Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01– «Информатика и вычислительная техника»

**Лабораторная работа**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

**на тему**

**«Задача коммивояжера»**

Выполнил студент гр. ИВТ-23-1б

Бушмаков Кирилл Дмитриевич

Проверил:

Доцент каф. ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

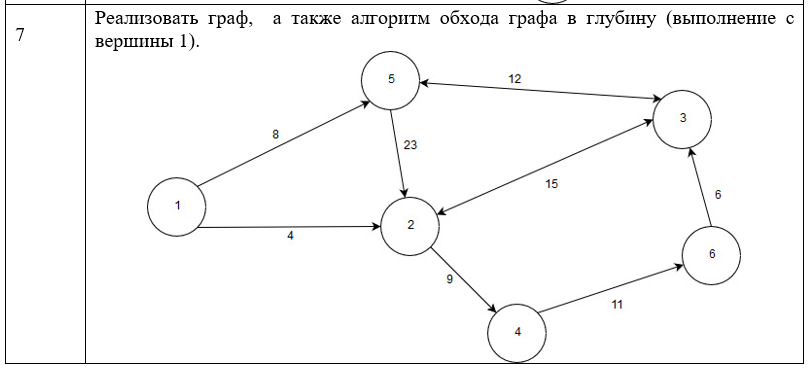
(дата)

г. Пермь, 2024

**Цель и задачи работы**

Целью данной работы реализация задачи коммивояжера и

**Вариант 7:** взят за основу для решения задачи коммивояжера



**UML диаграмма**

На рисунке 1 изображена диаграмма класса для задачи коммивояжера

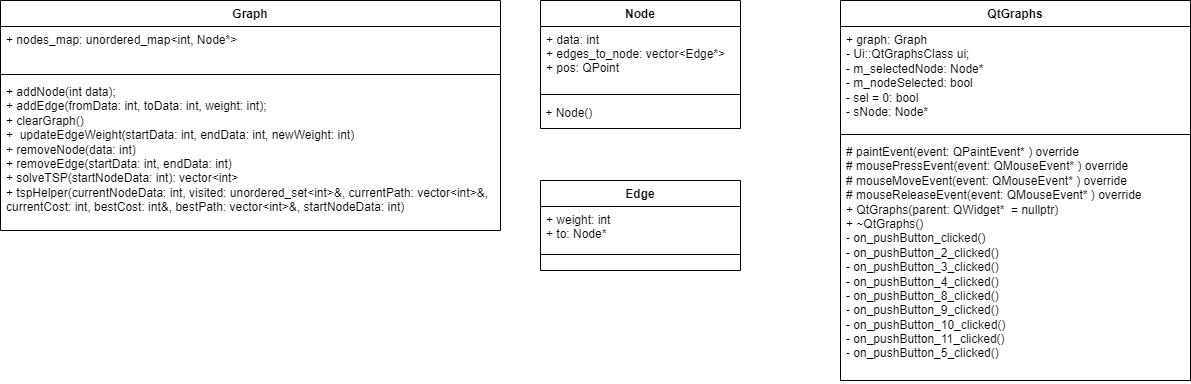


Рисунок 1

**Код программы для Коммивояжера**

**Файл QtGraphs.h**

﻿#pragma once

#include <QWidget>

#include <QMouseEvent>

#include <ui\_QtGraphs.h>

#include <unordered\_map>

#include <unordered\_set>

#include <vector>

#include <stack>

using namespace std;

class Edge;

class Node;

class Graph;

class Node

{

public:

int data;

vector<Edge\*> edges\_to\_node;

QPoint pos;

Node()

{

pos = QPoint(400, 200);

}

};

class Edge

{

public:

int weight;

Node\* to;

};

class Graph

{

public:

unordered\_map<int, Node\*> nodes\_map;

void addNode(int data);

void addEdge(int fromData, int toData, int weight);

void clearGraph();

void updateEdgeWeight(int startData, int endData, int newWeight);

void removeNode(int data);

void removeEdge(int startData, int endData);

vector<int> solveTSP(int startNodeData);

void tspHelper(int currentNodeData, unordered\_set<int>& visited, vector<int>& currentPath, int currentCost, int& bestCost, vector<int>& bestPath, int startNodeData);

};

class QtGraphs : public QMainWindow

{

Q\_OBJECT

public:

QtGraphs(QWidget\* parent = nullptr);

~QtGraphs();

Graph graph;

protected:

void paintEvent(QPaintEvent\* event) override;

void mousePressEvent(QMouseEvent\* event) override;

void mouseMoveEvent(QMouseEvent\* event) override;

void mouseReleaseEvent(QMouseEvent\* event) override;

private:

Ui::QtGraphsClass ui;

Node\* m\_selectedNode;

bool m\_nodeSelected;

bool sel = 0;

Node\* sNode;

void backtrackTSP(Graph& graph, int start, vector<int>& tour, vector<int>& nodes, double distance, vector<int>& shortestPath, double& shortestDistance);

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_8\_clicked();

void on\_pushButton\_9\_clicked();

void on\_pushButton\_10\_clicked();

void on\_pushButton\_11\_clicked();

void on\_pushButton\_5\_clicked();

};

**Файл QtGraphs.cpp**

﻿#include "QtGraphs.h"

#include <QPainter>

#include <vector>

#include <QLineEdit>

#include <QPushButton>

#include <cmath>

#include <unordered\_set>

#include <unordered\_map>

#include <chrono>

#include <thread>

#include <QTimer>

#include <queue>

#include <limits>

#include <random>

const double M\_PI = 3.1415;

void Graph::addNode(int data) {

if (nodes\_map.find(data) == nodes\_map.end()) {

Node\* newNode = new Node;

newNode->data = data;

nodes\_map[data] = newNode;

}

else {

}

}

void Graph::addEdge(int fromData, int toData, int weight) {

for (Edge\* edge : nodes\_map[fromData]->edges\_to\_node) {

if (edge->to == nodes\_map[toData]) {

return;

}

}

Edge\* newEdge = new Edge();

newEdge->to = nodes\_map[toData];

newEdge->weight = weight;

nodes\_map[fromData]->edges\_to\_node.push\_back(newEdge);

}

void Graph::clearGraph()

{

// Óäàëåíèå âñåõ óçëîâ

for (auto& pair : nodes\_map)

{

Node\* node = pair.second;

delete node;

}

// Î÷èñòêà õýø-òàáëèöû

nodes\_map.clear();

}

void Graph::updateEdgeWeight(int startData, int endData, int newWeight) {

if (nodes\_map.find(startData) == nodes\_map.end() || nodes\_map.find(endData) == nodes\_map.end()) {

return;

}

Node\* startNode = nodes\_map[startData];

Node\* endNode = nodes\_map[endData];

for (Edge\* edge : startNode->edges\_to\_node) {

if (edge->to == endNode) {

edge->weight = newWeight;

return;

}

}

}

void Graph::removeNode(int data)

{

for (auto& pair : nodes\_map)

{

Node\* node = pair.second;

vector<Edge\*> edges\_to\_remove;

for (Edge\* edge : node->edges\_to\_node)

{

if (edge->to->data == data)

{

edges\_to\_remove.push\_back(edge);

}

}

for (Edge\* edge : edges\_to\_remove)

{

auto it = find(node->edges\_to\_node.begin(), node->edges\_to\_node.end(), edge);

if (it != node->edges\_to\_node.end())

{

node->edges\_to\_node.erase(it);

delete edge;

}

}

}

auto it = nodes\_map.find(data);

if (it != nodes\_map.end())

{

delete it->second;

nodes\_map.erase(it);

}

}

void Graph::removeEdge(int startData, int endData)

{

auto startNodeIt = nodes\_map.find(startData);

auto endNodeIt = nodes\_map.find(endData);

if (startNodeIt == nodes\_map.end() || endNodeIt == nodes\_map.end())

{

return;

}

Node\* startNode = startNodeIt->second;

Node\* endNode = endNodeIt->second;

// Ïîèñê è óäàëåíèå ðåáðà ìåæäó óçëàìè

Edge\* edgeToRemove = nullptr;

for (Edge\* edge : startNode->edges\_to\_node)

{

if (edge->to->data == endData)

{

edgeToRemove = edge;

break;

}

}

if (edgeToRemove)

{

auto it = find(startNode->edges\_to\_node.begin(), startNode->edges\_to\_node.end(), edgeToRemove);

if (it != startNode->edges\_to\_node.end())

{

startNode->edges\_to\_node.erase(it);

delete edgeToRemove; // Îñâîáîæäåíèå ïàìÿòè

}

}

else

{

}

}

QtGraphs::QtGraphs(QWidget\* parent)

: QMainWindow(parent)

{

ui.setupUi(this);

connect(ui.pushButton, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_clicked);

connect(ui.pushButton\_2, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_2\_clicked);

connect(ui.pushButton\_4, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_3\_clicked);

connect(ui.pushButton\_3, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_4\_clicked);

connect(ui.pushButton\_5, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_5\_clicked);

connect(ui.pushButton\_8, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_8\_clicked);

connect(ui.pushButton\_9, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_9\_clicked);

connect(ui.pushButton\_10, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_10\_clicked);

connect(ui.pushButton\_11, &QPushButton::clicked, this, &QtGraphs::on\_pushButton\_11\_clicked);

}

QtGraphs::~QtGraphs()

{

}

void QtGraphs::paintEvent(QPaintEvent\* event) {

QPainter painter(this);

QFont font = painter.font();

font.setPointSize(16);

painter.setFont(font);

for (const auto& pair : graph.nodes\_map) {

Node\* node = pair.second;

for (Edge\* edge : node->edges\_to\_node) {

QPoint pos\_f;

QPoint pos\_t;

int d = 20 \* sin(atan(1));

double angles = atan2(-(edge->to->pos.y() - node->pos.y()), (edge->to->pos.x() - node->pos.x()));

pos\_f = QPoint(node->pos.x() + 20 \* cos(angles), node->pos.y() - 20 \* sin(angles));

pos\_t = QPoint(edge->to->pos.x() - 20 \* cos(angles), edge->to->pos.y() + 20 \* sin(angles));

painter.drawLine(pos\_f, pos\_t);

int x\_t = pos\_f.x() + 4 \* (pos\_t.x() - pos\_f.x()) / 5;

int y\_t = pos\_f.y() - 4 \* (pos\_f.y() - pos\_t.y()) / 5;

painter.drawText(x\_t - 10, y\_t + 10, QString::number(edge->weight));

QLine line(pos\_f, pos\_t);

double angle = atan2(-line.dy(), line.dx()) - M\_PI / 2;

double arrowSize = 15;

double arrowLength = 20;

QPointF arrowP1 = pos\_t + QPointF(sin(angle - M\_PI / 10) \* arrowSize, cos(angle - M\_PI / 10) \* arrowSize);

QPointF arrowP2 = pos\_t + QPointF(sin(angle + M\_PI / 10) \* arrowSize, cos(angle + M\_PI / 10) \* arrowSize);

QPolygonF arrowHead;

arrowHead << pos\_t << arrowP1 << arrowP2;

painter.drawPolygon(arrowHead);

}

}

for (const auto& pair : graph.nodes\_map) {

Node\* node = pair.second;

painter.drawEllipse(node->pos, 20, 20);

painter.drawText(node->pos.x() - 9, node->pos.y() + 8, QString::number(node->data));

}

if (sel) {

painter.drawEllipse(100, 100, 40, 40);

painter.setBrush(Qt::green);

painter.drawEllipse(sNode->pos, 20, 20);

painter.drawText(sNode->pos.x() - 9, sNode->pos.y() + 8, QString::number(sNode->data));

}

}

void QtGraphs::mousePressEvent(QMouseEvent\* event)

{

if (event->button() == Qt::LeftButton) {

m\_nodeSelected = false; // Ñáðîñ ôëàãà âûáðàííîãî óçëà

for (const auto& pair : graph.nodes\_map) {

Node\* node = pair.second;

if ((event->pos() - node->pos).manhattanLength() < 30) {

m\_selectedNode = node;

m\_nodeSelected = true;

break;

}

}

update();

}

}

void QtGraphs::mouseMoveEvent(QMouseEvent\* event)

{

if (m\_nodeSelected && m\_selectedNode) {

m\_selectedNode->pos = event->pos();

update();

}

}

void QtGraphs::mouseReleaseEvent(QMouseEvent\* event)

{

if (event->button() == Qt::LeftButton && m\_nodeSelected) {

m\_nodeSelected = false;

m\_selectedNode = nullptr;

update();

}

}

vector<int> Graph::solveTSP(int startNodeData) {

if (nodes\_map.find(startNodeData) == nodes\_map.end() || nodes\_map.size() < 2) {

return {}; // Пустой вектор, если начальный узел не найден или узлов меньше двух

}

int bestCost = numeric\_limits<int>::max();

vector<int> bestPath;

unordered\_set<int> visited;

vector<int> currentPath;

visited.insert(startNodeData);

currentPath.push\_back(startNodeData);

tspHelper(startNodeData, visited, currentPath, 0, bestCost, bestPath, startNodeData);

if (!bestPath.empty()) {

bestPath.push\_back(startNodeData); // Возвращаемся в начальный узел

}

return bestPath;

}

void Graph::tspHelper(int currentNodeData, unordered\_set<int>& visited, vector<int>& currentPath, int currentCost, int& bestCost, vector<int>& bestPath, int startNodeData) {

if (visited.size() == nodes\_map.size()) {

// Проверяем наличие пути обратно в начальный узел

for (Edge\* edge : nodes\_map[currentNodeData]->edges\_to\_node) {

if (edge->to->data == startNodeData) {

int totalCost = currentCost + edge->weight;

if (totalCost < bestCost) {

bestCost = totalCost;

bestPath = currentPath;

}

break;

}

}

return;

}

Node\* currentNode = nodes\_map[currentNodeData];

for (Edge\* edge : currentNode->edges\_to\_node) {

if (visited.find(edge->to->data) == visited.end()) {

visited.insert(edge->to->data);

currentPath.push\_back(edge->to->data);

tspHelper(edge->to->data, visited, currentPath, currentCost + edge->weight, bestCost, bestPath, startNodeData);

visited.erase(edge->to->data);

currentPath.pop\_back();

}

}

}

void QtGraphs::on\_pushButton\_clicked() {

QString text = ui.lineEdit->text();

if (text.isEmpty()) {

return;

}

int nodeValue = text.toInt();

graph.addNode(nodeValue);

ui.lineEdit->clear();

update();

}

void QtGraphs::on\_pushButton\_2\_clicked() {

if (ui.lineEdit\_2->text().isEmpty() or ui.lineEdit\_3->text().isEmpty() or ui.lineEdit\_4->text().isEmpty()) {

return;

}

int fromNode = ui.lineEdit\_2->text().toInt();

int toNode = ui.lineEdit\_3->text().toInt();

int weight = ui.lineEdit\_4->text().toInt();

if (graph.nodes\_map.find(fromNode) != graph.nodes\_map.end() && graph.nodes\_map.find(toNode) != graph.nodes\_map.end()) {

graph.addEdge(fromNode, toNode, weight);

ui.lineEdit\_2->clear();

ui.lineEdit\_4->clear();

ui.lineEdit\_3->clear();

update();

}

else {

}

}

void QtGraphs::on\_pushButton\_3\_clicked()

{

if (ui.lineEdit\_5->text().isEmpty()) {

return;

}

int del = ui.lineEdit\_5->text().toInt();

graph.removeNode(del);

ui.lineEdit\_5->clear();

update();

}

void QtGraphs::on\_pushButton\_4\_clicked()

{

if (ui.lineEdit\_7->text().isEmpty() or ui.lineEdit\_6->text().isEmpty()) {

return;

}

int s = ui.lineEdit\_7->text().toInt();

int f = ui.lineEdit\_6->text().toInt();

graph.removeEdge(s, f);

ui.lineEdit\_7->clear();

ui.lineEdit\_6->clear();

update();

}

void QtGraphs::on\_pushButton\_8\_clicked()

{

graph.addNode(1);

graph.addNode(2);

graph.addNode(3);

graph.addNode(4);

graph.addNode(5);

graph.addNode(6);

graph.addEdge(1, 2, 4);

graph.addEdge(1, 5, 8);

graph.addEdge(5, 3, 12);

graph.addEdge(3, 5, 12);

graph.addEdge(5, 2, 23);

graph.addEdge(2, 4, 9);

graph.addEdge(4, 6, 11);

graph.addEdge(2, 3, 15);

graph.addEdge(3, 2, 15);

graph.addEdge(6, 3, 6);

update();

}

void QtGraphs::on\_pushButton\_9\_clicked()

{

graph.addNode(1);

graph.addNode(2);

graph.addNode(3);

graph.addNode(4);

graph.addNode(5);

graph.addNode(6);

graph.addEdge(1, 2, 8);

graph.addEdge(2, 1, 8);

graph.addEdge(2, 3, 12);

graph.addEdge(3, 2, 12);

graph.addEdge(3, 4, 16);

graph.addEdge(4, 3, 16);

graph.addEdge(2, 5, 10);

graph.addEdge(5, 2, 10);

graph.addEdge(1, 6, 11);

graph.addEdge(6, 1, 11);

graph.addEdge(6, 4, 9);

graph.addEdge(4, 6, 9);

graph.addEdge(6, 5, 6);

graph.addEdge(5, 6, 6);

graph.addEdge(4, 5, 5);

graph.addEdge(5, 4, 5);

update();

}

void QtGraphs::on\_pushButton\_10\_clicked()

{

if (ui.lineEdit\_12->text().isEmpty() or ui.lineEdit\_13->text().isEmpty() or ui.lineEdit\_14->text().isEmpty()) {

return;

}

int s = ui.lineEdit\_12->text().toInt();

int t = ui.lineEdit\_14->text().toInt();

int w = ui.lineEdit\_13->text().toInt();

graph.updateEdgeWeight(s, t, w);

ui.lineEdit\_12->text().clear();

ui.lineEdit\_14->text().clear();

ui.lineEdit\_13->text().clear();

update();

}

void QtGraphs::on\_pushButton\_11\_clicked()

{

graph.clearGraph();

update();

}

void QtGraphs::on\_pushButton\_5\_clicked()

{

ui.textBrowser->clear();

int s = ui.lineEdit\_8->text().toInt();

vector<int>shortestPath = graph.solveTSP(s);

QString resultString;

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

resultString.append(QString::number(shortestPath[i]));

if (i < shortestPath.size() - 1) {

resultString.append(", ");

}

}

static int idx = 0;

QTimer\* timer = new QTimer(this);

connect(timer, &QTimer::timeout, [=]() {

if (shortestPath.size() != 0 and idx < shortestPath.size()) {

Node\* nod = graph.nodes\_map[shortestPath[idx]];

sNode = nod;

sel = 1;

update();

idx++;

}

else {

ui.textBrowser->setText(resultString);

timer->stop();

timer->deleteLater();

sel = 0;

ui.lineEdit\_8->clear();

update();

idx = 0;

}

});

timer->start(500);

}

**Файл main.cpp**

#include "QtGraphs.h"

#include <QtWidgets/QApplication>

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

{

QApplication a(argc, argv);

QtGraphs w;

w.show();

return a.exec();

}

**Демонстрация работы программ:**

**https://youtu.be/J8ZGGTRBfGI**