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
на тему
«Задача Коммивояжера и АРМ товароведа»

Выполнил студент гр. ИВТ-23-1б
Пискунов Дмитрий Александрович

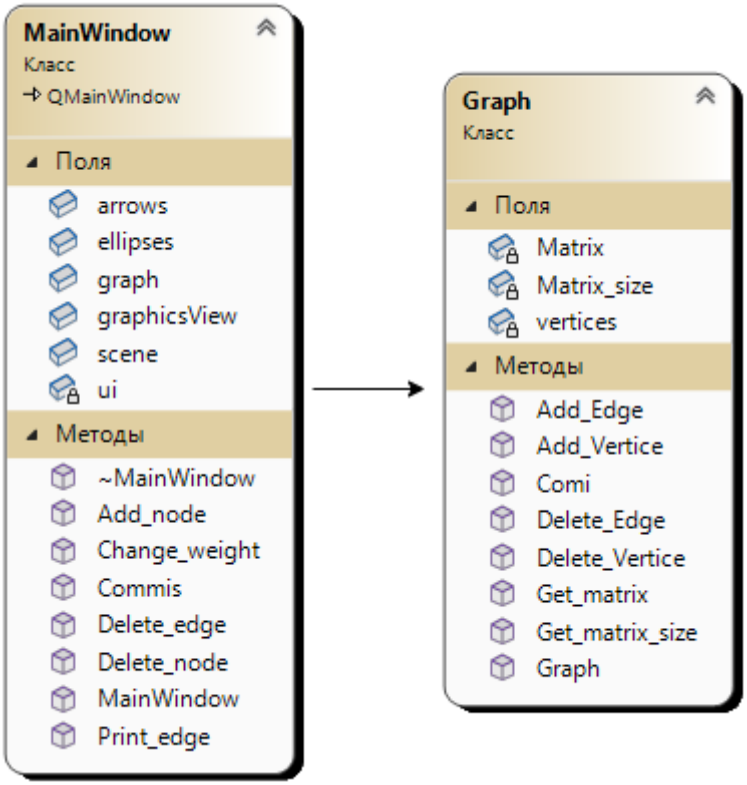
Проверил:
Доцент каф. ИТАС
Полякова Ольга Андреевна

(оценка) (подпись)

(дата)

г. Пермь, 2024

UMLдиаграмма Коммивояжёра



Код программы по решению задачи Коммивояжёра

Файл graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#include<vector>
#include <queue>
#include<iostream>

class Graph{
std::vector<int> vertices{0};
int** Matrix;
int Matrix_size = 1;
public:
    Graph();
    void Add_Vertice();
    void Add_Edge(int,int,int);
    void Delete_Vertice(int);
    void Delete_Edge(int,int);
    int Get_matrix_size();
    int** Get_matrix();
    int Comi();
};

#endif // GRAPH_H
```

Файл mainwindow.h

```h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "graph.h"
#include <QPainter>
#include <QGraphicsScene>
#include <QGraphicsView>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    Graph graph;
    QGraphicsScene *scene;
    QGraphicsView *graphicsView;
    std::vector<QGraphicsEllipseItem*> ellipses = {0};
    std::vector<QGraphicsItemGroup*> arrows = {0};
public slots:
    void Add_edge();
    void Add_node();
    void Print_edge(int,int,int);
    void Delete_edge();
    void Delete_node();
    void Change_weight();
    void Commis();
private:
    Ui::MainWindow *ui;

};
#endif // MAINWINDOW_H
```

Файл graph.cpp

```cpp
#include "graph.h"
#include<stack>
#include<set>
#include<cmath>
#include<map>
#include <algorithm>

Graph::Graph(){

}
void Graph::Add_Vertice(){
    vertices.push_back(vertices.size());
    int c = vertices.size();
    int** Matrix_temp = new int* [c];
    for (int i = 0; i < c; i++) {
        Matrix_temp[i] = new int[c];
        for (int j = 0; j < c; j++) {
            if (i < Matrix_size and j < Matrix_size and c >2) {
                Matrix_temp[i][j] = Matrix[i][j];
            }
            else {
                Matrix_temp[0][j] = j;
                Matrix_temp[i][0] = i;
                Matrix_temp[i][j] = 0;
            }
        }
    }
    Matrix = Matrix_temp;
    Matrix_size++;
}
void Graph::Add_Edge(int name_1,int name_2,int weight){
    Matrix[name_1][name_2] = weight;
}
void Graph::Delete_Vertice(int name){
    vertices[name] = 0;
    for(int i = 0; i<Matrix_size;i++){
         Matrix[name][i] = 0;
         Matrix[i][name] = 0;
    }
}
void Graph::Delete_Edge(int name_1,int name_2){
     Matrix[name_1][name_2] = 0;
}
int Graph::Get_matrix_size(){
    return Matrix_size;
}
int** Graph::Get_matrix(){
    return Matrix;
}
```

```
49      }
50      int Graph::Comi(){//FIX
51          int s = 1;
52          std::vector<int> vertex = {0};
53          for (int i = 1; i < vertices.size(); i++){
54              if (i != s){
55                   vertex.push_back(i);
56              }
57          }
58          int sum = 99999999;
59          do {
60              int current_pathweight = 0;
61              int k = s   ;
62              for (int i = 1; i < vertex.size(); i++) {
63                  if(k!=0){
64                      current_pathweight += Matrix[k][vertex[i]];
65                      k = vertex[i];
66                  }
67              }
68              if (k!=0){
69                  current_pathweight += Matrix[k][s];
70                  if(sum > current_pathweight){
71                      sum = current_pathweight;
72                  }
73              }
74
75
76          } while (next_permutation(vertex.begin(), vertex.end()));
77
78          return sum;
79      }
```

Файл mainwindow.cpp

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

#include <QDebug>
#include <QGraphicsTextItem>
#include <QRandomGenerator>
#include <qmath.h>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(ui->Add_Node_Btn, SIGNAL(clicked()),this,SLOT(Add_node()));
    connect(ui->Add_Edge_Btn, SIGNAL(clicked()), this,SLOT(Add_edge()));
    connect(ui->Delete_Edge_Btn, SIGNAL(clicked()), this, SLOT(Delete_edge()));
    connect(ui->Delete_Node_Btn, SIGNAL(clicked()), this, SLOT(Delete_node()));
    connect(ui->Change_weight_Btn, SIGNAL(clicked()), this, SLOT(Change_weight()));
    connect(ui->Commis_Btn, SIGNAL(clicked()), this, SLOT(Commis()));

    graphicsView = ui -> graphicsView;
    scene = new QGraphicsScene;
    graphicsView -> setScene(scene);
}

MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::Add_node(){
    graph.Add_Vertice();
    QString node = QString("%1").arg(graph.Get_matrix_size()-1);
    QGraphicsEllipseItem *ellipse = scene->addEllipse(640, 275, 50, 50, QPen(Qt::black), QBrush(Qt::lightGray));
    ellipse->setFlag(QGraphicsItem::ItemIsMovable);
    QGraphicsTextItem *textItem = scene->addText(node);
    textItem->setPos(ellipse->boundingRect().center().x() - textItem->boundingRect().width() / 2,
                     ellipse->boundingRect().center().y() - textItem->boundingRect().height() / 2);
    textItem->setParentItem(ellipse);
    scene->installEventFilter(this);
    ellipses.push_back(ellipse);
}
void MainWindow::Add_edge(){
    if((ui->Output_lineEdit->text() == "") or (ui->Input_lineEdit->text() == "") or (ui->weight_lineEdit->text() =="")){
        qDebug() << "Не все поля заполнены";
        return;
    }
    int out = (ui->Output_lineEdit->text()).toInt();
    int in = (ui->Input_lineEdit->text()).toInt();
    int weight = (ui->weight_lineEdit->text()).toInt();
    if(out < 1 or in < 1 or weight < 1){
        qDebug() << "ни какое из значений не может быть меньше единицы";
        return;
    }
    if(graph.Get_matrix_size()-1 < out or graph.Get_matrix_size()-1 < in){
        qDebug() << "Таких(ой) вершин(ы) нет";
```

```
            qDebug() << "Таких(ой) вершин(ы) нет";
            return;
        }
        graph.Add_Edge(out, in, weight);
        Print_edge(out, in, weight);
    }
    void MainWindow::Print_edge(int ou, int inn, int we){
        if(we == 0){
            return;
        }
        int out = ou;
        int in = inn;
        int weight = we;


        if(in != out){                          //стрелка
            QGraphicsEllipseItem *out_ellipse = ellipses[out];
            QGraphicsEllipseItem *in_ellipse = ellipses[in];

            QPointF center1 = out_ellipse->mapToScene(out_ellipse->boundingRect().center());
            QPointF center2 = in_ellipse->mapToScene(in_ellipse->boundingRect().center());


            qreal angle = qAtan2(center2.y() - center1.y(), center2.x() - center1.x());

            QPointF new_out(center1.x() + 25 * qCos(angle), center1.y() + 25 * qSin(angle));
            QPointF new_in(center2.x() + 25 * qCos(angle + M_PI), center2.y() + 25 * qSin(angle + M_PI));

            QGraphicsLineItem *line1 = new QGraphicsLineItem();
            line1->setLine(QLineF(new_out, new_in));
            scene->addItem(line1);

            QPolygonF arrowHead;

            qreal angle_arrow = qAtan2(new_in.y() - new_out.y(), new_in.x() - new_out.x());
            qreal arrowLength = 10.0;

            qreal arrowAngle = M_PI / 6.0;

            QPointF arrowP1 = new_in - QPointF(arrowLength * std::cos(angle_arrow + arrowAngle), arrowLength * std::sin(angle_arrow + arrowAngle));
            QPointF arrowP2 = new_in - QPointF(arrowLength * std::cos(angle_arrow - arrowAngle), arrowLength * std::sin(angle_arrow - arrowAngle));

            arrowHead << new_in << arrowP1 << arrowP2;

            QGraphicsPolygonItem *arrow1 = new QGraphicsPolygonItem(arrowHead);
            arrow1->setBrush(Qt::black);
            arrow1->setPen(Qt::NoPen);
            scene->addItem(arrow1);

            QPointF textPos2 = arrowP2;
            QGraphicsTextItem* textItem2 = scene->addText(QString::number(weight));
            textItem2->setPos(textPos2);

```

```
108
109             QList<QGraphicsItem*> items;
110             items << arrow1 << textItem2 <<line1;
111             QGraphicsItemGroup *group = scene->createItemGroup(items);
112             arrows.push_back(group);
113
114             out_ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
115             in_ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
116         }
117
118         if(in == out){                                      //петля
119             QGraphicsEllipseItem *ellipse = ellipses[out];
120
121             QPointF center = ellipse->mapToScene(ellipse->boundingRect().center());
122
123             qreal radius = ellipse->boundingRect().width() / 2.0;
124
125             qreal angle_loop = 45 * M_PI / 180;
126
127             QPointF start(center.x() + radius * qCos(angle_loop), center.y() + radius * qSin(angle_loop));
128             QPointF end(center.x() + radius * qCos(angle_loop + M_PI), center.y() + radius * qSin(angle_loop + M_PI));
129
130             radius *= 4;
131
132             QPointF controlPoint1(center.x() + radius * qCos(angle_loop - M_PI / 4), center.y() + radius * qSin(angle_loop - M_PI / 4));
133             QPointF controlPoint2(center.x() + radius * qCos(angle_loop + M_PI + M_PI / 4), center.y() + radius * qSin(angle_loop + M_PI + M_PI / 4));
134
135             QPainterPath loopPath;
136             loopPath.moveTo(start);
137             loopPath.cubicTo(controlPoint1, controlPoint2, end);
138             scene->addPath(loopPath);
139
140
141             QPointF textPos2(center.x() + radius * qCos(angle_loop + M_PI + M_PI / 4), center.y() + 25 + radius * qSin(angle_loop + M_PI + M_PI / 4));
142             QGraphicsTextItem* textItem2 = scene->addText(QString::number(weight));
143             textItem2->setPos(textPos2);
144
145             QGraphicsPathItem *loopItem = new QGraphicsPathItem(loopPath);
146
147             QList<QGraphicsItem*> items;
148             items << loopItem << textItem2;
149             QGraphicsItemGroup *group = scene->createItemGroup(items);
150             arrows.push_back(group);
151
152
153             ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
154         }
155     }
156 void MainWindow::Delete_edge(){
157     if(ui->Output_Delete_lineEdit->text() == "" or ui->Input_Delete_lineEdit->text() == ""){
158         qDebug() <<"Заполните все поля";
159         return;
```

```
159             return;
160         }
161         int out = (ui->Output_Delete_lineEdit->text()).toInt();
162         int in = (ui->Input_Delete_lineEdit->text()).toInt();
163         if(graph.Get_matrix_size()-1 < out or graph.Get_matrix_size()-1 < in){
164             qDebug() << "Таких(ой) вершин(ы) нет";
165             return;
166         }
167         int** matrix = graph.Get_matrix();
168         if(matrix[out][in] == 0){
169             qDebug() << "Такого ребра нет";
170             return;
171         }
172         graph.Delete_Edge(out, in);
173         matrix = graph.Get_matrix();
174         for (unsigned long long int i = 0; i < arrows.size(); i++){
175             scene->removeItem(arrows[i]);
176             delete arrows[i];
177         }
178         arrows.clear();
179         for(int i = 1; i< graph.Get_matrix_size(); i++){
180             for(int j = 1; j < graph.Get_matrix_size(); j++){
181                     Print_edge(i, j, matrix[i][j]);
182             }
183         }
184     }
185     void MainWindow::Delete_node(){
186         if(ui->Delete_node_lineEdit->text() == ""){
187             qDebug() <<"Заполните все поля";
188             return;
189         }
190         int node = (ui->Delete_node_lineEdit->text()).toInt();
191         if(node > graph.Get_matrix_size()){
192             qDebug() << "Такой вершины нет";
193             return;
194         }
195         for (unsigned long long int i = 0; i < ellipses.size(); i++){
196             if(node == i){
197                 scene->removeItem(ellipses[i]);
198                 ellipses[i] = 0;
199             }
200         }
201         for (int i = 1; i < graph.Get_matrix_size(); i++) {
202             for (int j = 1; j < graph.Get_matrix_size(); j++) {
203                 if(i == node or j == node){
204                     graph.Delete_Edge(i, j);
205                 }
206             }
207         }
208         for (unsigned long long int i = 0; i < arrows.size(); i++){
209             scene->removeItem(arrows[i]);
210             delete arrows[i];
211         }
```

```
        }
        arrows.clear();
        int** matrix = graph.Get_matrix();
        for(int i = 1; i< graph.Get_matrix_size(); i++){
            for(int j = 1; j < graph.Get_matrix_size(); j++){
                    Print_edge(i, j, matrix[i][j]);
            }
        }
    }
    void MainWindow::Change_weight(){
        if((ui->Output_lineEdit->text() == "") or (ui->Input_lineEdit->text() == "") or (ui->weight_lineEdit->text() =="")){
            qDebug() << "Не все поля заполнены";
            return;
        }
        int out = (ui->Output_lineEdit->text()).toInt();
        int in = (ui->Input_lineEdit->text()).toInt();
        int weight = (ui->weight_lineEdit->text()).toInt();
        if(out < 1 or in < 1 or weight < 1){
            qDebug() << "ни какое из значений не может быть меньше единицы";
            return;
        }
        if(graph.Get_matrix_size()-1 < out or graph.Get_matrix_size()-1 < in){
            qDebug() << "Таких(ой) вершин(ы) нет";
            return;
        }
        graph.Add_Edge(out, in, weight);
        for (unsigned long long int i = 0; i < arrows.size(); i++){
            scene->removeItem(arrows[i]);
            delete arrows[i];
        }
        arrows.clear();
        int** matrix = graph.Get_matrix();
        for(int i = 1; i< graph.Get_matrix_size(); i++){
            for(int j = 1; j < graph.Get_matrix_size(); j++){
                    Print_edge(i, j, matrix[i][j]);
            }
        }
    }
    void MainWindow::Commis(){
        ui->Output_algoritms->clear();
        int c = graph.Comi();
        QString result;
        result.append("Решение задачи Коммивояжёра = ");
        result.append(QString::number(c));
        ui->Output_algoritms->setText(result);

    }
```

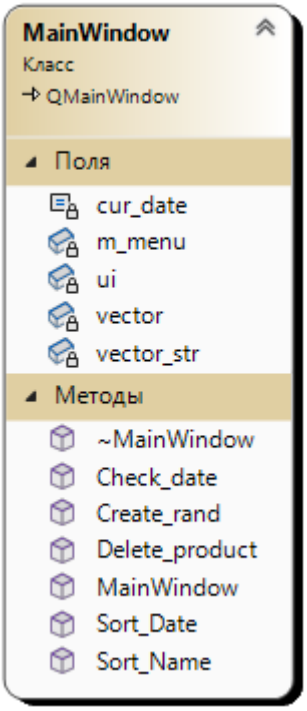
Файл main.cpp

```
#include "mainwindow.h"

#include <QApplication>

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

UML-Диаграмма АРМ товароведа



Код программы АРМ товароведа

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDate>
#include <QtDebug>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
public slots:
    void Add_product();
    void Delete_product();
    void Sort_Date();
    void Sort_Name();
    void Check_date();
    void Create_rand();
private:
    Ui::MainWindow  *ui;
    const QDate cur_date = QDate::currentDate();
    std::vector<QDate> vector;
    std::vector<QString> vector_str;
    QMenu *m_menu;
};

#endif // MAINWINDOW_H
```

Файл mainwindow.cpp

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QMenu>
#include <ctime>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    connect(ui->Add_Btn, SIGNAL(clicked()), this, SLOT(Add_product()));
    connect(ui->Delete_Btn,SIGNAL(clicked()), this, SLOT(Delete_product()));

    Create_rand();

    m_menu = new QMenu(ui->Sorts_Btn);
    m_menu->addAction("По дате", this, SLOT(Sort_Date()));
    m_menu->addAction("По алфавиту",this, SLOT(Sort_Name()));


    ui->Sorts_Btn->setMenu(m_menu);
}

MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::Add_product(){
    QString name = ui->Name_lineEdit->text();
    QString date = ui->dateEdit->text();
    if (name=="" or date ==""){
        QMessageBox msgBox;
        msgBox.setText("Заполнены не все поля");
        msgBox.exec();
        return;
    }
    if(cur_date > ui->dateEdit->date()){
        QMessageBox msgBox;
        msgBox.setText("Срок годности товара истёк :(");
        msgBox.exec();
        return;
    }
    ui->tableWidget->setRowCount(ui->tableWidget->rowCount()+1);
    ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 0, new QTableWidgetItem(name));
    ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 1, new QTableWidgetItem(date));
    ui->Name_lineEdit->clear();
    vector.push_back(ui->dateEdit->date());
    vector_str.push_back(name);
    Check_date();
}
void MainWindow::Delete_product(){
    QModelIndexList selectedRows = ui->tableWidget->selectionModel()->selectedRows();
```

```
54          QModelIndexList selectedRows = ui->tableWidget->selectionModel()->selectedRows();
55          if(selectedRows.empty()){
56              QMessageBox msgBox;
57              msgBox.setText("Выделите строку которую хотите удалить");
58              msgBox.exec();
59              return;
60          }
61          QMessageBox::StandardButton reply;
62            reply = QMessageBox::question(this, "Предупреждение", "Вы уверены что хотите удалить выделенную(ые) строку(и)?",
63                                          QMessageBox::Yes|QMessageBox::No);
64            if (reply == QMessageBox::No) {
65              return;
66            }
67
68              while (!selectedRows.empty()) {
69
70                  ui->tableWidget->removeRow(selectedRows[0].row());
71                  selectedRows = ui->tableWidget->selectionModel()->selectedRows();
72              }
73
74             vector.clear();
75             vector_str.clear();
76             for(int i = 0; i < ui->tableWidget->rowCount(); i++){
77                 QTableView* myTable = ui->tableWidget;
78                 QVariant myData;
79                 QModelIndex myIndex;
80                 myIndex = myTable->model()->index(i, 1 , QModelIndex());
81                 myData = myTable->model()->data( myIndex, Qt::DisplayRole);
82                 QString tmp = myData.toString();
83                 QDate temp = QDate::fromString(tmp, "dd.MM.yyyy");
84                 vector.push_back(temp);
85                 QVariant myData2;
86                 QModelIndex myIndex2;
87                 myIndex2 = myTable->model()->index(i, 0 , QModelIndex());
88                 myData2 = myTable->model()->data( myIndex2, Qt::DisplayRole);
89                 QString tmp2 = myData2.toString();
90                 vector_str.push_back(tmp2);
91             }
92      }
93      void MainWindow::Sort_Date(){
94          int c =vector.size();
95          while(c--){
96              bool swapped = true;
97              for(int i = 0; i < vector.size()-1; i++){
98                  if(vector[i] > vector[i+1]){
99                      std::swap(vector[i],vector[i+1]);
100                     std::swap(vector_str[i],vector_str[i+1]);
101                     swapped = false;
102                 }
103             }
104             if(swapped == true){
105                 break;
```

```
                break;
            }
        }

        while(ui->tableWidget->rowCount() > 0){
            ui->tableWidget->removeRow(0);
        }
        for(int i = 0; i < vector.size();i++){
            QString name = vector_str[i];
            QString date = vector[i].toString("dd.MM.yyyy");
            ui->tableWidget->setRowCount(ui->tableWidget->rowCount()+1);
            ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 0, new QTableWidgetItem(name));
            ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 1, new QTableWidgetItem(date));
        }
        Check_date();
    }
    void MainWindow::Check_date(){
        for(int i =0; i < vector.size(); i++){
            QString str = ui->tableWidget->item(i,1)->text();
            QDate product_date = QDate::fromString(str, "dd.MM.yyyy");

            if(cur_date >= product_date){
                ui->tableWidget->item(i,0)->setBackgroundColor(Qt::red);
                ui->tableWidget->item(i,1)->setBackgroundColor(Qt::red);
            }
            else if(cur_date.daysTo(product_date) <= 3){
                ui->tableWidget->item(i,0)->setBackgroundColor(Qt::yellow);
                ui->tableWidget->item(i,1)->setBackgroundColor(Qt::yellow);
            }
        }
    }
    void MainWindow::Sort_Name(){
        int c =vector.size();
        while(c--){
            bool swapped = true;
            for(int i = 0; i < vector.size()-1; i++){
               if(vector_str[i][0] > vector_str[i+1][0]){
                   std::swap(vector_str[i], vector_str[i+1]);
                   std::swap(vector[i], vector[i+1]);
                   swapped = false;
               }
            }
            if(swapped == true){
                break;
            }
        }
        while(ui->tableWidget->rowCount() > 0){
            ui->tableWidget->removeRow(0);
        }
        for(int i = 0; i < vector.size();i++){
            QString name = vector_str[i];
            QString date = vector[i].toString("dd.MM.yyyy");
```

```
            QString date = vector[i].toString("dd.MM.yyyy");
            ui->tableWidget->setRowCount(ui->tableWidget->rowCount()+1);
            ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 0, new QTableWidgetItem(name));
            ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 1, new QTableWidgetItem(date));
        }
        Check_date();
    }
void MainWindow::Create_rand(){
    srand(time(0));
    QString products[30] = {"Молоко", "Хлеб", "Яблоки", "Мясо", "Яйца", "Рис", "Картофель", "Вода", "Сахар", "Масло","Соль", "
    for(int i = 0; i< 50; i++){
        int r = rand()%30;
        int day = rand()%27+1;
        int month = rand()%11+1;
        int year = rand()%15+2020;
        QString d = QString::number(day);
        QString m = QString::number(month);
        QString y = QString::number(year);
        QString date;
        if(day<10 and month<10){
            date ='0'+ d+'.' + '0' + m + '.' + y;
        }
        else if(day < 10){
            date ='0'+ d+'.' + m + '.' + y;
        }
        else if(month < 10){
             date = d+'.'+ '0' + m + '.' + y;
        }
        else{
             date = d+'.' + m + '.' + y;
        }

        ui->tableWidget->setRowCount(ui->tableWidget->rowCount()+1);
        ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 0, new QTableWidgetItem(products[r]));
        ui->tableWidget->setItem(ui->tableWidget->rowCount()-1, 1, new QTableWidgetItem(date));
        QDate temp = QDate::fromString(date, "dd.MM.yyyy");
        vector.push_back(temp);
        vector_str.push_back(products[r]);
    }
    Check_date();
}
```

Видео на youtube

Решение задачи Коммивояжёра - https://youtu.be/SQJx8Cv8IF8

АРМ товароведа - https://youtu.be/Ftu6zwCdvOA