Министерство образования и науки Российской Федерации

Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по творческой работе часть 2**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Решение задачи коммивояжера

Вариант 18

Выполнил:

Студент группы РИС-20-1б

Тараканов Д. М.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

Пермь

2021 год

**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 или через графический интерфейс.

**Анализ задачи**

1. Определить какие действия необходимо выполнить для решения задачи:

* Разработать интерфейс программы средствами qt
* Реализовать поля ввода данных
* Реализовать добавление и удаление данных матрицы смежности с полей для ввода данных при нажатии соответствующих кнопок
* Реализовать вывод результата расчёта или сообщения об ошибке при нажатии на кнопку рассчитать оптимальный путь

1. С какими типами данных действие надо сделать, в каком виде эти данные будут представлены:

* Экземпляры всех элементов интерфейса хранятся в переменной ui.

Ui::PolishNotation\* ui;

* Вектора вершин графа tops типа TOP. Координаты вершин, позиция вершины в графе хранятся в классе TOP.

QVector <TOP> tops;

*class* **TOP**

{

*public*:

**TOP**();

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

*public*:

int x;

int y;

int pos;

};

* Вектора связей вершин графа edges типа EDGE. Координаты вершин начала и конца, тип связи, вес связи хранятся в классе EDGE.

QVector <EDGE> edges;

*class* **EDGE**

{

*public*:

**EDGE**();

**EDGE**(int X\_from, int Y\_from, int X\_to, int Y\_to, int Weight, int Type);

*public*:

int x\_from;

int y\_from;

int x\_to;

int y\_to;

int weight;

int type;

};

* Вектор векторов map типа int – матрица смежности

QVector <QVector<int>> map;

1. Для решения задачи данные были представлены в виде строки с оптимальным маршрутом типа QString и длине этого маршрута типа int
2. Реализация операций ввода и вывода данных:

* Ввод элементов реализован при помощи добавления веса связей между вершинами в матрицу смежности, а также добавления координат вершин и связей между ним. Данные для ввода берутся из соответствующих LineEdit полей. На примере нажатия на кнопку “Добавить связь (->) рассматривается как заполняется таблица смежности и добавляются новые связи в графе.

void TSP::**on\_Btn\_Add\_Edge\_1\_clicked**()

{

ui->L\_Errors->setText("");

ui->L\_Path\_length->setText("");

ui->L\_Errors->setText("");

*if*(tops.size()>1)

{

QString new\_edge = ui->LE\_Add\_Edge\_1\_Top\_1->text();

int from = new\_edge.toInt()-1;

new\_edge = ui->LE\_Add\_Edge\_1\_Top\_2->text();

int to = new\_edge.toInt()-1;

new\_edge = ui->LE\_Add\_Edge\_1\_Weight->text();

int weight = new\_edge.toInt();

*if*(from == -1 || to == -1 || weight == 0)

{

ui->L\_Errors->setText("Заполните пустые поля Добавить ребро (->)!");

}

*else* *if*(from + 1 > knot\_count || to + 1 > knot\_count)

{

ui->L\_Errors->setText("Введеной вершины в графе нет!");

}

*else*

{

EDGE edge(tops[from].x, tops[from].y, tops[to].x, tops[to].y, weight, 1);

edges.push\_back(edge);

map[from][to] = weight;

map[to][from] = 0;

openGlW->redraw();

}

}

ui->LE\_Add\_Edge\_1\_Top\_1->setText("");

ui->LE\_Add\_Edge\_1\_Top\_2->setText("");

ui->LE\_Add\_Edge\_1\_Weight->setText("");

}

* Вывод элементов реализован двумя способами. Первый это отрисовка графа, а второй вывод результат работы программы в соответствующие поля. Пример методов отрисовки графа которые используются при добавлении и удалении вершин, построение связей между вершинами графа послужат методы draw, draw\_edge, draw\_top.

void myPainter::**draw**(QPainter \*painter, QPaintEvent \*event)

{

painter->fillRect(event->rect(),background);

painter->setBrush(shape);

painter->setPen(textPen);

painter->setFont(glFont);

*extern* QVector <TOP> tops;

*extern* QVector <EDGE> edges;

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

{

draw\_top(tops[i].x, tops[i].y, painter, QString::number(i+1));

}

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

{

draw\_edge(edges[i].x\_from, edges[i].y\_from, edges[i].x\_to, edges[i].y\_to, painter, edges[i].weight, edges[i].type);

}

}

void myPainter::**draw\_top**(int x, int y, QPainter \*painter, QString name)

{

painter->drawEllipse(QRectF(x - knot\_R, y - knot\_R, 2 \* knot\_R, 2 \* knot\_R));

int text\_x = x - knot\_R / 2;

int text\_y = y - text\_size / 2;

painter->drawText(QRect(text\_x, text\_y, knot\_R, text\_size), Qt::*AlignCenter*, name);

}

void myPainter::**draw\_edge**(int x1, int y1, int x2, int y2, QPainter \*painter, int weight, int type)

{

double line\_length = sqrt((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1));

int l\_x1 = (x2 - x1) / line\_length \* knot\_R + x1;

int l\_y1 = (y2 - y1) / line\_length \* knot\_R + y1;

int l\_x2 = x2 - (x2 - x1) / line\_length \* knot\_R;

int l\_y2 = y2 - (y2 - y1) / line\_length \* knot\_R;

painter->drawLine(l\_x1, l\_y1, l\_x2, l\_y2);

*if*(type == 1)

{

double C;

*if* (x2==l\_x2 && l\_y2 < y2)

{

C = pi \* 3 / 2;

}

*else* *if* (x2==l\_x2 && l\_y2 > y2)

{

C = pi / 2;

}

*else* *if* (y2 == l\_y2 && l\_x2 < x2)

{

C = pi;

}

*else* *if* (y2 == l\_y2 && l\_x2 < x2)

{

C = 0;

}

*else* *if* (y2 < l\_y2 && l\_x2 < x2)

{

C = atan(1.0 \* (y2 - l\_y2) / (x2 - l\_x2)) + pi;

}

*else* *if* (y2 > l\_y2 && l\_x2 < x2)

{

C = atan(1.0 \* (y2 - l\_y2) / (x2 - l\_x2)) + pi;

}

*else* *if* (y2 < l\_y2 && l\_x2 > x2)

{

C = atan(1.0 \* (y2 - l\_y2) / (x2 - l\_x2)) + 2 \* pi;

}

*else* *if* (y2 > l\_y2 && l\_x2 > x2)

{

C = atan(1.0 \* (y2 - l\_y2) / (x2 - l\_x2)) + 0;

}

int a\_x1 = cos(C + arrowAngle) \* arrowLength + l\_x2;

int a\_x2 = cos(C - arrowAngle) \* arrowLength + l\_x2;

int a\_y1 = sin(C + arrowAngle) \* arrowLength + l\_y2;

int a\_y2 = sin(C - arrowAngle) \* arrowLength + l\_y2;

painter->drawLine(l\_x2, l\_y2, a\_x1, a\_y1);

painter->drawLine(l\_x2, l\_y2, a\_x2, a\_y2);

}

int text\_x = (l\_x2 + l\_x1) / 2 - knot\_R / 2;

int text\_y = (l\_y2 + l\_y1) / 2 - text\_size / 2;

painter->drawRect(QRectF(text\_x, text\_y, knot\_R, text\_size));

painter->drawText(QRect(text\_x, text\_y, knot\_R, text\_size), Qt::*AlignCenter*, QString::number(weight));

}

1. Разбор классов

* Класс TSP отвечает за работу кнопок интерфейса

*class* **TSP** : *public* QMainWindow

{

Q\_OBJECT

*public*:

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

~***TSP***();

myGlWidget\* openGlW;

*private* slots:

void **on\_Btn\_Add\_Top\_clicked**();

void **on\_Btn\_Del\_Top\_clicked**();

void **on\_Btn\_Add\_Edge\_1\_clicked**();

void **on\_Btn\_Add\_Edge\_2\_clicked**();

void **on\_Btn\_Calculate\_clicked**();

void **on\_L\_Clear\_clicked**();

*private*:

Ui::TSP \*ui;

};

* Класс myGlWidget отвечает за графическую реализацию работы графа

*class* **myGlWidget** : *public* QOpenGLWidget

{

Q\_OBJECT

*public*:

**myGlWidget**(QWidget \*parent);

void **redraw**();

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

*private*:

myPainter\* mypainter;

};

* Класс myPainter отвечает за отрисовку вершин и связей графа

*class* **myPainter**

{

*public*:

**myPainter**();

*public*:

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

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

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

QBrush shape;

*private*:

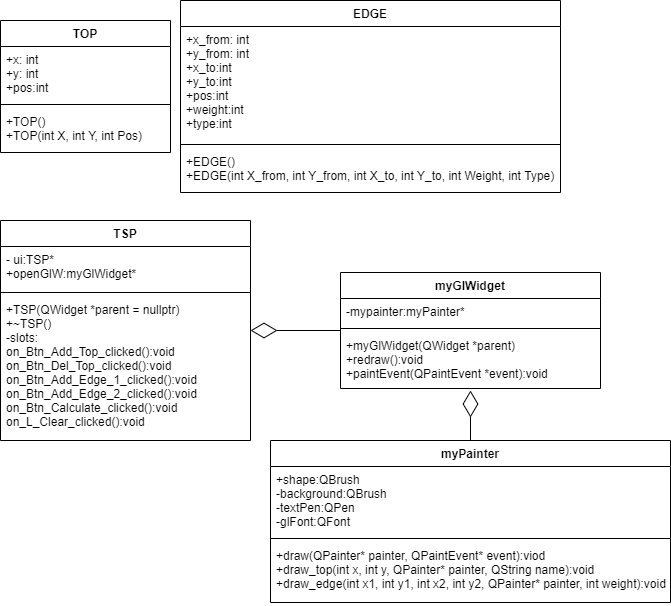
QBrush background;

QPen textPen;

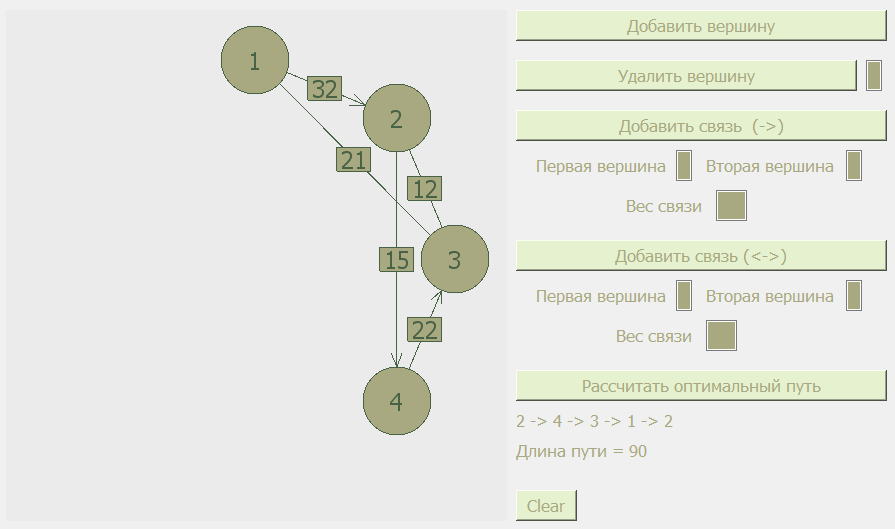
QFont glFont;

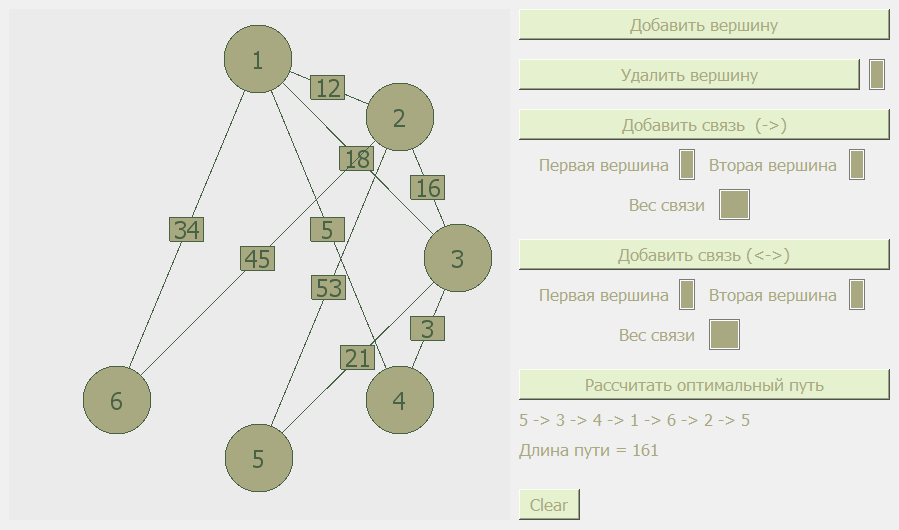
};

**UML диаграмма классов**



**Скриншоты тестов**

****

****