Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по творческому заданию (часть 2)**

Дисциплина: основы алгоритмизации и программирования

Тема: Решение задачи коммивояжёра

Вариант 20

Выполнил:

Студент группы РИС-20-1б

Кузнецов Михаил Сергеевич

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Постановка задачи**

1. Взять свой вариант из лабораторной работы по графам. Модифицировать граф таким образом, чтобы для этого графа можно было решить задачу коммивояжера. Можно придумать альтернативную задачу, которую можно решить методом ветвей и границ.
2. Проработать визуализирующую часть, а именно—построение графа, средствами QT.
3. Исходные данные должны приниматься с консоли и через графический интерфейс.

**Анализ задачи**

1. Определить какие действия необходимо выполнить для решения задачи.

* Разработать интерфейс программы средствами qt.
* Реализовать поля ввода данных.
* При нажатии кнопки, считать данные, выполнить расчёты, вывести результат.

1. Данные о вершинах и рёбрах хранятся в классах TOP и EDGE.

*class* **TOP**

{

*public*:

**TOP**();

**TOP**(int X,int Y);

*public*:

int x;

int y;

bool mark;

};

*class* **EDGE**

{

*public*:

**EDGE**();

**EDGE**(int X1,int Y1,int X2,int Y2, int Type,int Mass);

*public*:

int x1;

int y1;

int x2;

int y2;

int type;

int mass;

};

1. Работа с интерфейсом, расчёт и вывод оптимального пути реализовано в классе MainWindow.

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

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

~***MainWindow***();

void **SwitchState**();

void **CalculateRoute**();

*public*:

QButtonGroup\* G=*new* QButtonGroup;

QGridLayout \*grid;

myGLWidget \*openGLW;

QPushButton \*btn\_calc;

QLineEdit\* LE\_ent1;

QLineEdit\* LE\_ent2;

QLabel \*L\_res;

*private*:

Ui::MainWindow \*ui;

};

1. Отслеживание событий мыши и изменение текущего графа реализовано в классе myGLWidget.

*class* **myGLWidget** : *public* QOpenGLWidget

{

Q\_OBJECT

*public*:

**myGLWidget**( QWidget \*parent);

*public* slots:

void **redraw**();

*protected*:

void ***paintEvent***(QPaintEvent \*event) *override*;

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

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

void ***mouseMoveEvent***(QMouseEvent\*event) *override*;

*private*:

myPainter \*mypainter;

};

1. Алгоритмы отрисовки графа находятся в классе myPainter.

*class* **myPainter**

{

*public*:

**myPainter**();

*public*:

void **draw**(QPainter \*painter, QPaintEvent \*event);

void **draw\_edge**(int x1,int y1,int x2,int y2, int type,QPainter \*painter,QString text);

void **draw\_top**(int x,int y,QString text,QPainter \*painter );

QBrush front;

*private*:

QBrush back;

QFont glFont;

QPen textPen;

};

**Функция main()**

#include "mainwindow.h"

#include "top.h"

#include "edge.h"

#include <QApplication>

#include <QLineEdit>

QString status="none";

int len=0;

int mem\_x;

int mem\_y;

int mem\_num;

QVector <TOP> graph1;

QVector <EDGE> graph2;

QVector <QVector<int>> matrix;

QString statusBook[]={"top","edge1.1","edge2.1","move","del","none"};

QString text\_for\_arrows;

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

{

QApplication a(*argc*, argv);

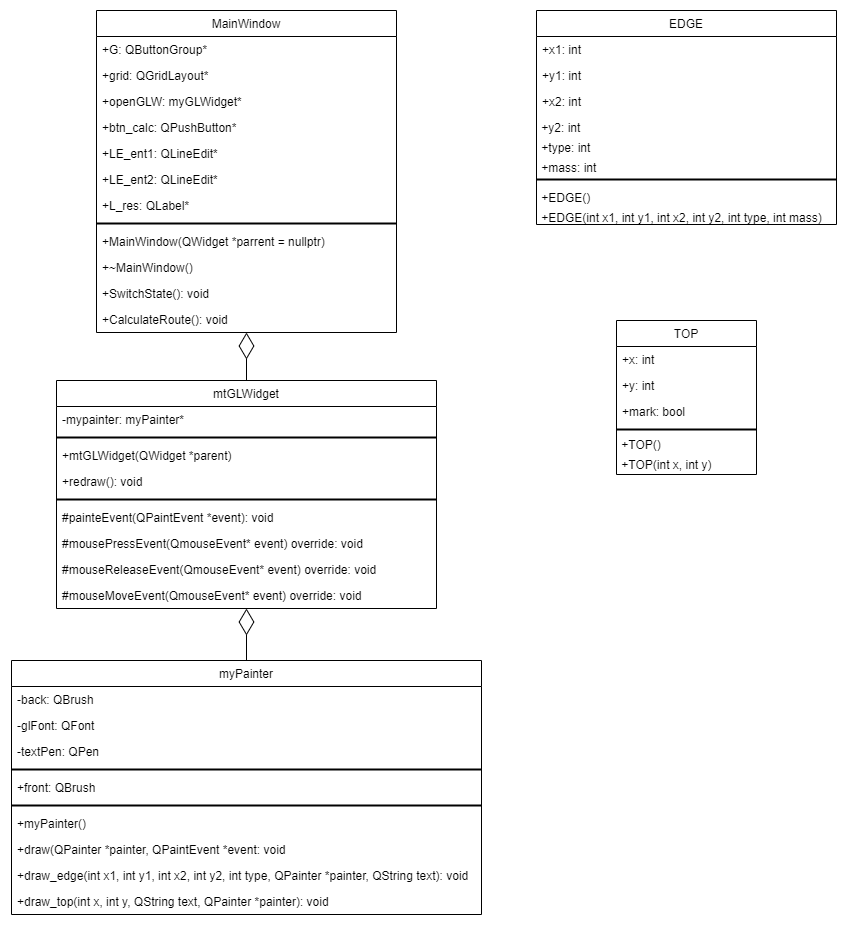
MainWindow\* w = *new* MainWindow;

w->show();

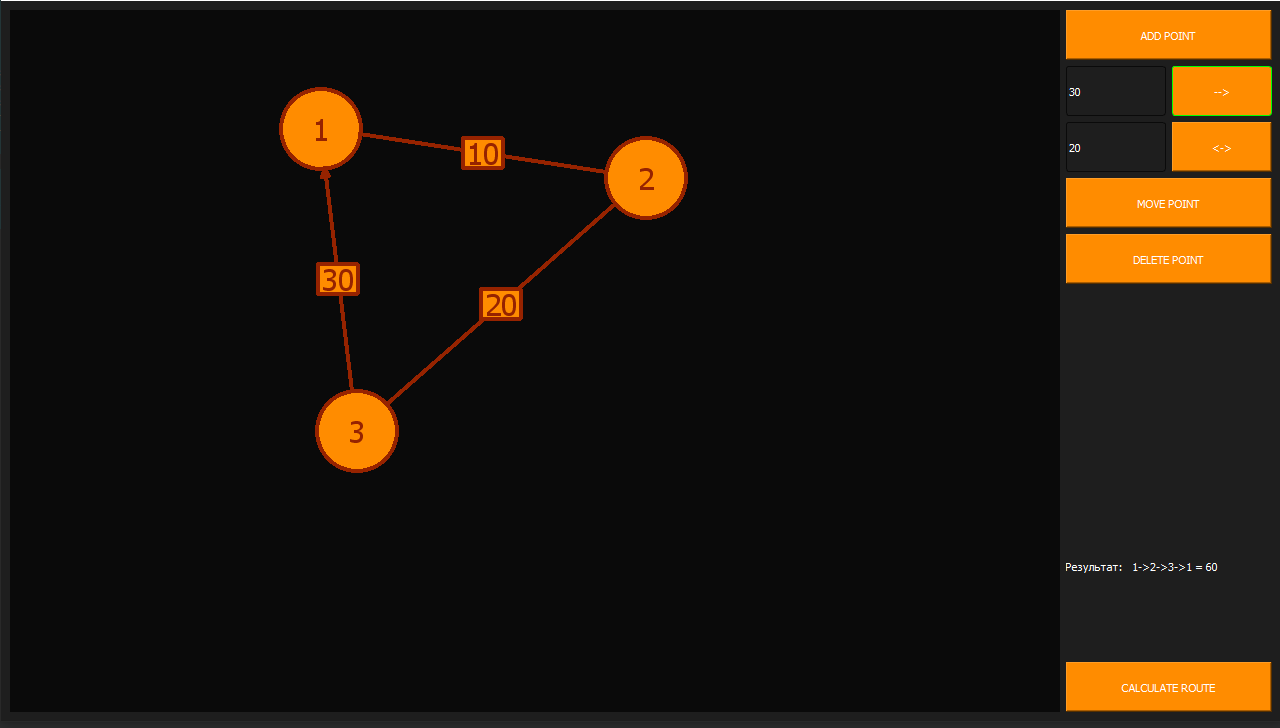
*return* a.exec();

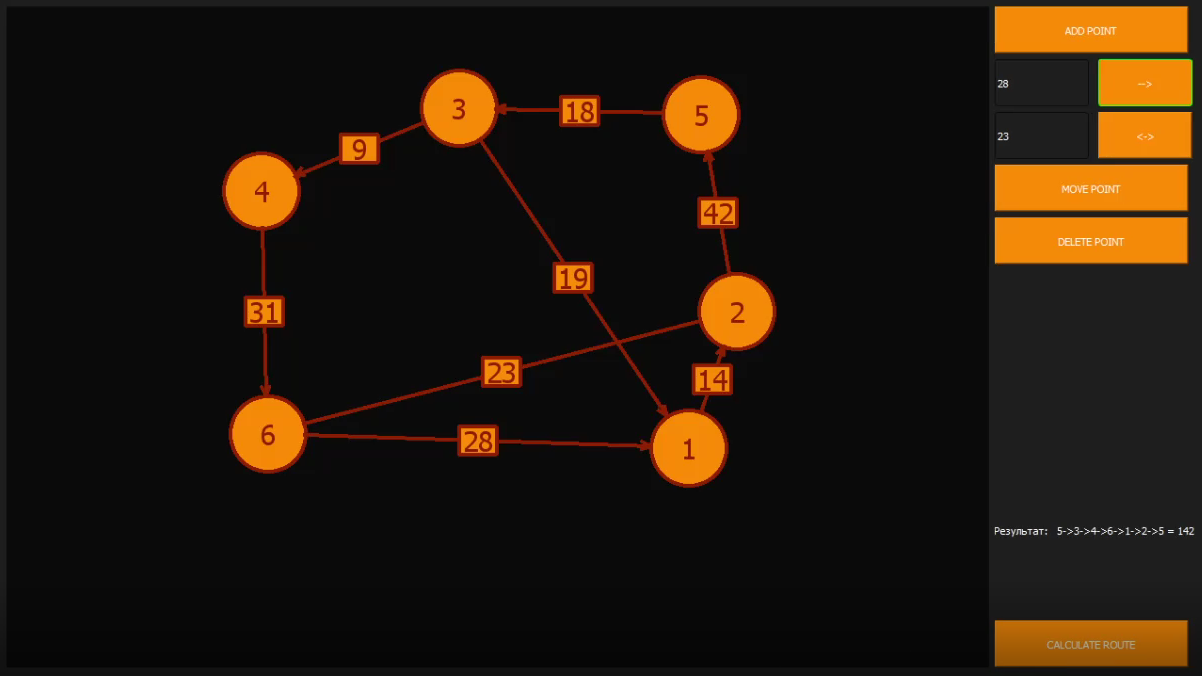
}

**UML диаграмма**

****

**Скриншоты тестов**

****

****