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» Вариант 2

Выполнил студент гр. ИВТ-23-1б
Попонин Михаил Александрович

Проверил:
Доцент каф. ИТАС
Яруллин Денис Владимирович

| | |
|---|---|
| (оценка) | (подпись) |
| | (дата) |

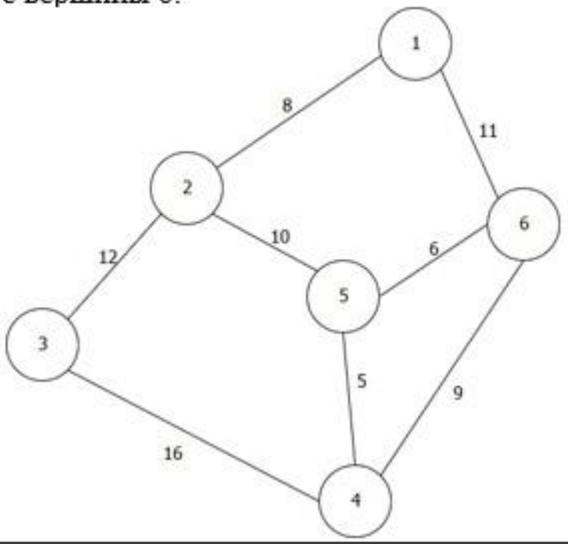
г. Пермь, 2024

## Цель и задачи работы

Целью данной работы реализация задачи коммивояжера и АРМ-специалиста поливки полей

**Вариант 2:** взят за основу для решения задачи коммивояжера

Реализовать граф, а также алгоритм Дейкстры, выполнив все необходимые действия.
Выполнение начать с вершины 6.



## UML диаграмма

На рисунке 1 изображена диаграмма класса для задачи коммивояжера

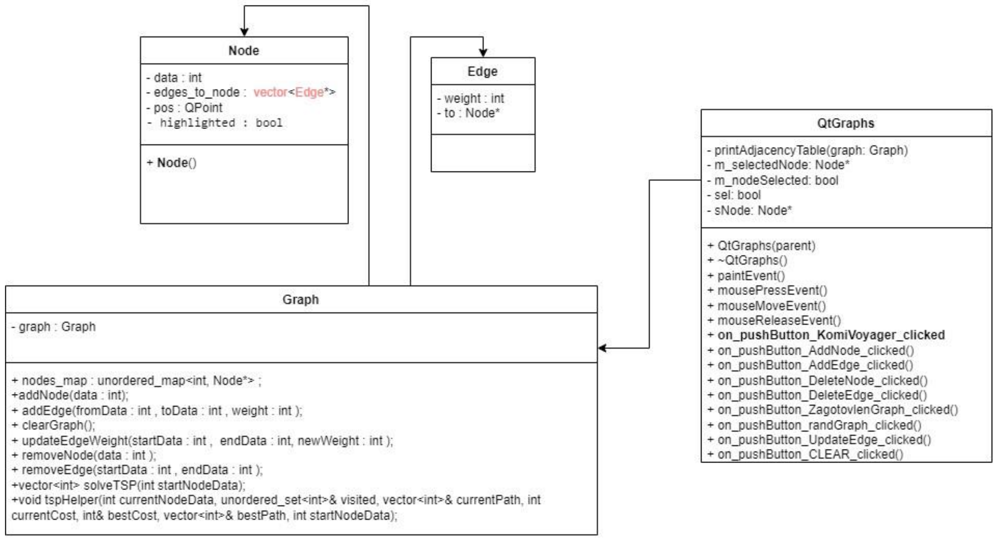
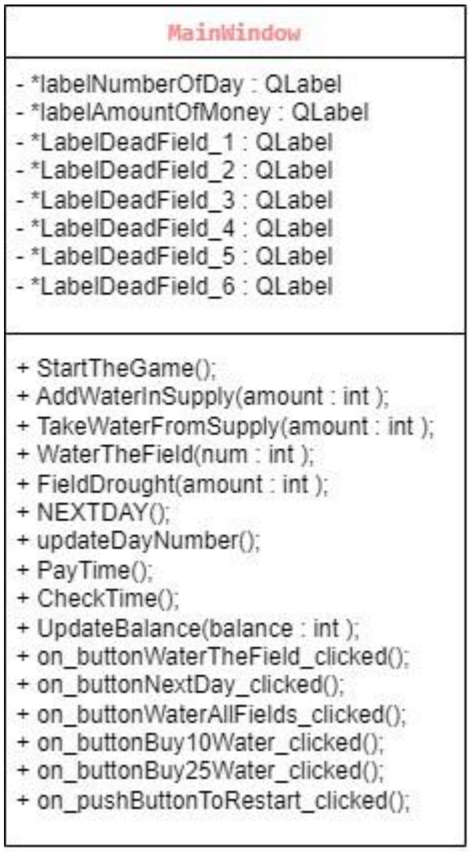

Node
- data : int
- edges_to_node : vector<Edge*>
- pos : QPoint
- highlighted : bool
+ **Node**()
Edge
- weight : int
- to : Node*
QtGraphs
- printAdjacencyTable(graph: Graph)
- m_selectedNode: Node*
- m_nodeSelected: bool
- sel: bool
- sNode: Node*
+ QtGraphs(parent)
+ ~QtGraphs()
+ paintEvent()
+ mousePressEvent()
+ mouseMoveEvent()
+ mouseReleaseEvent()
+ **on_pushButton_KomiVoyager_clicked**
+ on_pushButton_AddNode_clicked()
+ on_pushButton_AddEdge_clicked()
+ on_pushButton_DeleteNode_clicked()
+ on_pushButton_DeleteEdge_clicked()
+ on_pushButton_ZagotovlenGraph_clicked()
+ on_pushButton_randGraph_clicked()
+ on_pushButton_UpdateEdge_clicked()
+ on_pushButton_CLEAR_clicked()
Graph
- graph : Graph
+ nodes_map : unordered_map<int, Node*> ;
+addNode(data : int);
+ addEdge(fromData : int , toData : int , weight : int );
+ clearGraph();
+ updateEdgeWeight(startData : int , endData : int, newWeight : int );
+ removeNode(data : int );
+ removeEdge(startData : int , endData : int );
+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);

Рисунок 1

На рисунке 2 изображена диаграмма класса АРМ специалиста



```
- *labelNumberOfDay : QLabel
- *labelAmountOfMoney : QLabel
- *LabelDeadField_1 : QLabel
- *LabelDeadField_2 : QLabel
- *LabelDeadField_3 : QLabel
- *LabelDeadField_4 : QLabel
- *LabelDeadField_5 : QLabel
- *LabelDeadField_6 : QLabel

+ StartTheGame();
+ AddWaterInSupply(amount : int );
+ TakeWaterFromSupply(amount : int );
+ WaterTheField(num : int );
+ FieldDrought(amount : int );
+ NEXTDAY();
+ updateDayNumber();
+ PayTime();
+ CheckTime();
+ UpdateBalance(balance : int );
+ on_buttonWaterTheField_clicked();
+ on_buttonNextDay_clicked();
+ on_buttonWaterAllFields_clicked();
+ on_buttonBuy10Water_clicked();
+ on_buttonBuy25Water_clicked();
+ on_pushButtonToRestart_clicked();
```

Рисунок 2

## Код программы для Коммивояжера

Таблица 1 – файл QtGraphs.h

```
#pragma once
#include <QWidget>
#include <QMouseEvent>
#include <ui_QtGraphs.h>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <stack>
#include <iostream>
using namespace std;
class Edge;
class Node;
class Graph;
class Node
{
public:
    int data;
    vector<Edge*> edges_to_node;
    QPoint pos;
    bool highlighted;
    Node()
    {
        pos = QPoint(600 + (rand()%600 - 300), 300 + (rand()%600 - 300));
    }
};
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::QtGraphs 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_KomiVoyager_clicked();
    void on_pushButton_AddNode_clicked();
    void on_pushButton_AddEdge_clicked();
    void on_pushButton_DeleteNode_clicked();
    void on_pushButton_DeleteEdge_clicked();
    void on_pushButton_ZagotovlenGraph_clicked();
    void on_pushButton_randGraph_clicked();
    void on_pushButton_UpdateEdge_clicked();
    void on_pushButton_CLEAR_clicked();
};
```

Таблица 2 – файл main.cpp

```
#include "QtGraphs.h"
#include <QApplication>

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

Таблица 3 – файл QtGraphs.cpp

```
#include "QtGraphs.h"
#include <QPainter>
#include <vector>
#include <QLineEdit>
#include <QPushButton>
#include <cmath>
#include <unordered_set>
#include <chrono>
#include <thread>
#include <QTimer>
#include <queue>
#include <limits>
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::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()
{
   for (auto& pair : nodes_map)
   {
      Node* node = pair.second;
      delete node;
   }
   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;
    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; // Освобождение памяти
        }
    }
}
QtGraphs::QtGraphs(QWidget* parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);
    connect(ui.pushButton_AddNode, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_AddNode_clicked);
    connect(ui.pushButton_AddEdge, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_AddEdge_clicked);
    connect(ui.pushButton_DeleteNode, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_DeleteNode_clicked);
    connect(ui.pushButton_DeleteEdge, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_DeleteEdge_clicked);
    connect(ui.pushButton_ZagotovlenGraph, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_ZagotovlenGraph_clicked);
    connect(ui.pushButton_randGraph, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_randGraph_clicked);
    connect(ui.pushButton_UpdateEdge, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_UpdateEdge_clicked);
    connect(ui.pushButton_CLEAR, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_CLEAR_clicked);
```

```
   connect(ui.pushButton_KomiVoyager, &QPushButton::clicked, this,
&QtGraphs::on_pushButton_KomiVoyager_clicked);
}
QtGraphs::~QtGraphs()
{
}
void QtGraphs::paintEvent(QPaintEvent* event)
{
   QPainter painter(this);
   QFont font = painter.font();
   font.setPointSize(16);
   painter.setFont(font);
   painter.setPen(QPen(Qt::black, 2)); // Черный цвет для текста
   for (const auto& pair : graph.nodes_map) {
      Node* node = pair.second;
      for (Edge* edge : node->edges_to_node) {
         QPoint pos_f;
         QPoint pos_t;
         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 = 20;
         QPointF arrowP1 = pos_t + QPointF(sin(angle - M_PI / 12) * arrowSize, cos(angle -
M_PI / 12) * arrowSize);
         QPointF arrowP2 = pos_t + QPointF(sin(angle + M_PI / 12) * arrowSize, cos(angle +
M_PI / 12) * arrowSize);
         QPolygonF arrowHead;
         arrowHead << pos_t << arrowP1 << arrowP2;
         QPainterPath path;
         path.moveTo(pos_t);
         path.lineTo(arrowP1);
         path.lineTo(arrowP2);
         painter.fillPath(path, Qt::magenta);
         painter.drawPolygon(arrowHead);
      }
   }
   painter.setBrush(Qt::NoBrush);
   painter.setPen(QPen(Qt::black, 2));
   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::magenta);
        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;
        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();
    }
}
void QtGraphs::on_pushButton_AddNode_clicked()
{
    QString text = ui.lineEdit_addDelNodeValue->text();
    if (text.isEmpty())
    {
        return;
    }
    int nodeValue = text.toInt();
    graph.addNode(nodeValue);
    ui.lineEdit_addDelNodeValue->clear();
    update();
    ui.statusbar->showMessage("Вершина добавлена!");
}
void QtGraphs::on_pushButton_AddEdge_clicked() {
    if (ui.LineEdit_FirstNode->text().isEmpty() or ui.LineEdit_SecondNode->text().isEmpty() or
ui.lineEdit_Weight->text().isEmpty()) {
        return;
    }
    int fromNode = ui.LineEdit_FirstNode->text().toInt();
    int toNode = ui.LineEdit_SecondNode->text().toInt();
    int weight = ui.lineEdit_Weight->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_FirstNode->clear();
        ui.lineEdit_Weight->clear();
        ui.LineEdit_SecondNode->clear();
        update();
        ui.statusbar->showMessage("Грань добавлена!");
    }
}
void QtGraphs::on_pushButton_DeleteNode_clicked()
{
    if (ui.lineEdit_addDelNodeValue->text().isEmpty())
    {
        return;
    }
    int del = ui.lineEdit_addDelNodeValue->text().toInt();
    graph.removeNode(del);
    ui.lineEdit_addDelNodeValue->clear();
    update();
    ui.statusbar->showMessage("Вершина удалена!");
}
void QtGraphs::on_pushButton_DeleteEdge_clicked()
{
    if (ui.LineEdit_FirstNode->text().isEmpty() or ui.LineEdit_SecondNode->text().isEmpty()) {
        return;
    }
    int s = ui.LineEdit_FirstNode->text().toInt();
    int f = ui.LineEdit_SecondNode->text().toInt();
    graph.removeEdge(s, f);
    ui.LineEdit_FirstNode->clear();
    ui.LineEdit_SecondNode->clear();
    update();
    ui.statusbar->showMessage("Грань удалена!");
}
void QtGraphs::on_pushButton_ZagotovlenGraph_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(1, 6, 11);

    graph.addEdge(6, 5, 6);
    graph.addEdge(6, 4, 9);
    graph.addEdge(6, 1, 11);

    graph.addEdge(4, 5, 5);
    graph.addEdge(4, 3, 16);
    graph.addEdge(4, 6, 9);

    graph.addEdge(3, 2, 12);
    graph.addEdge(3, 4, 16);


    graph.addEdge(2, 1, 8);
    graph.addEdge(2, 3, 12);
    graph.addEdge(2, 5, 10);

    graph.addEdge(5, 2, 10);
    graph.addEdge(5, 4, 5);
    graph.addEdge(5, 6, 6);

    update();
    ui.statusbar->showMessage("Загатовленный граф призван!");
}
void QtGraphs::on_pushButton_randGraph_clicked()
{
    for (int i = 0; i < 10; i++)
    {
        int c = rand() % 10;
        int b = rand() % 10;
        graph.addNode(c);
        graph.addNode(b);
        int chance = rand() % 5;
        if (!(chance == 0))
        {
            int m = rand() % 10;
            graph.addEdge(c,b,m);
            if(chance == 1)
            {
                graph.addEdge(c,b,m);
            }
        }
    }
    update();
    ui.statusbar->showMessage("Случайный граф призван!");
}
void QtGraphs::on_pushButton_UpdateEdge_clicked()
{
    if (ui.LineEdit_FirstNode->text().isEmpty() or ui.lineEdit_Weight->text().isEmpty() or
ui.LineEdit_SecondNode->text().isEmpty())
    {
        return;
    }
    int s = ui.LineEdit_FirstNode->text().toInt();
    int t = ui.LineEdit_SecondNode->text().toInt();
    int w = ui.lineEdit_Weight->text().toInt();
    graph.updateEdgeWeight(s, t, w);
    ui.LineEdit_FirstNode->clear();
    ui.lineEdit_Weight->clear();
    ui.LineEdit_SecondNode->clear();
    update();
    ui.statusbar->showMessage("Грань обновлена!");
}
void QtGraphs::on_pushButton_CLEAR_clicked()
{
    graph.clearGraph();
    update();
}
vector<int> Graph::solveTSP(int startNodeData)
{
    if (nodes_map.find(startNodeData) == nodes_map.end() || nodes_map.size() < 2)
    {
        return {};
    }

    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);
    }

    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())
    {
        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_KomiVoyager_clicked()
{
    ui.textBrowser_KomiVoyager->clear();
    int s = ui.lineEdit_KomiVoyager->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_KomiVoyager->setText(resultString);
            timer->stop();
            timer->deleteLater();
            sel = 0;
            ui.lineEdit_KomiVoyager->clear();
            update();
            idx = 0;
        }
        });
    timer->start(500);
}
```

## Код программы для АРМ специалиста поливки полей

Таблица 4 – файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <cstdlib>
```

```
#include <QLabel>
#include <QMap>
#include <QDebug>
#include <QMessageBox>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
   Q_OBJECT
public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();
   QLabel *labelNumberOfDay;
   QLabel *labelAmountOfMoney;
   QLabel *LabelDeadField_1;
   QLabel *LabelDeadField_2;
   QLabel *LabelDeadField_3;
   QLabel *LabelDeadField_4;
   QLabel *LabelDeadField_5;
   QLabel *LabelDeadField_6;
private slots:
   void StartTheGame();
   void AddWaterInSupply(int amount);
   void TakeWaterFromSupply(int amount);
   void WaterTheField(int num);
   void FieldDrought(int amount);
   void NEXTDAY();
   void updateDayNumber();
   void PayTime();
   void CheckTime();
   void UpdateBalance(int balance);

   void on_buttonWaterTheField_clicked();
   void on_buttonNextDay_clicked();
   void on_buttonWaterAllFields_clicked();
   void on_buttonBuy10Water_clicked();
   void on_buttonBuy25Water_clicked();
   void on_pushButtonToRestart_clicked();
private:
   QMap<QString, QLabel*> labelMap;
   Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Таблица 5 – файл main.cpp

```
#include "mainwindow.h"

#include <QApplication>

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

Таблица 6 – файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   StartTheGame();
   labelNumberOfDay = ui->labelNumberOfDay;
   labelAmountOfMoney = ui->labelAmountOfMoney;
   LabelDeadField_1 = ui->LabelDeadField_1;
   LabelDeadField_2 = ui->LabelDeadField_2;
   LabelDeadField_3 = ui->LabelDeadField_3;
   LabelDeadField_4 = ui->LabelDeadField_4;
   LabelDeadField_5 = ui->LabelDeadField_5;
   LabelDeadField_6 = ui->LabelDeadField_6;
}
MainWindow::~MainWindow()
{
   delete ui;
}
void MainWindow::StartTheGame()
{
   ui->progressBar_WaterSupply->setValue(100);
   ui->progressBar_1field->setValue(100);
   ui->progressBar_2field->setValue(100);
   ui->progressBar_3field->setValue(100);
   ui->progressBar_4field->setValue(100);
   ui->progressBar_5field->setValue(100);
   ui->progressBar_6field->setValue(100);
   ui->labelNumberOfDay->clear();
   ui->labelAmountOfMoney->clear();
   ui->LabelDeadField_1->clear();
   ui->LabelDeadField_2->clear();
   ui->LabelDeadField_3->clear();
   ui->LabelDeadField_4->clear();
   ui->LabelDeadField_5->clear();
   ui->LabelDeadField_6->clear();
   ui->pushButtonToRestart->hide();
}
void MainWindow::AddWaterInSupply(int amount)
{
   int currentValue = ui->progressBar_WaterSupply->value();
   ui->progressBar_WaterSupply->setValue(qMin(currentValue + amount, ui-
>progressBar_WaterSupply->maximum()));
}
void MainWindow::TakeWaterFromSupply(int amount)
{
   int currentValue = ui->progressBar_WaterSupply->value();
   ui->progressBar_WaterSupply->setValue(qMax(currentValue - amount, 0));
}
void MainWindow::WaterTheField(int num)
{
   switch (num)
   {
   case 1:
      if(ui->progressBar_1field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
      {
         ui->statusbar->showMessage("Полить поле невозможно!");
         return;
      }
      TakeWaterFromSupply(2);
      ui->progressBar_1field->setValue(qMin(ui->progressBar_1field->value() + 10, ui-
>progressBar_1field->maximum()));
      break;
   case 2:
      if(ui->progressBar_2field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
      {
         ui->statusbar->showMessage("Полить поле невозможно!");
         return;
      }
      TakeWaterFromSupply(2);
      ui->progressBar_2field->setValue(qMin(ui->progressBar_2field->value() + 10, ui-
>progressBar_2field->maximum()));
      break;
   case 3:
      if(ui->progressBar_3field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
      {
         ui->statusbar->showMessage("Полить поле невозможно!");
```

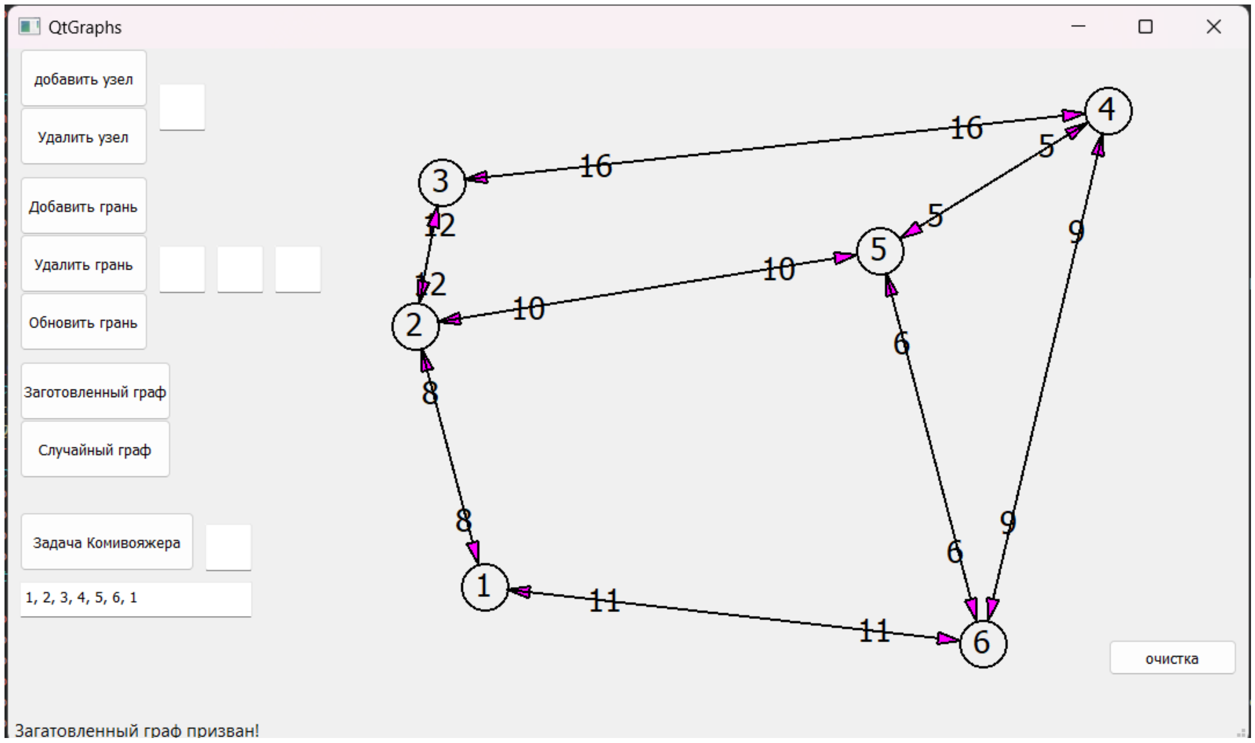
```
            return;
        }
        TakeWaterFromSupply(2);
        ui->progressBar_3field->setValue(qMin(ui->progressBar_3field->value() + 10, ui-
>progressBar_3field->maximum()));
        break;
    case 4:
        if(ui->progressBar_4field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
        {
            ui->statusbar->showMessage("Полить поле невозможно!");
            return;
        }
        TakeWaterFromSupply(2);
        ui->progressBar_4field->setValue(qMin(ui->progressBar_4field->value() + 10, ui-
>progressBar_4field->maximum()));
        break;
    case 5:
        if(ui->progressBar_5field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
        {
            ui->statusbar->showMessage("Полить поле невозможно!");
            return;
        }
        TakeWaterFromSupply(2);
        ui->progressBar_5field->setValue(qMin(ui->progressBar_5field->value() + 10, ui-
>progressBar_5field->maximum()));
        break;
    case 6:
        if(ui->progressBar_6field->value() == 0 || ui->progressBar_WaterSupply->value()< 2)
        {
            ui->statusbar->showMessage("Полить поле невозможно!");
            return;
        }
        TakeWaterFromSupply(2);
        ui->progressBar_6field->setValue(qMin(ui->progressBar_6field->value() + 10, ui-
>progressBar_6field->maximum()));
        break;
    default:
        ui->statusbar->showMessage("Укажите корректный номер поля (1 - 6)!");
        break;
    }
}
void MainWindow::on_buttonWaterTheField_clicked()
{
    if(ui->progressBar_WaterSupply->value() == 0)
    {
        ui->statusbar->showMessage("запасы воды закончились!");
        return;
    }
    QString Number = ui->lineEdit_NumberOfField->text();
    int num = Number.toInt();
    WaterTheField(num);
}
void MainWindow::on_buttonWaterAllFields_clicked()
{
    WaterTheField(1);
    WaterTheField(2);
    WaterTheField(3);
    WaterTheField(4);
    WaterTheField(5);
    WaterTheField(6);
    ui->statusbar->showMessage("Все живые поля были политы!");
}
void MainWindow::FieldDrought(int amount)
{
    ui->progressBar_1field->setValue(qMax(ui->progressBar_1field->value() -
(amount+rand()%10), 0));
    ui->progressBar_2field->setValue(qMax(ui->progressBar_2field->value() -
(amount+rand()%10), 0));
    ui->progressBar_3field->setValue(qMax(ui->progressBar_3field->value() -
(amount+rand()%10), 0));
    ui->progressBar_4field->setValue(qMax(ui->progressBar_4field->value() -
(amount+rand()%10), 0));
    ui->progressBar_5field->setValue(qMax(ui->progressBar_5field->value() -
(amount+rand()%10), 0));
    ui->progressBar_6field->setValue(qMax(ui->progressBar_6field->value() -
(amount+rand()%10), 0));
}
void MainWindow::CheckTime()
{
    bool anyFieldAlive = false;
    if (ui->progressBar_1field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_1->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_1->setPixmap(scaledPixmap);
    }
    if (ui->progressBar_2field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_2->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_2->setPixmap(scaledPixmap);
    }
    if (ui->progressBar_3field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_3->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_3->setPixmap(scaledPixmap);
    }
    if (ui->progressBar_4field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_4->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_4->setPixmap(scaledPixmap);
    }
    if (ui->progressBar_5field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_5->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_5->setPixmap(scaledPixmap);
    }
    if (ui->progressBar_6field->value() > 0) {anyFieldAlive = true;}
    else {
        QPixmap pixmap(":/image/img/WASTED.png");
        QSize labelSize = ui->LabelDeadField_6->size();
        QPixmap scaledPixmap = pixmap.scaled(labelSize, Qt::KeepAspectRatio);
        ui->LabelDeadField_6->setPixmap(scaledPixmap);
    }
    if (!anyFieldAlive)
    {
        QMessageBox::critical(this, "Смерть", "Вся рассада погибла");
        ui->pushButtonToRestart->show();
    }
}
void MainWindow::PayTime()
{
    QString Day = labelNumberOfDay->text();
    int day = Day.toInt();
    if((day % 7) == 0)
    {
        QString Balance = labelAmountOfMoney->text();
        int balance = Balance.toInt();
        if(ui->progressBar_1field->value() != 0){balance += 50;}
        if(ui->progressBar_2field->value() != 0){balance += 50;}
        if(ui->progressBar_3field->value() != 0){balance += 50;}
        if(ui->progressBar_4field->value() != 0){balance += 50;}
        if(ui->progressBar_5field->value() != 0){balance += 50;}
        if(ui->progressBar_6field->value() != 0){balance += 50;}
        UpdateBalance(balance);
    }
```

```
}
void MainWindow::NEXTDAY()
{
   updateDayNumber();
   PayTime();
   FieldDrought(rand()%10);
   CheckTime();
}
void MainWindow::updateDayNumber()
{
   QString text = labelNumberOfDay->text();
   int value = text.toInt();
   ++value;
   labelNumberOfDay->setText(QString::number(value));
   QFont font = labelNumberOfDay->font();
   font.setPointSize(12);
   labelNumberOfDay->setFont(font);
   labelNumberOfDay->setStyleSheet("color: #aa5500;");
}
void MainWindow::on_buttonNextDay_clicked()
{
   NEXTDAY();
}
void MainWindow::UpdateBalance(int balance)
{
   labelAmountOfMoney->setText(QString::number(balance));
   QFont font = labelAmountOfMoney->font();
   font.setPointSize(12);
   labelAmountOfMoney->setFont(font);
   labelAmountOfMoney->setStyleSheet("color: #55aa00;");
}
void MainWindow::on_buttonBuy10Water_clicked()
{
   QString Balance = labelAmountOfMoney->text();
   int balance = Balance.toInt();
   if(balance >= 45)
   {
      AddWaterInSupply(10);
      balance -= 45;
      UpdateBalance(balance);
   }
}
void MainWindow::on_buttonBuy25Water_clicked()
{
   QString Balance = labelAmountOfMoney->text();
   int balance = Balance.toInt();
   if(balance >= 100)
   {
        AddWaterInSupply(25);
        balance -= 100;
        UpdateBalance(balance);
    }
}
void MainWindow::on_pushButtonToRestart_clicked()
{
    StartTheGame();
}
```

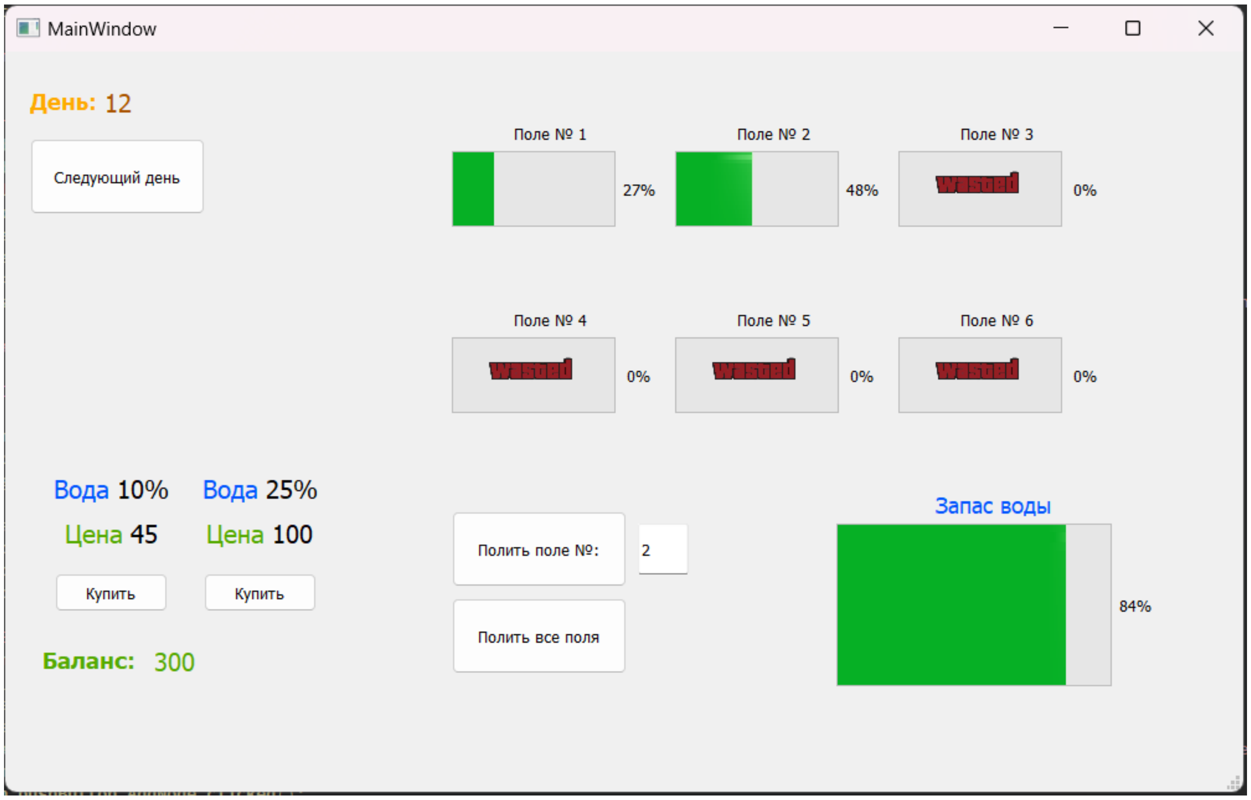
## Демонстрация работы программ:

**https://youtu.be/Rdih7zFSVn0?si=shPdgBdnHlH-xm1Q**

## Скриншоты работы программ:



Заготовленный граф призван!


MainWindow
День: 12
Следующий день
Поле № 1
27%
Поле № 2
48%
Поле № 3
0%
Поле № 4
0%
Поле № 5
0%
Поле № 6
0%
Вода 10%
Цена 45
Купить
Вода 25%
Цена 100
Купить
Баланс: 300
Полить поле №:
2
Полить все поля
Запас воды
84%