Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе

Тема: «Решение задачи Коммивояджера»

Выполнил

Студент группы РИС-22-2б

Худеньких В.Д.

Проверил доц. Кафедры ИТАС

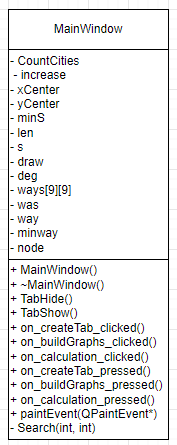
Полякова О.А.

Пермь 2023

# Постановка задачи

Реализовать алгоритмы для собственного варианта графа, имеющего не менее 6 вершин.  
  
Алгоритмы:  
1. Обход в ширину.  
2. Обход в глубину.  
3. Алгоритм Дейкстры.  
Требования:  
1. Пользовательский интерфейс на усмотрение разработчика с условием кроссплатформенности (поощряется использование Qt или иных фреймворков)  
2. Визуализация графа с использованием любой доступной графической библиотеки (SFML, SDL, OpenGL и подобных)  
3. Реализованные алгоритмы должны справляться как с графом, представленным в задании варианта, так и с другими на усмотрение проверяющего.  
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).  
5. Выполнить отчет.

UML – диаграмма



Код программы

**edge.cpp**

#include "edge.h"

EDGE::**EDGE**()

{

/\*

x\_from = 0;

y\_from = 0;

x\_to = 0;

y\_to = 0;

weight = 0;

type = 0;

\*/

}

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

{

x\_from = X\_from;

y\_from = Y\_from;

x\_to = X\_to;

y\_to = Y\_to;

weight = Weight;

type = Type;

}

**glwidget.cpp**

#include "glwidget.h"

#include "top.h"

#include "edge.h"

#include <QPainter>

extern QVector<TOP> tops;

extern QVector<EDGE> edges;

GlWidget::**GlWidget**(QWidget \*parent)

:QOpenGLWidget(*parent*)

{

mypainter=new Painter;

}

void GlWidget::**redraw**()

{

this->update();

}

void GlWidget::***paintEvent***(QPaintEvent \*event)

{

QPainter painter;

painter.begin(this);

mypainter->draw(*&painter*,*event*);

painter.end();

}

**painter.cpp**

#include "painter.h"

#include "top.h"

#include "edge.h"

#include <QPainter>

#include <QPaintEvent>

#include <QtMath>

const double pi = 3.1415;

const int text\_size = 24;

const int knot\_R = 34;

const int arrowLength = 15;

const double arrowAngle=pi/8;

Painter::**Painter**()

{

shape = QBrush(QColor(234, 255, 0));

background = QBrush(QColor(34, 255, 0));

textPen = QPen(QColor(0, 0, 0));

glFont.setPixelSize(text\_size);

}

void Painter::**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 Painter::**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 Painter::**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));

}

**top.cpp**

#include "top.h"

TOP::**TOP**()

{

/\*

x=0;

y=0;

pos=0;

\*/

}

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

{

x=X;

y=Y;

pos=Pos;

}

**tsp.cpp**

#include "tsp.h"

#include "ui\_tsp.h"

#include "top.h"

#include "edge.h"

#include <QPainter>

#include <QtMath>

#include <QIntValidator>

extern QVector<TOP> tops;

extern QVector<EDGE> edges;

extern QVector<QVector<int>> map;

const int graph\_R = 200;

const int shift\_x = 250;

const int shift\_y = 250;

const double pi = 3.1415;

const int max\_knot\_count = 8;

bool flag[max\_knot\_count] = {0, 0, 0, 0, 0, 0, 0, 0};

int knot\_count = 0;

TSP::**TSP**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::TSP)

{

ui->setupUi(this);

openGlW = new GlWidget(this);

ui->verticalLayout->addWidget(*openGlW*, 0, 0);

ui->LE\_Del\_Top\_index->setValidator(new QIntValidator(1,8,this));

ui->LE\_Add\_Edge\_1\_Top\_1->setValidator(new QIntValidator(1,8,this));

ui->LE\_Add\_Edge\_1\_Top\_2->setValidator(new QIntValidator(1,8,this));

ui->LE\_Add\_Edge\_2\_Top\_1->setValidator(new QIntValidator(1,8,this));

ui->LE\_Add\_Edge\_2\_Top\_2->setValidator(new QIntValidator(1,8,this));

ui->LE\_Add\_Edge\_1\_Weight->setValidator(new QIntValidator(1,99,this));

ui->LE\_Add\_Edge\_2\_Weight->setValidator(new QIntValidator(1,99,this));

}

TSP::~***TSP***()

{

delete ui;

}

// функция для обработки событий кнопки "Clear"

void TSP::**on\_L\_Clear\_clicked**()

{

knot\_count=0;

tops.clear();

tops.squeeze();

edges.clear();

edges.squeeze();

map.clear();

map.squeeze();

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

{

flag[i]=0;

}

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

ui->LE\_Del\_Top\_index->setText("");

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

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

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

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

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

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

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

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

openGlW->redraw();

}

// функция для обработки событий конпки "добавить вершину"

void TSP::**on\_Btn\_Add\_Top\_clicked**()

{

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

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

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

if(knot\_count != max\_knot\_count)

{

int id = 0;

while(flag[id] != 0 && id < 7)

{

id++;

}

flag[id] = 1;

double angel = pi \* 3/2 + pi \* 2 / max\_knot\_count \* id;

int knot\_x = cos(angel) \* graph\_R + shift\_x;

int knot\_y = sin(angel) \* graph\_R + shift\_y;

TOP top(knot\_x, knot\_y, id);

tops.push\_back(top);

knot\_count++;

QVector<int> row(knot\_count);

row.fill(0);

map.append(row);

for(int i = 0; i < knot\_count - 1; i++)

{

map[i].append(0);

}

openGlW->redraw();

}

}

// функция для обработки событий кнопки "Добавить ребро (ориентированный)"

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("");

}

// функция для обработки событий кнопки "Добавить ребро (не ориентированный)"

void TSP::**on\_Btn\_Add\_Edge\_2\_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\_2\_Top\_1->text();

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

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

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

new\_edge = ui->LE\_Add\_Edge\_2\_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, 0);

edges.push\_back(edge);

map[from][to] = weight;

map[to][from] = weight;

openGlW->redraw();

}

}

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

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

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

}

//решение задачи коммивояжера

// функция для обработки событий кнопки "Коммивояжёр"

void TSP::**on\_Btn\_Calculate\_clicked**()

{

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

try

{

int X = 1000;

QVector<QVector<int>> matrix = map;

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

{

for(int j = 0; j < knot\_count; j++)

{

if(i==j || matrix[i][j]==0)

{

matrix[i][j] = X;

}

}

}

QVector<int> from;

QVector<int> to;

bool calculation\_flag = true;

int sum = 0;

while(calculation\_flag)

{

//Редукция строк и столбцов

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

{

int row\_min = X;

for(int j = 0; j < knot\_count; j++)

{

row\_min=qMin(row\_min, matrix[i][j]);

}

for(int j = 0; j < knot\_count; j++)

{

if(row\_min != X && matrix[i][j] != X)

{

matrix[i][j] -= row\_min;

}

}

}

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

{

int column\_min = X;

for(int j = 0; j < knot\_count; j++)

{

column\_min=qMin(column\_min, matrix[j][i]);

}

for(int j = 0; j < knot\_count; j++)

{

if(column\_min != X && matrix[j][i] != X)

{

matrix[j][i] -= column\_min;

}

}

}

//Оценка нулей

int max = -1;

int max\_x;

int max\_y;

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

{

for (int j = 0; j < knot\_count; j++)

{

if(matrix[i][j] == 0)

{

int row\_min = X;

for(int k = 0; k < knot\_count; k++)

{

if(k != j)

{

row\_min = qMin(row\_min, matrix[i][k]);

}

}

int column\_min = X;

for(int k = 0; k < knot\_count; k++)

{

if(k != i)

{

column\_min = qMin(column\_min, matrix[k][j]);

}

}

if(row\_min + column\_min > max)

{

max = row\_min + column\_min;

max\_x = i;

max\_y = j;

}

}

}

}

sum += map[max\_x][max\_y];

matrix[max\_y][max\_x] = X;

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

{

matrix[max\_x][i] = X;

matrix[i][max\_y] = X;

}

from.push\_back(max\_x + 1);

to.push\_back(max\_y + 1);

//Проверка

calculation\_flag = false;

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

{

for(int j = 0; j < knot\_count; j++)

{

if(matrix[i][j] != X)

{

calculation\_flag = true;

}

}

}

}

//Построение маршрута

QString path = QString::number(from[0]) + " -> " + QString::number(to[0]);

int new\_from = to[0];

from.erase(from.begin());

to.erase(to.begin());

while(from.size() != 0)

{

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

{

if(from[i] == new\_from)

{

path += " -> " + QString::number(to[i]);

new\_from = to[i];

from.erase(from.begin() + i);

to.erase(to.begin() + i);

}

}

}

ui->L\_Result->setText(path);

QString result = "Длина пути = ";

result += QString::number(sum);

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

}

catch(...)

{

ui->L\_Result->setText("Error: Невозможно вычилить оптимальный путь");

}

ui->LE\_Del\_Top\_index->setText("");

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

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

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

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

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

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

}

// функция для обработки событий конпки "удалить врешину"

void TSP::**on\_Btn\_Del\_Top\_clicked**()

{

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

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

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

if(knot\_count != 0)

{

QString del = ui->LE\_Del\_Top\_index->text();

int del\_index = del.toInt()-1;

if(del\_index == -1)

{

ui->L\_Errors->setText("Заполните пустые поля Удалить вершину!");

}

else if(del\_index+1>knot\_count)

{

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

}

else

{

TOP del\_top = tops[del\_index];

flag[del\_top.pos] = 0;

tops.erase(tops.begin() + del\_index);

QVector <EDGE> new\_edges;

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

{

if(!(edges[i].x\_from == del\_top.x && edges[i].y\_from == del\_top.y) && !(edges[i].x\_to == del\_top.x && edges[i].y\_to == del\_top.y))

{

new\_edges.push\_back(edges[i]);

}

}

edges=new\_edges;

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

{

map[i].erase(map[i].begin() + del\_index);

}

map.erase(map.begin() + del\_index);

knot\_count--;

openGlW->redraw();

}

}

ui->LE\_Del\_Top\_index->setText("");

}

Работа программы

