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

## «АРМ специалист и задача коммивояжера»

Выполнил студент гр. ИВТ-23-1б

Дагелис Александр Юрьевич

Проверил:

Доцент каф. ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

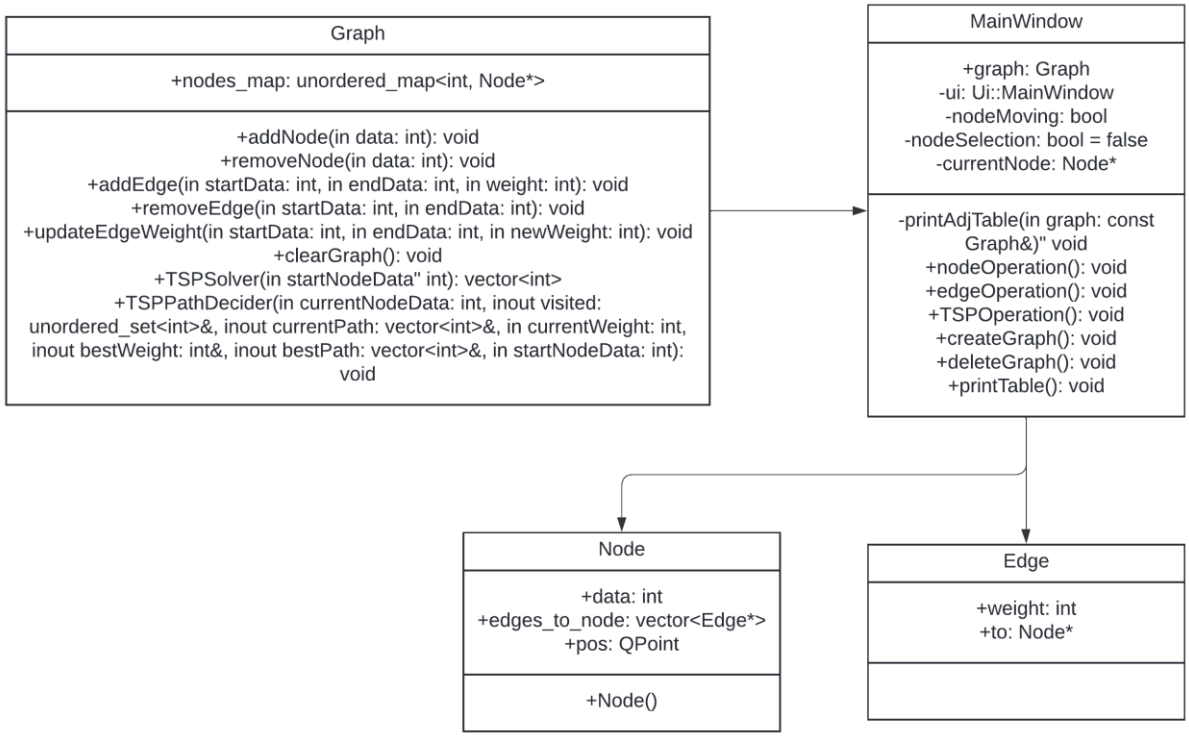
(дата)

г. Пермь, 2024

## Цель и задачи работы

Целью данной работы является реализация задачи коммивояжера и автоматизированного рабочего места.

Задача коммивояжера.




Функция main

```cpp
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

mainwindow.h

```cpp
#include <QWidget>
#include <QMouseEvent>
#include <ui_mainwindow.h>
#include <unordered_map>
#include <unordered_set>

using namespace std;

// Предопределение классов
class Edge;
class Node;
class Graph;

class Node{
public:
    int data;
    vector<Edge*> edges_to_node;
    QPoint pos;
    Node(){
        pos = QPoint(800 + (rand()%400 - 400), 300 + (rand()%300 - 300));
    }
};

class Edge{
public:
    int weight;
    Node* to;
};

class Graph{
public:
    unordered_map<int, Node*> nodes_map;

    void addNode(int data);
    void removeNode(int data);

    void addEdge(int fromData, int toData, int weight);
    void removeEdge(int startData, int endData);
    void updateEdgeWeight(int startData, int endData, int newWeight);

    void clearGraph();

    vector<int> TSPSolver(int startNodeData);
    void TSPPathDecider(int currentNodeData, unordered_set<int>& visited, vector<int>& currentPath, int currentWeight, int& bestWeight, vector<int>& bestPath, int startNodeData);
};

class MainWindow : public QMainWindow{
    Q_OBJECT
public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();
    Graph graph;
```

```cpp
protected:
    void paintEvent(QPaintEvent* event) override;
    void mousePressEvent(QMouseEvent* event) override;
    void mouseMoveEvent(QMouseEvent* event) override;
    void mouseReleaseEvent(QMouseEvent* event) override;
private:
    Ui::MainWindow ui;
    Node* selectedNode;
    bool nodeMoving;
    bool nodeSelection = false;
    Node* sNode;

    void printAdjTable(const Graph& graph);

public slots:
    void nodeOperation();
    void edgeOperation();
    void TSPOperation();
    void createGraph();
    void deleteGraph();
    void printTable();
};
```

mainwindow.cpp

```cpp
#include "mainwindow.h"

#include <QPainter>
#include <cmath>
#include <QTimer>
#include <queue>
#include <stack>

MainWindow::MainWindow(QWidget* parent): QMainWindow(parent){
    ui.setupUi( this );

    connect(ui.nodeOperationButton, &QPushButton::pressed, this, &MainWindow::nodeOperation);
    connect(ui.edgeOperationButton, &QPushButton::pressed, this, &MainWindow::edgeOperation);
    connect(ui.TSPButton, &QPushButton::pressed, this, &MainWindow::TSPOperation);
    connect(ui.createGraphButton, &QPushButton::clicked, this, &MainWindow::createGraph);
    connect(ui.deleteGraphButton, &QPushButton::clicked, this, &MainWindow::deleteGraph);
    connect(ui.printTableButton, &QPushButton::clicked, this, &MainWindow::printTable);
}

// Операции над узлами
void Graph::addNode(int data){
    if (nodes_map.find(data) == nodes_map.end()){
        Node* newNode = new Node;
        newNode->data = data;
        nodes_map[data] = newNode;
    }
}
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 MainWindow::nodeOperation(){
    if (ui.nodeValue->text().isEmpty()){ return; }

    int operation = ui.nodeOperations->currentIndex();
    int nodeValue = ui.nodeValue->text().toInt();
```

```
    switch(operation){
    case 0: graph.addNode(nodeValue); break;
    case 1: graph.removeNode(nodeValue); break;
    }
    ui.nodeValue->clear(); update();
    ui.statusText->setText("Операция над узлом проведена.");
}

// Операции над рёбрами
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::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;
    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;
        }
    }
}
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;
        }
    }
}
```

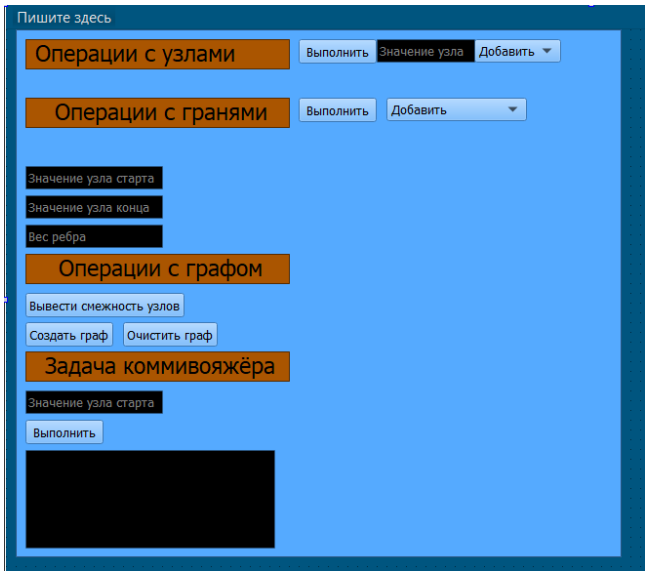
```cpp
void MainWindow::edgeOperation(){
    if (ui.nodeStart->text().isEmpty() || ui.nodeEnd->text().isEmpty()) { return; }

    int nodeStart = ui.nodeStart->text().toInt();
    int nodeEnd = ui.nodeEnd->text().toInt();
    int edgeWeight = ui.edgeWeight->text().toInt();

    int operation = ui.edgeOperations->currentIndex();

    // 0 - Добавить, 1 - Удалить, 2 - Редактировать вес
    switch(operation){
    case 0: if(ui.edgeWeight->text().isEmpty()){ return; } graph.addEdge(nodeStart, nodeEnd, edgeWeight);  break;
    case 1: graph.removeEdge(nodeStart, nodeEnd); break;
    case 2: if(ui.edgeWeight->text().isEmpty()){ return; } graph.updateEdgeWeight(nodeStart, nodeEnd, edgeWeight);  break;

    ui.nodeStart->clear(); ui.nodeEnd->clear(); ui.edgeWeight->clear(); update();
    ui.statusText->setText("Операция над гранью проведена.");
    }
}

// Удаление и создание графа
void Graph::clearGraph(){
    for (auto& pair : nodes_map){
        Node* node = pair.second;
        delete node;
    }

    nodes_map.clear();
}
void MainWindow::createGraph(){
    graph.addNode(1);
    graph.addNode(2);
    graph.addNode(3);
    graph.addNode(4);
    graph.addNode(5);
    graph.addNode(6);

    graph.addEdge(1, 2, 2);
    graph.addEdge(1, 6, 57);

    graph.addEdge(2, 1, 2);
    graph.addEdge(2, 6, 13);
    graph.addEdge(2, 4, 8);
    graph.addEdge(2, 3, 3);

    graph.addEdge(3, 2, 3);
    graph.addEdge(3, 4, 5);

    graph.addEdge(4, 3, 5);
    graph.addEdge(4, 2, 8);
    graph.addEdge(4, 6, 21);
    graph.addEdge(4, 5, 34);

    graph.addEdge(5, 6, 45);
    graph.addEdge(5, 4, 34);

    graph.addEdge(6, 1, 57);
    graph.addEdge(6, 2, 13);
    graph.addEdge(6, 4, 21);
    graph.addEdge(6, 5, 45);
```

```
    update();
    ui.statusText->setText("Граф задания создан.");
}
void MainWindow::deleteGraph(){
    graph.clearGraph();

    ui.statusText->setText("Граф удалён.");
    update();
}

// Визуализация графа
void MainWindow::paintEvent(QPaintEvent* event){
    QPainter painter(this);

    QFont font = painter.font();
    font.setPointSize(16);
    painter.setFont(font);
    painter.setPen(QPen(Qt::cyan));

    for (const auto& pair : graph.nodes_map) {
        Node* node = pair.second;

        for (Edge* edge : node->edges_to_node) {
            painter.setOpacity(0.2);
            QPoint edgeStart;
            QPoint edgeEnd;

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

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

            painter.drawLine(edgeStart, edgeEnd);

            int x_t = edgeStart.x() + 4 * (edgeEnd.x() - edgeStart.x()) / 5 ;
            int y_t = edgeStart.y() - 4 * (edgeStart.y() - edgeEnd.y()) / 5;

            painter.setPen(QPen(Qt::black, 2));
            painter.setOpacity(1);
            painter.drawText(x_t-10, y_t+10, QString::number(edge->weight));
            painter.setOpacity(0.2);
            painter.setPen(QPen(Qt::cyan, 2));

            QLine line(edgeStart, edgeEnd);

            double angle = atan2(-line.dy(), line.dx())-M_PI/2;
            double arrowSize = 20;

            QPointF arrowP1 = edgeEnd + QPointF(sin(angle - M_PI / 12) * arrowSize, cos(angle - M_PI / 12) * arrowSize);
            QPointF arrowP2 = edgeEnd + QPointF(sin(angle + M_PI / 12) * arrowSize, cos(angle + M_PI / 12) * arrowSize);

            QPolygonF arrowHead;

            arrowHead << edgeEnd << arrowP1 << arrowP2;

            QPainterPath path;
```

```
            path.moveTo(edgeEnd);
            path.lineTo(arrowP1);
            path.lineTo(arrowP2);
            painter.fillPath(path, Qt::darkCyan);
            painter.drawPolygon(arrowHead);
            painter.setOpacity(1);
        }
    }
    painter.setBrush(Qt::NoBrush);
    painter.setPen(QPen(Qt::white, 2));

    for (const auto& pair : graph.nodes_map) {
        Node* node = pair.second;
        painter.drawEllipse(node->pos, 20, 20);
        painter.setPen(QPen(Qt::black, 2));
        painter.drawText(node->pos.x() - 9, node->pos.y() + 8, QString::number(node->data));
        painter.setPen(QPen(Qt::white, 2));
    }

    if (nodeSelection){
        painter.drawEllipse(100,100, 40, 40);
        painter.setBrush(Qt::yellow);
        painter.drawEllipse(sNode->pos, 20, 20);
        painter.setPen(QPen(Qt::black, 2));
        painter.drawText(sNode->pos.x() - 9, sNode->pos.y() + 8, QString::number(sNode->data));
    }
}
void MainWindow::mousePressEvent(QMouseEvent* event){
    if (event->button() == Qt::LeftButton){
        nodeMoving = false;
        for (const auto& pair : graph.nodes_map){
            Node* node = pair.second;
            if ((event->pos() - node->pos).manhattanLength() < 30){
                selectedNode = node;
                nodeMoving = true;
                break;
            }
        }
        update();
    }
}
void MainWindow::mouseMoveEvent(QMouseEvent* event){
    if (nodeMoving && selectedNode){
        selectedNode->pos = event->pos();
        update();
    }
}
void MainWindow::mouseReleaseEvent(QMouseEvent* event){
    if (event->button() == Qt::LeftButton && nodeMoving){
        nodeMoving = false;
        selectedNode = nullptr;
        update();
    }
}
```

```cpp
// Вывод таблицы смежностей узлов
void MainWindow::printTable(){
    printAdjTable(graph);
}
void MainWindow::printAdjTable(const Graph& graph){
    QString result;
    for (const auto& pair : graph.nodes_map){
        int node = pair.first;
        Node* nodeConnections = pair.second;

        result += "Узел " + QString::number(node) + ": \n";
        unordered_set<int> printedNodes;
        for (Edge* edge : nodeConnections->edges_to_node){
            if (printedNodes.find(edge->to->data) == printedNodes.end()){
                result += "    Связь с " + QString::number(edge->to->data) + ", вес ребра - " + QString::number(edge->weight) + ". \n";
                printedNodes.insert(edge->to->data);
            }
        }
        ui.statusText->setText(result);
    }
}

// Задача коммивояжёра
vector<int> Graph::TSPSolver(int startNodeData){
    int bestWeight = numeric_limits<int>::max();

    vector<int> bestPath;
    unordered_set<int> visited;
    vector<int> currentPath;

    visited.insert(startNodeData);
    currentPath.push_back(startNodeData);

    TSPPathDecider(startNodeData, visited, currentPath, 0, bestWeight, bestPath, startNodeData);

    if (!bestPath.empty()) {
        bestPath.push_back(startNodeData);
    }

    return bestPath;
}
void Graph::TSPPathDecider(int currentNodeValue, unordered_set<int>& visited, vector<int>& currentPath, int currentWeight, int& bestWeight, vector<int>& bestPath, int startNodeData){
    if (visited.size() == nodes_map.size()){
        for (Edge* edge : nodes_map[currentNodeValue]->edges_to_node){
            if (edge->to->data == startNodeData){
                int totalCost = currentWeight + edge->weight;

                if (totalCost < bestWeight){
                    bestWeight = totalCost;
                    bestPath = currentPath;
                }
                break;
            }
        }
        return;
    }
    Node* currentNode = nodes_map[currentNodeValue];

    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);

            TSPPathDecider(edge->to->data, visited, currentPath, currentWeight + edge->weight, bestWeight, bestPath, startNodeData);

            visited.erase(edge->to->data);
            currentPath.pop_back();
        }
    }
}
void MainWindow::TSPOperation(){
    int nodeStart = ui.TSPStart->text().toInt();

    vector<int>shortestPath = graph.TSPSolver(nodeStart);

    QString TSPResult;

    for (unsigned int i = 0; i < shortestPath.size(); i++){
        TSPResult.append(QString::number(shortestPath[i]));

        if (i < shortestPath.size() - 1){
            TSPResult.append(" -> ");
        }
    }
    static unsigned int idx = 0;

    // Смена активного узла при перемещении
    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;
            nodeSelection = true;
            update();
            idx++;
        }else{
            timer->stop();
            timer->deleteLater();
            nodeSelection = false;
            update();
            idx = 0;
        }
    });

    ui.statusText->setText("Результат задачи коммивояджёра: " + TSPResult);
    timer->start(1000);
}

MainWindow::~MainWindow(){}
```

mainwindow.ui



**Код АРМ прогресс бар.**

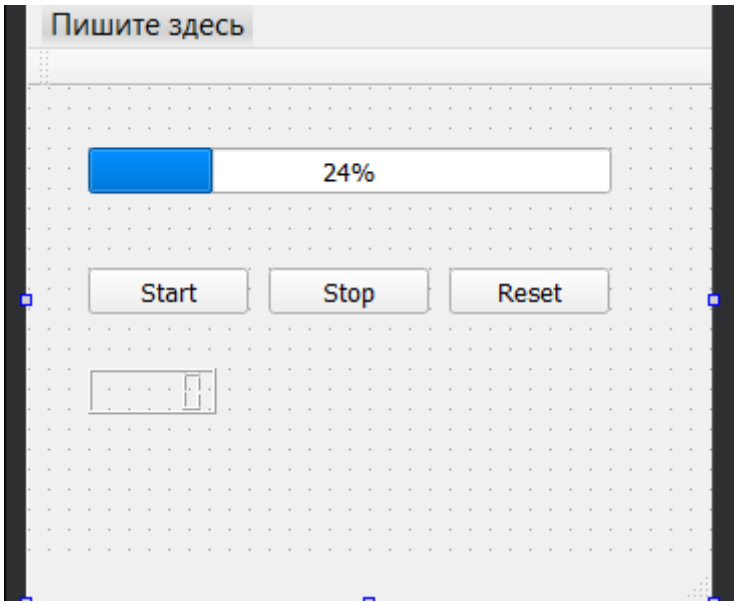
mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtGui>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    QTimer *qt;
    int tval;
private slots:
    void start();
    void stop();
    void reset();
    void timeout();
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    qt=new QTimer;ui->progressBar->setValue(0);
    tval=0;

    connect(ui->pbStart,SIGNAL(clicked()),this,SLOT(start()));
    connect(ui->pbStop,SIGNAL(clicked()),this,SLOT(stop()));
    connect(ui->pbReset,SIGNAL(clicked()),this,SLOT(reset()));
    connect(this->qt,SIGNAL(timeout()),this,SLOT(timeout()));
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::timeout()
{
    if(tval<=100)
        ui->progressBar->setValue(tval++);
    else
    {
        qt->stop();
        QMessageBox qmb;
        qmb.setText("Timer stops");
        qmb.exec();

    }
}

void MainWindow::start()
{
  qt->setInterval(500);
  qt->start();
}
void MainWindow::stop()
{
  qt->stop();
}
void MainWindow::reset()
{
  tval=0;
}
```

mainwindow.ui



## Скриншоты работы программ.