTextStream stream(&temp);

while(!stream.atEnd())

{

QString line = add\_spaces\_and\_simplifie(stream.readLine(220));

if(line.isEmpty()) continue;

QStringList list = line.split(":");

if(list.first() == list.last()) return false;

QString t = list.first().trimmed();

if (nodes.count(t) == 0)

nodes << t;

QString t2 = list.last().trimmed();

QStringList list\_of\_children = t2.split(" , ",QString::SkipEmptyParts);

child\_of\_nodes << list\_of\_children;

}

QList<QString> all\_n; // Все вершины

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

if(!all\_n.contains(nodes[i]))

all\_n << nodes[i];

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

for(int k = 0; k < child\_of\_nodes[i].size();k++)

if(!all\_n.contains(child\_of\_nodes[i][k]))

all\_n << child\_of\_nodes[i][k];

//проверка, что все вершины это числа

bool ok = true;

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

for(int k = 0; k < child\_of\_nodes[i].size();k++)

{

child\_of\_nodes[i][k].toInt(&ok);

}

if (!ok)

{

return false;

}

int radius = 200;

double segments = (/\*TwoPi\*/6.28)/all\_n.size();

if (all\_n.size() >= 16)

radius = all\_n.size()\*70/6;

double current\_angle = 0;

foreach(QString str,all\_n)

{

// Создание всех вершин и размещение их по кругу

QPointF pos(radius\*cos(current\_angle)+250,radius\*sin(current\_angle)+250);

int k = str.toInt();

//чтобы получить целые координаты для сохранения в файл

int a, b;

a = static\_cast<int>(pos.rx());

b = static\_cast<int>(pos.ry());

pos.setX(a); pos.setY(b);

addNode1(pos,k);

update();

current\_angle += segments;

}

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

{

Node \* a = findNode(nodes[i].toInt());

//по номеру надо найти вершину

for(int k = 0; k < child\_of\_nodes[i].size(); k++)

{

Node\* b = findNode(child\_of\_nodes[i][k].toInt());

if (a && b) // Елси найдены

{

addEdge(a, b);

}

else

continue;

}

}

return true;

}

QString GraphWidget::add\_spaces\_and\_simplifie(QString str\_for\_work)

{

QChar symbs[2]{':',','};

for(int i = 0; i < 2; i++)

{

int a = -2;

while((a = str\_for\_work.indexOf(symbs[i],a+2))!=-1)

{

{

str\_for\_work.insert(a, ' ');

str\_for\_work.insert(a+2, ' ');

}

}

}

str\_for\_work = str\_for\_work.simplified();

return str\_for\_work;

}

Node\* GraphWidget::findNode(int val)//по номеру надо найти вершину

{

Node \*a;

//по номеру надо найти вершину

foreach (Node \*itemNode, listOfNode)

{

if (itemNode->getIndex() == val)

{

a = static\_cast<Node\*>(itemNode); // Привести текушую вершину к типу Node

return a;

}

}

return nullptr;

}