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 кладовщик»

Выполнил студент гр. ИВТ-23-1б
Давыдов Андрей Юрьевич

Проверил:
Доцент каф. ИТАС
Полякова Ольга Андреевна

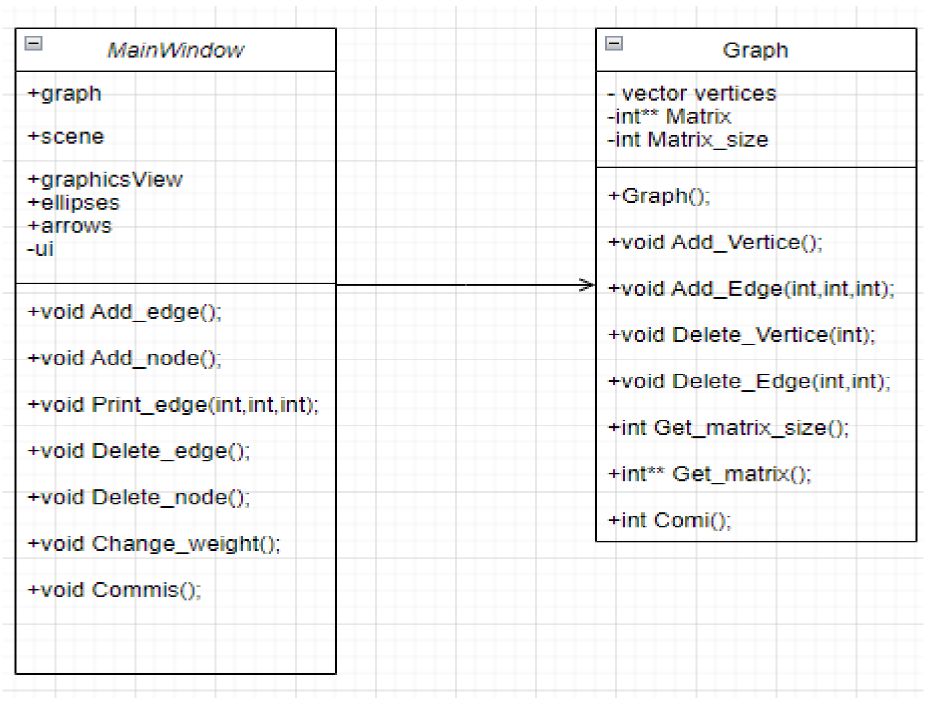
____

______________ ______________
(оценка) (подпись)

____
(дата)

г. Пермь, 2024

UML –диаграмма Коммивояжера

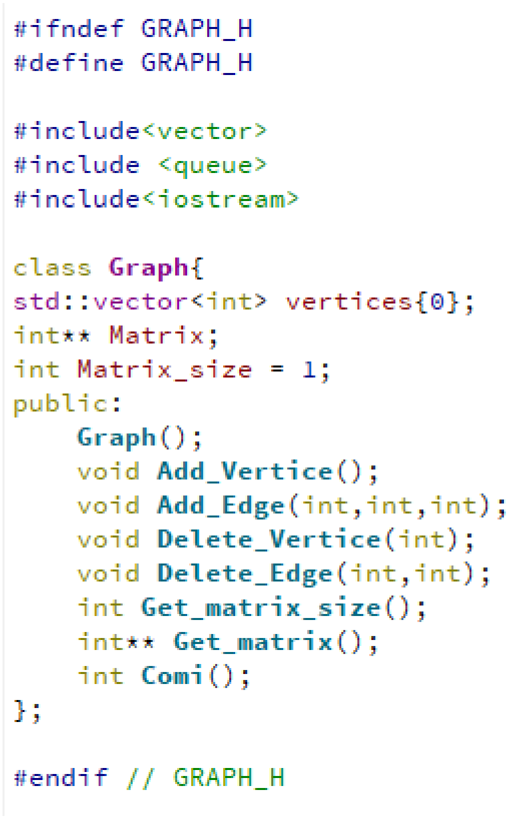


Код программы по решению задачи Коммивояжера

Файл mainwindow.h

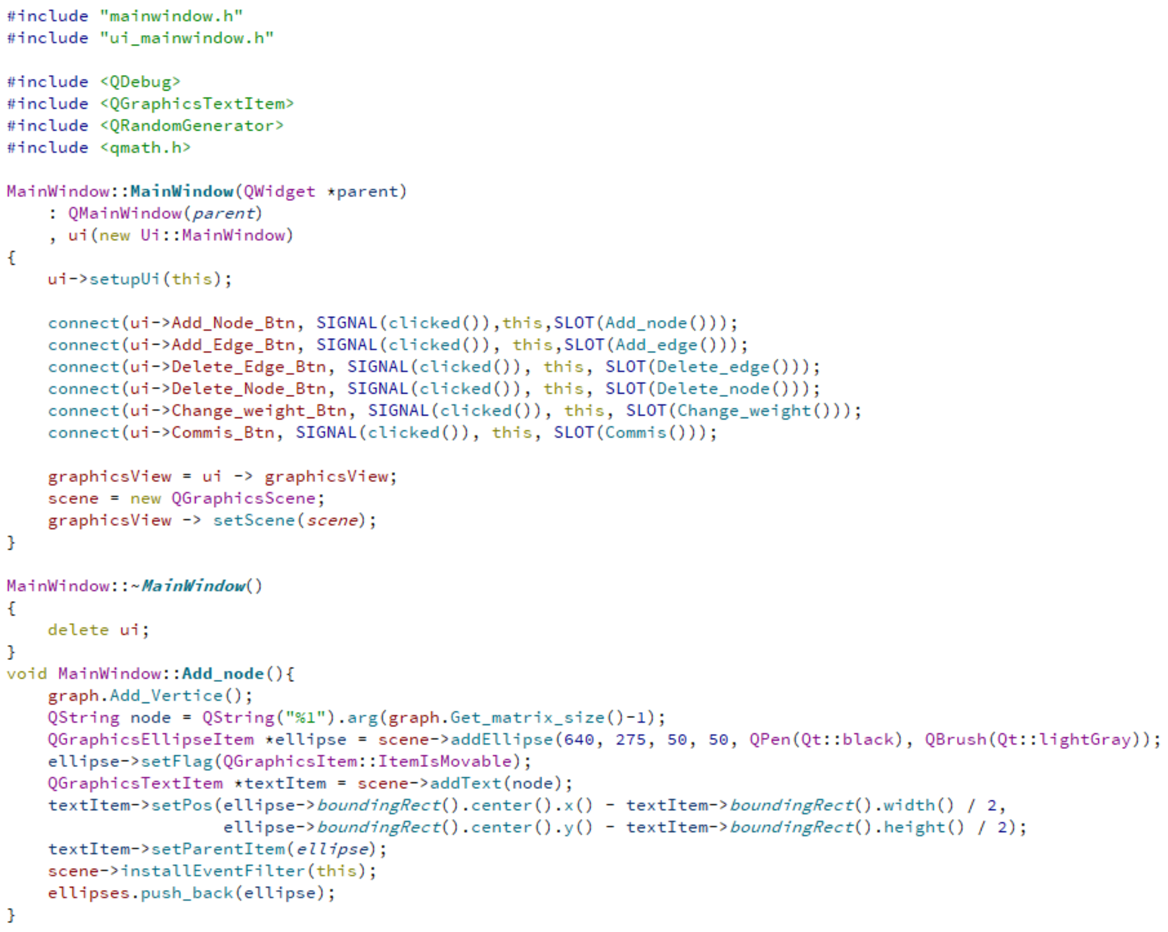
```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#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.h

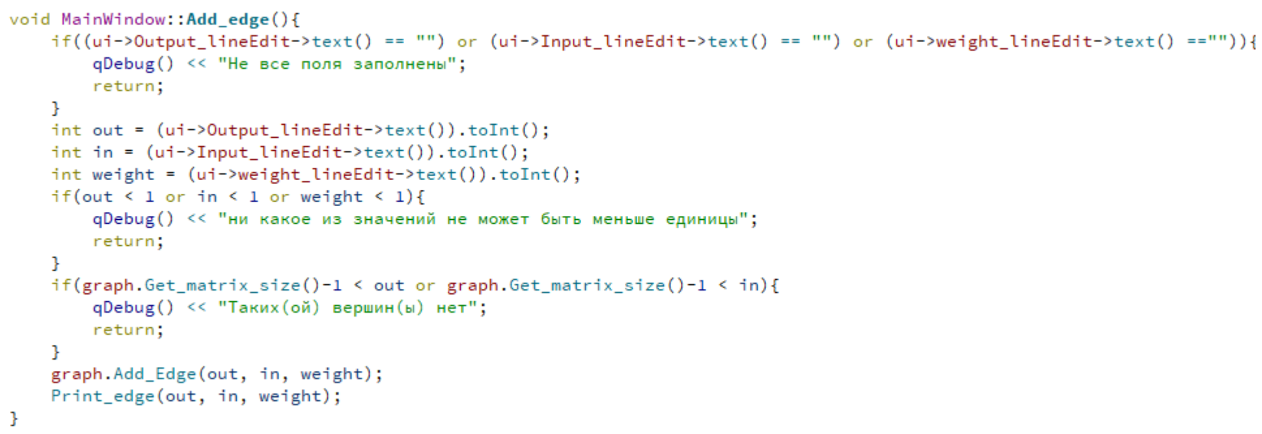


```cpp
#ifndef GRAPH_H
#define GRAPH_H

#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
```

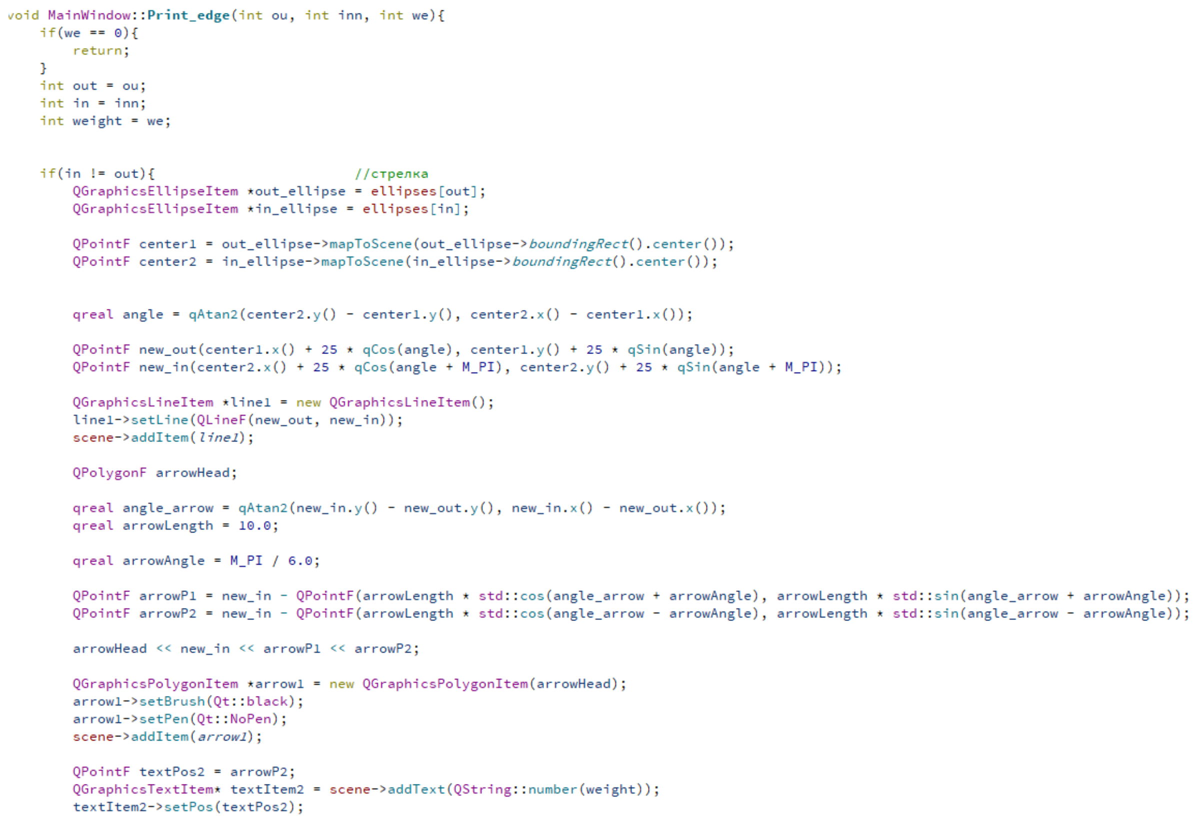
Файл mainwindow.cpp



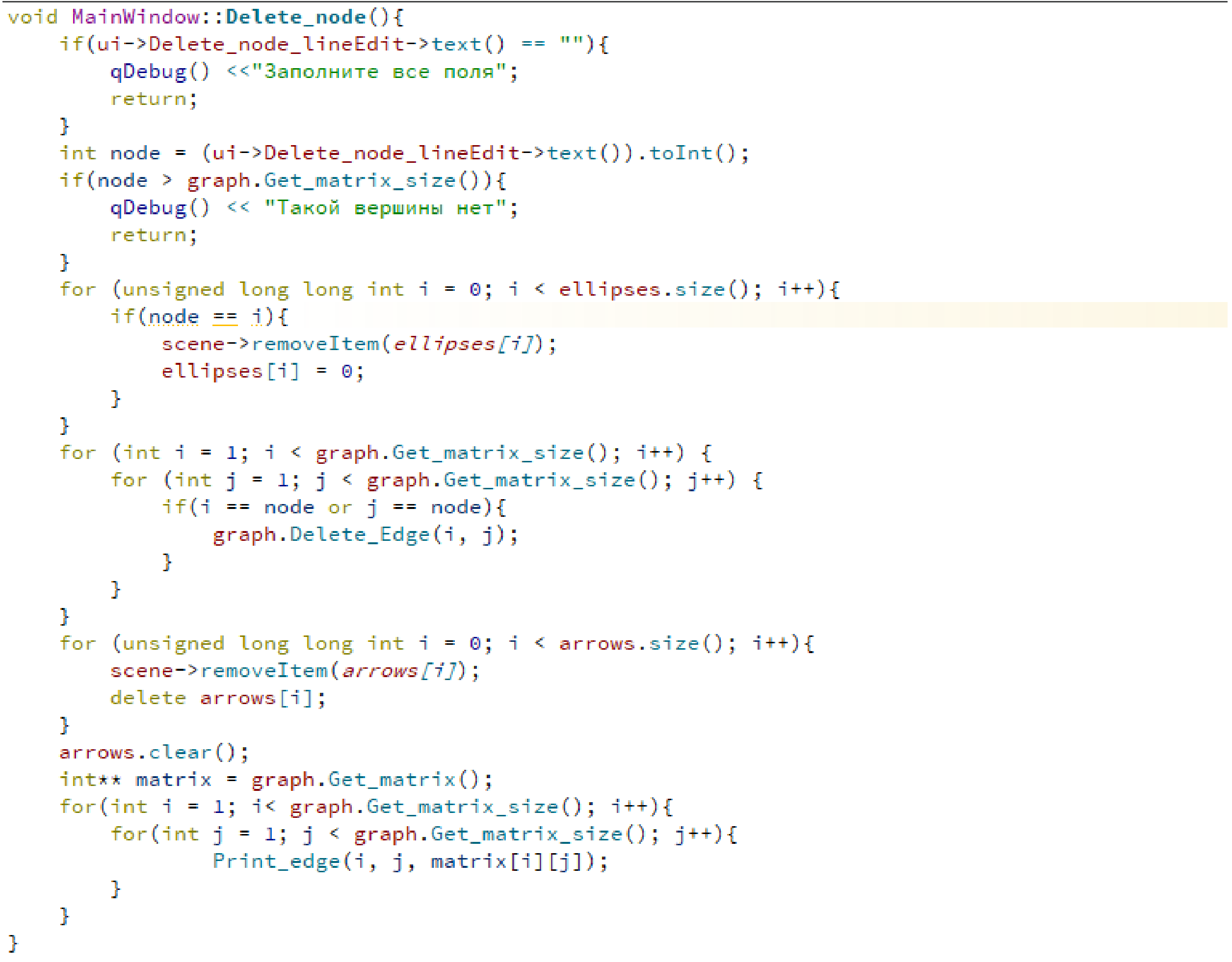
```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

#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);
}
```



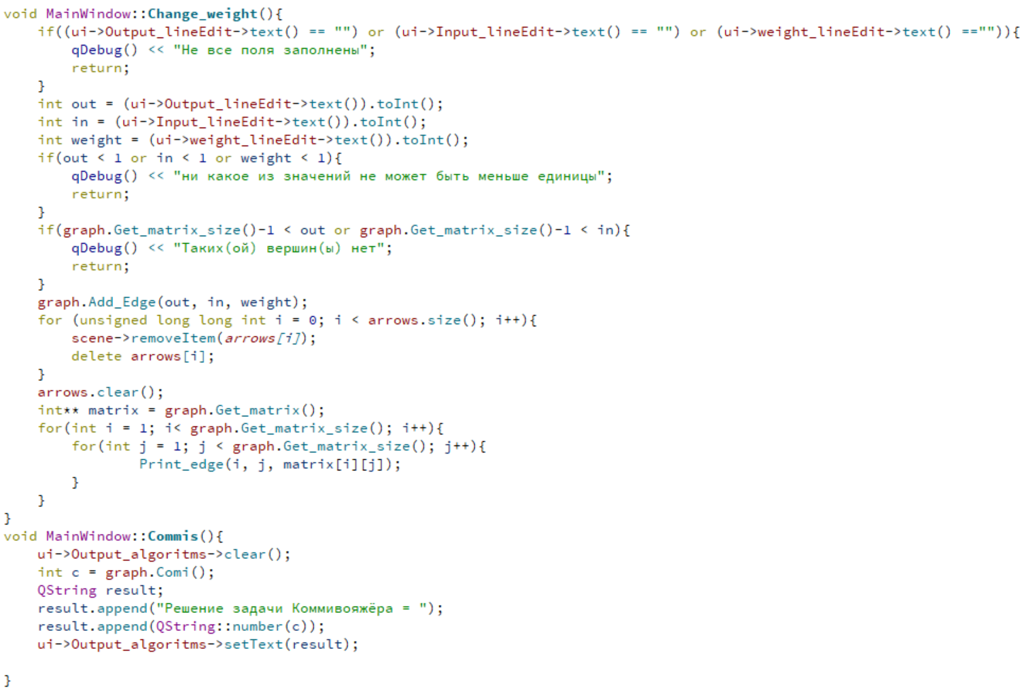
```cpp
void MainWindow::Add_edge(){
    if((ui->Output_lineEdit->text() == "") or (ui->Input_lineEdit->text() == "") or (ui->weight_lineEdit->text() ==""")){
        qDebug() << "Не все поля заполнены";
        return;
    }
    int out = (ui->Output_lineEdit->text()).toInt();
    int in = (ui->Input_lineEdit->text()).toInt();
    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";
        return;
    }
    graph.Add_Edge(out, in, weight);
    Print_edge(out, in, weight);
}
```



```cpp
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);
        QList<QGraphicsItem*> items;
        items << arrow1 << textItem2 <<line1;
        QGraphicsItemGroup *group = scene->createItemGroup(items);
        arrows.push_back(group);

        out_ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
        in_ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
    }

    if(in == out){                                    //петля
         QGraphicsEllipseItem *ellipse = ellipses[out];

         QPointF center = ellipse->mapToScene(ellipse->boundingRect().center());

         qreal radius = ellipse->boundingRect().width() / 2.0;

         qreal angle_loop = 45 * M_PI / 180;

         QPointF start(center.x() + radius * qCos(angle_loop), center.y() + radius * qSin(angle_loop));
         QPointF end(center.x() + radius * qCos(angle_loop + M_PI), center.y() + radius * qSin(angle_loop + M_PI));

         radius *= 4;

         QPointF controlPoint1(center.x() + radius * qCos(angle_loop - M_PI / 4), center.y() + radius * qSin(angle_loop - M_PI / 4));
         QPointF controlPoint2(center.x() + radius * qCos(angle_loop + M_PI + M_PI / 4), center.y() + radius * qSin(angle_loop + M_PI + M_PI / 4));

         QPainterPath loopPath;
         loopPath.moveTo(start);
         loopPath.cubicTo(controlPoint1, controlPoint2, end);
         scene->addPath(loopPath);


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

         QGraphicsPathItem *loopItem = new QGraphicsPathItem(loopPath);

         QList<QGraphicsItem*> items;
         items << loopItem << textItem2;
         QGraphicsItemGroup *group = scene->createItemGroup(items);
         arrows.push_back(group);


         ellipse->setFlag(QGraphicsItem::ItemIsMovable, false);
    }
}

void MainWindow::Delete_edge(){
    if(ui->Output_Delete_lineEdit->text() == "" or ui->Input_Delete_lineEdit->text() == ""){
        qDebug() <<"Заполните все поля";
        return;
    }
    int out = (ui->Output_Delete_lineEdit->text()).toInt();
    int in = (ui->Input_Delete_lineEdit->text()).toInt();
    if(graph.Get_matrix_size()-1 < out or graph.Get_matrix_size()-1 < in){
        qDebug() << "Таких(ой) вершин(ы) нет";
        return;
    }
    int** matrix = graph.Get_matrix();
    if(matrix[out][in] == 0){
        qDebug() << "Такого ребра нет";
        return;
    }
    graph.Delete_Edge(out, in);
    matrix = graph.Get_matrix();
    for (unsigned long long int i = 0; i < arrows.size(); i++){
        scene->removeItem(arrows[i]);
        delete arrows[i];
    }
    arrows.clear();
    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]);
        }
    }
}
```



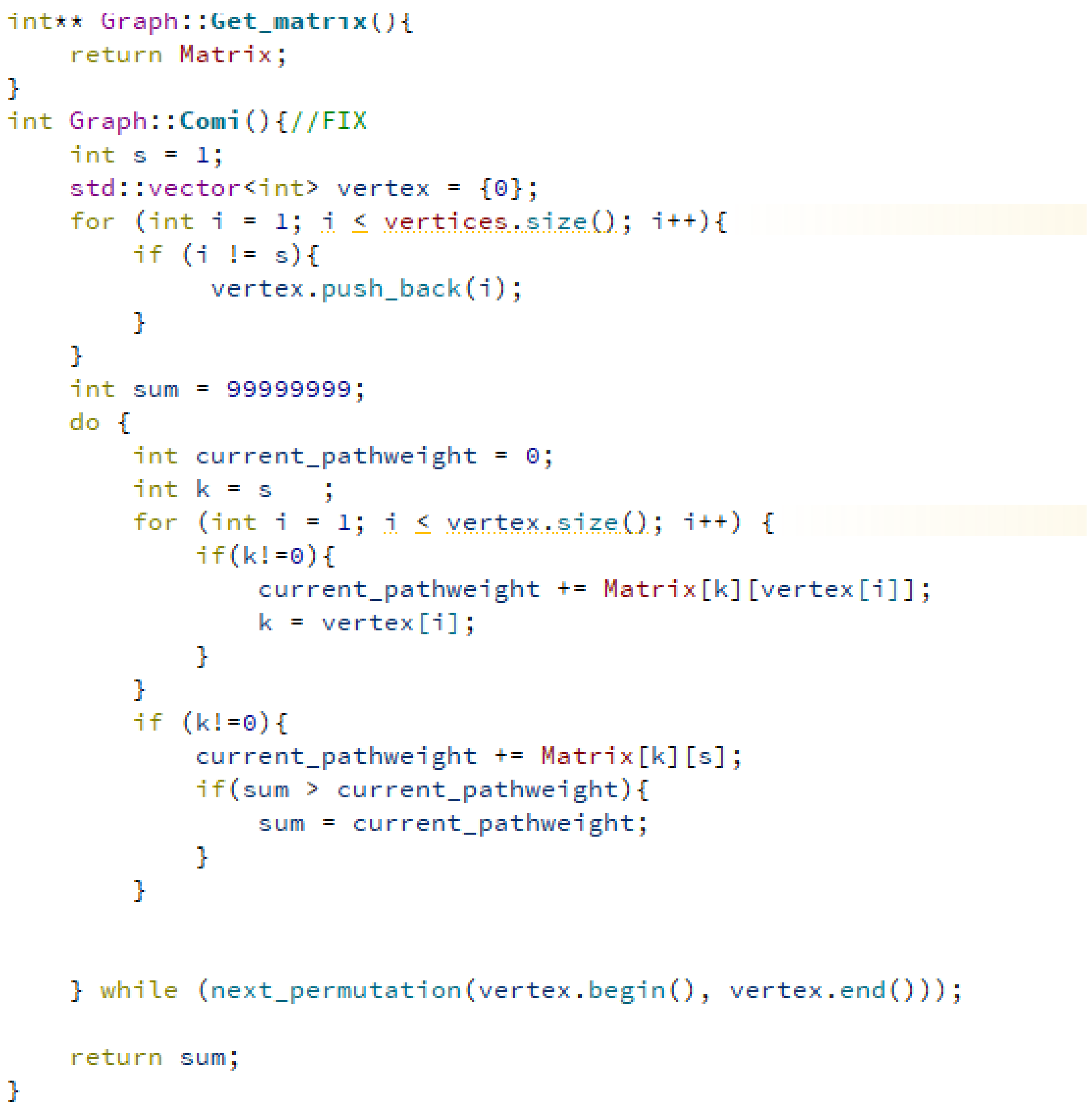
```cpp
void MainWindow::Delete_node(){
    if(ui->Delete_node_lineEdit->text() == ""){
        qDebug() <<"Заполните все поля";
        return;
    }
    int node = (ui->Delete_node_lineEdit->text()).toInt();
    if(node > graph.Get_matrix_size()){
        qDebug() << "Такой вершины нет";
        return;
    }
    for (unsigned long long int i = 0; i < ellipses.size(); i++){
        if(node == i){
            scene->removeItem(ellipses[i]);
            ellipses[i] = 0;
        }
    }
    for (int i = 1; i < graph.Get_matrix_size(); i++) {
        for (int j = 1; j < graph.Get_matrix_size(); j++) {
            if(i == node or j == node){
                graph.Delete_Edge(i, j);
            }
        }
    }
    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]);
        }
    }
}
```



```cpp
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);

}
```

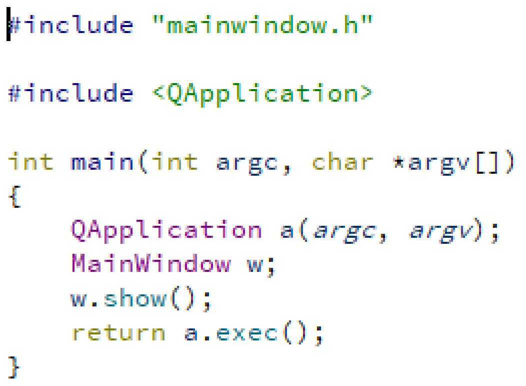
Файл graph.cpp

```
#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;
}
```



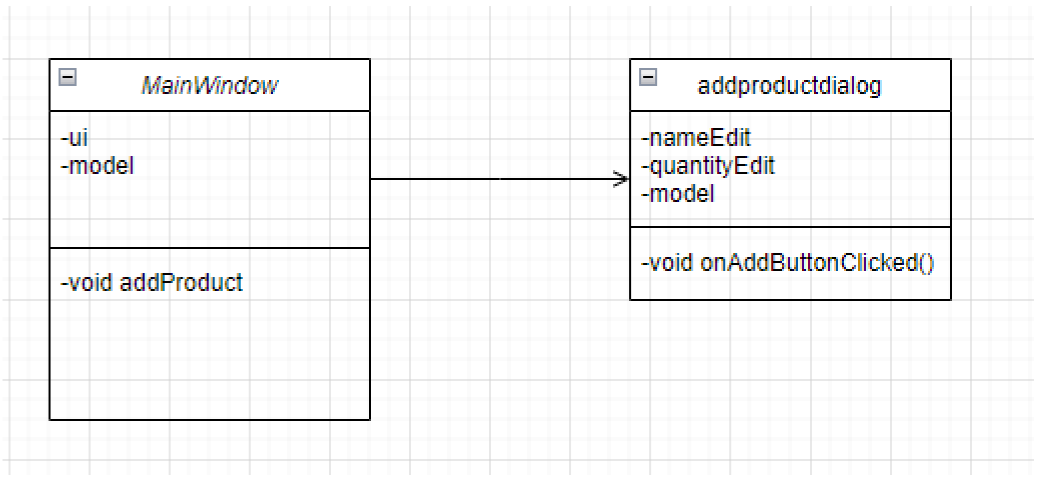
```
int** Graph::Get_matrix(){
    return Matrix;
}
int Graph::Comi(){//FIX
    int s = 1;
    std::vector<int> vertex = {0};
    for (int i = 1; i <= vertices.size(); i++){
        if (i != s){
             vertex.push_back(i);
        }
    }
    int sum = 99999999;
    do {
        int current_pathweight = 0;
        int k = s    ;
        for (int i = 1; i <= vertex.size(); i++) {
            if(k!=0){
                current_pathweight += Matrix[k][vertex[i]];
                k = vertex[i];
            }
        }
        if (k!=0){
            current_pathweight += Matrix[k][s];
            if(sum > current_pathweight){
                sum = current_pathweight;
            }
        }


    } while (next_permutation(vertex.begin(), vertex.end()));

    return sum;
}
```

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 кладовщик



Код программ АРМ кладовщик:

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStandardItemModel>

QT_BEGIN_NAMESPACE
class QTableView;
class QPushButton;
QT_END_NAMESPACE

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow {
    Q_OBJECT

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

private slots:
    void addProduct();

private:
    Ui::MainWindow *ui;
    QStandardItemModel *model;
};

#endif // MAINWINDOW_H
```

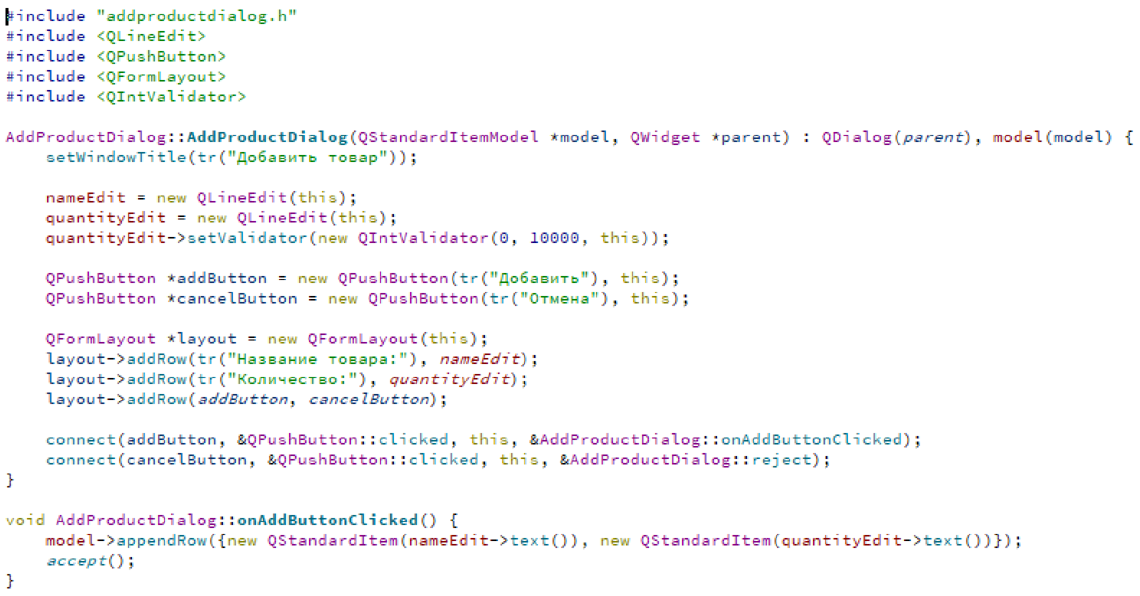
Файл addproducdialog.h

```
#ifndef ADDPRODUCTDIALOG_H
#define ADDPRODUCTDIALOG_H

#include <QDialog>
#include <QStandardItemModel>

QT_BEGIN_NAMESPACE
class QLineEdit;
QT_END_NAMESPACE

class AddProductDialog : public QDialog {
    Q_OBJECT

public:
    AddProductDialog(QStandardItemModel *model, QWidget *parent = nullptr);

private slots:
    void onAddButtonClicked();

private:
    QLineEdit *nameEdit;
    QLineEdit *quantityEdit;
    QStandardItemModel *model;
};

#endif // ADDPRODUCTDIALOG_H
```

Файл mainwindow.cpp

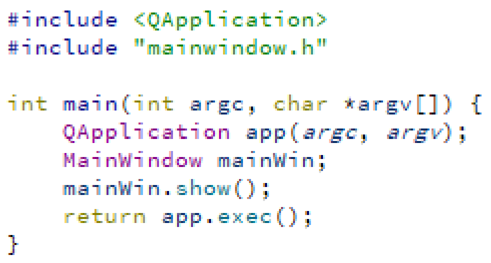
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "addproductdialog.h"
#include <QTableView>
#include <QPushButton>
#include <QVBoxLayout>
#include <QStandardItemModel>
#include <QRandomGenerator>
#include <QStyledItemDelegate>
#include <QPainter>

class HighlightDelegate : public QStyledItemDelegate {
public:
    HighlightDelegate(QObject *parent = nullptr) : QStyledItemDelegate(parent) {}

    void paint(QPainter *painter, const QStyleOptionViewItem &option, const QModelIndex &index) const override {
        QVariant quantity = index.model()->data(index, Qt::DisplayRole);
        if (quantity.toInt() < 10) {
            painter->fillRect(option.rect, Qt::yellow);
        }
        QStyledItemDelegate::paint(painter, option, index);
    }
};

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

    model = new QStandardItemModel(this);
    ui->tableView->setModel(model);
    ui->tableView->setItemDelegate(new HighlightDelegate(this));

    QStringList productsList = {"Молоко", "Хлеб", "Яйца", "Сыр", "Йогурт", "Масло", "Мука", "Сахар", "Рис", "Макароны", "Картофель", "Овощи", "Фрукты", "Мясо", "Рыба", "Колбаса", "Консервы", "Сухофрукты", "Орехи", "Зелень"};
    for (const QString &productName : productsList) {
        QStandardItem *productItem = new QStandardItem(productName);
        QStandardItem *quantityItem = new QStandardItem(QString::number(QRandomGenerator::global()->bounded(1, 20)));
        model->appendRow({productItem, quantityItem});
    }

    connect(ui->addButton, &QPushButton::clicked, this, &MainWindow::addProduct);
}

void MainWindow::addProduct() {
    AddProductDialog *dialog = new AddProductDialog(model, this);
    dialog->exec();
}

MainWindow::~MainWindow() {
    delete ui;
}
```

Файл addproducdialog.cpp



```
#include "addproductdialog.h"
#include <QLineEdit>
#include <QPushButton>
#include <QFormLayout>
#include <QIntValidator>

AddProductDialog::AddProductDialog(QStandardItemModel *model, QWidget *parent) : QDialog(parent), model(model) {
    setWindowTitle(tr("Добавить товар"));

    nameEdit = new QLineEdit(this);
    quantityEdit = new QLineEdit(this);
    quantityEdit->setValidator(new QIntValidator(0, 10000, this));

    QPushButton *addButton = new QPushButton(tr("Добавить"), this);
    QPushButton *cancelButton = new QPushButton(tr("Отмена"), this);

    QFormLayout *layout = new QFormLayout(this);
    layout->addRow(tr("Название товара:"), nameEdit);
    layout->addRow(tr("Количество:"), quantityEdit);
    layout->addRow(addButton, cancelButton);

    connect(addButton, &QPushButton::clicked, this, &AddProductDialog::onAddButtonClicked);
    connect(cancelButton, &QPushButton::clicked, this, &AddProductDialog::reject);
}

void AddProductDialog::onAddButtonClicked() {
    model->appendRow({new QStandardItem(nameEdit->text()), new QStandardItem(quantityEdit->text())});
    accept();
}
```

Файл main.cpp



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

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

Видео на YouTube

https://youtu.be/ZAWIfe1v2AM