Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» («ПНИПУ»)

Творческая работа

выполнил:

студент группы РИС-23-3Б

Богомягков Василий Александрович

проверила:

доцент кафедры ИТАС О. А. Полякова

2024 г.

Введение

Цель работы – разработать интерфейс решения задачи коммивояжера, разработать АРМ специалиста.

Содержание:

1. АРМ

АРМ специалиста орнитолога. Программа, которая позволяет следить за состоянием камер, в которых находятся наблюдаемые особи птиц. Можно узнать вид и возраст особи, состояние камеры, в которой эта особь находится (состояние работоспособности вентиляционной системы, количество оставшейся еды от обычной порции, то, насколько чистая камера), в процентах.

UML

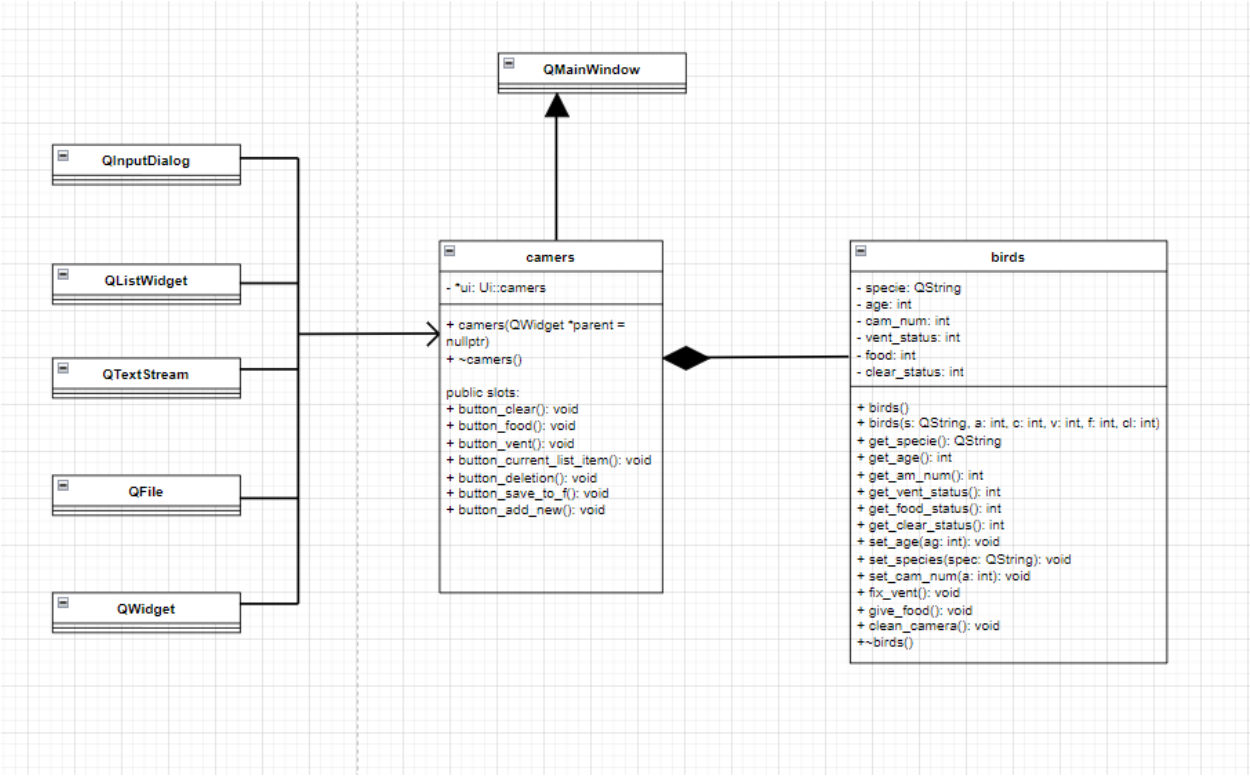

Рисунок 1 - UML-диаграмма АРМ

Анализ

Для интерфейса была использована среда Qt.

Используемые библиотеки: QFile – для открытия файлов, QMainWindow – основное окно, QListWidget – список элементов, QTextStream – потоки для текста, QInputDialog – диалог ввода данных, vector – вектор, fstream – потоковый ввод/вывод у файлов.

Вне функций хранится глобальный вектор vector<birds>, в котором будут храниться все объекты класса birds.

Класс birds:

- Поля – private

Методы:

- Конструктор
- Деструктор
- Методы, начинающиеся с “get_” – получение данных полей объекта
- Методы, начинающиеся с “set_”, fix_vent, clean_camera, give_food – установка данных полей объекта

Класс основного окна:

- При запуске программы считываются данные из файла bird_cams.txt в вектор vector<birds>, в список на экране выводятся некоторые данные об элементах (вид птицы и номер камеры для каждого элемента вектора).

Методы:

- button_clear – кнопка очистки камеры, вызывает функцию clean_camera для текущего объекта
- button_vent – кнопка починки вентиляционной системы, вызывает функцию fix_vent для текущего объекта
- button_food – кнопка добавления порции еды, вызывает функцию give_food для текущего объекта
- button_deletion – удаляет выделенный в списке объект из списка и из вектора объектов, текущий объект – объект без данных
- button_save_to_f – основной файл открывается на запись, очищается с помощью QIODevice::Truncate, в него записываются данные из всех объектов вектора
- button_add_new – добавление нового элемента с указанными параметрами, использует диалоговые окна QInputDialog
- button_current_list_item – берёт индекс выделенного объекта списка, текущий объект – объект из вектора с этим же индексом

Код

camers.h:

```
#ifndef CAMERS_H
#define CAMERS_H
#include <fstream>
#include <QFile>
#include <QMainWindow>
#include <QListWidget>
#include <QTextStream>
#include <QInputDialog>
#include <vector>
QT_BEGIN_NAMESPACE
namespace Ui { class camers; }
QT_END_NAMESPACE
```

```
class birds
{
private:
    QString specie;
    int age;
    int cam_num;
    int vent_status;
    int food;
    int clear_status;
public:
    birds()
    {
        specie="";
        cam_num=0;
        vent_status=100;
        age=0;
        food=100;
        clear_status=100;
    }
    birds(QString s, int a, int c, int v, int f, int cl)
    {
        specie=s;
        cam_num=c;
        vent_status=v;
        age=a;
        food=f;
        clear_status=cl;
    }
    QString get_specie()
    {
        return specie;
    }
    int get_age()
    {
        return age;
    }
    int get_cam_num()
    {
        return cam_num;
    }
    int get_vent_status()
    {
        return vent_status;
    }
    int get_food_status()
    {
        return food;
    }
    int get_clear_status()
    {
        return clear_status;
    }
    void set_age(int ag)
    {
        age=ag;
    }
    void set_species(QString spec)
    {
        specie=spec;
    }
    void set_cam_num(int a)
    {
        cam_num=a;
```

```
    }
    ~birds(){}
    void fix_vent()
    {
        vent_status=100;
    }
    void give_food()
    {
        food=100;
    }
    void clean_camera()
    {
        clear_status=100;
    }
};

class camers : public QMainWindow
{
    Q_OBJECT

public:
    camers(QWidget *parent = nullptr);
    ~camers();
private:
    Ui::camers *ui;
public slots:
    void button_clear();
    void button_food();
    void button_vent();
    void button_current_list_item();
    void button_deletion();
    void button_save_to_f();
    void button_add_new();
};
#endif // CAMERS_H
```

camers.cpp:

```
#include "camers.h"
#include "ui_camers.h"
std::vector<birds> v;
birds curren;
birds defaul;
camers::camers(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::camers)
{
    ui->setupUi(this);
    ui->clear_fixat->setEnabled(false);
    ui->vent_fixat->setEnabled(false);
    ui->food_fixat->setEnabled(false);
    ui->sohran_v_file->setEnabled(false);
    ui->del_el_from_l->setEnabled(false);
    //ui->age_output->setText(QString::number(defaul.get_age()));
    //ui->species_output->setText(defaul.get_specie());
    //ui->clear_output->setText(QString::number(defaul.get_clear_status()));
    //ui->food_output->setText(QString::number(defaul.get_food_status()));
    //ui->vent_output->setText(QString::number(defaul.get_vent_status()));
    ui->age_output->setText("Не выбрана особь");
    ui->species_output->setText("");
    ui->clear_output->setText("");
    ui->food_output->setText("");
```

```
    ui->vent_output->setText("");
    QFile file("bird_cams.txt");
    if (file.open(QIODevice::ReadOnly|QIODevice::Text))
    {
        QTextStream steram(&file);
        while (!steram.atEnd())
        {
            QString l1, l2, l3, l4, l5, l6;
            l1 = steram.readLine();
            l2 = steram.readLine();
            l3 = steram.readLine();
            l4 = steram.readLine();
            l5 = steram.readLine();
            l6 = steram.readLine();
            QString ltemp = l1+" ("+l3+")";
            birds temp(l1, l2.toInt(), l3.toInt(),
l4.toInt(),l5.toInt(),l6.toInt());
            v.push_back(temp);
            ui->list_cams->addItem(ltemp);
        }
        file.close();
    }
    connect(ui->choose_cur, &QPushButton::clicked, this,
&camers::button_current_list_item);
    connect(ui->clear_fixat, &QPushButton::clicked, this,
&camers::button_clear);
    connect(ui->food_fixat, &QPushButton::clicked, this,
&camers::button_food);
    connect(ui->vent_fixat, &QPushButton::clicked, this,
&camers::button_vent);
    connect(ui->del_el_from_l, &QPushButton::clicked, this,
&camers::button_deletion);
    connect(ui->add_el_to_list, &QPushButton::clicked, this,
&camers::button_add_new);
    connect(ui->sohran_v_file, &QPushButton::clicked, this,
&camers::button_save_to_f);
}
void camers::button_current_list_item()
{
    int ind = ui->list_cams->currentRow();
    curren = v.at(ind);
    ui->age_output->setText("Возраст: " + QString::number(curren.get_age()));
    ui->species_output->setText("Вид: " + curren.get_specie());
    ui->clear_output->setText(QString::number(curren.get_clear_status()));
    ui->food_output->setText(QString::number(curren.get_food_status()));
    ui->vent_output->setText(QString::number(curren.get_vent_status()));
    if (curren.get_clear_status()<40)
    {
        ui->clear_fixat->setEnabled(true);
    }
    if (curren.get_food_status()<15)
        ui->food_fixat->setEnabled(true);
    if (curren.get_vent_status()<35)
        ui->vent_fixat->setEnabled(true);
    ui->sohran_v_file->setEnabled(true);
    ui->del_el_from_l->setEnabled(true);
}
void camers::button_food()
{
    curren.give_food();
    ui->food_output->setText(QString::number(curren.get_food_status()));
    ui->food_fixat->setEnabled(false);
}
```

```
void camers::button_clear()
{
    curren.clean_camera();
    ui->clear_output->setText(QString::number(curren.get_clear_status()));
    ui->clear_fixat->setEnabled(false);
}
void camers::button_vent()
{
    curren.fix_vent();
    ui->vent_output->setText(QString::number(curren.get_vent_status()));
    ui->vent_fixat->setEnabled(false);
}
void camers::button_deletion()
{
    unsigned int ind = ui->list_cams->currentRow();
    delete ui->list_cams->takeItem(ind);
    for (unsigned int i=ind; i<v.size() - 1; i++)
    {
        v.at(i)=v.at(i+1);
    }
    v.pop_back();
    //ui->age_output->setText("Возраст: " +
QString::number(defaul.get_age()));
    //ui->species_output->setText("Вид: " + defaul.get_specie());
    //ui->clear_output->setText(QString::number(defaul.get_clear_status()));
    //ui->food_output->setText(QString::number(defaul.get_food_status()));
    //ui->vent_output->setText(QString::number(defaul.get_vent_status()));
    ui->age_output->setText("Не выбрана особь");
    ui->species_output->setText("");
    ui->clear_output->setText("");
    ui->food_output->setText("");
    ui->vent_output->setText("");
    ui->clear_fixat->setEnabled(false);
    ui->vent_fixat->setEnabled(false);
    ui->food_fixat->setEnabled(false);
    ui->sohran_v_file->setEnabled(false);
    ui->del_el_from_l->setEnabled(false);
}
void camers::button_save_to_f()
{
    QFile file("bird_cams.txt");
    if (file.open(QIODevice::ReadWrite|QIODevice::Truncate|QIODevice::Text))
    {
        QTextStream steram(&file);
        for (unsigned int i=0; i<v.size(); i++)
        {
            steram<<v[i].get_specie()<<endl;
            steram<<v[i].get_age()<<endl;
            steram<<v[i].get_cam_num()<<endl;
            steram<<v[i].get_vent_status()<<endl;
            steram<<v[i].get_food_status()<<endl;
            steram<<v[i].get_clear_status()<<endl;
        }
        file.close();
    }
}

void camers::button_add_new()
{
    int num_to_add;
    QString text_to_add;
    bool ok1, ok2, ok3;
    birds temp_bird;
```

```
    text_to_add = QInputDialog::getText(this, "Ввести", "Вид особи: ",
QLineEdit::Normal, "", &ok1);
    if (ok1 && !text_to_add.isEmpty())
    {
        temp_bird.set_species(text_to_add);
        num_to_add=QInputDialog::getInt(this, "Ввести", "Возраст особи: ", 0,
0, 60, 1, &ok2);
        if (ok2)
        {
            temp_bird.set_age(num_to_add);
            num_to_add=QInputDialog::getInt(this, "Ввести", "Номер камеры: ",
1, 1, 122345, 1, &ok3);
            if (ok3)
            {
                temp_bird.set_age(num_to_add);
                v.push_back(temp_bird);
                ui->list_cams->addItem(text_to_add+"
("+QString::number(num_to_add)+")");
            }
        }
    }
}

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

Пример работы программы

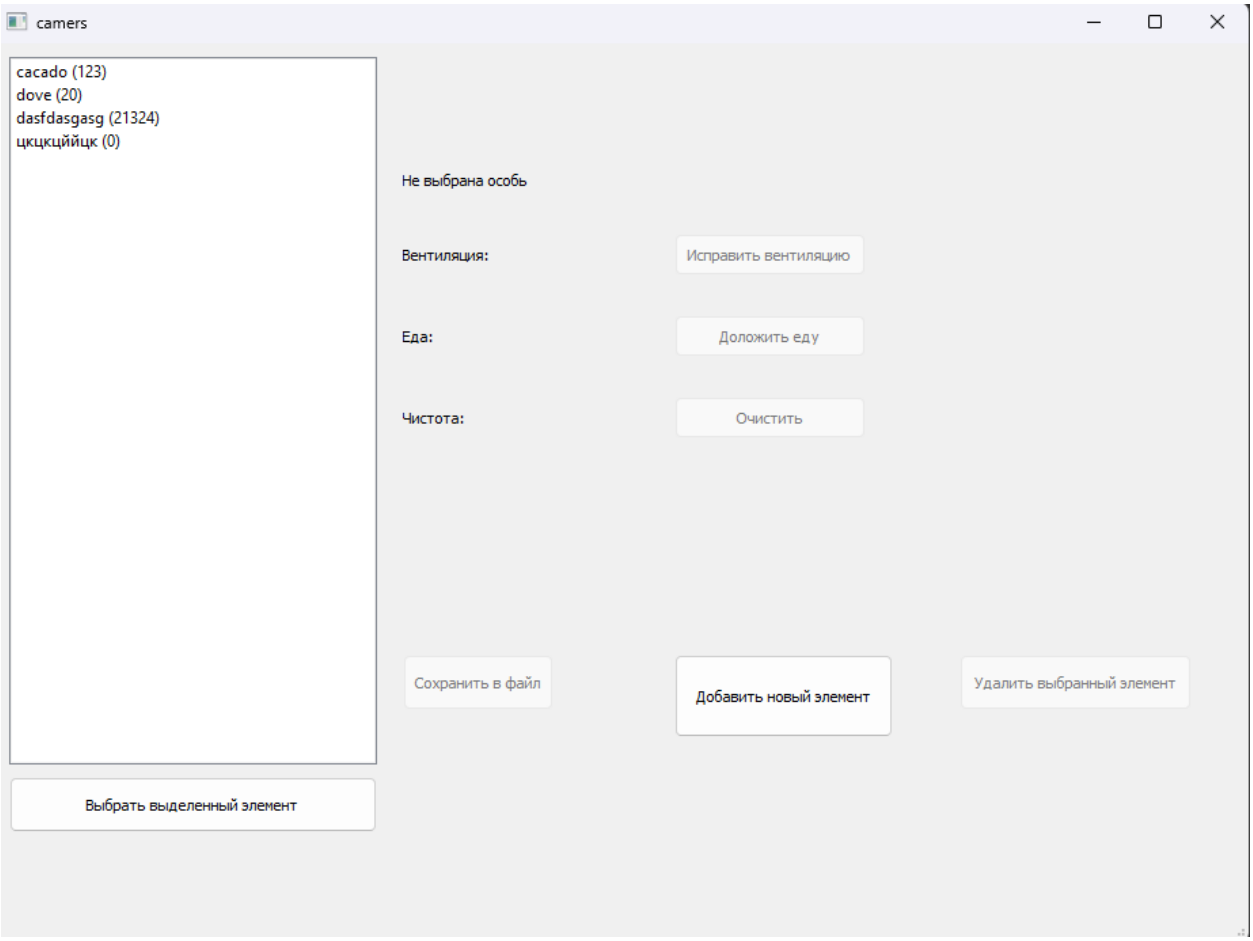

Рисунок 2 - начало работы АРМ

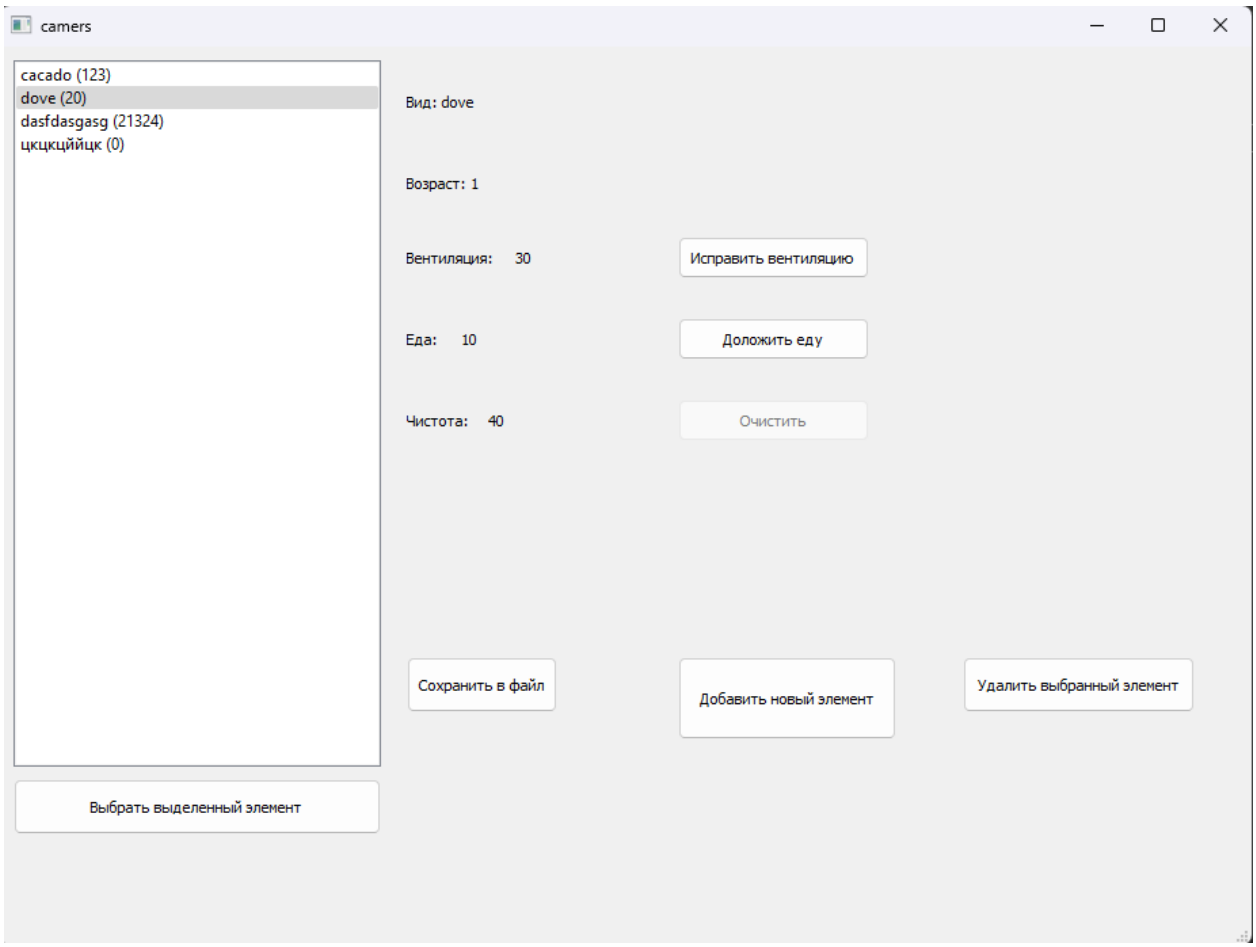

Рисунок 3 - выбран один из элементов

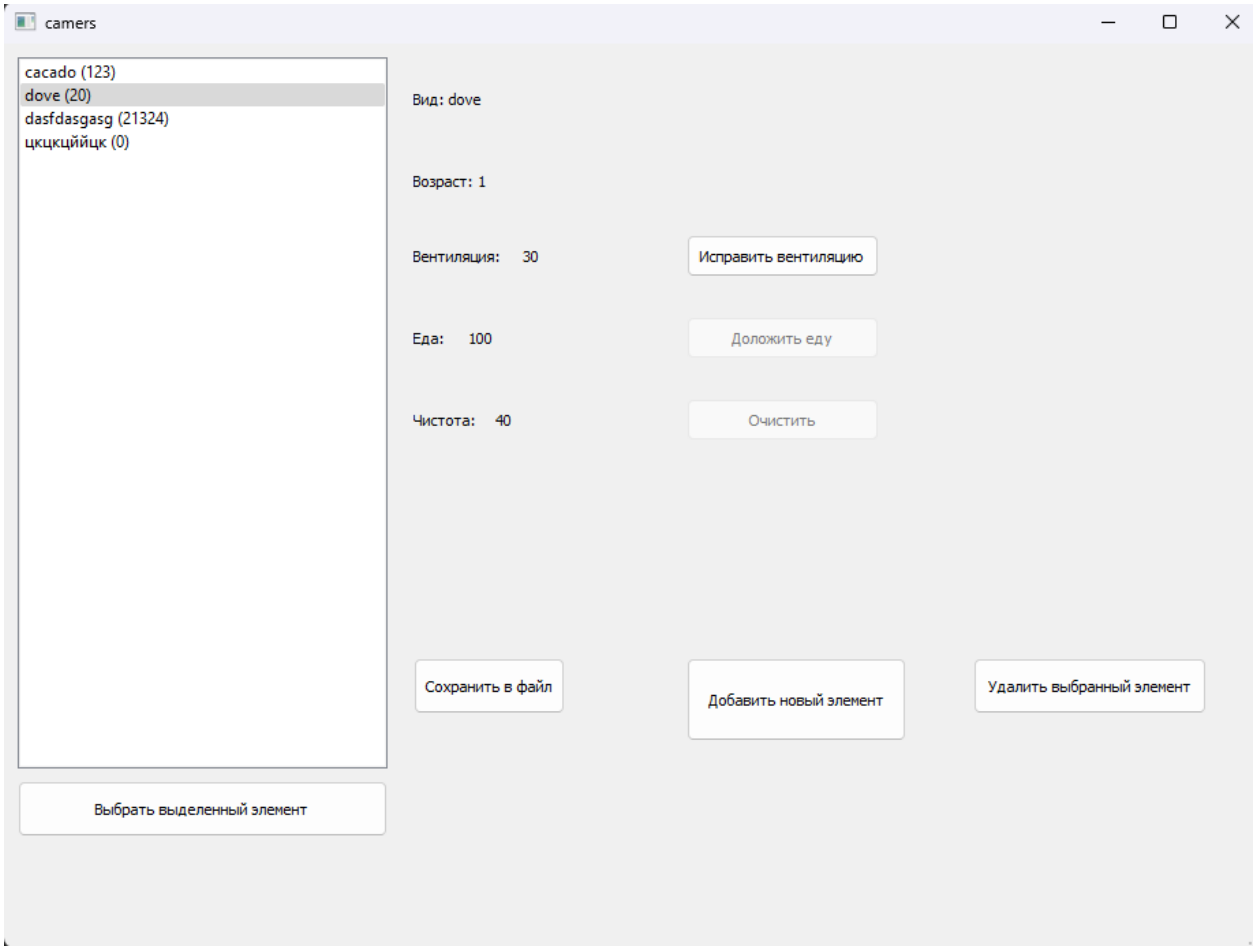

Рисунок 4 - для особи была дана порция еды

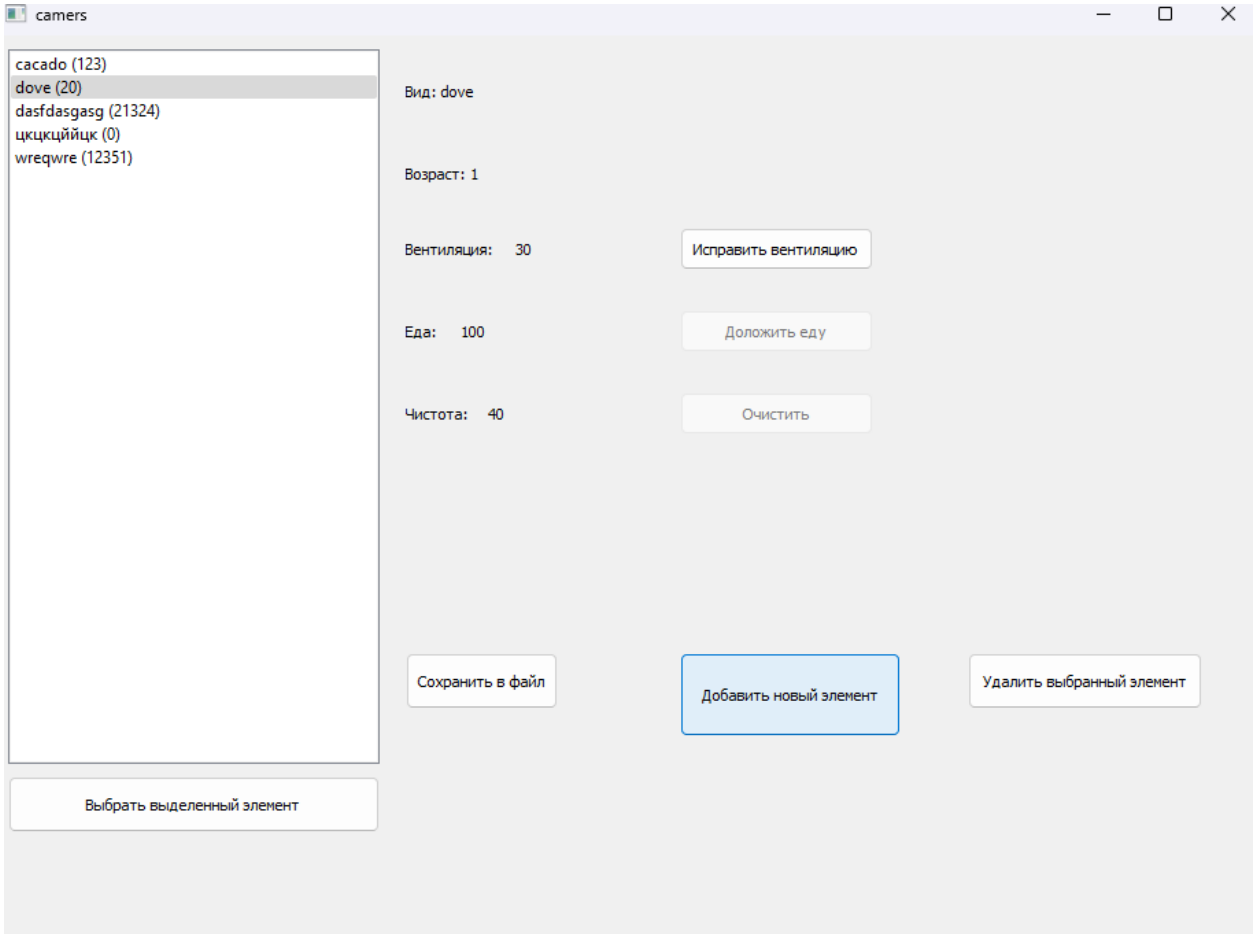

Рисунок 5 - был добавлен новый элемент

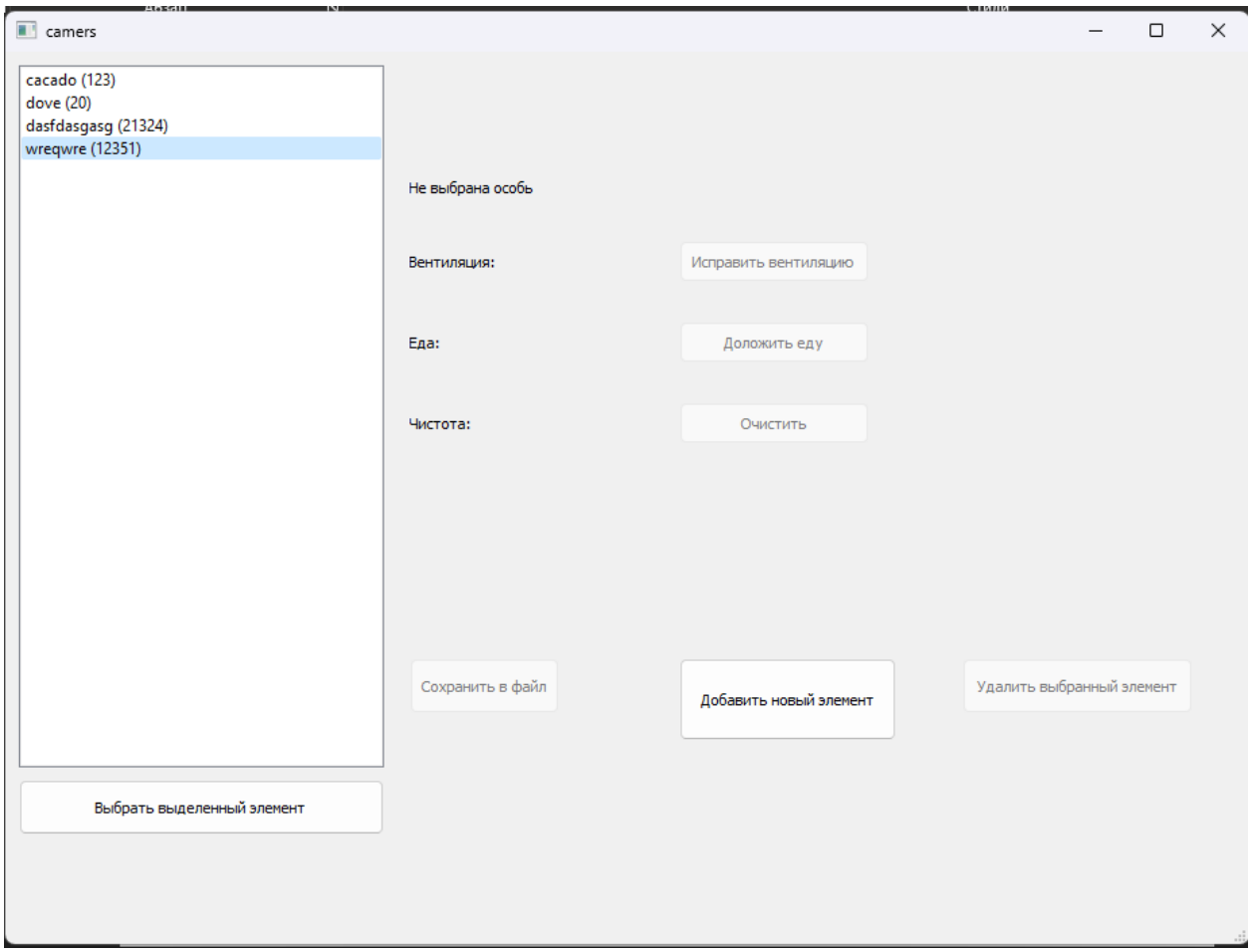

Рисунок 6 - был удалён один из элементов

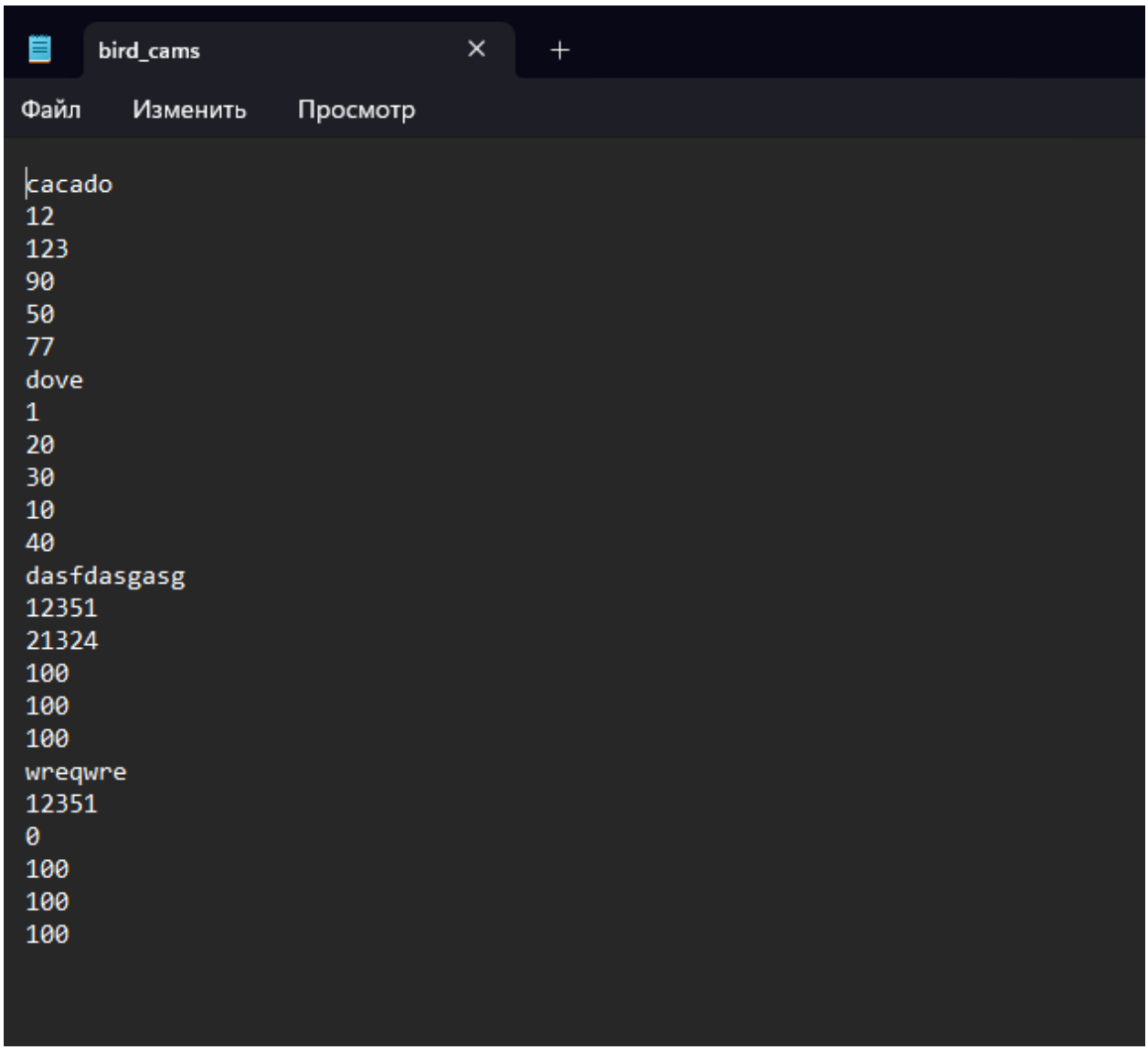

Рисунок 7 - результат сохранения данных в файл

2. Задача коммивояжера

Задача коммивояжера – задача нахождения самого оптимального маршрута обхода графа. Существует много разных способов решения этой задачи. В данном случае будет рассматриваться метод ветвей и границ.

UML

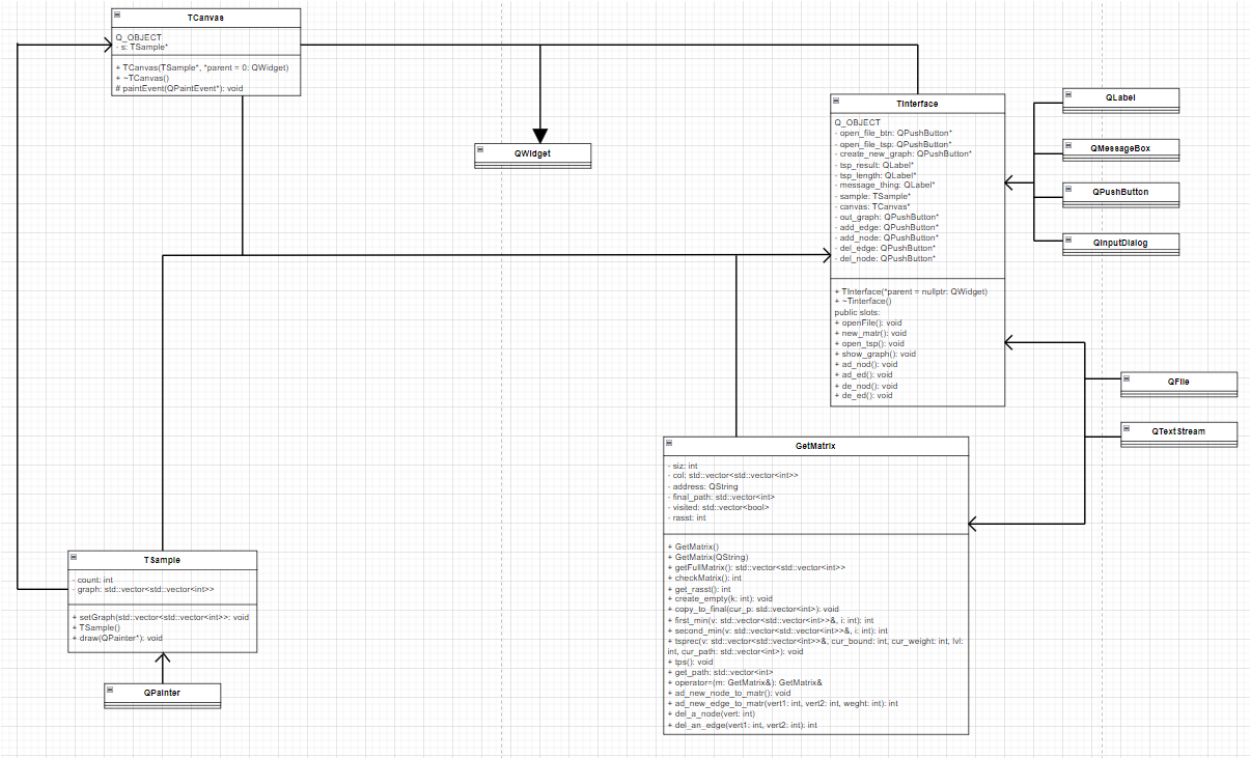

Рисунок 8 - UML-диаграмма решения задачи коммивояжера

Анализ

Для интерфейса используется среда Qt.

Используемые библиотеки: QPainter – для рисования графов, QWidget – встроенные виджеты, QLabel – текстовый заголовок, QMessageBox – окно с сообщением, QPushButton – кнопка, QInputDialog – диалоговое окно ввода данных, QFile и QTextStream – потоковое чтение из файлов.

Класс TCanvas – холст, на котором выводится визуализация графа. Функция paintEvent – событие рисования графа.

Класс TSample служит для рисования графа по матрице смежности. Функция draw рисует граф на холсте.

Класс GetMatrix – граф, в виде матрицы смежности. Кроме самого графа в объекте находятся адрес файла, откуда был взят граф, вектор final_path – финальный путь прохода графа, вектор visited – хранит данные о посещённых узлах, rasst – финальное расстояние прохода по графу.

Кроме функций инициализации графа и вычисления решения задачи коммивояжера присутствуют функции добавления и удаления узлов и рёбер, есть функция создания пустого графа.

Класс TInterface – интерфейс программы. Присутствуют кнопки выполнения функций – получение графа из файла, создание пустого графа, вычисление решения задачи коммивояжера, добавление и удаление рёбер и узлов.

Код

canvas.h:

```
#ifndef VIEWER_H
#define VIEWER_H

#include <QWidget>
#include "sample.h"

class TCanvas : public QWidget
{
    Q_OBJECT

    TSample* s;
public:
    TCanvas(TSample*,QWidget *parent = 0);
    ~TCanvas();

protected:
    void paintEvent(QPaintEvent*); //вызов события рисования
};

#endif // VIEWER_H
```

getmatrix.h:

```
#ifndef GETMATRIX_H
#define GETMATRIX_H
#include <QTextStream>
#include <QFile>
#include <vector>

class GetMatrix
{
private:
    int siz; //размер
    std::vector<std::vector<int>> col; //матрица смежности
    QString address; //адрес
    std::vector<int> final_path; //финальный путь прохода
    std::vector<bool>visited; //посещённые узлы
    int rasst; //финальное расстояние
public:
    GetMatrix();
    GetMatrix(QString);
    std::vector<std::vector<int>> getFullMatrix(); //возврат основной матрицы
    int checkMatrix(); //открытие матрицы из файла, проверка матрицы
    int get_rasst();
    void create_empty(int k); //сделать пустую матрицу
    //tsp
    void copy_to_final(std::vector<int> cur_p); //сделать временный маршрут
финальным
```

```
    int first_min(std::vector<std::vector<int>>& v, int i); //нахождение
первого минимального ребра от узла i
    int second_min(std::vector<std::vector<int>>& v, int i);//нахождение
второго минимального ребра от узла i
    void tsprec(std::vector<std::vector<int>>& v, int cur_bound, int
cur_weight, int lvl, std::vector<int> cur_path); //основной цикл задачи
коммивояжера
    void tsp(); //приготовления к решению задачи коммивояжера
    //misc
    std::vector<int> get_path();
    GetMatrix& operator=(GetMatrix& m);
    void ad_new_node_to_matr(); //добавить узел
    int ad_new_edge_to_matr(int vert1, int vert2, int weight); //добавить
ребро
    int del_a_node(int vert); //удалить узел
    int del_an_edge(int vert1, int vert2); //удалить ребро
};

#endif // GETMATRIX_H
```

interface.h:

```
#ifndef TINTERFACE_H
#define TINTERFACE_H

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QFileDialog>
#include <QInputDialog>
#include <QMessageBox>
#include "getmatrix.h"
#include "sample.h"
#include "canvas.h"

class TInterface : public QWidget
{
    Q_OBJECT

    QPushButton* open_file_btn;
    QPushButton* open_file_tsp;
    QPushButton* create_new_graph;
    QLabel* tsp_result;
    QLabel* tsp_lenght;
    QLabel* message_thing;
    TSample* sample;
    TCanvas* canvas;
    QPushButton* out_graph;
    QPushButton* add_edge;
    QPushButton* add_node;
    QPushButton* del_edge;
    QPushButton* del_node;

public:
    TInterface(QWidget *parent = nullptr);
    ~TInterface();

public slots:
    void openFile(); //открыть файл с графом
    void new_matr(); //создать пустой граф
    void open_tsp(); //вычислить задачу коммивояжера
    void show_graph(); //вывести граф
    void ad_nod(); //добавить узел
```

```
    void ad_ed(); //добавить ребро
    void de_nod(); //удалить узел
    void de_ed(); //удалить ребро
};
#endif // TINTERFACE_H
```

sample.h:

```
#ifndef SAMPLE_H
#define SAMPLE_H

#include <QPainter>

class TSample
{
    int count;
     std::vector <std::vector<int>> graph;
public:
    void setGraph(std::vector <std::vector<int>>);
    TSample();
    void draw(QPainter*);
};

#endif // SAMPLE_H
```

canvas.cpp:

```
#include "canvas.h"

TCanvas::TCanvas(TSample* f, QWidget *parent)
    : QWidget(parent)
{
    setWindowTitle("graph");
    s = f;
    setFixedSize(500,500);
}

TCanvas::~TCanvas()
{

}

void TCanvas::paintEvent(QPaintEvent*)
{
    QPainter p;
    p.begin(this);
    s->draw(&p);
    p.end();
}
```

getmatrix.cpp:

```
#include "getmatrix.h"
GetMatrix::GetMatrix()
{
    address="";
    siz=0;
    rasst=1000000;
}
GetMatrix::GetMatrix(QString t)
{
    address = t;
    siz=0;
    rasst=100000;
}
```

```
int GetMatrix::checkMatrix()
{
    col.clear();
    QFile file(address);
    if (!file.open(QIODevice::ReadOnly)) //если файл не открыть
    {
        qWarning("fail failed to open");
        return 1;//файл не открылся
    }
    else
    {
        QTextStream in(&file);
        int columnCount = 0;
        int rowCount = 0;
        int total_amount = in.readLine().toInt();
        while (!in.atEnd()) //пока не конец файл
        {
             //читать по строкам
            QString temp = in.readLine();
            temp=temp+"|";
            std::vector<int> rows;
            QString num="";
             //выделение чисел из строки, числа разделены вертикальной чертой
            for (int i=0; i<temp.size(); i++)
            {
                if (temp.at(i)!="|")
                {
                    num+=temp.at(i);
                }
                else
                {
                    rows.push_back(num.toInt());
                    num="";
                }
            }
            col.push_back(rows);
            final_path.push_back(-1);
            rowCount++;
        }
        if (rowCount!=0)
            columnCount=total_amount/rowCount;
        if (rowCount!=columnCount)
        {
            qWarning("not square");
            col.clear();
            return 3;//матрица не симметрична
        }
        siz = rowCount;
    }
    for (int i=0; i<siz-1; i++)
    {
        for (int j=i+1; j<siz; j++)
        {
            if (col[i][j]!=0 && col[j][i]!=0)
            {
                if (col[i][j]!=col[j][i])
                {
                    qWarning("different paths");
                    col.clear();
                    return 4; //расстояния из A в B и из B в A разные
                }
            }
```

```
        }
    }
    final_path.push_back(-1);
    return 0;
}

std::vector<std::vector<int>> GetMatrix::getFullMatrix()
{
    return col;
}


int GetMatrix::first_min(std::vector<std::vector<int>> &v, int i)
{
    int mi = 1000000;
    for (int k=0; k<siz; k++)
    {
        if (v[i][k]<mi && i!=k)
            mi=v[i][k];
    }
    return mi;
}
int GetMatrix::second_min(std::vector<std::vector<int>> &v, int i)
{
    int f=100000, s=100000;
    for (int j=0; j<siz; j++)
    {
        if (i!=j)
        {
            if (v[i][j]<=f)
            {
                s=f;
                f=v[i][j];
            }
            else
            {
                if (v[i][j]<=s && v[i][j]!=f)
                    s=v[i][j];
            }
        }
    }
    return s;
}
//аргументы:
//cur_bound - нижняя граница исходного узла
//cur_weight - хранит текущий вес пути
//cur_path - текущий путь
//lvl - уровень итерации
void GetMatrix::tsprec(std::vector<std::vector<int>> &v, int cur_bound, int
cur_weight, int lvl, std::vector<int> cur_path)
{
    //если достигнут последний уровень итерации - конец матрицы
    if (lvl==siz)
    {
        //если есть ребро из последнего узла в пути назад к первому узлу
        if (v[cur_path[lvl-1]][cur_path[0]]!=0)
        {
            //вычисляется суммарный вес
            int cur_res=cur_weight+v[cur_path[lvl-1]][cur_path[0]];
            //если результат лучше, то его надо обновить
            if (cur_res<rasst)
            {
                copy_to_final(cur_path);
```

```
                rasst=cur_res;
            }
        }
        return;
    }
    //для всех остальных уровней итерации
    for (int i=0; i<siz; i++)
    {
        //если следующий узел не он же сам
        if (v[cur_path[lvl-1]][i]!=0 && visited[i]==false)
        {
            int tmp = cur_bound;
            cur_weight+=v[cur_path[lvl-1]][i];
            //разные вычисления границы для разных уровней итерации
            if (lvl==1)
                cur_bound-=((first_min(v, cur_path[lvl-
1])+first_min(v,i))/2);
            else
                cur_bound-=((second_min(v, cur_path[lvl-
1])+first_min(v,i))/2);
            //cur_bound+cur_weight - действительная нижняя граница для узла
            //на котором алгоритм сейчас находится
            //если она меньше конечного пути, надо исследовать узел дальше
            if (cur_bound+cur_weight<rasst)
            {
                cur_path[lvl]=i;
                visited[i]=true;//пометка, что узел посещён
                //рекурсивный вызов следующего уровня
                tsprec(v, cur_bound, cur_weight, lvl+1, cur_path);
            }
            //иначе надо отменить изменения к cur_weight и cur_bound
            cur_weight-=v[cur_path[lvl-1]][i];
            cur_bound=tmp;
            //в том числе отменить изменения к вектору посещённых узлов
            for (unsigned int k=0;k<visited.size();k++)
            {
                visited[k]=false;
            }
            for (int j=0;j<=lvl-1;j++)
            {
                visited[cur_path[j]]=true;
            }
        }
    }
}
void GetMatrix::tsp()
{
    //текущий путь
    std::vector<int> cur_path;
    int cur_bound = 0;
    //начальная инициализация используемых векторов
    for (int i=0; i<=siz;i++)
    {
        final_path[i]=-1;
        cur_path.push_back(-1);
    }
    for (int i=0;i<siz;i++)
    {
        visited.push_back(false);
    }
    //вычисление исходной нижней границы для начального узла
    for (int i=0;i<siz;i++)
    {
```

```
            cur_bound+=(first_min(col,i)+second_min(col,i));
        }
        //округление до целого числа
        cur_bound=(cur_bound&1)?cur_bound/2+1:cur_bound/2;
        //начало - самый первый узел
        visited[0]=true;
        cur_path[0]=0;
        tsprec(col, cur_bound, 0, 1, cur_path);
}
//копирование пути в final_path
void GetMatrix::copy_to_final(std::vector<int> cur_p)
{
        for (unsigned int i=0; i<cur_p.size(); i++)
        {
            final_path[i]=cur_p[i];
        }
        final_path[siz]=cur_p[0];
}
int GetMatrix::get_rasst()
{
        return rasst;
}
std::vector<int> GetMatrix::get_path()
{
        return final_path;
}

GetMatrix& GetMatrix::operator=(GetMatrix &m)
{
        this->siz=m.siz;
        this->col=m.col;
        this->address=m.address;
        this->final_path=m.final_path;
        this->visited=m.visited;
        this->rasst=m.rasst;
        return *this;
}


void GetMatrix::ad_new_node_to_matr() //добавить новый узел
{
        std::vector<int> temp_v;
        for (int i=0; i<siz; i++)
        {
            col[i].push_back(0);
            temp_v.push_back(0);
        }
        temp_v.push_back(0);
        col.push_back(temp_v);
        siz++;
}

int GetMatrix::ad_new_edge_to_matr(int vert1, int vert2, int weight)
//добавить новое ребро
{
        if (vert1==vert2)
        {
            return 2; //если они равны
        }
        if (col[vert1-1][vert2-1]!=0)
        {
            return 3; //если уже есть узел на этом месте
        }
```

```
    if (weight==0)
    {
        return 4; //если новое ребро равно нулю
    }
    col[vert1-1][vert2-1]=weight;
    return 1;
}

int GetMatrix::del_a_node(int vert) //удаление узла
{
    int ind = vert-1;
    for (int i=ind; i<siz-1; i++)
    {
        for (int j=0; j<siz; j++)
        {
            col[i][j]=col[i+1][j];
        }
    }
    for (int j=ind; j<siz-1; j++)
    {
        for (int i=0; i<siz-1; i++)
        {
            col[i][j]=col[i][j+1];
        }
    }
    col.pop_back();
    for (int i=0; i<siz-1; i++)
    {
        col[i].pop_back();
    }
    siz-=1;
    return 1;
}

int GetMatrix::del_an_edge(int vert1, int vert2) //удаление ребра
{
    if (col[vert1-1][vert2-1]==0)
    {
        return 2; //нет ребра
    }
    col[vert1-1][vert2-1]=0;
    return 1;
}

void GetMatrix::create_empty(int k)
{
    //создать пустую матрицу смежности размером k
    col.clear();
    siz=k;
    final_path.clear();
    rasst=0;
    address="";
    std::vector<int> tmp;
    for (int i=0; i<k; i++)
    {
        tmp.clear();
        for (int j=0; j<k; j++)
        {
            tmp.push_back(0);
        }
        col.push_back(tmp);
        final_path.push_back(-1);
    }
```

```
    final_path.push_back(-1);
}
```

interface.cpp:

```
#include "interface.h"

GetMatrix curren_matr;

TInterface::TInterface(QWidget *parent)
    : QWidget(parent)
{
    setWindowTitle("TSP");
    setFixedSize(500,400);

    open_file_btn = new QPushButton("открыть граф\nиз файла", this);
    open_file_btn->setGeometry(100,145,100,35);

    create_new_graph = new QPushButton("создать пустой\nграф", this);
    create_new_graph->setGeometry(250, 145, 100, 35);

    open_file_tsp=new QPushButton("коммивояжёр", this);
    open_file_tsp->setGeometry(100,200, 100, 30);
    open_file_tsp->setEnabled(false);

    out_graph=new QPushButton("вывести граф", this);
    out_graph->setGeometry(100, 250, 100, 30);
    out_graph->setEnabled(false);

    add_edge=new QPushButton("добавить ребро", this);
    add_edge->setGeometry(100, 300, 100, 30);
    add_edge->setEnabled(false);

    add_node=new QPushButton("добавить узел", this);
    add_node->setGeometry(250, 200, 100, 30);
    add_node->setEnabled(false);

    del_edge=new QPushButton("удалить ребро", this);
    del_edge->setGeometry(250, 250, 100, 30);
    del_edge->setEnabled(false);

    del_node=new QPushButton("удалить узел", this);
    del_node->setGeometry(250, 300, 100, 30);
    del_node->setEnabled(false);

    sample = new TSample();
    canvas = new TCanvas(sample);
    tsp_lenght=new QLabel(this);
    tsp_lenght->setGeometry(10,100,100,30);
    tsp_result=new QLabel(this);
    tsp_result->setGeometry(10, 50, 200, 60);
    message_thing=new QLabel(this);
    message_thing->setGeometry(10,50, 300, 100);
    canvas->setAttribute( Qt::WA_QuitOnClose, false );

    connect(open_file_btn, SIGNAL(pressed()),this,SLOT(openFile()));
    connect(open_file_tsp, SIGNAL(pressed()), this, SLOT(open_tsp()));
    connect(out_graph, SIGNAL(pressed()),this,SLOT(show_graph()));
    connect(add_node, SIGNAL(pressed()), this, SLOT(ad_nod()));
    connect(add_edge, SIGNAL(pressed()), this, SLOT(ad_ed()));
    connect(del_edge, SIGNAL(pressed()), this, SLOT(de_ed()));
    connect(del_node, SIGNAL(pressed()), this, SLOT(de_nod()));
    connect(create_new_graph, SIGNAL(pressed()), this, SLOT(new_matr()));
}
```

```
TInterface::~TInterface()
{
    delete open_file_btn;
    delete sample;
    delete canvas;
    delete open_file_tsp;
    delete tsp_lenght;
    delete tsp_result;
}

void TInterface::openFile()
{
    QString fname = QFileDialog::getOpenFileName(this, "Файл", "C://");
    QFile file(fname);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
            return;

    GetMatrix matrix(fname);
    int getResult = matrix.checkMatrix();

    if(getResult == 0) //если файл с матрицей корректный
    {
        curren_matr=matrix;
        open_file_tsp->setEnabled(true);
        out_graph->setEnabled(true);
        add_edge->setEnabled(true);
        add_node->setEnabled(true);
        del_edge->setEnabled(true);
        del_node->setEnabled(true);
    }
    else
    {
        QMessageBox messageBox;
        messageBox.critical(0,"Ошибка","Неверный формат матрицы");
        messageBox.setFixedSize(500,200);
    }
}

void TInterface::show_graph()
{
    sample->setGraph(curren_matr.getFullMatrix());
    canvas->update();
    canvas->show();
}

void TInterface::open_tsp()
{
    curren_matr.tsp();
    std::vector<int> tmp_path = curren_matr.get_path();
    int tmp_rasst = curren_matr.get_rasst();
    QString res;
    res=res+"Проход по задаче коммивояжёра:\n";
    res+=QString::number(tmp_path[0]+1);
    for (int i=1; i<tmp_path.size();i++)
    {
        res=res+"->"+QString::number(tmp_path[i]+1);
    }
    res=res+"\nВычисленные затраты: "+QString::number(tmp_rasst);
    message_thing->setText(res);
}

void TInterface::ad_nod()
```

```
{
    curren_matr.ad_new_node_to_matr();
    message_thing->setText("Добавлен новый узел");
}

void TInterface::ad_ed()
{
    bool ok1, ok2, ok3;
    int vert1 = QInputDialog::getInt(this, "Окно ввода", "Узел 1:", 1, 1,
curren_matr.getFullMatrix().size(), 1, &ok1);
    if (ok1)
    {
        int vert2 = QInputDialog::getInt(this, "Окно ввода", "Узел 2:", 1, 1,
curren_matr.getFullMatrix().size(), 1, &ok2);
        if (ok2)
        {
            int weight = QInputDialog::getInt(this, "Окно ввода", "Вес
ребра:", 0, 0, 10000, 1, &ok3);
            if (ok3)
            {
                int msg = curren_matr.ad_new_edge_to_matr(vert1, vert2,
weight);
                switch (msg)
                {
                case 1:
                {
                    message_thing->setText("Новое ребро успешно добавлено");
                    break;
                }
                case 2:
                {
                    message_thing->setText("Рёбра равны");
                    break;
                }
                case 3:
                {
                    message_thing->setText("На указанном месте узел уже
существует");
                    break;
                }
                case 4:
                {
                    message_thing->setText("Указан пустой вес ребра");
                    break;
                }
                }
            }
        }
    }
}
void TInterface::de_ed()
{
    bool ok1, ok2;
    int vert1 = QInputDialog::getInt(this, "Окно ввода", "Узел 1:", 1, 1,
curren_matr.getFullMatrix().size(), 1, &ok1);
    if (ok1)
    {
        int vert2 = QInputDialog::getInt(this, "Окно ввода", "Узел 2:", 1, 1,
curren_matr.getFullMatrix().size(), 1, &ok2);
        if (ok2)
        {
            int msg = curren_matr.del_an_edge(vert1, vert2);
            switch (msg)
```

```
            {
            case 1:
            {
                message_thing->setText("Указанное ребро успешно удалено");
                break;
            }
            case 2:
            {
                message_thing->setText("Указанного ребра уже не существует");
                break;
            }
            }
        }
    }
}

void TInterface::de_nod()
{
    bool ok;
    int vert = QInputDialog::getInt(this, "Окно ввода", "Узел:", 1, 1,
curren_matr.getFullMatrix().size(), 1, &ok);
    if (ok)
    {
        curren_matr.del_a_node(vert);
        message_thing->setText("Указанный узел успешно удалён");
    }
}

void TInterface::new_matr()
{
    bool ok;
    int new_siz = QInputDialog::getInt(this, "Окно", "Количество узлов:", 2,
2, 13, 1, &ok);
    if (ok)
    {
        curren_matr.create_empty(new_siz);
        message_thing->setText("Создан новый пустой граф");
        open_file_tsp->setEnabled(true);
        out_graph->setEnabled(true);
        add_edge->setEnabled(true);
        add_node->setEnabled(true);
        del_edge->setEnabled(true);
        del_node->setEnabled(true);
    }
}
```

sample.cpp:

```
#include "sample.h"
#include <math.h>
TSample::TSample()
{
}

void TSample::setGraph(std::vector<std::vector<int>> s)
{
    graph = s;
}

void TSample::draw(QPainter* p)
{
    count = graph.size();
```

```
    QRect r(0,0,500,500);
    qreal cw = 0.5*r.width();
    qreal ch = 0.5*r.height();
    qreal cr = 0.9*(cw>ch?ch:cw);
    qreal a = 2.0*acos(-1.0)/count;
    QPointF *t = new QPointF[count];
    p->setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::FlatCap));
    p->setBrush(QBrush(Qt::white, Qt::SolidPattern));
    QFont font;
    font.setPointSize(15);
    font.setBold(true);
    p->setFont(font);
    std::vector<QPointF> tem;

    for (int i = 0; i<count; ++i)
    {
        QString str;
        t[i] = QPointF(cw+cr*sin(i*a),ch-cr*cos(i*a));
        p->drawEllipse(QPointF(cw+cr*sin(i*a),ch-cr*cos(i*a)),25,25);
        p->drawText(QPointF(cw+cr*sin(i*a)-6,ch-
cr*cos(i*a)+6),str.setNum(i+1));
        tem.push_back(QPointF(cw+cr*sin(i*a)-6,ch-cr*cos(i*a)+6));
    }
    for (int i=0; i<count-1; i++)
    {
        for (int j=i+1; j<count; j++)
        {
            int y_offset=0;
            int x_offset=0;
            if (abs(tem[i].x()-tem[j].x())<10)//(tem[i].x()==tem[j].x())
            {
                y_offset=40;
            }
            if (abs(tem[i].y()-tem[j].y())<10)
            {
                x_offset=40;
            }
            if ((graph[i][j]!=0 && graph[j][i]!=0) || (graph[i][j]!=0 &&
graph[j][i]==0))
            {
                p-
>drawText(abs(tem[i].x()+tem[j].x())/2+x_offset,abs(tem[i].y()+tem[j].y())/2
- y_offset,QString::number(graph[i][j]));
            }
            if (graph[i][j]==0 && graph[j][i]!=0)
            {
                p-
>drawText(abs(tem[i].x()+tem[j].x())/2+x_offset,abs(tem[i].y()+tem[j].y())/2
- y_offset,QString::number(graph[j][i]));
            }
        }
    }

    for(int i = 0; i < graph.size(); ++i)
    {
        for(int j = 0; j < graph.size(); ++j)
        {
            if(graph[i][j] > 0)
            {
                if(i == j)
                {
                    p->drawText(QPointF(cw+cr*sin(i*a)+4,ch-
cr*cos(i*a)),"*");
```

```
}
QPoint p1, p2;
p1.setX(int(t[i].x()));
p1.setY(int(t[i].y()));
p2.setX(int(t[j].x()));
p2.setY(int(t[j].y()));

if(p1.x() < p2.x())
{
    if(p1.y() == p2.y())
    {
        p1.setX(t[i].x()+25);
        p2.setX(t[j].x()-25);
    }
    else
    {
        p1.setX(t[i].x()+18);
        p2.setX(t[j].x()-18);
    }
}
if(p1.x() > p2.x())
{
    if(p1.y() == p2.y())
    {
        p1.setX(t[i].x()-25);
        p2.setX(t[j].x()+25);
    }
    else
    {
        p1.setX(t[i].x()-18);
        p2.setX(t[j].x()+18);
    }

}
if(p1.y() < p2.y())
{
    if(p1.x() == p2.x())
    {
        p1.setY(t[i].y()+25);
        p2.setY(t[j].y()-25);
    }
    else
    {
        p1.setY(t[i].y()+18);
        p2.setY(t[j].y()-18);
    }

}
if(p1.y() > p2.y())
{
    if(p1.x() == p2.x())
    {
        p1.setY(t[i].y()-25);
        p2.setY(t[j].y()+25);
    }
    else
    {
        p1.setY(t[i].y()-18);
        p2.setY(t[j].y()+18);
    }
}
QLineF line(p2,p1);
```

```
                qreal arrowSize = 10;

                double angle = std::atan2(-line.dy(), line.dx());
                QPointF arrowP1 = line.p1() + QPointF(sin(angle + M_PI / 3) *
arrowSize,
                                                      cos(angle + M_PI / 3) *
arrowSize);
                QPointF arrowP2 = line.p1() + QPointF(sin(angle + M_PI - M_PI
/ 3) * arrowSize,
                                                      cos(angle + M_PI - M_PI
/ 3) * arrowSize);
                QPolygonF arrowHead;
                arrowHead.clear();
                arrowHead << line.p1() << arrowP1 << arrowP2;


                p->drawLine(p1,p2);
                p->setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::FlatCap));
                p->setBrush(QBrush(Qt::black, Qt::SolidPattern));
                if(i!=j)
                {
                p->drawPolygon(arrowHead);
                }
            }
        }
    }

    delete [] t;

}
```

Пример работы программы

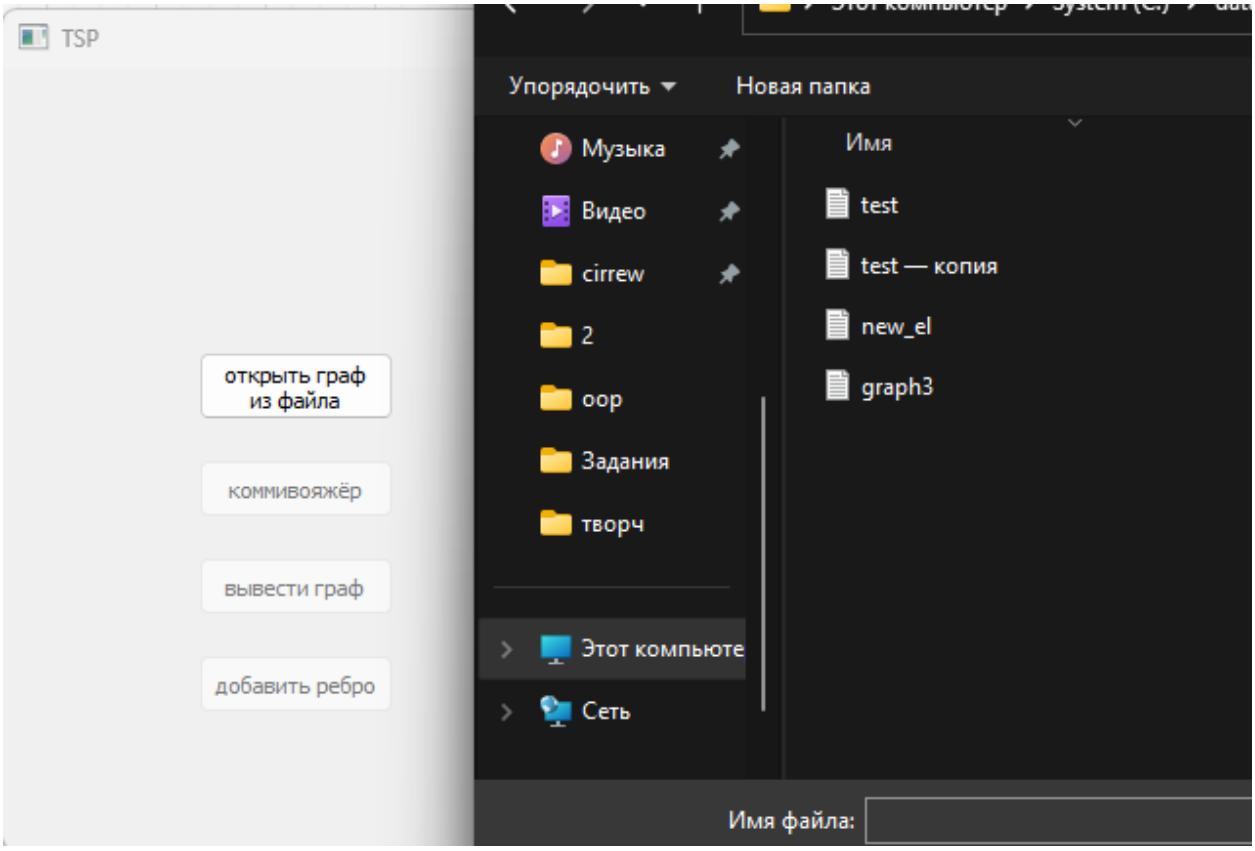

Рисунок 9 - диалог открытия файла с графом

Рисунок 10 - вычисленная задача коммивояжера для открытого графа

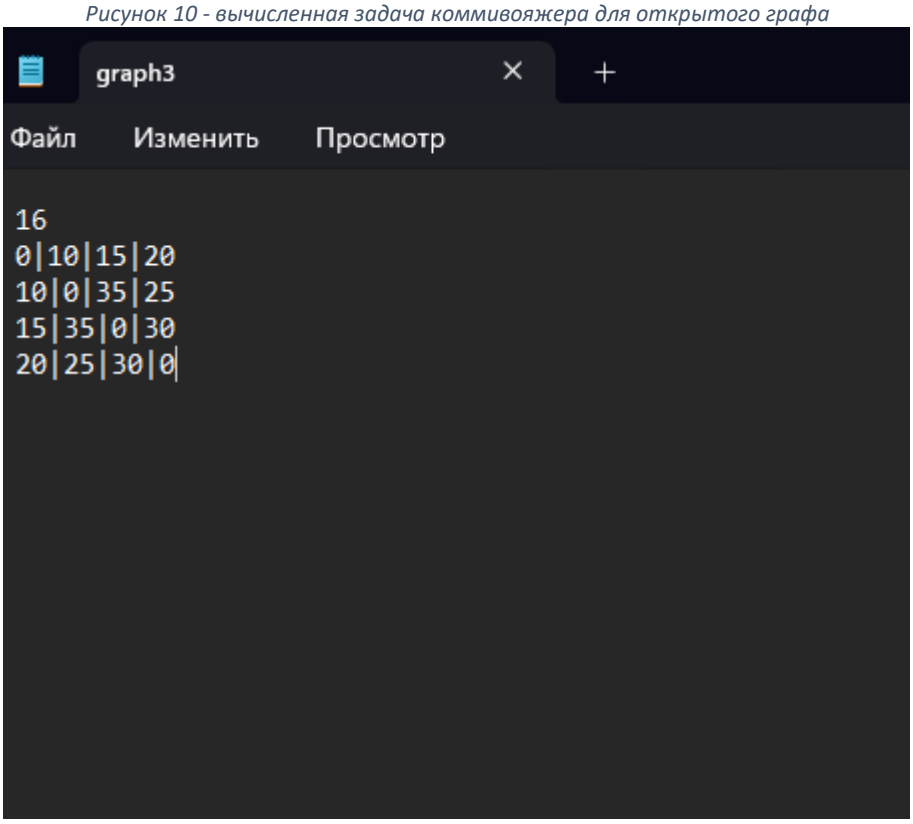

Рисунок 11 - формат матрицы в текстовом файле

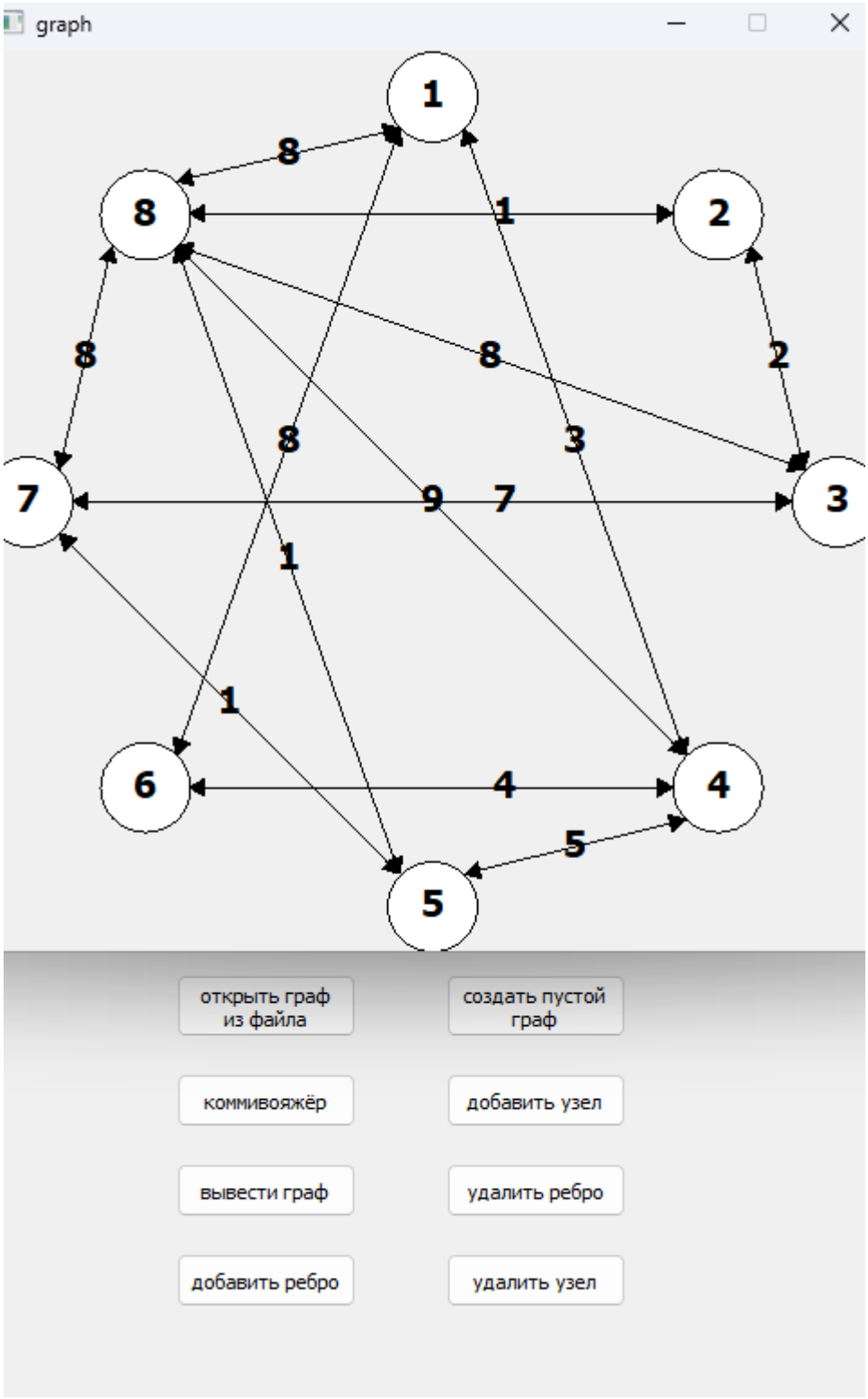

Рисунок 12 - другой граф из другого файла, визуализация

Проход по задаче коммивояжёра:
1->6->4->5->7->3->2->8->1
Вычисленные затраты: 36

Рисунок 13 - решение задачи коммивояжера для второго графа

Заключение

Цель работы была достигнута. Необходимый код был написан. Код не является идеальным. В АРМ интерфейс относительно простой.

Алгоритм решения задачи коммивояжера тоже не идеальный. Начальный узел – всегда узел с индексом 0, нельзя выбрать узел, из которого начинается путь. Алгоритм не работает, если нет пути обратно в начальный узел, в том числе, если есть узлы только с одним ребром.

Используемые программы:

- OBS – запись видео с объяснением
- Qt Creator – написание кода
- drawio – создание UML-диаграммы

Используемые источники

1. QInputDialog: https://doc.qt.io/qt-6/qinputdialog.html
2. QPainter: https://doc.qt.io/qt-6/qpainter.html
3. QListWidget: https://doc.qt.io/qt-6/qlistwidget.html
4. QFile: https://doc.qt.io/qt-6/qfile.html
5. Одна из вариаций решения задачи коммивояжера методом ветвей и границ: https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/amp/
6. Один из вариантов вывода графа + вариант открытия графа из файла: https://github.com/3xwr/DrawGraph-Qt?tab=readme-ov-file