

Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Томский государственный университет систем управления и
радиоэлектроники» (ТУСУР)
Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

АЛГОРИТМ БЛИЖАЙШЕГО СОСЕДА В ЗАДАЧЕ КОММИВОЯЖЁРА

Курсовая работа по дисциплине «Основы программирования»

Пояснительная записка

Студент гр. 728-2

_____ Геворгян Д.Р.

Руководитель

Доцент кафедры БИС

_____ Харченко С.С.

_____ оценка

Реферат

Курсовая работа содержит 50 страниц пояснительной записки, 34 рисунка, 6 приложений, 7 источников.

Программа «Алгоритм ближайшего соседа в задаче Коммивояжёра».

Цель работы – разработка программы «Алгоритм ближайшего соседа в задаче Коммивояжёра», которая предназначена для нахождения кратчайшего пути между точками.

В результате работы была создана программа, которая корректно создаёт точки, а также соединяет их графически, находя кратчайший путь между ними.

Разработка программы проводилась на языке программирования C#.

Курсовая работа выполнена в текстовом редакторе Microsoft Word 2019.

Пояснительная записка была оформлена согласно ОС ТУСУР 02-2013.

The abstract

The course work contains 50 pages of explanatory note, 34 figures, 6 annexes, 7 sources.

Nearest neighbor algorithm in travelling salesman problem.

The purpose of the work is to develop the program “Nearest neighbor algorithm in travelling salesman problem”, which is designed to find the shortest path between points.

As a result of the work, a program was created that correctly creates points, and also connects them graphically, finding the shortest way between them.

The development of the program was carried out in the C# programming language.

The course work is done in the Microsoft word 2019 text editor.

The explanatory note was issued according to TUSUR 02-2013.

Оглавление

1. Введение	5
2. Обзор темы	6
2.1 История задачи коммивояжёра.....	6
2.2 Краткий обзор метода ближайшего соседа.....	7
2.3 Другие методы решения задачи Коммивояжёра	8
3. Проектирование.....	10
3.1 Обоснование выбранных технологий	10
3.2 Описание алгоритма.....	10
3.3 Сложность алгоритма.....	11
3.4 UML диаграммы.....	17
4. Разработка.....	19
5. Тестирование	29
6. Заключение	37
7. Список использованной литературы.....	39
Приложение А	40
Приложение Б.....	41
Приложение В	43
Приложение Г	45
Приложение Д	48
Приложение Е.....	52

1. Введение

Цель курсовой работы:

- разработать программу «Алгоритм ближайшего соседа в задаче Коммивояжёра», которая предназначена для нахождения кратчайшего пути между точками;
- приобрести навыки и методы программирования поставленных задач;
- подготовиться к выполнению дипломного проекта.

2. Обзор темы

2.1 История задачи коммивояжёра

Данная работа посвящена теме «Задача о коммивояжере». Задача коммивояжера заключается в определении такой последовательности объезда городов, которая обеспечит минимальное время переезда, или минимальную стоимость проезда, или минимальное расстояние переезда.

Наиболее ярким и характерным примером применения задачи "О коммивояжере" стала оптимизация операций на конвейере: в 1984 году, после того как был проведен анализ последовательности и временных затрат на операции на конвейерах заводов компании "General Motors" и приняты рекомендованные меры, удалось увеличить общую производительность почти на 13% при неизменном числе рабочих и том же оборудовании. Расчеты производились на компьютерах IBM 360gr, которые в то время являлись одними из самых производительных систем в мире. Просчет и оптимизация 200 основных и 87 вспомогательных операций занял около 230 часов. Считается, что это было первое коммерческое применение компьютерных технологий в области управления производством в целом.

Сейчас решение данной задачи необходимо во многих областях связанных с замкнутыми и при этом жестко связанными по времени системами, такими как: конвейерное производство, многооперационные обрабатывающие комплексы, судовые и железнодорожные погрузочные системы, перевозки грузов по замкнутому маршруту, расчет авиационных линий.

Поэтому данная проблема на современном этапе развития общества имеет не самое последнее по значимости место.

В коммерческой деятельности коммерсанты, торговые агенты постоянно проводят работу по поиску партнеров или клиентов для заключения договоров на поставку и покупку товаров. Для решения этих задач коммерсантам необходимо выезжать в командировки, выполнять вояж по целой сети городов как по нашей стране, так и за рубежом. Поскольку продолжительность командировки и транспортные расходы следует сокращать, то необходимо перед поездкой составить кратчайший маршрут, предусматривающий посещение каждого пункта только один раз, и вернуться обратно.

2.2 Краткий обзор метода ближайшего соседа

Алгоритм ближайшего соседа — один из простейших эвристических алгоритмов решения задачи коммивояжёра. Относится к категории «жадных» алгоритмов.

Формулируется следующим образом: «Пункты обхода плана последовательно включаются в маршрут, причем каждый очередной включаемый пункт должен быть ближайшим к последнему выбранному пункту среди всех остальных, ещё не включенных в состав маршрута».

Шаги алгоритма:

1. Инициализация всех вершин как не посещенные;
2. Выбор произвольной вершины, установка ее в качестве текущей вершины u , а также отметка её посещённой;
3. Поиск самого короткого ребра, соединяющего текущую вершину u и любую из не посещенных вершин, установка второй вершины как v ;
4. Установка v в качестве текущей вершины u . Отметка вершины v посещенной;
5. Если все вершины посещены, то конец. Иначе пункт 3.

Алгоритм прост в реализации, быстро выполняется, но, как и другие «жадные» алгоритмы, может выдавать неоптимальные решения. Одним из эвристических критериев оценки решения является правило: если путь, пройденный на последних шагах алгоритма, сравним с путём, пройденным на начальных шагах, то можно условно считать найденный маршрут приемлемым, иначе, вероятно, существуют более оптимальные решения. Другой вариант оценки решения заключается в использовании алгоритма нижней граничной оценки (lower bound algorithm).

Для любого количества городов, большего трех, в задаче коммивояжёра можно подобрать такое расположение городов (значение расстояний между вершинами графа и указание начальной вершины), что алгоритм ближайшего соседа будет выдавать наихудшее решение.

2.3 Другие методы решения задачи Коммивояжёра

Алгоритм имитации отжига – приближенный метод решения задачи поиска минимума функции. За основу взят физический процесс кристаллизации вещества, который применяют в металлургии для повышения однородности металла. Отжиг – это процесс остывания вещества, при котором молекулы на фоне замедляющегося теплового движения собираются в наиболее энергетически выгодные конфигурации. Как известно, у металла есть кристаллическая решетка, она описывает геометрическое положение атомов вещества. Совокупность позиций всех атомов будем называть маршрутом между вершинами. Каждому маршруту соответствует определенная протяженность между вершинами. Цель отжига – привести маршрут в состояние с наименьшей протяженностью. Чем ниже уровень энергии, тем «меньше» протяженность маршрута.

В ходе «отжига» сначала задается некоторая температура, затем начинается медленное и контролируемое понижение этой температуры.

Вершины выстраиваются в состояние с наименьшей протяжённостью, однако, с определенной вероятностью они могут перейти и в состояние с большей протяженностью. Эта вероятность уменьшается вместе с температурой. Переход в худшее состояние помогает отыскать маршрут с меньшей протяженностью, чем начальная. Процесс завершается, когда температура падает до заранее заданного значения. Начальный маршрут формируется стохастическим путем.

Муравьиный алгоритм (алгоритм оптимизации муравьиной колонии) – позволяет находить приближенные решения задачи коммивояжёра. Основной идеей муравьиного алгоритма является имитация поведения муравьиной колонии при поиске еды. Проходя маршрут от одной вершины графа (жилища) до другой вершины графа (источника пищи), муравьи (агенты) оставляют после себя след феромона, то есть повышают значимость ребра, которая со временем уменьшается (испаряется). Значимость ребра на маршруте зависит от длины маршрута L и от количества агентов, которые прошли по маршруту.

Генетический алгоритм использует механизм эволюции – представляет собой естественный отбор. Его суть состоит в том, что более приспособленные особи имеют больше возможностей для выживания и размножения. При этом благодаря передаче генетической информации (генетическому наследованию) потомки наследуют от родителей основные их качества. Таким образом, потомки сильных индивидуумов также будут более эффективными, а их доля в общей массе особей будет возрастать. После смены нескольких десятков или сотен поколений средняя приспособленность особей данного вида заметно возрастает.

3. Проектирование

3.1 Обоснование выбранных технологий

Для выполнения курсовой работы были выбраны следующие технологии: Язык программирования C#, СУБД SQLite, .NET Framework.

Язык программирования C# был выбран исходя из того, что приложение будет разрабатываться для Windows, следовательно, кроссплатформенность не является весомым критерием выбора языка программирования, иначе выбор стоял бы между скриптовыми языками и Java/C++.

СУБД SQLite была выбрана исходя из простоты настройки и использования, а также высокой скорости доступа к файлам по сравнению с MySQL и PostgreSQL. Помимо этого, SQLite использует механизмы блокировки доступа к файлу на уровне ОС.

Для написания графического интерфейса был выбран .NET Framework, так как именно в этом фреймворке лучше всего реализованы механизмы работы с классическим приложением Windows Forms

3.2 Описание алгоритма

1. На вход подаются координаты x и y точки;
2. Создаётся объект `dot` с поданными координатами;
3. Создаются два списка – `CheckListX[]` и `CheckListY[]`, хранящие координаты «иксов» и «игреков»;
4. Координаты объекта `dot` сравниваются с содержимым списков `CheckListX[]` и `CheckListY[]`
5. При нахождении полных совпадений координат выводится сообщение о том, что такая точка уже существует, иначе `dot` добавляется в список `DotList[]` элементов класса `Dot`, при этом выдав об этом сообщение;
6. Если точек меньше трёх, то переходим к пункту 1, иначе к 7;

7. К списку `DotList[]` применяется метод `Algorithm(List<Dot> dots)`, суть которого состоит в том, чтобы создать новый список с уже упорядоченными объектами `dot` внутри списка `ListTrueDots[]` таким образом, чтобы вторая точка была самой близкой по отношению к первой. Для этого применяется формула расстояния между двумя точками: $AB =$

$$\sqrt{|x_B - x_A|^2 + |y_B - y_A|^2};$$

8. Сравниваются все расстояния между первой точкой и остальными точками, чтобы найти отрезок наименьшей длины;

9. После того, как была найдена ближайшая точка по отношению к первой точке, то она добавляется в список `ListTrueDots[]`, после чего процесс повторяется до тех пор, пока не будут отсортированы все точки;

10. Находится общая длина пути методом суммирования отрезков, соединяющих ближайшие точки;

11. Так как из последней точки нужно попасть обратно в первую, то к общей длине пути прибавляем расстояние от последней точки списка `ListTrueDots[]` до первой точки этого же списка по формуле расстояния между двумя точками;

12. Записываем полное расстояние в таблицу `history` базы данных `DB`;

13. Создаются два списка `x[]` и `y[]`, в которых будут храниться координаты соответствующих осей точек в списке `ListTrueDots[]`;

14. Инициализируем точки на графике `chart1`;

15. Соединяем точки по порядку списка `ListTrueDots[]`.

Ниже представлена блок-схема алгоритма (Рисунок 3.1-3.9).

3.3 Сложность алгоритма

В худшем случае алгоритм приводит к туру, который намного длиннее оптимального тура. Если быть точным, то для каждой константы r существует пример проблемы, связанной с путешествующим продавцом, так

что длина тура, вычисленная ближайшим соседним алгоритмом, больше, чем g -кратная длина оптимального тура. Более того, для каждого числа городов существует присвоение расстояний между городами, для которых алгоритм ближайшего соседа производит уникальный наихудший тур. (Если алгоритм применяется к каждой вершине в качестве начальной вершины, лучший найденный путь будет лучше, чем по крайней мере $N/2-1$ другие туры, где N - число вершин).

Сложность алгоритма в худшем случае $O(N^2)$.

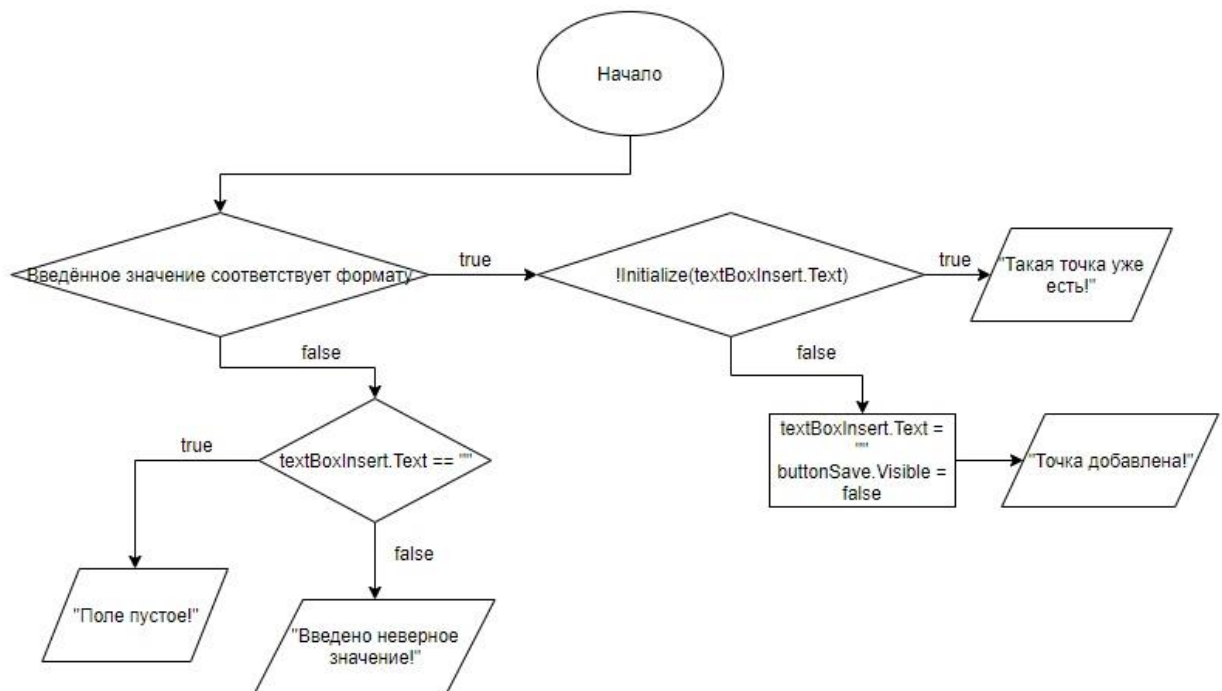


Рисунок 3.1 – Блок-схема алгоритма добавления точки

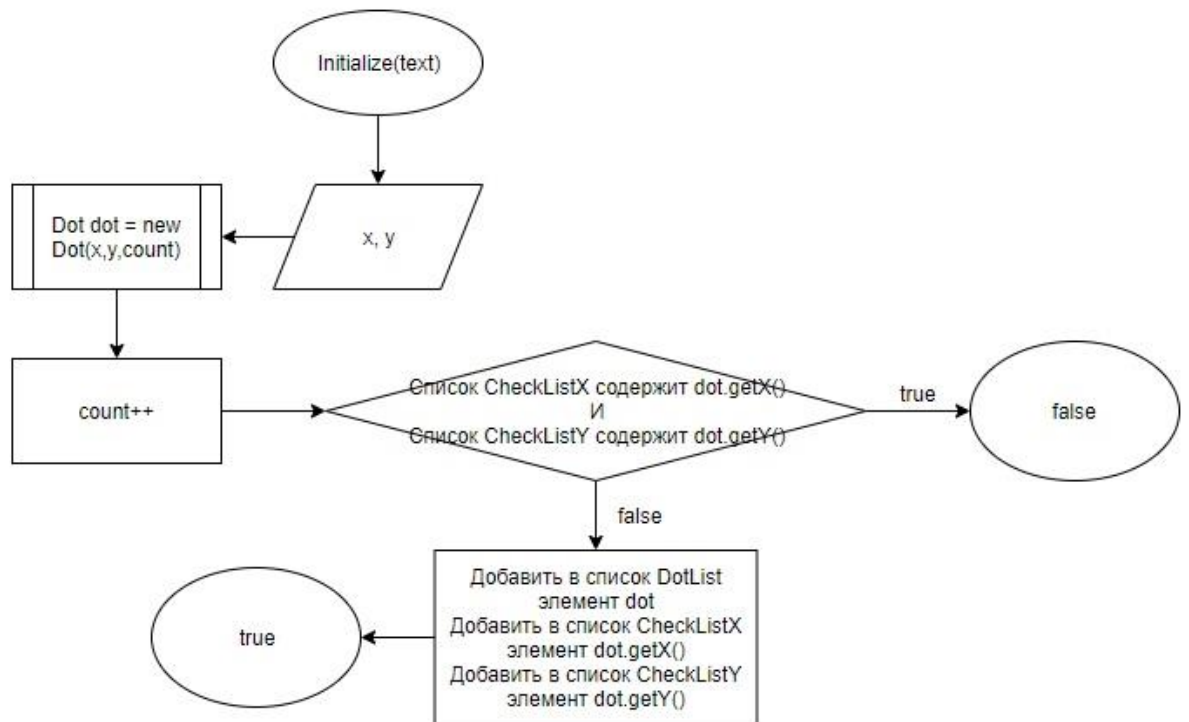


Рисунок 3.2 – Блок-схема метода Initialize

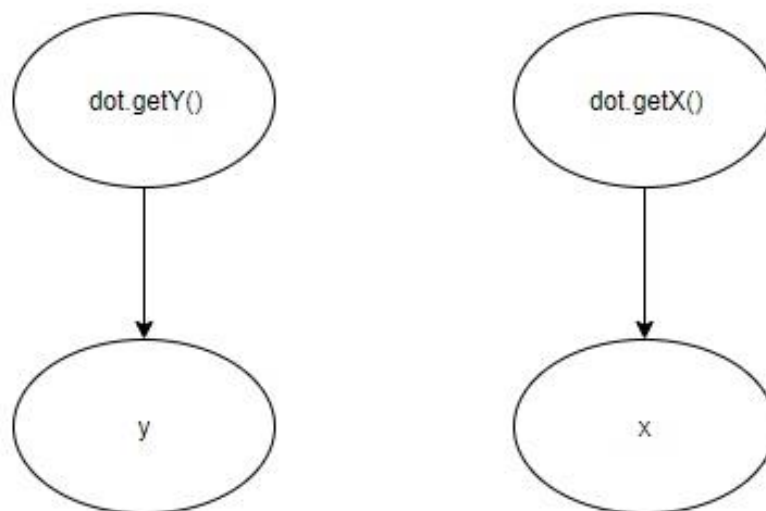


Рисунок 3.3 – Блок-схема методов getY и getX

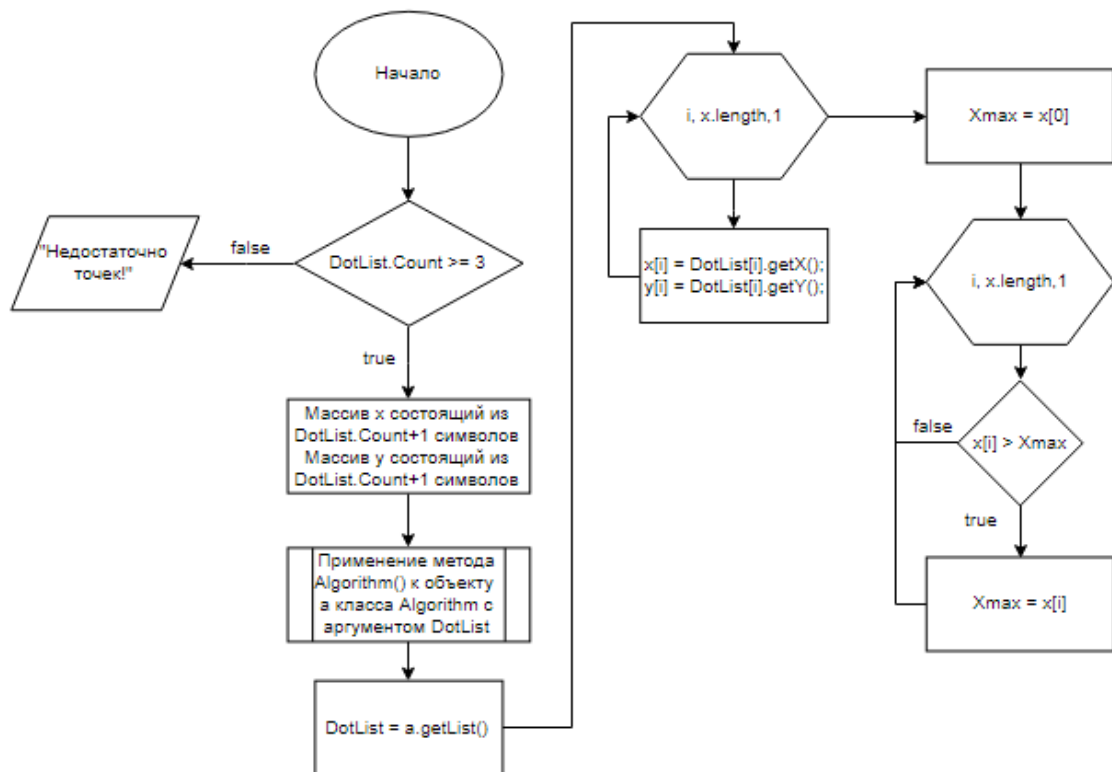


Рисунок 3.4 – Блок-схема алгоритма соединения точек

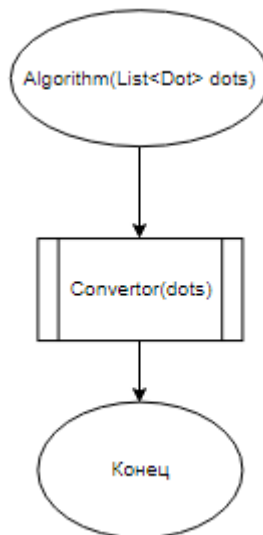
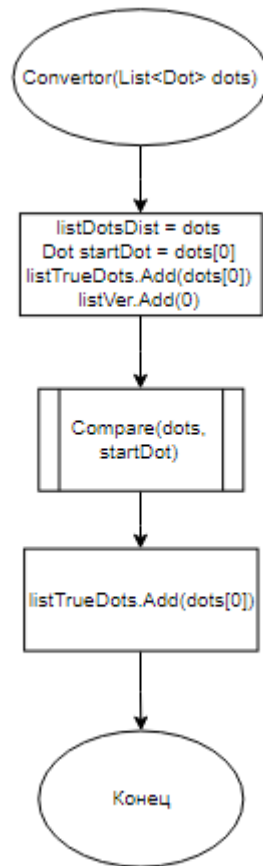
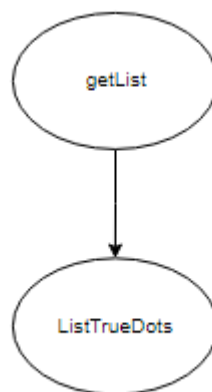


Рисунок 3.5 – Блок-схема метода Algorithm

Рисунок 3.6 – Блок-схема метода `Convertor`Рисунок 3.7 – Блок-схема метода `getList`

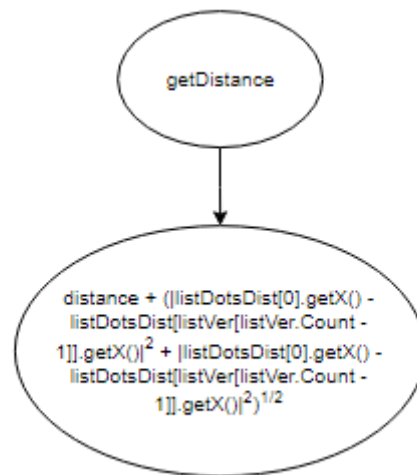


Рисунок 3.8 – Блок-схема метода getDistance

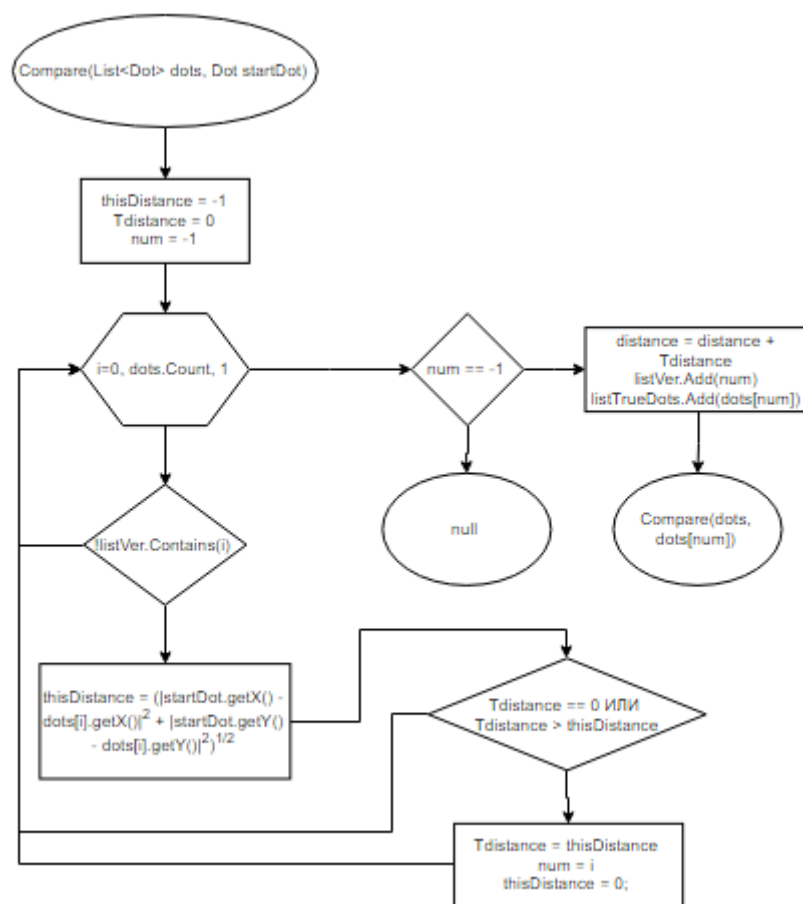


Рисунок 3.9 – Блок-схема метода Compare

3.4 UML диаграммы

Для представления структуры программы приведены диаграмма прецедентов (Рисунок 3.10), а также диаграмма взаимодействия классов (Рисунок 3.11).

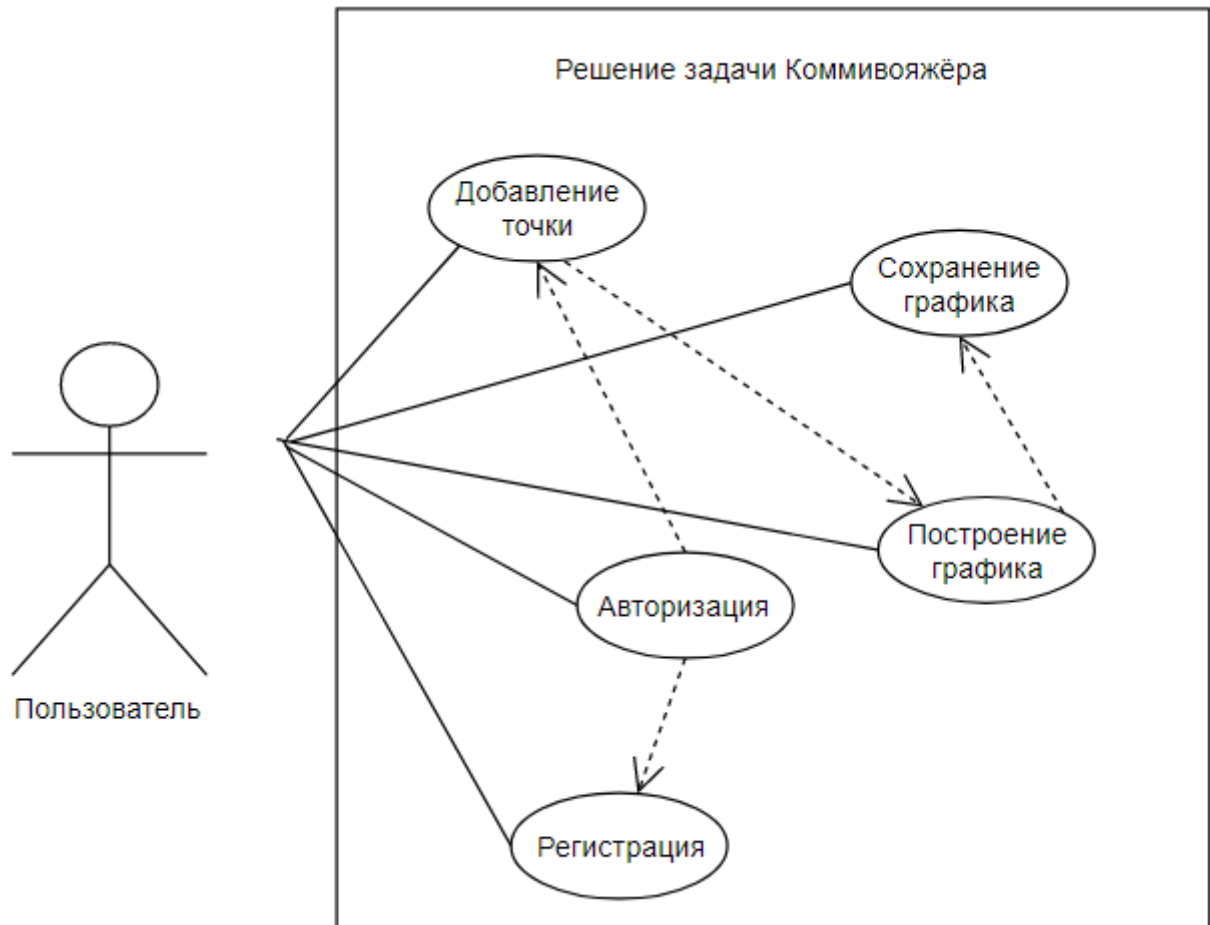


Рисунок 3.10 – Диаграмма прецедентов

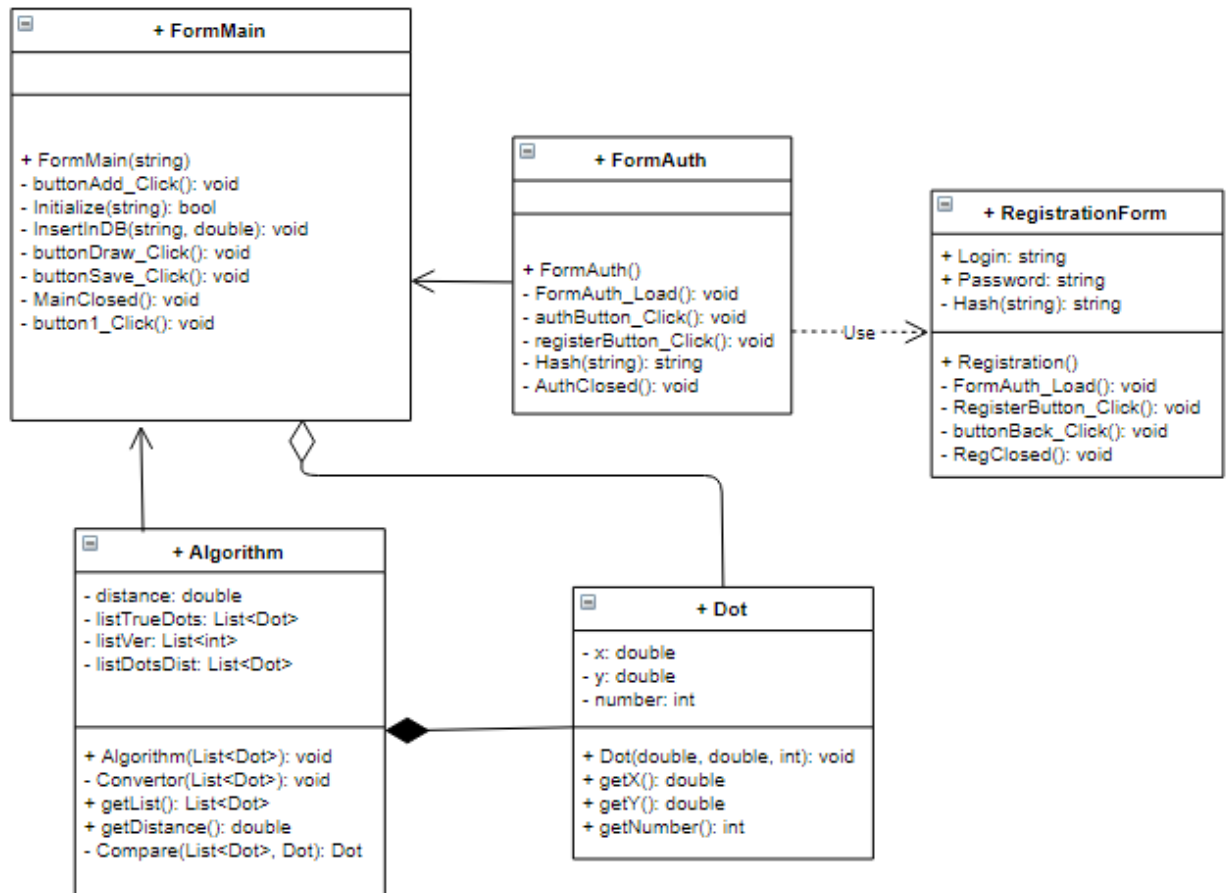


Рисунок 3.11 – Диаграмма классов

Ниже представлена ER-диаграмма базы данных (Рисунок 3.12)

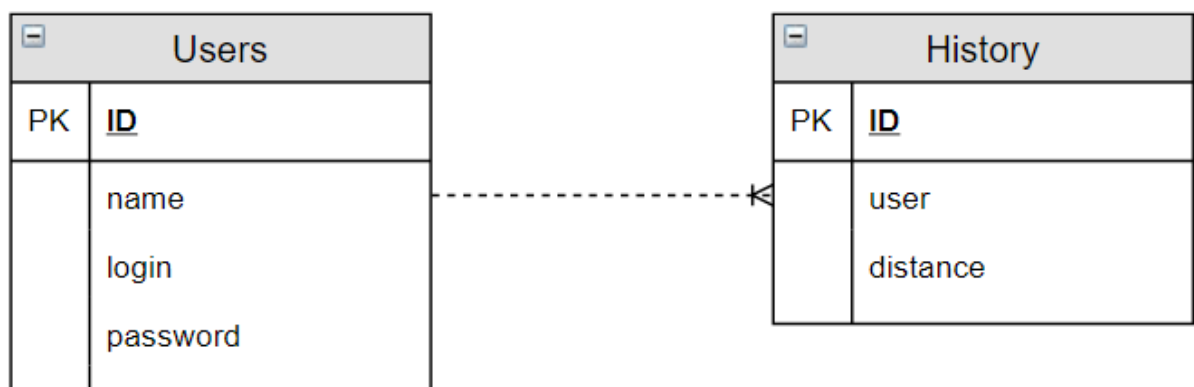


Рисунок 3.12 – ER-диаграмма базы данных

4. Разработка

Было создано два класса – Dot и Algorithm. Первый необходим для инициализации точек, а второй реализует непосредственно сам алгоритм работы.

В классе Dot присутствуют следующие поля:

- x – координата оси абсцисс;
- y – координата оси ординат;
- number – номер точки.

В классе Dot присутствуют следующие методы:

- Dot(double x, double y, int number) – хранит координаты точки, а также её номер;
- getX() – метод чтения координаты x;
- getY() – метод чтения координаты y;
- getNumber() – метод чтения порядкового номера точки.

В классе Algorithm присутствуют следующие поля:

- distance – общая длина пути;
- listTrueDots – упорядоченный список заданных точек;
- listVer – список номеров упорядоченных точек;
- listDotsDist – неупорядоченный список заданных точек;

В классе Algorithm используются следующие методы:

- Algorithm(List<Dot> dots) – непосредственно алгоритм;
- Convertor(List<Dot> dots) – метод создания упорядоченного списка точек на основе неупорядоченного;
- getList() – метод чтения упорядоченного списка точек;
- getDistance() – метод расчёта полной длины пути;
- Dot Compare(List<Dot> dots, Dot startdot) – метод, позволяющий сравнивать расстояния между точками для выбора кратчайшего из них.

Была создана форма авторизации (Рисунок 4.1)

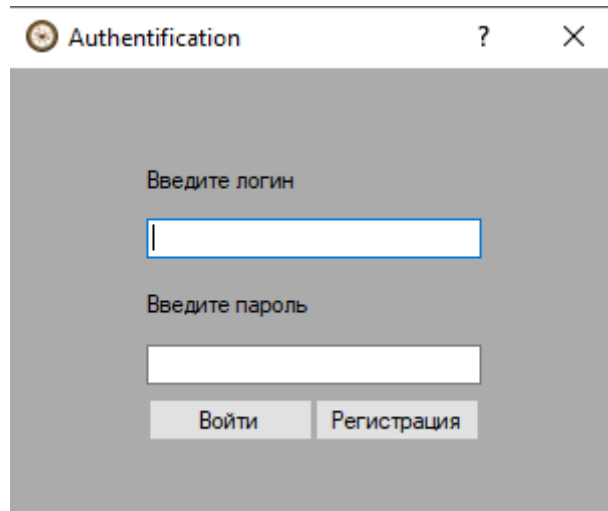
The image shows a window titled "Authentication" with a question mark icon and a close button. Inside the window, there are two text input fields. The first field is preceded by the label "Введите логин" (Enter login). The second field is preceded by the label "Введите пароль" (Enter password). Below the input fields, there are two buttons: "Войти" (Login) and "Регистрация" (Registration).

Рисунок 4.1 – Форма авторизации

В верхнее поле вводится логин, в нижнее – пароль. При нажатии на кнопку Войти происходит запрос в базу данных. Если пользователь найден в базе и введенный пароль совпадает с паролем, находящимся в базе, то происходит авторизация.

Если пользователю необходимо зарегистрироваться, то можно воспользоваться кнопкой Регистрация и зарегистрировать нового пользователя.

На рисунке 4.2 представлен внешний вид формы регистрации.

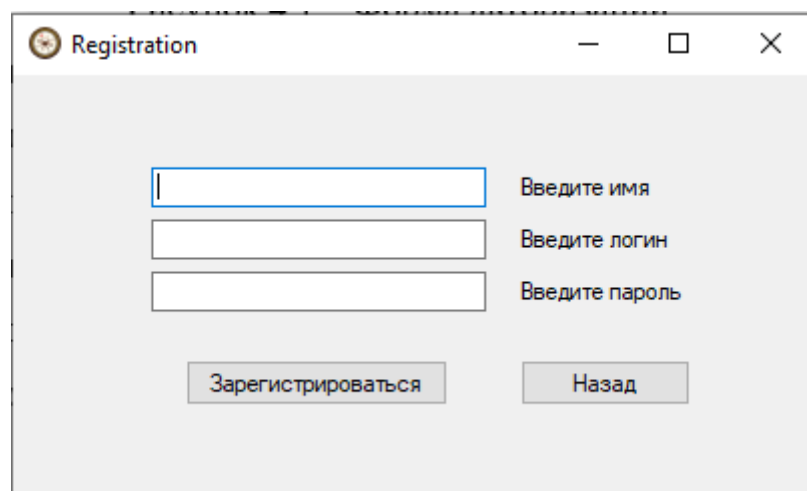
The image shows a window titled "Registration" with a question mark icon and standard window controls. Inside the window, there are three text input fields arranged vertically. To the right of each field is a label: "Введите имя" (Enter name) for the first, "Введите логин" (Enter login) for the second, and "Введите пароль" (Enter password) for the third. Below the input fields, there are two buttons: "Зарегистрироваться" (Register) and "Назад" (Back).

Рисунок 4.2 – Форма регистрации

В верхнем поле вводится имя пользователя, в среднем поле – его логин, а в нижнем – пароль. Если все требования к вводимым данным верны, то пользователь будет зарегистрирован при нажатии на кнопку Зарегистрироваться.

Кнопка Назад отвечает за переход к предыдущему окну (Авторизация).

Ниже продемонстрировано главная форма программы (Рисунок 4.3).

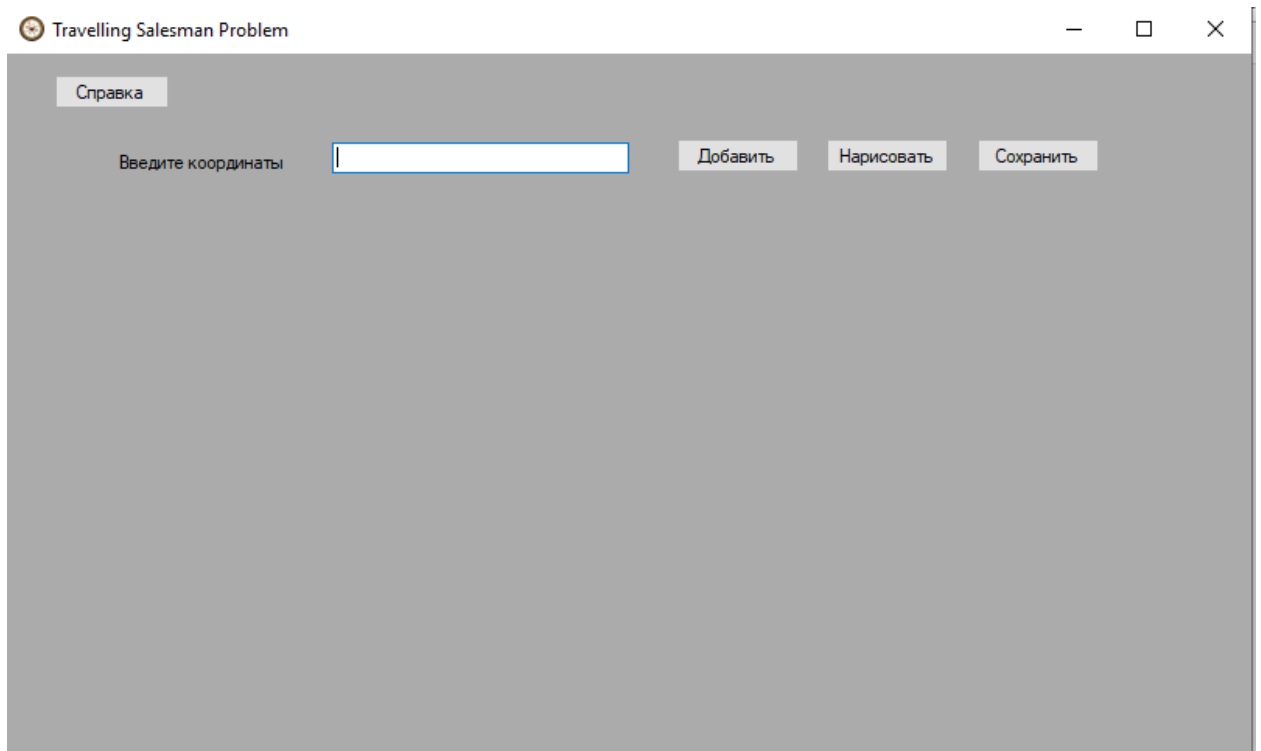


Рисунок 4.3 – Главная форма программы

На главном окне присутствует поле ввода координат, кнопка добавления точки, кнопка, позволяющая нарисовать график, кнопка сохранения графика в виде изображения в форматах png, bmp или jpg, а также кнопка справки.

Чтобы точка была добавлена, она должна соответствовать формату $([*],[*];[*],[*])$ иначе выведется сообщение о том, что точка не была добавлена.

Ниже продемонстрировано окно Справки (Рисунок 4.4) и главное окно с нарисованным графиком (Рисунок 4.5).

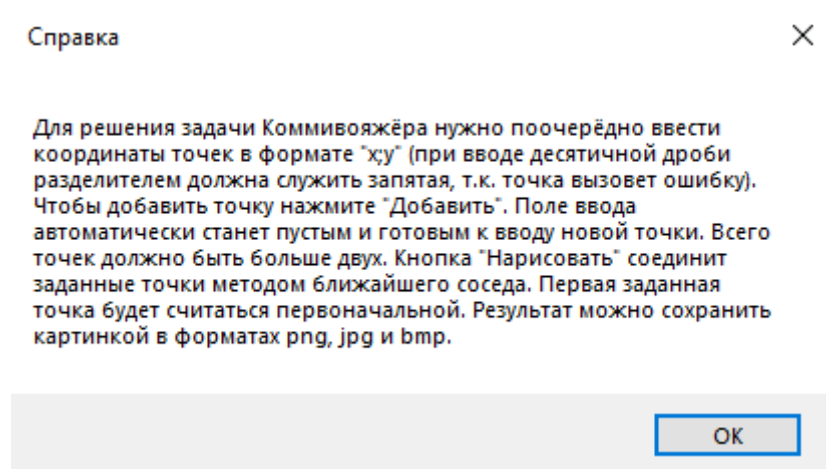


Рисунок 4.4 – Окно справки

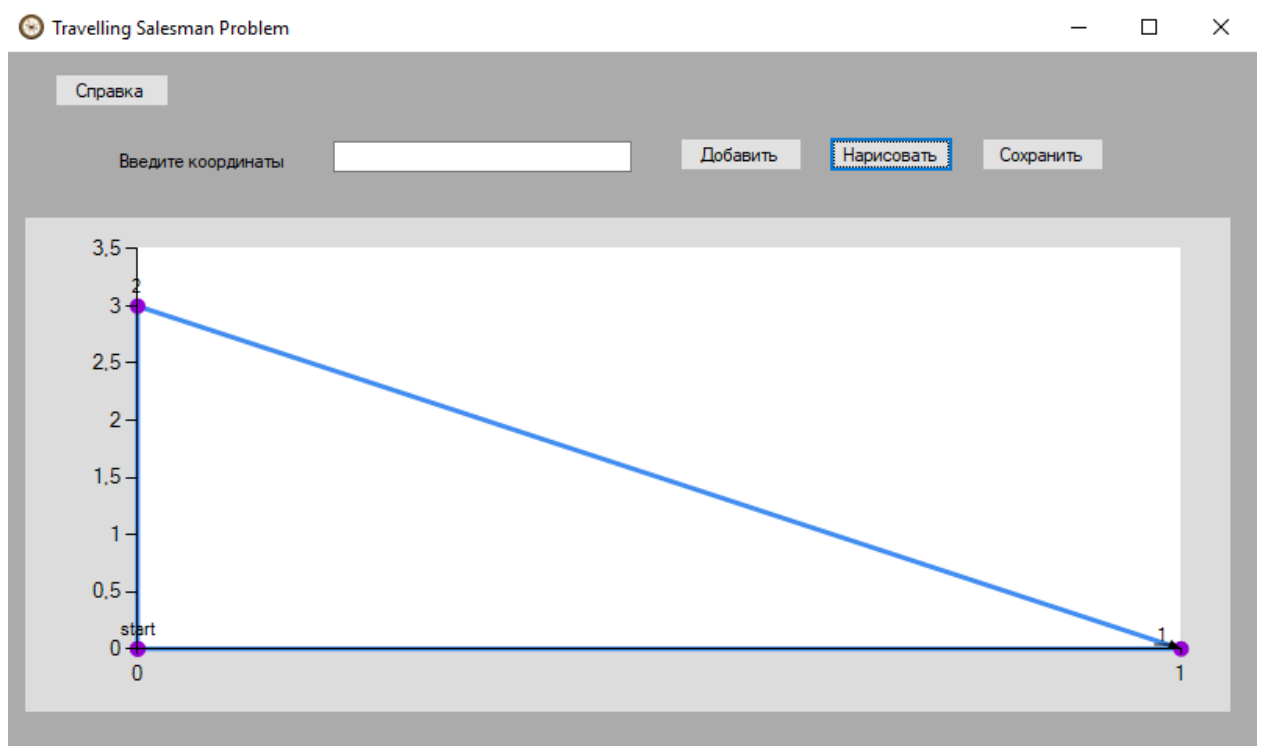


Рисунок 4.5 – Главное окно с нарисованным графиком

Окно сохранения графика представлено на рисунке 4.6

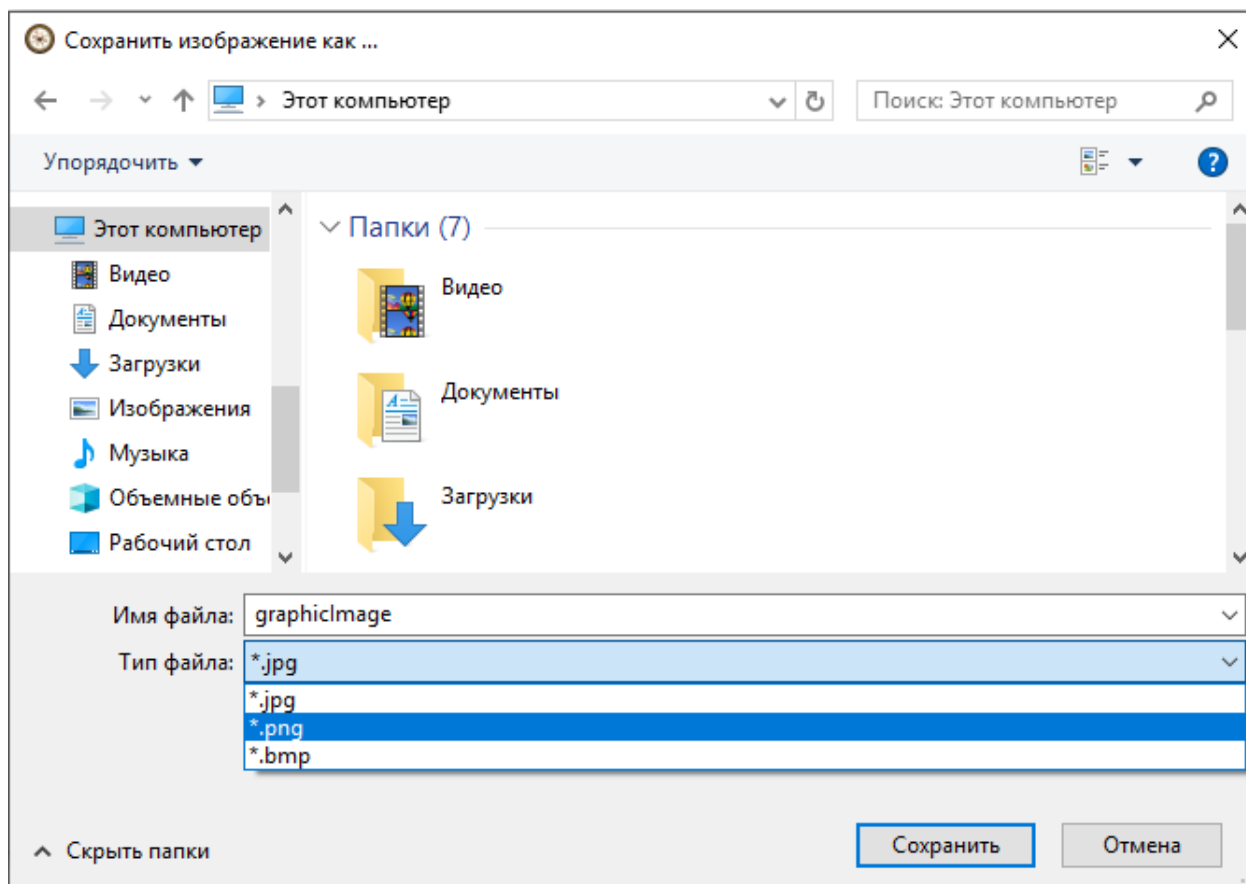


Рисунок 4.6 – Окно сохранения графика

Скриншот сохраненной картинки показан на рисунке 4.7

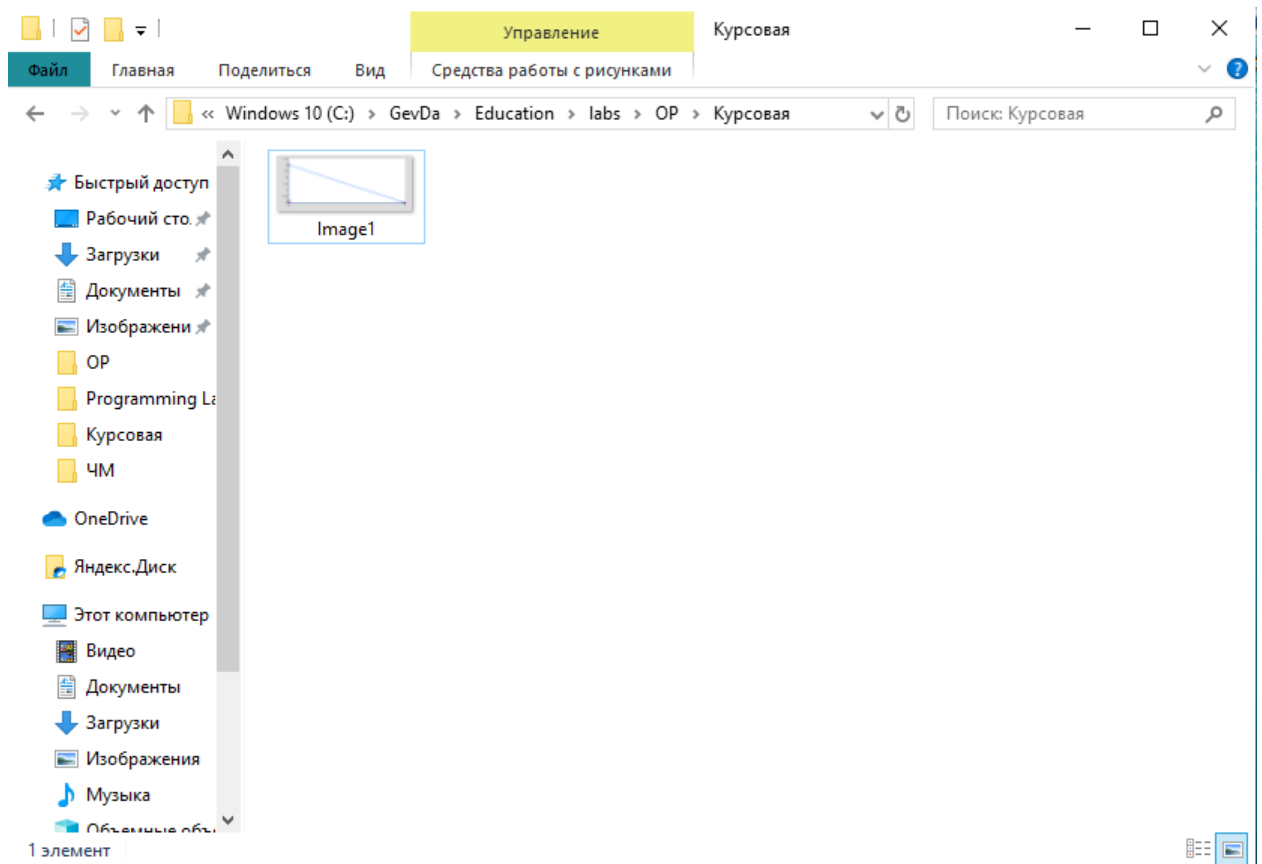


Рисунок 4.7 – Сохраненная картинка

Также стоит отметить, что механизм проверки точки на наличие уже существующей такой же реализован в форме главного окна при нажатии на кнопку добавления точки:

```
private void buttonAdd_Click(object sender, EventArgs e)
{
    if (Regex.IsMatch(textBoxInsert.Text, "^(\\s+)?[0-9]*[.,]?[0-9]+(\\s+)?\\; (\\s+)?[0-9]*[.,]?[0-9]+(\\s+)?$"))
    {
        if (!Initialize(textBoxInsert.Text))
        {
            textBoxInsert.Text = "";
            MessageBox.Show("Такая точка уже есть!");
        }
        else
        {
            textBoxInsert.Text = "";
            MessageBox.Show("Точка добавлена!");
            buttonSave.Visible = false;
        }
    }
    else
    {
        if (textBoxInsert.Text == "")
```



```

    {
        MessageBox.Show("Поле пустое!");
    }
    MessageBox.Show("Введено неверное значение!");
}
}

```

Более того, в форме главного окна присутствует вспомогательный метод инициализации точки, привязанный, собственно, к кнопке добавления точки:

```

private bool Initialize(string text)
{
    text = text.Replace(" ", "");
    double x = Double.Parse(text.Substring(0, text.IndexOf(";")));
    double y = Double.Parse(text.Substring(text.LastIndexOf(";") + 1));
    Dot dot = new Dot(x, y, count);
    count++;
    if ((CheckListX.Contains(dot.getX())) && (CheckListY.Contains(dot.getY())))
    {
        return false;
    }
    else
    {
        DotList.Add(dot);
        CheckListX.Add(dot.getX());
        CheckListY.Add(dot.getY());
        return true;
    }
}

```

Не стоит забывать, что в форме главного окна также реализована запись пройденного Коммивояжёром пути в таблицу базы данных:

```

private void InsertInDB(string user, double dist)
{
    chart1.Visible = true;
    try
    {
        SQLiteConnection DB = new SQLiteConnection("Data Source=DB.db;
Version=3");
        DB.Open();
        SQLiteCommand CMD = DB.CreateCommand();
        CMD.CommandText = "INSERT INTO 'history' ('user', 'dist') VALUES ('" +
name + "', '" + dist.ToString() + "')";
        CMD.ExecuteNonQuery();
        DB.Close();
    }
    catch
    {
    }
}

```

Регистрация нового пользователя реализована в окне регистрации при нажатии на кнопку Зарегистрироваться:

```

private void RegisterButton_Click(object sender, EventArgs e)
{
    try
    {
        if (!nameRegBox.Text.Equals("") && !passRegBox.Text.Equals("") &&
!logRegBox.Text.Equals(""))
        {
            if (!nameRegBox.Text.Trim().Contains(" ") &&
!logRegBox.Text.Trim().Contains(" "))
            {
                if (Char.IsLetter(nameRegBox.Text[0]) &&
Char.IsLetter(logRegBox.Text[0]))
                {
                    if (!File.Exists("DB.db")) // если базы данных нету, то...
                    {
                        SQLiteConnection.CreateFile("DB.db"); // создать базу
данных, по указанному пути создаётся пустой файл базы данных
                        using (SQLiteConnection Connect = new
SQLiteConnection("Data Source=DB.db;Version=3;")) // в строке указывается к какой базе
подключаемся
                        {
                            // строка запроса, который надо будет выполнить в
базе
                            string commandText = "CREATE TABLE users(id INTEGER
PRIMARY KEY AUTOINCREMENT NOT NULL, name VARCHAR (50) NOT NULL, " +
                                "login VARCHAR(50) NOT NULL, password VARCHAR(50)
NOT NULL);" +
                                "CREATE TABLE history(id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, user VARCHAR(50) NOT NULL, dist VARCHAR(50) NOT NULL);";
// создать таблицу, если её нет
                            SQLiteCommand Command = new
SQLiteCommand(commandText, Connect);
                            Connect.Open();
                            int _result = Command.ExecuteNonQuery();
                            Connect.Close();
                        }
                        SQLiteConnection DB = new SQLiteConnection("Data
Source=DB.db; Version=3");
                        DB.Open();
                        SQLiteCommand CMDa = DB.CreateCommand();
                        CMDa.CommandText = "select * from users where login =
@login";
                        CMDa.Parameters.Add("@login",
System.Data.DbType.String).Value = logRegBox.Text;
                        SQLiteDataReader SQL = CMDa.ExecuteReader();

                        if (!SQL.HasRows)
                        {
                            SQLiteCommand CMD = DB.CreateCommand();
                            CMD.CommandText = "insert into users(name, login,
password) values( @name, @login, @password)";
                            CMD.Parameters.Add("@name",
System.Data.DbType.String).Value = nameRegBox.Text;
                            CMD.Parameters.Add("@login",
System.Data.DbType.String).Value = logRegBox.Text;
                            CMD.Parameters.Add("@password",
System.Data.DbType.String).Value = Hash(passRegBox.Text);
                            CMD.ExecuteNonQuery();
                            DB.Close();
                        }
                    }
                }
            }
        }
    }
}

```

```

        DB = null;
        CMD = null;
        CMDa = null;
        MessageBox.Show("Вы успешно зарегистрировались!");
        this.Hide();
        FormAuth f = new FormAuth();
        f.Show();
    }
    else
    {
        SQL.Close();
        DB.Close();
        DB = null;
        CMDa = null;
        MessageBox.Show("Такой пользователь уже существует!");
        nameRegBox.Text = "";
        logRegBox.Text = "";
        passRegBox.Text = "";
    }
}
else
{
    MessageBox.Show("Имя и Логин должны начинаться с буквы!");
    nameRegBox.Text = "";
    logRegBox.Text = "";
    passRegBox.Text = "";
}
}
else
{
    MessageBox.Show("Имя и Логин не могут содержать пробелы!");
    nameRegBox.Text = "";
    logRegBox.Text = "";
    passRegBox.Text = "";
}
}
else
{
    MessageBox.Show("Вы ввели не все значения!");
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.StackTrace);
}
}

```

Процесс авторизации также был реализован при помощи СУБД SQLite в форме Авторизация при нажатии на кнопку Войти:

```

private void authButton_Click(object sender, EventArgs e)
{
    try
    {
        SQLiteConnection DB = new SQLiteConnection("Data Source=DB.db;
Version=3");
        DB.Open();
        SQLiteCommand CMD = DB.CreateCommand();
    }
}

```

```

        CMD.CommandText = "select * from users where login = @login and password
= @password";
        CMD.Parameters.Add("@login", System.Data.DbType.String).Value =
textBoxLogin.Text;
        CMD.Parameters.Add("@password", System.Data.DbType.String).Value =
Hash(textBoxPass.Text);

        SQLiteDataReader SQL = CMD.ExecuteReader();
        if (SQL.HasRows)
        {
            FormMain f = new FormMain(textBoxLogin.Text);
            f.Left = this.Left;
            f.Top = this.Top;
            f.Show();
            DB.Close();
            this.Hide();
        }
        else
        {
            MessageBox.Show("Неправильно введён логин и/или пароль!");
        }
    }
    catch
    {
        MessageBox.Show("Вы не зарегистрировались!", "Ошибка!");
    }
}

```

Пароль хранится в БД в зашифрованном при помощи алгоритма хэширования SHA1 виде, чтобы при несанкционированном доступе к БД злоумышленник не смог извлечь хотя бы пароль пользователя:

```

private string Hash(string input)
{
    SHA1Managed sha1 = new SHA1Managed();
    var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
    var sb = new StringBuilder(hash.Length * 2);
    foreach (byte b in hash)
    {
        sb.Append(b.ToString("x2"));
    }
    return sb.ToString();
}

```

5. Тестирование

Программа была вручную протестирована разработчиком.

На рисунке 5.1 показана попытка входа без указания логина и пароля.

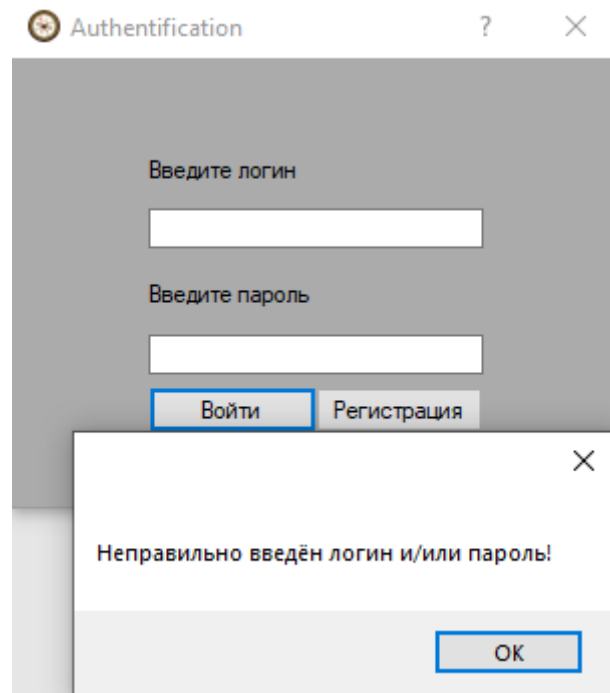


Рисунок 5.1 – Попытка входа без указания логина и пароля

На рисунке 5.2 показана попытка входа без указания логина.

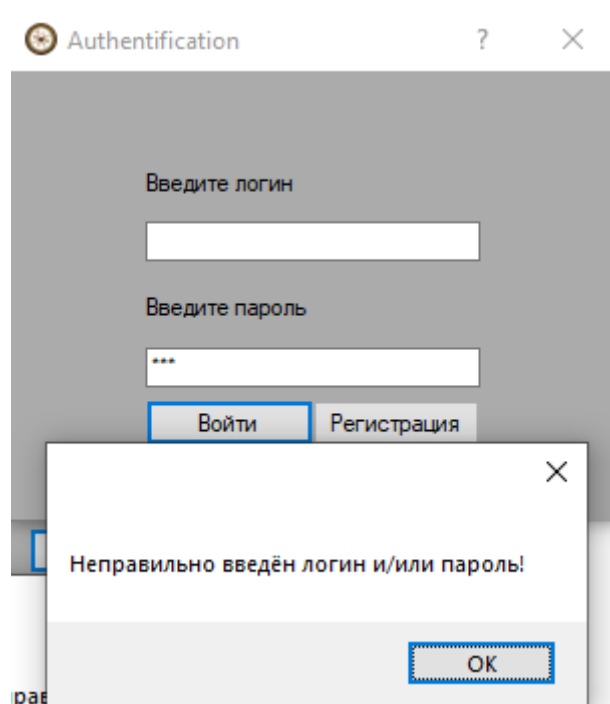


Рисунок 5.2 – Попытка входа без указания логина

На рисунке 5.3 показана попытка входа без указания пароля.

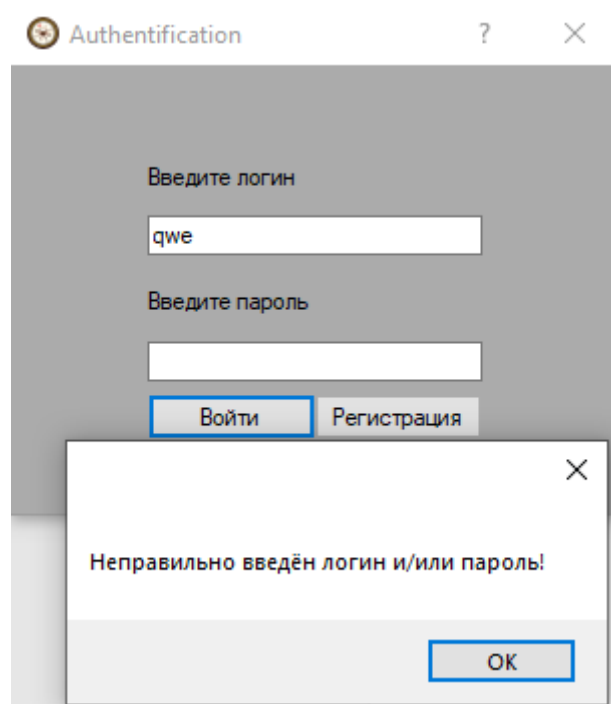


Рисунок 5.3 – Попытка входа без указания пароля

На рисунке 5.4 показана попытка входа с использованием неправильного пароля.

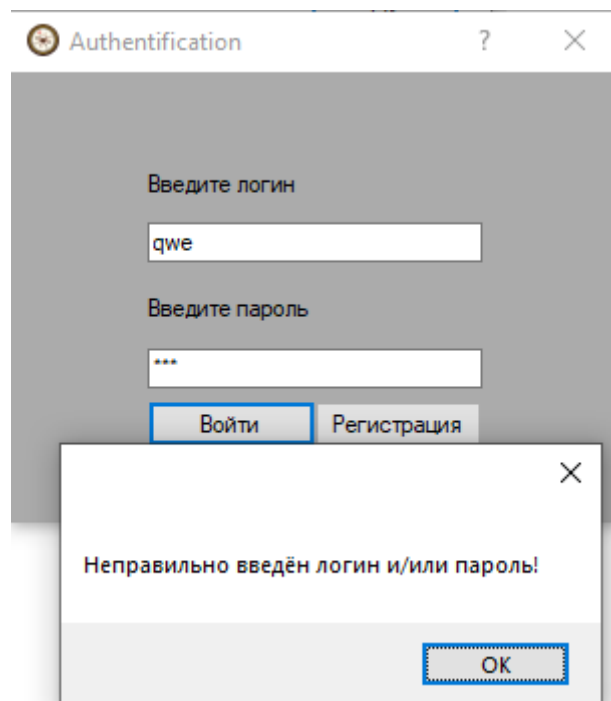


Рисунок 5.4 – Попытка входа с использованием неправильного пароля

На рисунке 5.5 показана попытка зарегистрировать уже существующего пользователя.

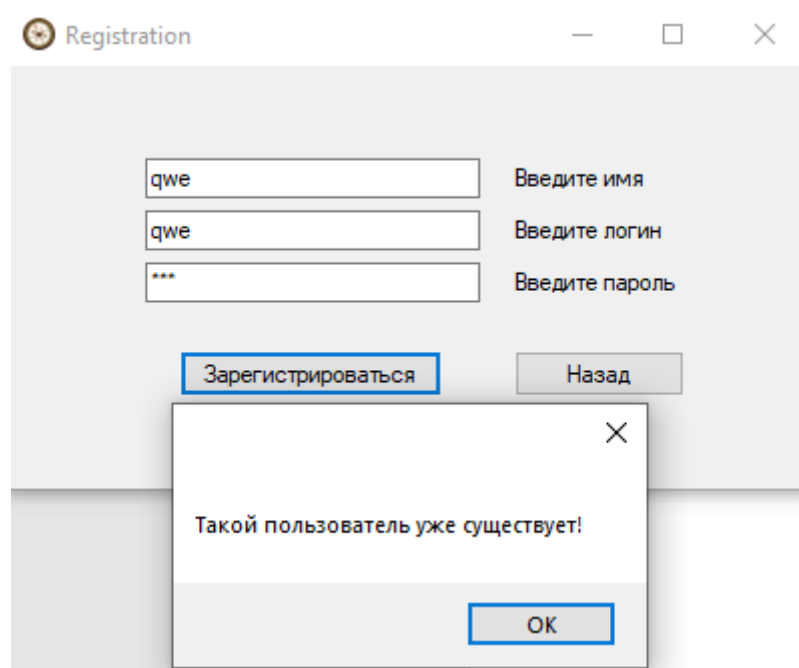


Рисунок 5.5 – Попытка зарегистрировать уже существующего пользователя

На рисунке 5.6 показана попытка регистрации пользователя с логином и именем, начинающимися с цифры.

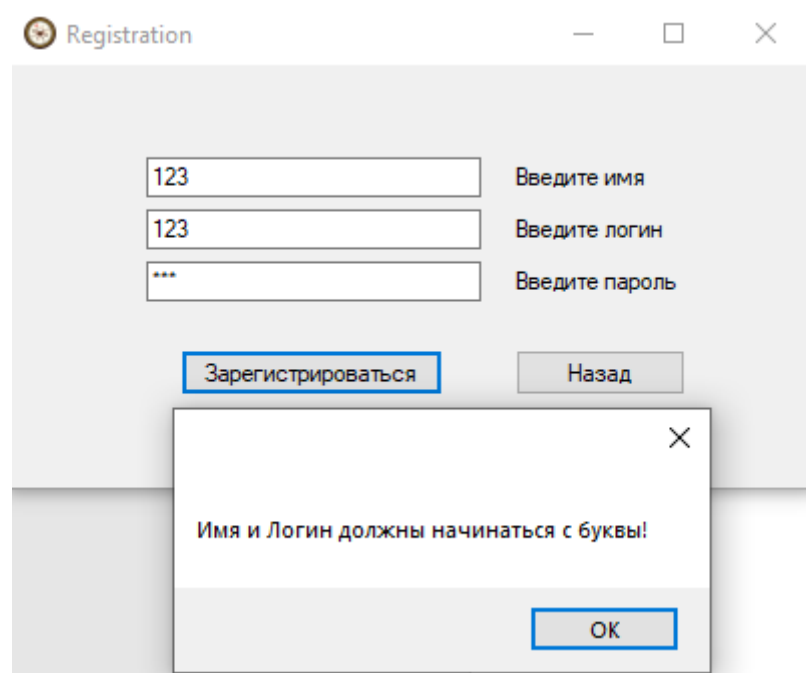


Рисунок 5.6 - Попытка регистрации пользователя с логином и именем, начинающимися с цифры

На рисунке 5.7 показана попытка регистрации пользователя с именем и логином, содержащими пробелы.

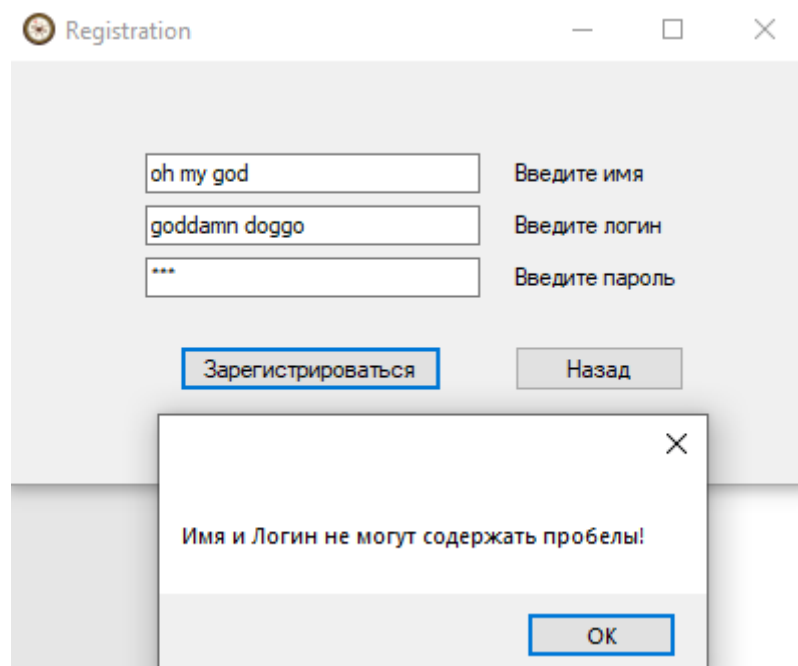


Рисунок 5.7 – Попытка регистрации пользователя с именем и логином, содержащими пробелы

На рисунке 5.8 показана попытка регистрации без введённого поля.

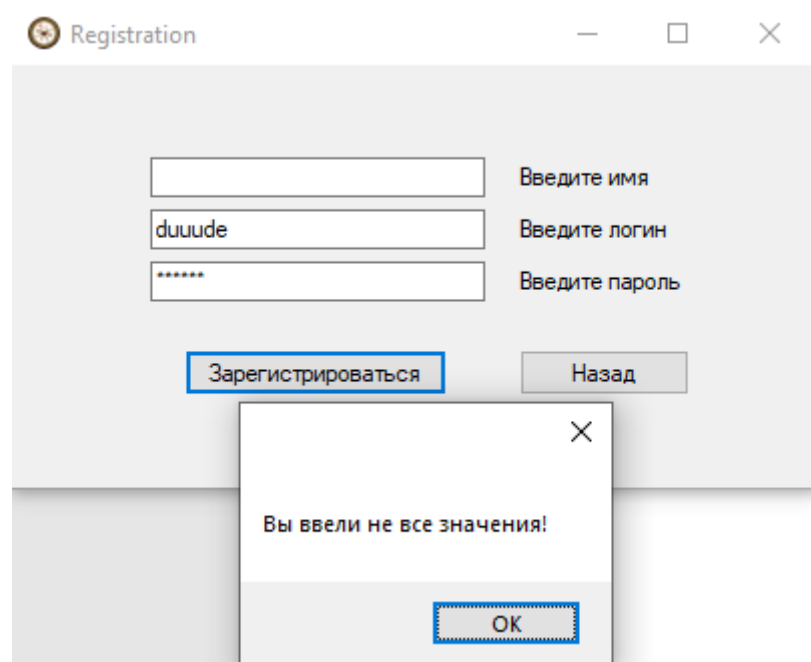


Рисунок 5.8 – Попытка регистрации без введённого поля

На рисунке 5.9 показана попытка регистрации с корректными данными.

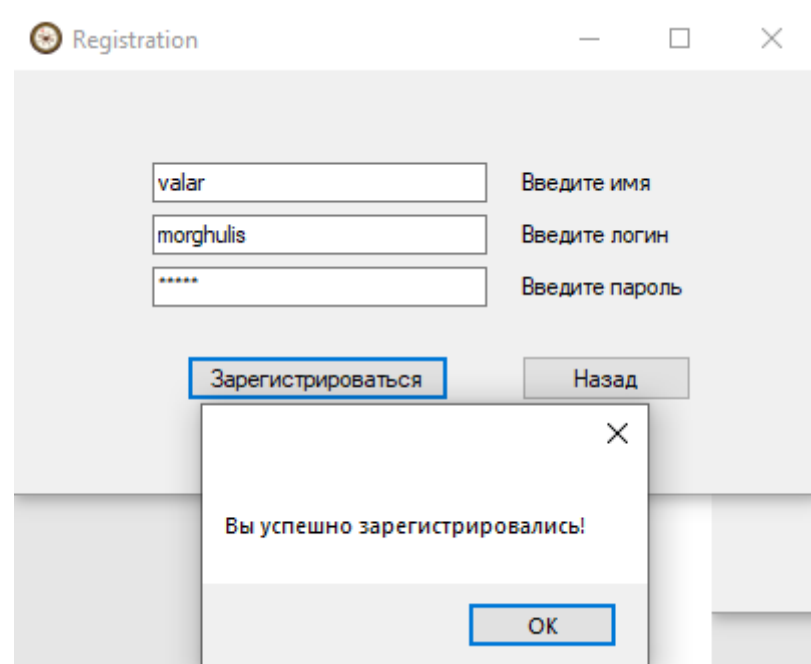


Рисунок 5.9 – Попытка регистрации с корректными данными

На рисунке 5.10 показана попытка ввести некорректное значение в поле ввода точки.

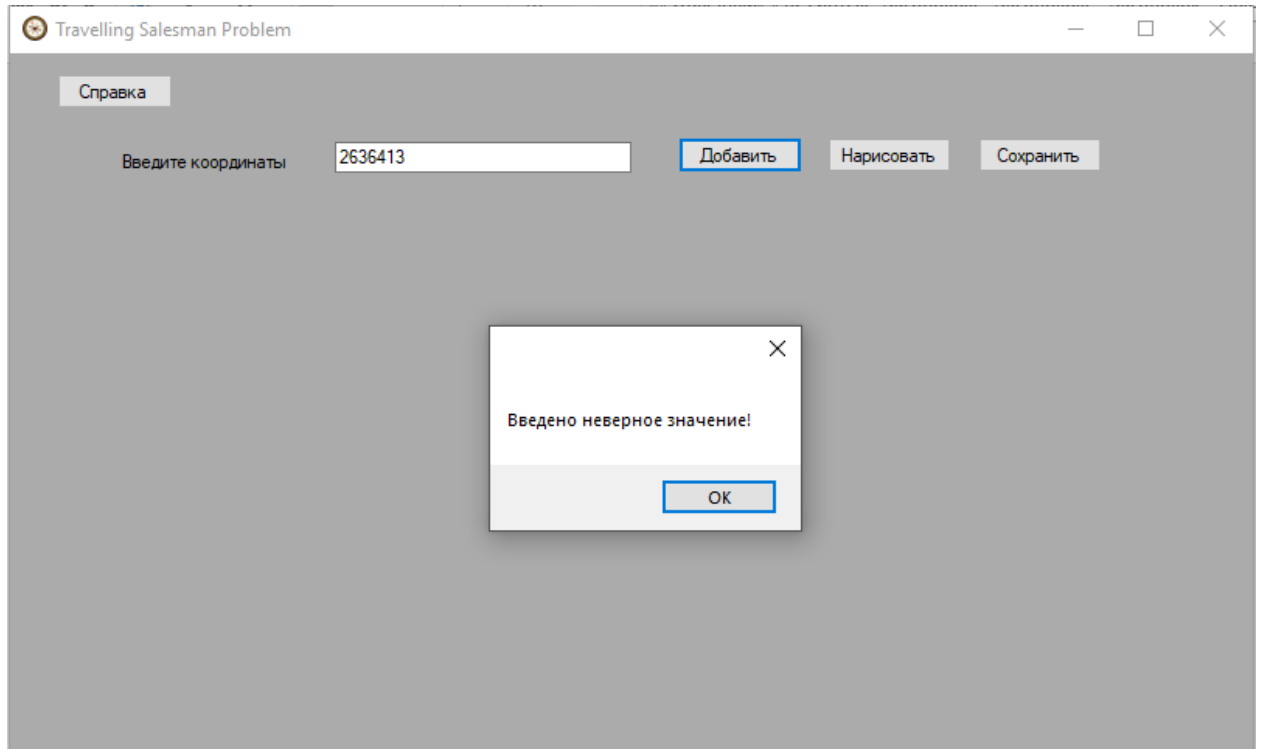


Рисунок 5.10 – Попытка ввода некорректных значений

На рисунке 5.11 показана попытка ввести отрицательные координаты.

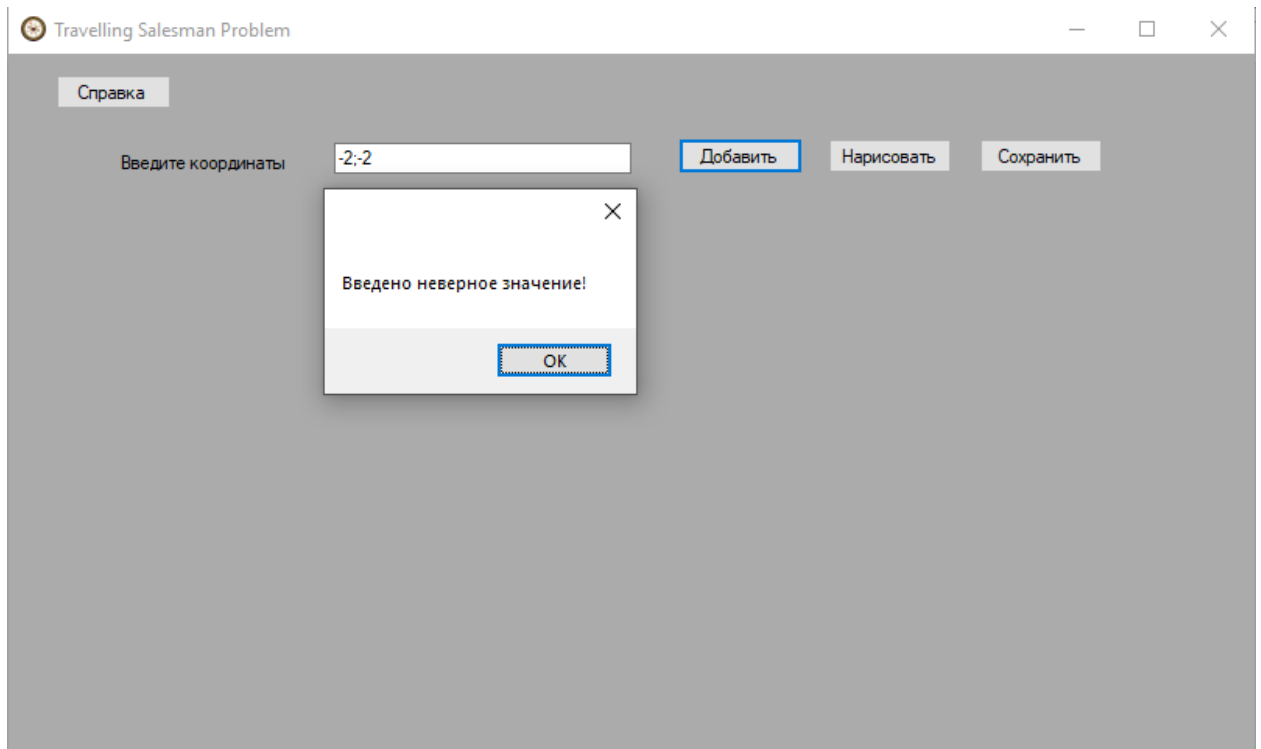


Рисунок 5.11 – Попытка ввода некорректных значений

На рисунке 5.12 показана попытка ввода точки с корректными значениями, а на рисунке 5.13 – результат.



Рисунок 5.12 – Попытка ввода точки с корректными значениями

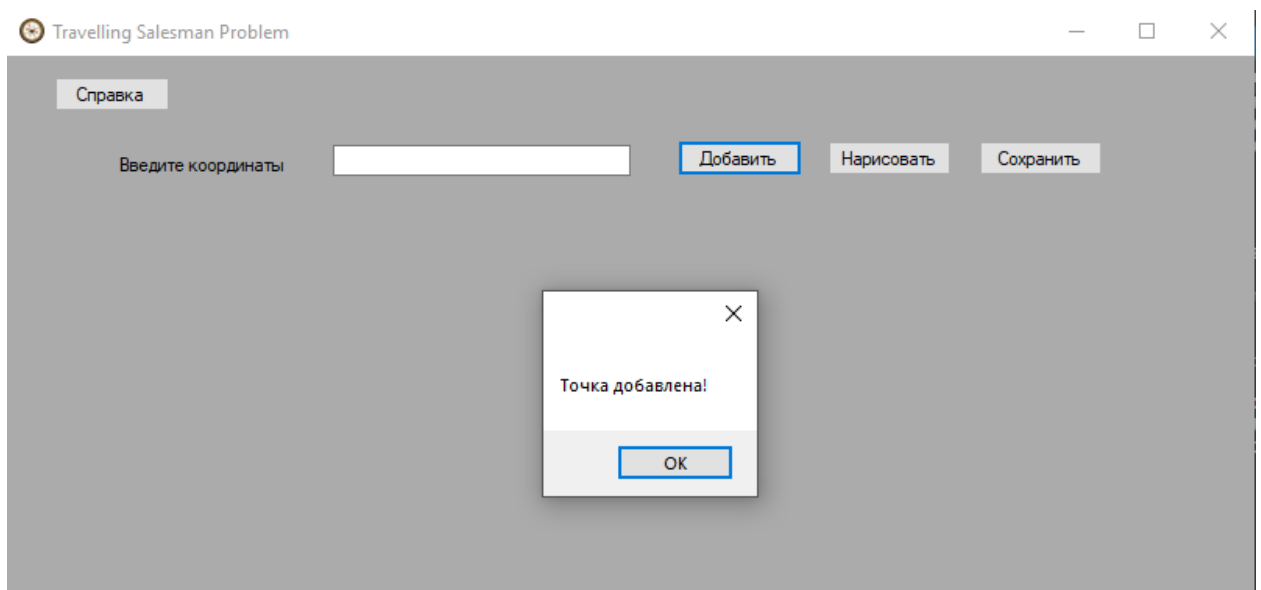


Рисунок 5.13 – Результат попытки ввода точки с корректными значениями

На рисунке 5.14 показана попытка ввода уже существующей точки

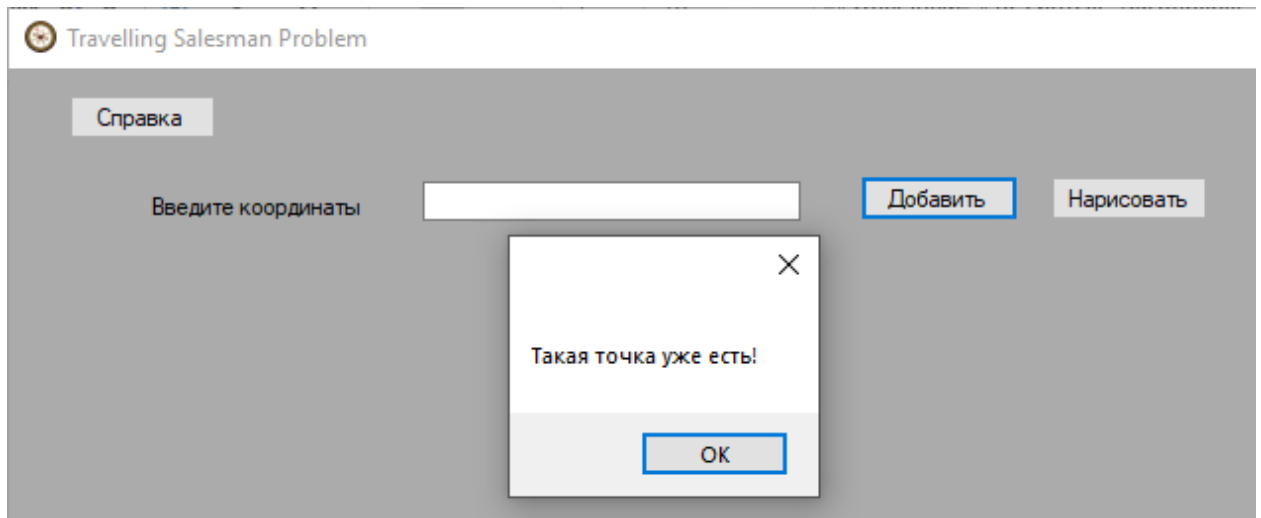


Рисунок 5.14 – Попытка ввода инициализированной точки

На рисунке 5.15 представлена попытка нарисовать график, инициализировав меньше 3 точек.

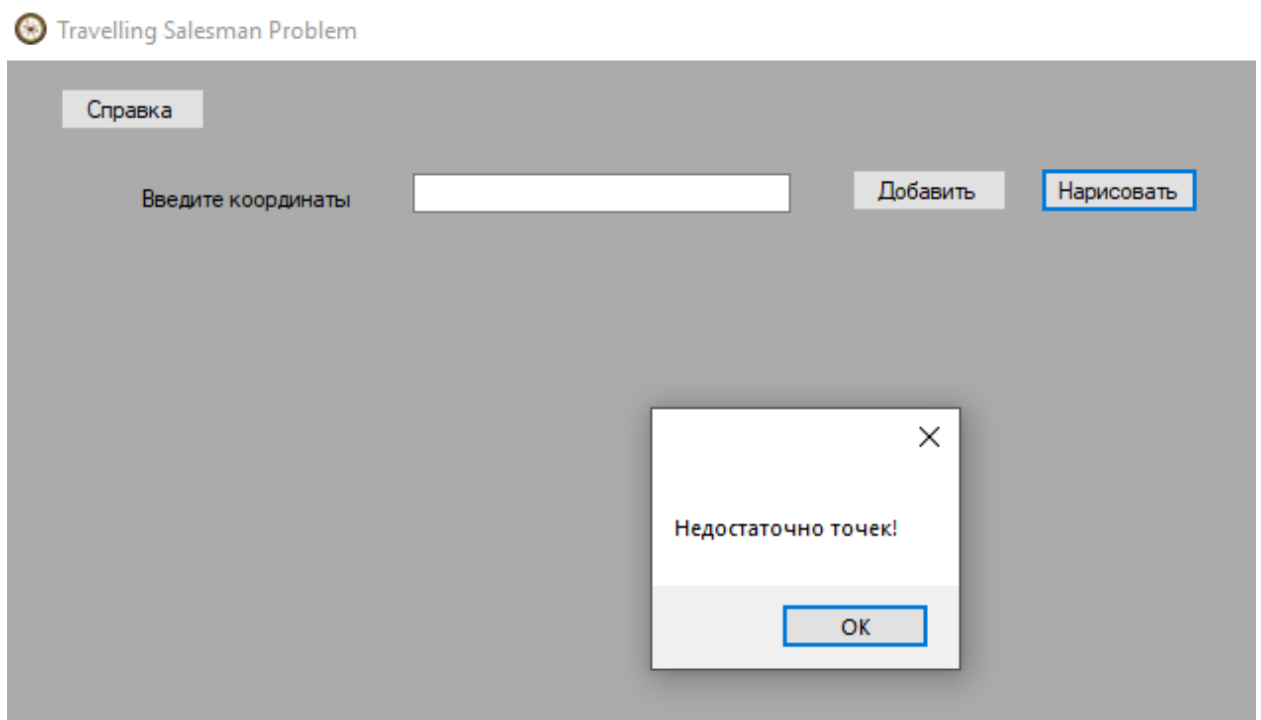


Рисунок 5.15 – Попытка нарисовать график с недостаточным количеством точек

Проведённые тесты указывают на работоспособность программы.

6. Заключение

В ходе выполнения данной курсовой работы было разработано приложение «Алгоритм ближайшего соседа в задаче Коммивояжёра», которое очень просто в использовании. В данном приложении пользователь проходит процесс регистрации и авторизации, вводятся точки для инициализации, а также строится график с возможностью сохранения в файл в виде картинки в форматах jpg, bmp и png.

В ходе выполнения курсовой работы приобретены следующие умения и навыки:

- умение реализации механизма авторизации;
- умение реализации механизма регистрации;
- навыки работы с SQLite;
- навыки работы с .NET Framework;
- навыки работы с Windows Forms;
- расширение знания работы с языком программирования C#.

Для достижения поставленной цели были решены следующие задачи:

- анализ темы «Алгоритм ближайшего соседа в задаче Коммивояжёра»;
- описание алгоритма программы;
- построение блок-схемы алгоритма;
- построение UML диаграммы прецедентов, диаграммы классов и баз данных;
- реализация механизма авторизации и регистрации;
- разработка программы «Алгоритм ближайшего соседа в задаче Коммивояжёра»;
- тестирование программы.

Также была приобретена следующая компетенция: способность применять языки, системы и инструментальные средства программирования в профессиональной деятельности.

7. Список использованной литературы

1. Электронный ресурс:

https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm;

2. Электронный ресурс:

https://en.wikipedia.org/wiki/Travelling_salesman_problem;

3. Документация по C#: <https://docs.microsoft.com/ru-ru/dotnet/csharp>;

4. Документация по SQLite: <https://www.sqlite.org/docs.html>;

5. Образовательный стандарт вуза ОС ТУСУР 02-2013:

https://storage.tusur.ru/files/40669/rules_gum_02-2013.pdf;

6. Электронный ресурс:

https://en.wikipedia.org/wiki/Time_complexity;

7. С.С. Харченко, «Учебно-методическое пособие по курсовой работе»;

Приложение А

Dot.cs

```
namespace kursa4
{
    public class Dot
    {
        private double x { get; set; }
        private double y { get; set; }
        private int number { get; set; }

        public Dot(double x, double y, int number)
        {
            this.number = number;
            this.x = x;
            this.y = y;
        }

        public double getX()
        {
            return x;
        }

        public double getY()
        {
            return y;
        }
        public int getNumber()
        {
            return number;
        }
    }
}
```


Приложение Б

Algorithm.cs

```

using System.Collections.Generic;
using static System.Math;

namespace kursa4
{
    public class Algorithm
    {
        private double distance;
        private List<Dot> listTrueDots = new List<Dot>();
        private List<int> listVer = new List<int>();
        private List<Dot> listDotsDist;
        public Algorithm(List<Dot> dots)
        {
            Convertor(dots);
        }

        private void Convertor(List<Dot> dots)
        {
            listDotsDist = dots;
            Dot startDot = dots[0];
            listTrueDots.Add(dots[0]);
            listVer.Add(0);
            Compare(dots, startDot);
            listTrueDots.Add(dots[0]);
        }

        public List<Dot> getList()
        {
            return listTrueDots;
        }

        public double getDistance()
        {
            return distance + Sqrt(Pow(Abs(listDotsDist[0].getX() -
listDotsDist[listVer[listVer.Count - 1]].getX()), 2)
+ Pow(Abs(listDotsDist[0].getY() -
listDotsDist[listVer[listVer.Count - 1]].getY()), 2));
        }

        private Dot Compare(List<Dot> dots, Dot startDot)
        {
            double thisDistance = -1;
            double Tdistance = 0;
            int num = -1;
            for (int i = 0; i < dots.Count; i++)
            {
                if (!listVer.Contains(i))
                {
                    thisDistance = Sqrt(Pow(Abs(startDot.getX() - dots[i].getX()), 2)
+ Pow(Abs(startDot.getY() - dots[i].getY()), 2));
                    if (Tdistance == 0 || Tdistance > thisDistance)
                    {
                        Tdistance = thisDistance;
                        num = i;
                        thisDistance = 0;
                    }
                }
            }
        }
    }
}

```

```
        }  
    }  
}  
if (num == -1)  
{  
    return null;  
}  
else  
{  
    distance = distance + Tdistance;  
    listVer.Add(num);  
    listTrueDots.Add(dots[num]);  
    return Compare(dots, dots[num]);  
}  
}  
}
```

Приложение В

FormAuth.cs

```

using System;
using System.Data.SQLite;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;

namespace kursa4
{
    public partial class FormAuth : Form
    {
        public FormAuth()
        {
            InitializeComponent();
        }

        private void FormAuth_Load(object sender, EventArgs e)
        {
        }

        private void authButton_Click(object sender, EventArgs e)
        {
            try
            {
                SQLiteConnection DB = new SQLiteConnection("Data Source=DB.db;
Version=3");
                DB.Open();
                SQLiteCommand CMD = DB.CreateCommand();
                CMD.CommandText = "select * from users where login = @login and password
= @password";
                CMD.Parameters.Add("@login", System.Data.DbType.String).Value =
textBoxLogin.Text;
                CMD.Parameters.Add("@password", System.Data.DbType.String).Value =
Hash(textBoxPass.Text);

                SQLiteDataReader SQL = CMD.ExecuteReader();
                if (SQL.HasRows)
                {
                    FormMain f = new FormMain(textBoxLogin.Text);
                    f.Left = this.Left;
                    f.Top = this.Top;
                    f.Show();
                    DB.Close();
                    this.Hide();
                }
                else
                {
                    MessageBox.Show("Неправильно введён логин и/или пароль!");
                }
            }
            catch
            {
                MessageBox.Show("Вы не зарегистрировались!", "Ошибка!");
            }
        }
    }
}

```

```

    }

    private void registerButton_Click(object sender, EventArgs e)
    {
        this.Hide();
        var new_user = new Registration();
        new_user.Closed += (s, args) => this.Close();
        new_user.Show();
    }

    private string Hash(string input)
    {
        SHA1Managed sha1 = new SHA1Managed();
        var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sb = new StringBuilder(hash.Length * 2);
        foreach (byte b in hash)
        {
            sb.Append(b.ToString("x2"));
        }
        return sb.ToString();
    }

    private void AuthClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }
}

```

Приложение Г

RegistrationForm.cs

```

using System;
using System.Data.SQLite;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace kursa4
{
    public partial class Registration : Form
    {
        public Registration()
        {
            InitializeComponent();
        }

        private void RegistrationForm_Load(object sender, EventArgs e)
        {
        }

        public string Login { get; }
        public string Password { get; }

        private string Hash(string input)
        {
            SHA1Managed sha1 = new SHA1Managed();
            var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
            var sb = new StringBuilder(hash.Length * 2);
            foreach (byte b in hash)
            {
                sb.Append(b.ToString("x2"));
            }
            return sb.ToString();
        }

        private void RegisterButton_Click(object sender, EventArgs e)
        {
            try
            {
                if (!nameRegBox.Text.Equals("") && !passRegBox.Text.Equals("") &&
                    !logRegBox.Text.Equals(""))
                {
                    if (!nameRegBox.Text.Trim().Contains(" ") &&
                        !logRegBox.Text.Trim().Contains(" "))
                    {
                        if (Char.IsLetter(nameRegBox.Text[0]) &&
                            Char.IsLetter(logRegBox.Text[0]))
                        {
                            if (!File.Exists("DB.db")) // если базы данных нету, то...
                            {
                                SQLiteConnection.CreateFile("DB.db"); // создать базу
                                данных, по указанному пути создаётся пустой файл базы данных
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        using (SQLiteConnection Connect = new
SQLiteConnection("Data Source=DB.db;Version=3;")) // в строке указывается к какой базе
подключаемся
        {
            // строка запроса, который надо будет выполнить на
базе
            string commandText = "CREATE TABLE users(id INTEGER
PRIMARY KEY AUTOINCREMENT NOT NULL, name VARCHAR (50) NOT NULL, " +
"login VARCHAR(50) NOT NULL, password VARCHAR(50)
NOT NULL);" +
"CREATE TABLE history(id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, user VARCHAR(50) NOT NULL, dist VARCHAR(50) NOT NULL);";
// создать таблицу, если её нет
            SQLiteCommand Command = new
SQLiteCommand(commandText, Connect);
            Connect.Open();
            int _result = Command.ExecuteNonQuery();
            Connect.Close();
        }
        SQLiteConnection DB = new SQLiteConnection("Data
Source=DB.db; Version=3");
        DB.Open();
        SQLiteCommand CMDa = DB.CreateCommand();
        CMDa.CommandText = "select * from users where login =
@login";
        CMDa.Parameters.Add("@login",
System.Data.DbType.String).Value = logRegBox.Text;
        SQLiteDataReader SQL = CMDa.ExecuteReader();

        if (!SQL.HasRows)
        {
            SQLiteCommand CMD = DB.CreateCommand();
            CMD.CommandText = "insert into users(name, login,
password) values( @name, @login, @password)";
            CMD.Parameters.Add("@name",
System.Data.DbType.String).Value = nameRegBox.Text;
            CMD.Parameters.Add("@login",
System.Data.DbType.String).Value = logRegBox.Text;
            CMD.Parameters.Add("@password",
System.Data.DbType.String).Value = Hash(passRegBox.Text);
            CMD.ExecuteNonQuery();
            DB.Close();
            DB = null;
            CMD = null;
            CMDa = null;
            MessageBox.Show("Вы успешно зарегистрировались!");
            this.Hide();
            FormAuth f = new FormAuth();
            f.Show();
        }
        else
        {
            SQL.Close();
            DB.Close();
            DB = null;
            CMDa = null;
            MessageBox.Show("Такой пользователь уже существует!");
            nameRegBox.Text = "";
            logRegBox.Text = "";
        }
    }
}

```

```

        passRegBox.Text = "";
    }

    }
    else
    {
        MessageBox.Show("Имя и Логин должны начинаться с буквы!");
        nameRegBox.Text = "";
        logRegBox.Text = "";
        passRegBox.Text = "";
    }
}
else
{
    MessageBox.Show("Имя и Логин не могут содержать пробелы!");
    nameRegBox.Text = "";
    logRegBox.Text = "";
    passRegBox.Text = "";
}
}
else
{
    MessageBox.Show("Вы ввели не все значения!");
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.StackTrace);
}
}

private void buttonBack_Click(object sender, EventArgs e)
{
    this.Hide();
    FormAuth Auth = new FormAuth();
    Auth.Show();
}

private void RegClosed(object sender, FormClosedEventArgs e)
{
    Application.Exit();
}
}
}

```

Приложение Д

FormMain.cs

```

using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace kursa4
{
    public partial class FormMain : Form
    {
        string name;
        List<Dot> DotList = new List<Dot>();
        List<double> CheckListX = new List<double>();
        List<double> CheckListY = new List<double>();
        int count = 0;

        public FormMain(string userName)
        {
            InitializeComponent();
            name = userName;
        }

        private void buttonAdd_Click(object sender, EventArgs e)
        {
            if (Regex.IsMatch(textBoxInsert.Text, "^(\\s+)?[0-9]*[.,]?[0-9]+(\\s+)?\\; (\\s+)?[0-9]*[.,]?[0-9]+(\\s+)?$"))
            {
                if (!Initialize(textBoxInsert.Text))
                {
                    textBoxInsert.Text = "";
                    MessageBox.Show("Такая точка уже есть!");
                }
                else
                {
                    textBoxInsert.Text = "";
                    MessageBox.Show("Точка добавлена!");
                    buttonSave.Visible = false;
                }
            }
            else
            {
                if (textBoxInsert.Text == "")
                {
                    MessageBox.Show("Поле пустое!");
                }
                MessageBox.Show("Введено неверное значение!");
            }
        }
    }
}

```



```

    }

    private bool Initialize(string text)
    {
        text = text.Replace(" ", "");
        double x = Double.Parse(text.Substring(0, text.IndexOf(";")));
        double y = Double.Parse(text.Substring(text.LastIndexOf(";") + 1));
        Dot dot = new Dot(x, y, count);
        count++;
        if ((CheckListX.Contains(dot.getX())) && (CheckListY.Contains(dot.getY())))
        {
            return false;
        }
        else
        {
            DotList.Add(dot);
            CheckListX.Add(dot.getX());
            CheckListY.Add(dot.getY());
            return true;
        }
    }

    private void InsertInDB(string user, double dist)
    {
        chart1.Visible = true;
        try
        {
            SQLiteConnection DB = new SQLiteConnection("Data Source=DB.db;
Version=3");
            DB.Open();
            SQLiteCommand CMD = DB.CreateCommand();
            CMD.CommandText = "INSERT INTO 'history' ('user', 'dist') VALUES ('" +
name + "', '" + dist.ToString() + "')";
            CMD.ExecuteNonQuery();
            DB.Close();
        }
        catch
        {
        }
    }

    private void buttonDraw_Click(object sender, EventArgs e)
    {
        if (DotList.Count >= 3)
        {
            double[] x = new double[DotList.Count + 1];
            double[] y = new double[DotList.Count + 1];

            Algorithm a = new Algorithm(DotList);
            DotList = a.getList();
            InsertInDB(name, a.getDistance());

            for (int i = 0; i < x.Length; i++)
            {
                x[i] = DotList[i].getX();
                y[i] = DotList[i].getY();
            }
        }
    }

```

```

double Xmax = x[0];
for (int i = 1; i < x.Length; i++)
{
    if (x[i] > Xmax)
        Xmax = x[i];
}

chart1.ChartAreas[0].AxisX.Minimum = 0;
chart1.ChartAreas[0].AxisX.Maximum = Xmax;
chart1.ChartAreas[0].AxisX.MajorGrid.Enabled = false;
chart1.ChartAreas[0].AxisY.MajorGrid.Enabled = false;
chart1.Series[0].Points.DataBindXY(x, y);

chart1.Series[0].ToolTip = "X = #VALX, Y = #VALY";

chart1.Series["Series1"].Points[0].Label = "start";
for (int i = 1; i < x.Length - 1; i++)
{
    chart1.Series["Series1"].Points[i].Label = i.ToString();
}

DotList.Clear();
CheckListX.Clear();
CheckListY.Clear();
buttonSave.Visible = true;
}
else
{
    MessageBox.Show("Недостаточно точек!");
}
}

private void buttonSave_Click(object sender, EventArgs e)
{
    using (SaveFileDialog sfd = new SaveFileDialog())
    {
        sfd.Title = "Сохранить изображение как ...";
        sfd.Filter = "*.jpg|*.jpg;|.png|.png;|.bmp|.bmp";
        sfd.AddExtension = true;
        sfd.FileName = "graphicImage";
        if (sfd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            switch (sfd.FilterIndex)
            {
                case 1: chart1.SaveImage(sfd.FileName,
ChartImageFormat.Jpeg); break;
                case 2: chart1.SaveImage(sfd.FileName, ChartImageFormat.Png);
break;
                case 3: chart1.SaveImage(sfd.FileName, ChartImageFormat.Bmp);
break;
            }
        }
    }
}

private void MainClosed(object sender, FormClosedEventArgs e)
{
    Application.Exit();
}

```

```

    }

    private void button1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Для решения задачи Коммивояжёра нужно поочерёдно ввести  

        координаты точек в формате \"x;y\" (при вводе десятичной дроби разделителем должна  

        служить запятая, т.к. точка вызовет ошибку). "+
        "Чтобы добавить точку нажмите \"Добавить\". Поле ввода автоматически  

        станет пустым и готовым к вводу новой точки. Всего точек должно быть больше двух. "+
        "Кнопка \"Нарисовать\" соединит заданные точки методом ближайшего соседа.  

        Первая заданная точка будет считаться первоначальной. Результат можно сохранить картинкой  

        в форматах png, jpg и bmp.", "Справка");
    }
}

```

Приложение Е

Program.cs

```
using System;
using System.Windows.Forms;

namespace kursa4
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormAuth());
        }
    }
}
```