МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
к. ф.-м. н., доцент
_____ А. С. Иванов

ОТЧЕТ О ПРАКТИКЕ

студента 4 курса 411 группы факультета КНиИТ
Власова Андрея Александровича

вид практики: учебная
кафедра: математической кибернетики и компьютерных наук
курс: 4
семестр: 8
продолжительность: 2 нед., с 01.05.2020 г. по 14.05.2020 г.

Руководитель практики от университета,

к. ф.-м. н. _____ С. В. Миронов

Руководитель практики от организации (учреждения, предприятия),

к. ф.-м. н. _____ С. В. Миронов

Тема практики:«Создание приложения для анализа генетического алгоритма поиска центральных вершин»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1 Описание задачи 5
2 Описание генетического алгоритма 6
2.1 Генерация начальной популяции 6
2.2 Этап мутации 6
2.3 Этап скрещивания 6
2.4 Этап естественного отбора 6
3 Описание приложения 7
4 Описание технологий и архитектуры приложения 12
4.1 Структура базы данных 12
4.2 Уровень доступа к данным 13
4.3 Уровень бизнес-логики 15
4.4 Уровень представления 16
4.4.1 Контроллеры 16
4.4.2 Модели данных и валидация 18
4.4.3 Аутентификация 20
4.5 Шифровка паролей 20
4.6 Внедрение зависимостей 21
5 Работа с генетическим алгоритмом 23
5.0.1 Чтение графа из файла 23
5.0.2 Запуск алгоритма на графах 23
5.0.3 Генетический алгоритм 24
ЗАКЛЮЧЕНИЕ 25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26
Приложение А Уровень доступа данных 28
Приложение Б Уровень бизнес-логики 32
Приложение В Уровень пользовательского интерфейса 38
Приложение Г Внедрение зависимостей 43
Приложение Д Генетический алгоритм 45
Приложение Е CD-диск с отчетом о выполненной работе 54

ВВЕДЕНИЕ

Зачастую задачи, хорошо исследованные с точки зрения теории не всегда могут быть оптимально решены в реальных условиях. Например, задача поиска центральных вершин в графах легко может быть решена при помощи классических алгоритмов [1–4]. Вместе с тем не всегда затраты по времени, которые эти алгоритмы выдают могут быть приемлемы. В связи с этим очень часто с использованием точных алгоритмов совместно могут разрабатываться и применяться эвристические подходы, в своей основе содержащие некоторую естественную идею, формализованную для решения той или иной задачи.

В частности к эвристическим алгоритмам относятся генетические алгоритмы, которые моделируют в своей работе эволюционный процесс за счет этапов мутации, скрещивания и естественного отбора. Сам процесс создания и адаптации генетического алгоритма для той или иной задачи не всегда представляет собой большую задачу для исследования. Как известно [5, 6], на работу любого генетического алгоритма ключевым образов влияют несколько факторов: параметр, отвечающий за вероятность мутации, параметр, связанный с вероятностью скрещивания и параметр, описывающий размер популяции. Именно подбор этих параметров для решения задачи может требовать большого изучения и временных затрат, как вычислительной техники, так и исследователя. Для задачи поиска центральных вершин был разработан генетический алгоритм, который необходимо проанализировать и выявить его сильные и слабые стороны, а так же требуется предоставить возможность его легко запускать с различными параметрами и на различных графах.

В связи с этим основной целью этой работы является создание приложения, позволяющего исследовать созданный алгоритм. Для достижения данной цели были поставлены следующие задачи:

- разработать способ загрузки и хранения графов,
- создать интерфейс, позволяющий пользователю запускать алгоритм,
- разработать систему регистрации для доступа пользователя к дополнительным возможностям приложения,
- создать гибкое программное решение, которое было бы легко изменять и использовать.

1 Описание задачи

В целом задачу поиска центральных вершин можно описать формулами и следующими словами. Существует невзвешенный неориентированный граф $G = (V, E)$, где V — множество вершин, E — множество ребер.

Тогда центральными вершинами будут являться те узлы, которые имеют минимальный эксцентриситет — длина кротчайшего пути до самой удаленной вершины в графе от заданной. Кроме этого вместе с центральными вершинами используется понятие радиуса графа — значение эксцентриситета на центральной вершине.

То есть всю задачу можно описать следующей формулой:

$$R = \min_{x \in V} \max_{y \in V} (d(x, y)),$$

где $d(x, y)$ — расстояние между вершинами x, y; R — радиус графа.

Так как граф невзвешенный, то длина кротчайшего пути определяется как число ребер в самом пути.

2 Описание генетического алгоритма

Генетический алгоритм в своей основе содержит итерационное выполнение генетических операторов преобразования, которые изменяют популяцию и выводят ее в некоторое оптимальное решение. Как уже говорилось ранее этими этапами является этап мутации, этап скрещивания и этап естественного отбора. Все эти этапы реализуются в зависимости от решаемой задачи поразному. При решении задачи поиска центральных вершин была разработана следующая реализация генетического алгоритма.

2.1 Генерация начальной популяции

В качестве начальной популяции выбирается случайный набор вершин графа. При этом этот размер указывается в качестве параметра N.

2.2 Этап мутации

При работе этапа мутации алгоритм проходится по всей популяции и для каждой вершины внутри популяции находятся ее соседние вершины, после чего с учетном заданного параметра мутации p_m из соседей выбирается один представитель, которые заменяет рассматриваемый элемент.

2.3 Этап скрещивания

Для работы оператора скрещивания был реализован следующий подход: из популяции выбирается два представителя, после чего между этими вершинами находится кротчайший путь и из этого пути извлекается одна из вершин, которая и представляет собой потомка для последующего поколения популяции. При этом оператор так же действует с учетом параметра, отвечающего за вероятность скрещивания p_c.

2.4 Этап естественного отбора

При реализации естественного отбора для каждой вершины в популяции в качестве функции оценки приспособленности «особи» выступает значение эксцентриситета. Для каждой вершины находится ее эксцентриситет после чего происходит вероятностный отбор, при этом приоритет для попадания в следующее поколение отдается вершинам с меньшим эксцентриситетом.

3 Описание приложения

Приложение представляет собой веб-сайт через который предоставляется возможность для работы с генетическим алгоритмом. С одной стороны у пользователя есть доступ к исследованию алгоритма и его запуску с различными параметрами, а с другой пользователь может заняться исследованием собственного графа и запустить алгоритм на нем.

При входе в приложение пользователь попадает на главную страницу см. рисунок 1.

Рисунок 1 – Главная страница

На главной странице пользователь видит небольшое описание алгоритма и главным образом навигационную панель, которая находится наверху страницы. На этой навигационной панели находятся ссылки на страницу с формой для поиска центральных вершин, страницу с формой для запуска алгоритма с различными параметрами и на различных графах, а так же ссылка для входа зарегистрированных пользователей.

После перехода на страницу входа (см. рисунок 2) пользователь может ввести свои логин и пароль и после успешной аутентификации он будет перенаправлен на главную страницу, при этом в навигационной панели появятся ссылки на страницу добавления новых пользователей, новых графов, а так же будет отображаться его логин (см. рисунок 3)

При переходе на страницу, с которой возможно загрузить граф для поиска

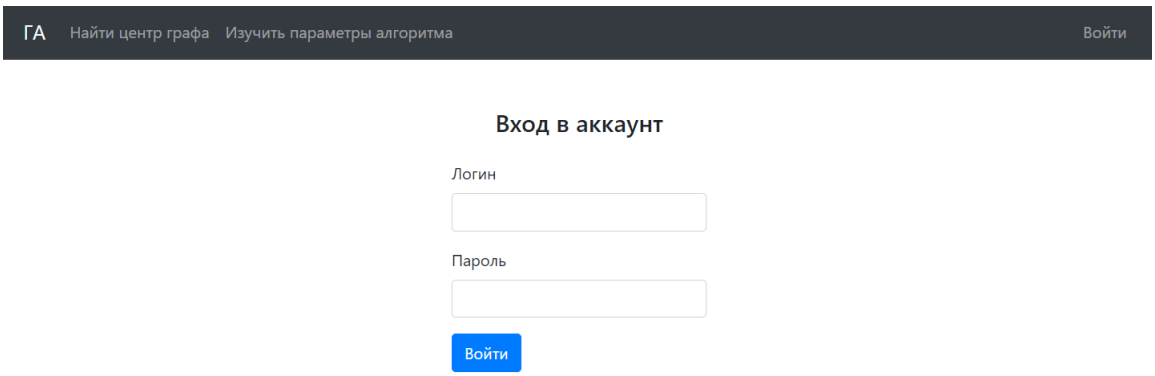

Рисунок 2 – Страница входа

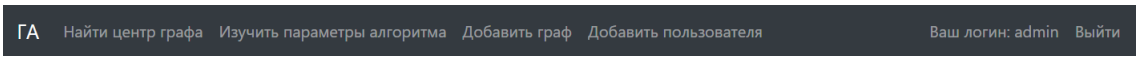

Рисунок 3 – Навигационная панель после входа в систему

радиуса, пользователю показывается форма, через которую загружается граф (см. рисунок 4).

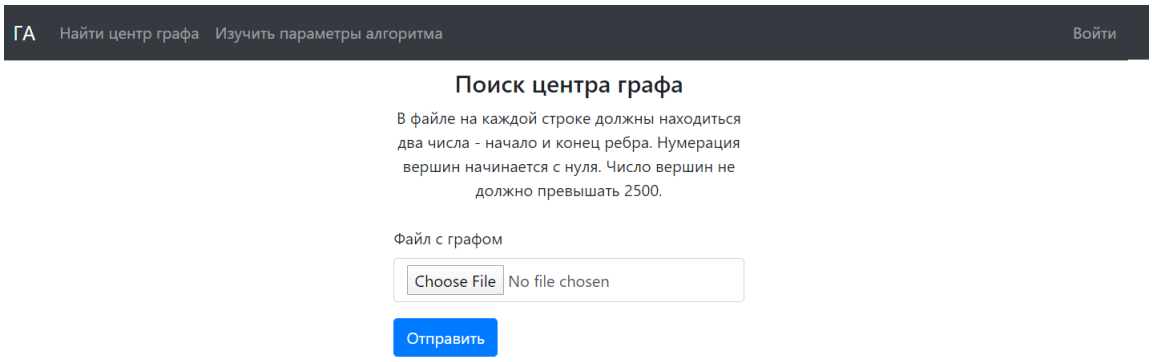

Рисунок 4 – Форма для загрузки графа с целью поиска радиуса

Похожая страница показывается пользователю при загрузке нового графа на сервер (см. рисунок 5).

Если пользователь не загрузит граф или загрузит файл в неверном формате, то получит страницу с сообщением об ошибке (см. рисунок 6, 7).

При верно загруженном файле пользователю открывается страница, на которой отображаются результаты запуска алгоритма: время его работы, полученный радиус графа и возможные центральные вершины (см. рисунок 8).

Кроме этого у пользователя есть возможность запускать генетический алгоритм с различными параметрами (см. рисунок 9).

На форме находятся несколько ползунков, через которые можно выставить основные параметры генетического алгоритма, а также там же находится

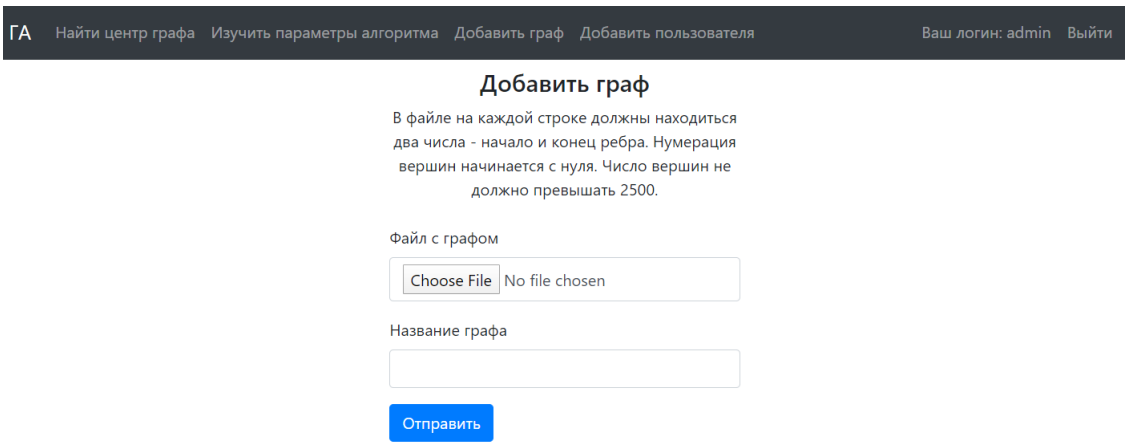

Рисунок 5 – Страница для добавления графа

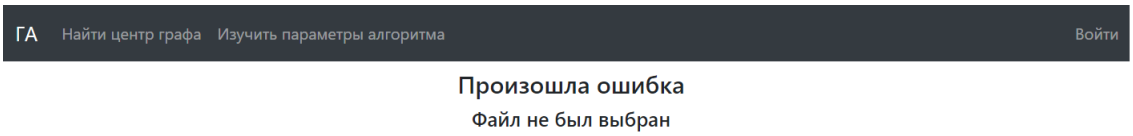

Рисунок 6 – Страница с сообщением об ошибке

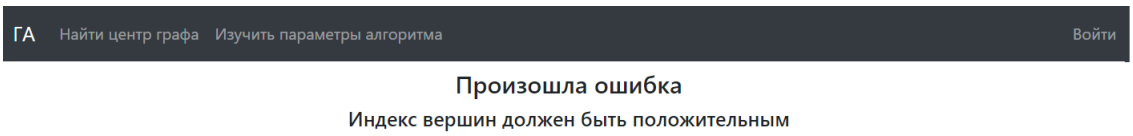

Рисунок 7 – Страница с сообщением об ошибке

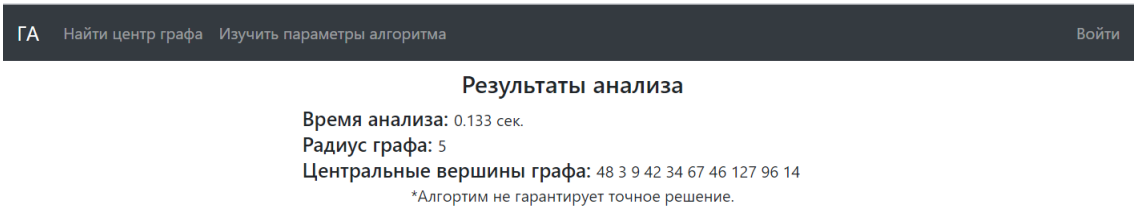

Рисунок 8 – Страница с результатами поиска центральных вершин

выпадающий список, в котором выбирается один из сохраненных графов. После того как пользователь выбрал соответствующие параметры и отправил форму на сервер с выбранными параметрами запускается алгоритм, причем несколько раз для того, чтобы получить эмпирическую оценку времени работы и процента неверных ответов. Именно эти результаты показываются на форме, которая показывается пользователю после работы алгоритма (см. рисунок 10).

Каждому зарегистрированному пользователю предоставляется возмож-

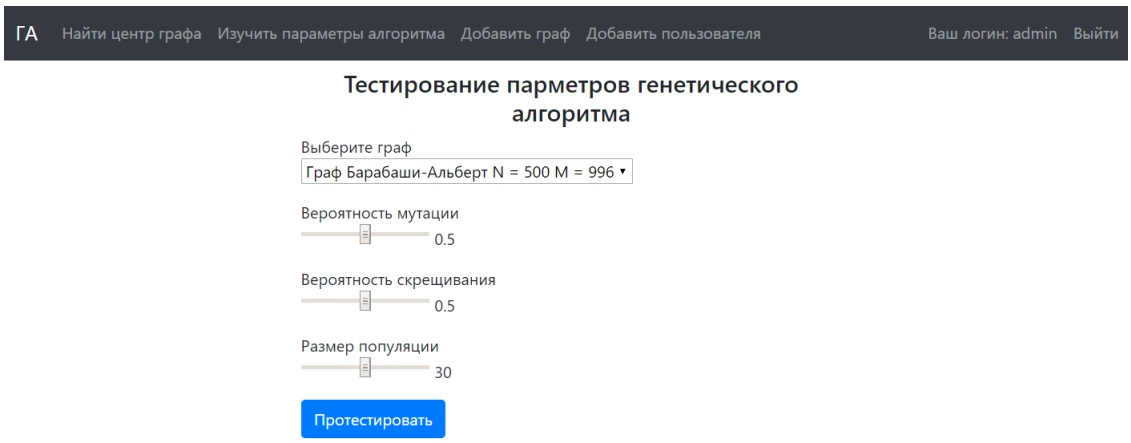

Рисунок 9 – Форма для исследования параметров алгоритма

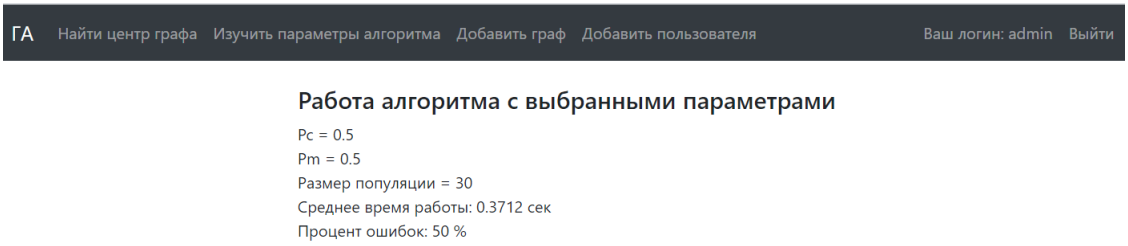

Рисунок 10 – Форма с результатами исследования параметров алгоритма

ность для добавления новых пользователей через соответствующую форму (см. рисунок 11). Данные которые вводит пользователь выдвигаются следующие требования: длина пароля и логина должна быть от 5 до 50 символов и пользователь с таким же логином не должен уже существовать в базе данных. Если эти условия не будут выполнены пользователь получит соответствующее сообщение об ошибке (см. рисунок 12).

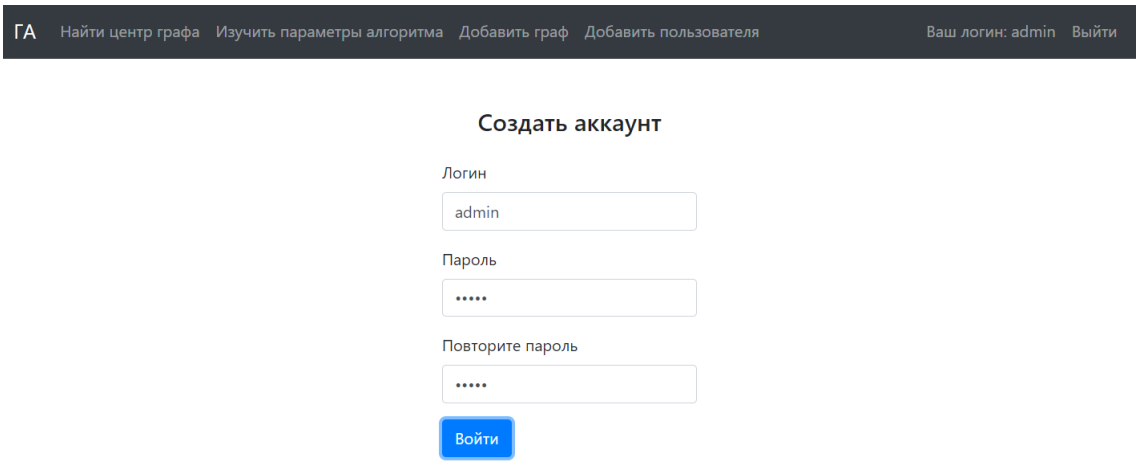

Рисунок 11 – Форма для регистрации нового пользователя

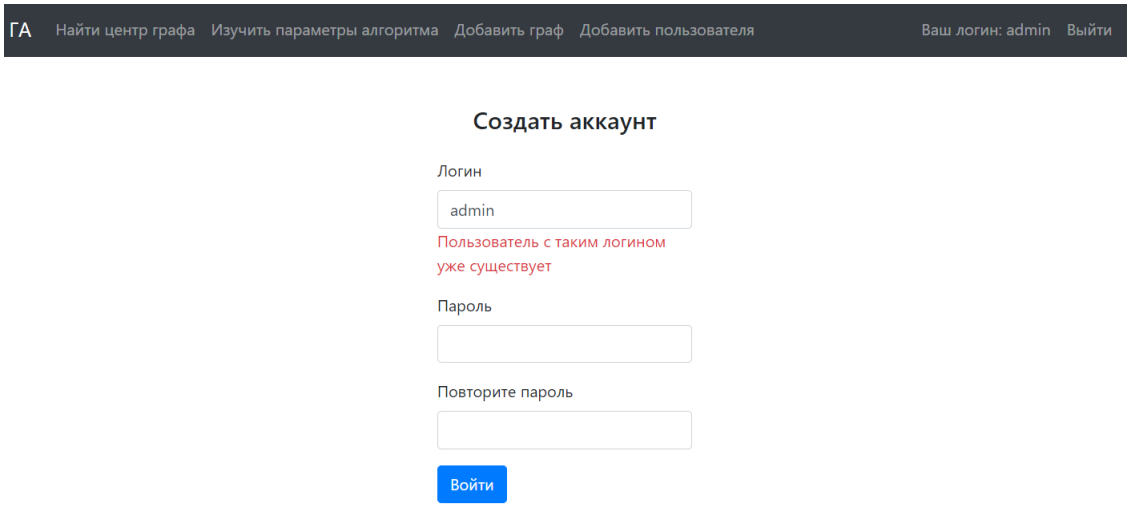

Рисунок 12 – Форма с ошибкой при неверных данных для регистрации пользователя

4 Описание технологий и архитектуры приложения

В качестве языка программирования создания проекта был выбран объектно-ориентированный язык C#. Вместе с тем для создания клиент-серверного приложения был использован фреймворк `ASP.NET MVC 5` [7, 8], который позволяет создавать веб-приложения с использованием архитектуры `MVC`. Кроме этого в качестве системы объектно-реляционного отображения используется технология `Entity Framework 6` [9]. При этом приложение разделено на три слоя абстракции — уровень доступа к данным, уровень бизнес-логики и уровень визуального представления.

4.1 Структура базы данных

Для создания базы данных используется технология `Entity Framework 6`, вместе с чем используется подход `Code-first`, согласно которому были созданы классы `Graph`, `GraphInfo`, `Edge`, `User`, описывающие модели данных:

```
public class GraphInfo
{
        public int Id { get; set; }
        public int N { get; set; }
        public int M { get; set; }
        public string Name { get; set; }
        public int R { get; set; }
}
```

```
public class Graph : GraphInfo
{
        public ICollection<Edge> Edges { get; set; }
        public Graph()
        {
                Edges = new List<Edge>();
        }
}
```

```
public class Edge
{
        public int Id { get; set; }
        public int V1 { get; set; }
        public int V2 { get; set; }
}
```

После чего фреймворк на основании созданных моделей создал структуру базы данных и связи между таблицами.

Хранение графов в базе данных были созданы следующие таблицы: `Graphs` и `Edges`. При этом таблица `Graphs` содержит следующие поля:

— поле `Id` (типа данных `INT`) — уникальный идентификатор, внутренний ключ,
— поле `N` (типа данных `INT`) — количество вершин в графе,
— поле `M` (типа данных `INT`) — количество ребер в графе,
— поле `Name` (типа данных `NVARCHAR`) — название графа,
— поле `R` (типа данных `INT`) — радиус графа.

Кроме этого таблица `Edges` состоит из следующих полей:

— поле `Id` (типа данных `INT`) — уникальный идентификатор, внутренний ключ,
— поле `V1` (типа данных `INT`) — одна из вершин, которые соединяет ребро,
— поле `V1` (типа данных `INT`) — вторая из вершин, которые соединяет ребро,
— поле `Graph_Id` (типа данных `INT`) — `Id` графа, которому принадлежит ребро, внешний ключ.

Кроме этого для хранения зарегистрированных пользователей существует таблица `Users`:

— поле `Id` (типа данных `INT`) — уникальный идентификатор, внутренний ключ,
— поле `Login` (типа данных `NVARCHAR`) — логин пользователя,
— поле `Password` (типа данных `VARBINARY`) — зашифрованный пароль пользователя.

Полная диаграмма таблиц представлена на рисунке 13:

4.2 Уровень доступа к данным

Для гибкой и стандартизированной работы с базой данных был создан ряд интерфейсов, в которые были вынесены основные методы для доступа к данным и их изменениям. Классы, реализующие эти интерфейсы представляют собой уровень доступа к данным, при этом использование интерфейсов позволяет с легкостью изменять реализацию этих классов, а также упрощает процесс тестирования.

Далее приводится код основных интерфейсов, которые используются при работе с данными:

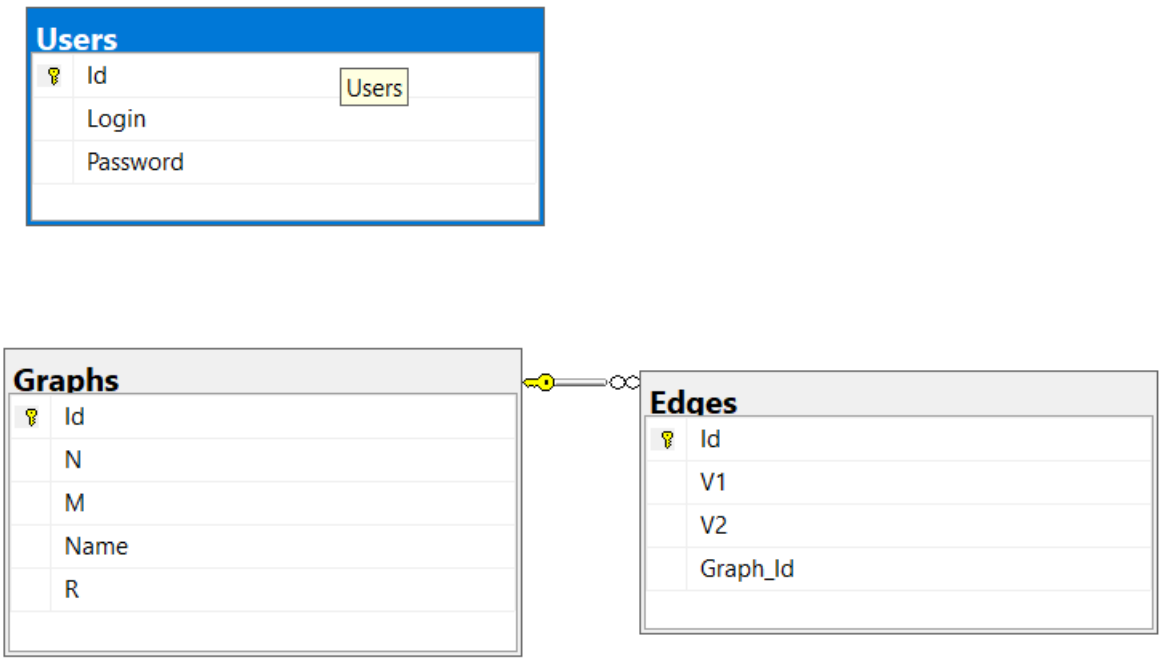

Рисунок 13 – Диаграмма базы данных

```
public interface IGraphDao
{
        IEnumerable<GraphInfo> GetAllGraphInfo();
        Graph GetById(int id);
        Graph Add(Graph graph);
}
```

```
public interface IUserDao
{
        User GetById(int id);
        User GetByName(string name);
        User Add(User user);
}
```

Вместе с тем для использования технологии Entity Framework созданы классы GraphContext и UserContext, которые наследуются от класса System.Data.Entity.DbContext, что позволяет получить возможность для легкого доступа к базе данных без написания SQL запросов. Классы GraphContext и UserContext содержат в себе поля типа DbSet<Graph> и DbSet<User>, через которые происходит добавление, чтение или изменение данных в базе данных. Кроме этого в этих классах описан статический конструктор, внутри которого указан способ начальной инициализации базы данных, за счет

классов UserContextInitializer и GraphContextInitializer. Эти классы наследуются от класса CreateDatabaseIfNotExists, что позволяет фреймворку выполнить начальное заполнение данными, если база данных еще не существует, за счет кода, который описан в переопределенном методе Seed. Внутри этого метода происходит чтение нескольких созданных графов из текстовых файлов, все это происходит внутри метода, описанного в классе GraphContextInitializer, в этом же методе в классе UserContextInitializer добавляется пользователь с логином admin и паролем admin. Полный код представлен в приложении А.

4.3 Уровень бизнес-логики

Уровень бизнес-логики представляет собой похожую структуру, как и уровень доступа данных — так же созданы ряд интерфейсов и классы, которые их реализуют. При этом многие из этих интерфейсов похожи на те, которые описаны в уровне доступа к данным, однако именно на этом уровне происходит запуск генетического алгоритма с различными параметрами и анализ загруженных графов. С определением интерфейсов и классов реализующих бизнес-логику приложения можно ознакомиться в приложении Б. Классы GraphBL и UserBL, реализуют интерфейсы IGraphBL и IUserBL. Они содержат ссылки на объекты, реализующие интерфейсы IGraphDao и IUserBL, при этом эти объекты передаются в качестве параметров в соответствующие конструкторы. При добавлении нового пользователя происходит проверка на существование записи с таким же логином, а кроме этого происходит шифровка пароля.

Также при добавлении нового графа в базу данных в классе, который отвечает за работу с моделью графа, происходит проверка графа на связность и его размеры. При неудачном прохождении проверки выбрасывается исключение, которое отлавливается на уровне представления.

Кроме этого для работы с генетическим алгоритмом создан интерфейс IAlgorithm:

```
1 public interface IAlgorithm
2     {
3         FindingVertexResponse FindCentralVertex(Graph graph);
4         ResearchAlgorithmResponse ResearchAlgorithm(ResearchRequest param);
5     }
```

Интерфейс определяет набор методов, в которых будет реализована логика для работы с генетическим алгоритмом. Вместе с тем класс `Algorithm` реализует данный интерфейс и в нем содержится вся логика работы с алгоритмом — получение результатов поиска центральных вершин, замеры времени работы и процента неверно найденных решений.

Результаты измерений возвращаются из методов при помощи классов `FindingVertexResponse` и `ResearchAlgorithmResponse`:

```
public class FindingVertexResponse
{
        public int[] Center { get; set; }
        public int R { get; set; }
        public double Time { get; set; }
}
```

```
public class ResearchAlgorithmResponse
    {
        public double AvgTime { get; set; }
        public double Error { get; set; }
    }
```

Полный код классов уровня бизнес-логики можно увидеть в приложении Б.

4.4 Уровень представления

Так как проект представляет собой веб-приложение, то уровень, отвечающий за пользовательский интерфейс реализован при помощи технологии `ASP .NET MVC 5`. В связи с чем весь код этого уровня разделен на:

- контроллеры, которые отвечают на HTTP запросы клиента с помощью представлений,
- представления, которые написаны с использованием технологии Razor, позволяющей внедрять серверный C# код,
- модели данных, внутри которых происходит передача данных от клиента серверу и обратно.

4.4.1 Контроллеры

Для взаимодействия с клиентской частью приложения и обработки пользовательских данных было создано несколько контроллеров: `HomeController`, `GraphController`, `LoginController`, `ResearchController`.

Класс `HomeController` содержит один метод `Index`, который отвечает на GET-запрос и возвращает домашнюю страницу.

Класс `GraphController` включает в себя методы, определяющие URL-адреса при взаимодействии с которыми клиентской части приложения предоставляется возможность запускать генетический алгоритм для поиска центральных вершин или добавлять новый граф в базу данных. В целом данный контроллер ответственен за обработку следующих запросов:

— `/Graph` — обрабатывает GET-запрос возвращает загрузочную страницу для поиска центральных вершин,

— `/Graph/FindCentralVertex` — обрабатывает POST-запрос, в котором передается файл с графом, после чего запускается генетический алгоритм. В качестве результата возвращается страница с сообщением об ошибке, которая могла произойти при обработке запроса из-за неверного формата данных в файле с графом, или слишком большого размера загружаемого графа, либо же при успешном запуске возвращает страницу с результатами работы — радиус графа и возможные центральные вершины,

— `/Graph/Add` — обрабатывает GET-запрос, который возвращает форму для загрузки графа в базу данных,

— `/Graph/Add` — обрабатывает POST-запрос, в котором пользователь предает на сервер для сохранения файл с графом и название графа. При успешном добавлении перенаправляет пользователя на домашнюю страницу, при возможной ошибке — страницу с описанием ошибки.

Класс `ResearchController` содержит методы через которые пользователю предоставляется возможность запускать генетический алгоритм с различными параметрами (p_c, p_m, N), а также граф на котором будет тестироваться алгоритм. Контроллер содержит следующие методы:

— `/Research/Index` — обрабатывает GET-запрос и возвращает форму с выбором параметров для генетического алгоритма, при этом в результат подгружаются описания графов, сохраненных в базе данных,

— `/Research/ResearchAlgorithm` — обрабатывает POST-запрос в который передаются выбранные параметры алгоритма и выбранный граф.

В классе `LoginController` определены методы, за счет которых происходит регистрация и аутентификация пользователей. Пользователь, который вошел в систему получает возможность для добавления новых графов и добав-

ление новых пользователей, по сути залогиненному пользователю предоставляются права администратора. В контроллере `LoginController` реализованы следующие методы:

— `/Login/Index` — обрабатывает GET-запрос и возвращает форму заполнения для входа в систему,

— `/Research/SignIn` — обрабатывает POST-запрос в который передается логин и пароль. В этом методе происходит валидация введенных данных, проверка принадлежности пароля введенному пользователю и при успешном прохождении пользователю отправляется набор cookie-данных, вследствие чего происходит аутентификация пользователя. При неверном логине или пароле пользователю возвращается сообщение об ошибке, при успешном прохождении аутентификации — метод возвращает переадресацию на домашнюю страницу,

— `/Login/SignUp` — обрабатывает GET-запрос и возвращает форму для регистрации нового пользователя,

— `/Login/SignUp` — обрабатывает POST-запрос и добавляет нового пользователя при успешном прохождении валидации и отсутствии пользователя с таким же логином.

Полный код контроллеров можно увидеть в приложении В.

4.4.2 Модели данных и валидация

Основными классами, с помощью объектов которых происходит передача данным в контроллеры, `AddGraphRequest`, `CreateUserRequest`, `LoginUserRequest`, `ResearchRequest`, при этом результаты вычислений возвращаются из контроллеров в виде представлений, которые представляют собой HTML разметку с внедрением данных переданных через классы `AlgorithmResultResponse`, `FindingVertexResponse`, `ResearchAlgorithmResponse`.

При этом очевидно, что при этом введенные пользовательские данные должны удовлетворять некоторым условиям. Для того, чтобы передаваемые данные можно было проверить из любого участка кода для некоторых свойств были использованы атрибуты валидации [10] `Required`, `Compare`, `StringLength`:

```
1 public class CreateUserRequest
2     {
```

```
        [Required(ErrorMessage = "Введите логин")]
        [StringLength(50, MinimumLength = 3,
        ErrorMessage = "Длина логина должна быть от 3 до 50 символов")]
        public string Login { get; set; }

        [Required(ErrorMessage = "Введите пароль")]
        [StringLength(50, MinimumLength = 3,
        ErrorMessage = "Длина логина должна быть от 3 до 50 символов")]
        public string Password { get; set; }

        [Required(ErrorMessage = "Повторите пароль")]
        [Compare("Password", ErrorMessage = "Пароли не совпадают")]
        public string ConfirmPassword { get; set; }
    }
```

```
public class LoginUserRequest
    {
        [Required(ErrorMessage = "Введите логин")]
        public string Login { get; set; }

        [Required(ErrorMessage = "Введите пароль")]
        public string Password { get; set; }
    }
```

При таком использовании атрибутов валидации проверить модель на соответствие выдвинутым требованиям можно при помощи следующего кода:

```
if (ModelState.IsValid) {
        ...
}
```

внутри любого из контроллеров, где `ModelState` — свойство класса `Controller`, которое инкапсулирует состояние модели, переданной в качестве параметра запроса. В случае неудачного прохождения валидации в свойство `ModelState` при помощи метода `AddModelError` добавляется сообщение об ошибке, которое затем будет вставлено в HTML разметку. Атрибут `Requred` установлен для логина и пароля, вводимого пользователем, что гарантирует тот факт, что в базу данных не будет помещена запись с пустыми полями. В добавок к этому у свойства `ConfirmPassword` установлен атрибут `Compare`, который требует, чтобы свойство, отвечающее за хранение пароля, совпадало с свойством, отвечающим за хранение повтора пароля. Также используется атрибут

StringLength, в котором устанавливаются минимальная и максимальная длина логина и пароля.

4.4.3 Аутентификация

Как уже отмечалось ранее доступ к возможности добавлять графы в базу данных и добавлять туда же новых пользователей имеют доступ только пользователи, которые вошли в систему. В связи с этим в качестве технологии аутентификации в созданном приложении используется аутентификация с помощью форм [11]. Для ее включения в файл Web.config были добавлены следующие строки:

```
<authentication mode="Forms">
      <forms loginUrl="~/login" timeout="60" />
</authentication>
```

в которых указывается по какому адресу будет отправлен пользователь в случае, если он не имеет прав доступа к запрошенным ресурсам и время действия cookieфайлов. При успешном прохождении проверки на принадлежность пользователю введенного им пароля при помощи следующей строки кода клиентская часть получает cookie-файлы, которые затем будут присоединяться ко всем остальным запросам:

```
FormsAuthentication.SetAuthCookie(user.Login, true);
```

Для того, чтобы к определенным методам был доступ только аутентифицированным пользователям к каждому методу применяется атрибут Authorize, который гарантирует проверку на доступность для пользователя этих методов. При этом для пользователя вошедшего в систему несколько изменяется HTML разметка, что достигается при помощи использования свойства User.Identity.IsAuthenticated.

4.5 Шифровка паролей

Очевидно, что хранение паролей пользователей в открытом виде представляет собой подход нарушающий основы требования к безопасности приложения. В связи с чем каждый пароль при регистрации пользователя хешируется и полученный хеш сохраняется в базе данных. Хеширование паролей происходит в классе Encryption (см. приложение Б), где определены публичные

методы `CreatePassword` и `CheckPassword`. Создание хеша пароля происходит при помощи объекта класса `Rfc2898DeriveBytes` [12]:

```
var pbkdf2 = new Rfc2898DeriveBytes(password, salt, iterations);
byte[] hash = pbkdf2.GetBytes(hashSize);
```

В конструктор класса передается строка с паролем, «соль» — псевдослучайная последовательность байт, которая используется для повышения криптоустойчивости хеша и параметр, отвечающий за искусственную временную задержку, которая позволяет избежать попытки грубого перебора. В базу данных сохраняется полученный хеш и сгенерированная «соль». При проверки подлинности пароля из базы данных извлекается хеш с «солью», после чего введенный пароль хешируется с сохраненной «солью» и результат сравнивается с тем, что было сохранено в базе данных.

4.6 Внедрение зависимостей

Ранее описывались независимые уровни, на которые разделено приложение, при этом гибкость и заменяемость каждого из уровней гарантируется описанием интерфейсов. Каждый из уровней содержит ссылки на объекты, которые реализуют тот или иной интерфейс при этом эти объекты передаются в качестве параметров в конструкторы. Для того, чтобы гарантировать тот факт, что во все конструкторы будут переданы одни и те же реализации интерфейсов и избежать дублирования кода в созданном приложении используется IoC-контейнер `Ninject`, который связывает интерфейсы с объектами, которые их реализуют и предоставлять их при необходимости. Связывание интерфейсов и реализации происходит в методе класса `NinjectRegistrations`:

```
public class NinjectRegistrations : NinjectModule
    {
        public override void Load()
        {
            Bind<IUserDao>().To<UserDao>();
            Bind<IGraphDao>().To<GraphDao>();
            Bind<IAlgorithm>().To<Algorithm>();
            Bind<IGraphBL>().To<GraphBL>();
            Bind<IUserBL>().To<UserBL>();
        }
    }
```

При этом в глобальном файле запуска приложения происходит регистрация этого класса в качестве основного способа разрешения зависимостей (см. приложение Г).

5 Работа с генетическим алгоритмом

Для изолированного доступа к методам генетического алгоритма и для корректного измерения временных затрат, а так же для корректного чтения графов из текстовых файлов были созданы следующие классы: `ExactAlgorithmCore`, `GeneticAlgorithmCore`, `GraphContext`, `GraphParser`.

5.0.1 Чтение графа из файла

Ранее говорилось о том, что графы передаются на сервер в текстовых файлах, при этом формат данных этого файла должен быть следующим: на каждой строке указывается ровно два числа — начало и конец ребра, при этом нумерация вершин должна начинаться с 0 и идти по порядку, т. е. если в графе N вершин, то максимальным числом в файле должно быть число $N-1$ и встречаться все числа от 0 до $N-1$. Кроме этого в файле не должно встречаться отрицательных чисел. С целью гарантированно принимать только файлы с корректными графами был создан статический класс `GraphParser`, в котором реализован статический метод `ParseTxtFormat`. Метод принимает массив строк, читает их и создает объект класса `Graph`, который возвращается в качестве результата. При этом если формат файла не удовлетворяет одному из описанных ранее требований метод выбрасывает исключение с сообщением о неверном формате данных, исключение отлавливается на уровне пользовательского интерфейса.

5.0.2 Запуск алгоритма на графах

При работе генетического алгоритма после запуска обхода в ширину такие результаты, как расстояние между вершинами и найденные кротчайшие пути сохраняются до окончания работы алгоритма, в связи с чем был создан класс `GraphContext`, который инкапсулирует структуры данных, хранящие описанные сведения, появляющиеся в процессе работы. Класс позволяет найти эксцентриситет вершин при помощи алгоритма поиска в ширину, а также позволяет проверять граф на связность с помощью этого же алгоритма. Расстояния, найденные в ходе процесса определения эксцентриситета хранятся в матрице $N \times N$, а кроме этого для хранения кротчайших путей используется такая же матрица, где каждым элементом является список вершин в пути. С полным кодом описанного класса можно ознакомиться в приложении Д.

5.0.3 Генетический алгоритм

Сам генетический алгоритм реализован в классе `GeneticAlgorithmCore`. При создании объекта этого класса в конструктор передаются такие параметры, как объект типа `Graph`, значение параметра для уровня вероятности мутации, параметр для уровня вероятности скрещивания и размер популяции. После того, как объект этого класса создан у него можно вызвать метод `StartAlgorithm`:

```
public FindingVertexResponse StartAlgorithm() {
        Init();
        _watch.Start();
        for (int i = 0; i < _step; i++) {
                EvolutionStep();
        }
        _watch.Stop();

        FindingVertexResponse res = new FindingVertexResponse() {
                Time = _watch.ElapsedMilliseconds / (double)1000,
        };
        GetBestResult(res);
        return res;
}
```

в котором происходит вызов метода ответственного за начальную инициализацию популяции генетического алгоритма, а также за создание нового объекта класса `GraphContext`. После чего итерационно запускаются процессы мутации, скрещивания и естественного отбора. В качестве результата возвращается время работы алгоритма и найденные центральные вершины.

Кроме этого при добавлении нового графа необходимо выяснить его реальный радиус и с этой целью создан класс `ExactAlgorithmCore`, который запускает алгоритм обхода в ширину, из каждой вершины, что позволяет гарантированно найти точный радиус графа.

Полноценный код всего приложения представлен на CD-диске E.

ЗАКЛЮЧЕНИЕ

В рамках выполнения практики была достигнута поставленная цель — создано веб-приложение, за счет которого существует возможность работать с генетическим алгоритмом поиска центральных вершин. При этом приложение позволяет исследовать генетический алгоритм, выявлять его сильные и слабые стороны.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Chan, T. M.* All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time / T. M. Chan // *ACM Trans. Algorithms*. — oct 2012. — Vol. 8, no. 4. — Pp. 34:1–34:17.

2 *Berman, P.* Faster approximation of distances in graphs // Algorithms and Data Structures / Ed. by F. Dehne, J.-R. Sack, N. Zeh. — Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

3 *Roditty, L.* Fast approximation algorithms for the diameter and radius of sparse graphs // Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing. — STOC '13. — New York, NY, USA: ACM, 2013. — Pp. 515–524.

4 *Aingworth, D.* Fast estimation of diameter and shortest paths (without matrix multiplication) / D. Aingworth, C. Chekuri, P. Indyk, R. Motwani // *SIAM J. Comput.* — 1999. — Vol. 28, no. 1. — Pp. 1167–1181.

5 *Holland, J.* Adaption in Natural and Artificial Systems Adaption in Natural and Artificial Systems / J. Holland. — University of Michigan Press, 1975.

6 *Панченко, Т.* Генетические алгоритмы / Т. Панченко. — Астрахань: Издательский дом «Астраханский университет», 2007.

7 *Фримен, А.* ASP.NET MVC 5 Framework с примерами на C# для профессионалов / А. Фримен. — Вильямс, 2015.

8 Начало работы с ASP.NET MVC 5 [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/aspnet/mvc/overview/getting-started/introduction/getting-started (Дата обращения 24.05.2020). Загл. с экр. Яз. рус.

9 Обзор Entity Framework 6 [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/ef/ef6/ (Дата обращения 24.05.2020). Загл. с экр. Яз. рус.

10 System.ComponentModel.DataAnnotations Пространство имен [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.componentmodel.dataannotations?view=netcore-3.1 (Дата обращения 24.05.2020). Загл. с экр. Яз. рус.

11 Класс FormsAuthentication [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.web.security.formsauthentication?view=netframework-4.8 (Дата обращения 24.05.2020). Загл. с экр. Яз. рус.

12 Rfc2898DeriveBytes Класс [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=netcore-3.1 (Дата обращения 24.05.2020). Загл. с экр. Яз. рус.

ПРИЛОЖЕНИЕ А

Уровень доступа данных

Далее приводится программный код основных файлов, описывающих уровень доступа к данным.

```
namespace GeneticAlgorithmWEB.Dao
{
    // Класс отвечает за предоставление
    // доступа к коллекции объектов графов.
    public class GraphContext : DbContext
    {
        static GraphContext() {
            // Установка класса, за счет которого происходит
            // начальная инициализация базы данных графами.
            Database.SetInitializer(new GraphContextInitializer());
        }
        public GraphContext() : base("DB") { }
        // Набор графов из базы данных.
        public DbSet<Graph> Graphs { get; set; }
    }
}
```

```
namespace GeneticAlgorithmWEB.Dao
{
    // Класс реализующий уровень доступа к данным.
    // В классе определены основные методы для
    // получения графов из базы данных.
    public class GraphDao : IGraphDao
    {
        // Сохраняет новый граф в базе данных.
        public Graph Add(Graph graph)
        {
            using (GraphContext context = new GraphContext())
            {
                Graph res = context.Graphs.Add(graph);
                context.SaveChanges();
                return res;
            }
        }

        // Получение описательной информации обо всех графах в базе данных.
        public IEnumerable<GraphInfo> GetAllGraphInfo()
```

```
        {
            List<GraphInfo> res = new List<GraphInfo>();
            using (GraphContext context = new GraphContext())
            {
                foreach (var graph in context.Graphs)
                {
                    res.Add(new GraphInfo()
                    {
                        Id = graph.Id,
                        N = graph.N,
                        M = graph.M,
                        Name = graph.Name,
                    });
                }
                return res;
            }
        }

        // Получение графа по Id.
        // Выбрасывает исключение,
        // если граф с переданным Id не удалось найти в БД.
        public Graph GetById(int id)
        {
            using (GraphContext context = new GraphContext())
            {
                // Поиск графа при помощи LINQ запроса к контексту с графами.
                Graph res = context.Graphs
                    .Where(g => g.Id == id)
                    .Include(g => g.Edges)
                    .FirstOrDefault();
                if (res == null)
                {
                    throw new ArgumentException($"Invalid graph id = {id}");
                }
                return res;
            }
        }
    }
}

namespace GeneticAlgorithmWEB.Dao
{
```

```
    public class UserContext : DbContext
    {
        // Класс отвечает за предоставление
        // доступа к коллекции объектов пользователей.
        static UserContext() {
            // Установка класса, за счет которого происходит
            // начальная инициализация базы данных пользователями.
            Database.SetInitializer(new UserContextInitializer());
        }
        public UserContext() : base("DB") {}
        // Набор графов из базы данных.
        public DbSet<User> Users { get; set; }
    }
}
```

```
namespace GeneticAlgorithmWEB.Dao
{
    class UserContextInitializer : CreateDatabaseIfNotExists<UserContext>
    {
        // Инициализация базы данных одним пользователем.
        protected override void Seed(UserContext context)
        {
            CreateUserRequest userRequest = new CreateUserRequest() {
                Login = "admin",
                Password = "admin"
            };
            // Шифровка пароля пользователя
            Encryption encryption = new Encryption();
            User user = new User() {
                Login = userRequest.Login,
                Password = encryption.CreatePassword(userRequest.Password),
            };
            context.Users.Add(user);
            context.SaveChanges();
        }
    }
}
```

```
namespace GeneticAlgorithmWEB.Dao
{
    // Класс реализующий уровень доступа к данным.
    // В классе определены основные методы для
    // получения пользователей из базы данных.
```

```
6       public  class  UserDao : IUserDao
7       {
8           // Сохраняет нового пользователя в базе данных.
9           public  User  Add(User user)
10          {
11              using  (UserContext  context  = new UserContext())
12              {
13                  User  res  =  context . Users . Add(user);
14                  context . SaveChanges();
15                  return  res ;
16              }
17          }
18          // Получение пользователя по Id
19          public  User GetById(int  id )
20          {
21              using  (UserContext  context  = new UserContext())
22              {
23                  return  context . Users . Where(u => u.Id  == id) . FirstOrDefault () ;
24              }
25          }
26          // Получение пользователя по логину
27          public  User GetByLogin(string  name)
28          {
29              using  (UserContext  context  = new UserContext())
30              {
31                  return  context . Users . Where(u => u.Login  == name). FirstOrDefault () ;
32              }
33          }
34      }
35  }
```

ПРИЛОЖЕНИЕ Б

Уровень бизнес-логики

Далее приводится программный код, за счет которого происходит реализация уровня бизнес-логики.

```
1  namespace GeneticAlgorithmWEB.BLL
2  {
3      public class Algorithm : IAlgorithm
4      {
5          private readonly IGraphBL _graphBL;
6          // Фиксированные параметры для работы генетического алгоритма.
7          private double _fixedPM = 0.4;
8          private double _fixedPC = 0.4;
9          private int _fixedPopSize = 30;
10         // Число запусков при тестировании алгоритма с заданными параметрами.
11         private int _testCount = 10;
12
13         public Algorithm(IGraphBL graphBL)
14         {
15             _graphBL = graphBL;
16         }
17         // Поиск центральных вершин при помощи генетического алгоритма.
18         // Алгоритм запускается с фиксированными параметрами мутации, скрещивания
               и размер популяции.
19         public FindingVertexResponse FindCentralVertex(Graph graph)
20         {
21             GeneticAlgorithmCore ga = new GeneticAlgorithmCore(graph, _fixedPopSize,
                   _fixedPM, _fixedPC);
22             return ga.StartAlgorithm();
23         }
24         // Иссдедование алгоритма с переданными параметрами.
25         public ResearchAlgorithmResponse ResearchAlgorithm(ResearchRequest param) {
26             Graph graph = _graphBL.GetById(param.GraphId);
27
28             double avgTime = 0.0;
29             int error = 0;
30             // Многократный запуск алгоритма для оценки времени работы и процента
                   ошибок.
31             GeneticAlgorithmCore ga = new GeneticAlgorithmCore(graph, param.
                   PopulationSize, param.Pm, param.Pc);
32             for (int i = 0; i < _testCount; i++)
```

```
33                {
34                    FindingVertexResponse algResult = ga.StartAlgorithm();
35                    if (algResult.R != graph.R) {
36                        error++;
37                    }
38                    avgTime += algResult.Time;
39                }
40                return new ResearchAlgorithmResponse() {
41                    AvgTime = avgTime / _testCount,
42                    Error = error / (double)_testCount * 100.0,
43                };
44            }
45        }
46    }
```

```
1   namespace GeneticAlgorithmWEB.BLL
2   {
3       // Класс отвечающий за хеширование паролей и их проверку.
4       public class Encryption
5       {
6           // Размер соли.
7           private readonly int saltSize = 12;
8           // Размер хеша.
9           private readonly int hashSize = 20;
10          // Число итераций для задержки.
11          private readonly int iterations = 10000;
12
13          // Создание хеша пароля.
14          // Результат представляет собой последовательно
15          // записанные соль и хеш пароля в виде массива байтов.
16          public byte[] CreatePassword(string password)
17          {
18              // Создание "соли" для пароля
19              byte[] salt = new byte[saltSize];
20              new RNGCryptoServiceProvider().GetBytes(salt);
21              // Хеширование пароля с солью
22              return HashPassword(password, salt);
23          }
24
25          private byte[] HashPassword(string password, byte[] salt)
26          {
27              // Генерация хеша.
```

```
28              var pbkdf2 = new Rfc2898DeriveBytes(password, salt, iterations);
29              // Выбор из хеша заданного числа байтов.
30              byte[] hash = pbkdf2.GetBytes(hashSize);
31              // Выделение памяти под результат.
32              byte[] result = new byte[saltSize + hashSize];
33              // КОпирование соли и хеша в единый массив данных.
34              Array.Copy(salt, 0, result, 0, saltSize);
35              Array.Copy(hash, 0, result, saltSize, hashSize);
36              return result;
37          }
38
39          // Проверка верности пароля.
40          // Передаются байты из базы данных и пароль для проверки.
41          public bool CheckPassword(byte[] realPassword, string check) {
42              // Выделение соли из массива байтов.
43              byte[] salt = GetSaltFromHash(realPassword);
44              // Хеширование пароля с сохраненной солью
45              byte[] hashedPassword = HashPassword(check, salt);
46              if (hashedPassword.Length != realPassword.Length) {
47                  throw new ArgumentException("Длина реального хеша не совпадет с длиной
                        хеша из базы данных");
48              }
49              // ПОбайтовая проверка на соответствие пароля и хеша.
50              for (int i = 0; i < hashedPassword.Length; i++) {
51                  if (hashedPassword[i] != realPassword[i]) {
52                      return false;
53                  }
54              }
55              return true;
56          }
57          // Выделение хеша из массива байтов.
58          private byte[] GetSaltFromHash(byte[] hash)
59          {
60              byte[] salt = new byte[saltSize];
61              Array.Copy(hash, 0, salt, 0, saltSize);
62              return salt;
63          }
64
65      }
66  }
```

```
1   namespace GeneticAlgorithmWEB.BLL
```

```
{
    // Уровень бизнес–логики.
    // Реализут работу с графами.
    public class GraphBL : IGraphBL
    {
        private readonly IGraphDao _graphDao;
        private readonly int _maxN = 2500;

        public GraphBL(IGraphDao graphDao)
        {
            _graphDao = graphDao;
        }

        // Добавление графа в базу данных.
        // Выбрасывается исключение, если граф превышает максимальные размеры или
        // не является связным.
        public Graph Add(Graph graph)
        {
            // Проверка графа на максимальный размер.
            if (graph.N > _maxN) {
                throw new FormatException($"Количество вершин в графе должно быть
                    меньше, чем {_maxN}");
            }
            GraphContext context = new GraphContext(graph);
            // Проверка на связность.
            if (!context.CheckConnectivity())
            {
                throw new FormatException("Граф должен быть связаным");
            }
            ExactAlgorithmCore exactAlgorithm = new ExactAlgorithmCore();
            int R = exactAlgorithm.FindRadius(context);
            graph.R = R;
            return _graphDao.Add(graph);
        }

        // Получение описания всей информации о графах.
        public IEnumerable<GraphInfo> GetAllGraphInfo()
        {
            return _graphDao.GetAllGraphInfo();
        }

```

```
        public Graph GetById(int id) {
            return _graphDao.GetById(id);
        }
    }
}

namespace GeneticAlgorithmWEB.BLL
{
    // Уровень бизнес-логики.
    // Реализут работу с пользователями.
    public class UserBL : IUserBL
    {
        private readonly IUserDao _userDao;
        private readonly Encryption _encryption;

        public UserBL(IUserDao userDao)
        {
            _userDao = userDao;
            _encryption = new Encryption();
        }
        // Добавление нового пользователя с шифрацией пароля
        public void Add(CreateUserRequest user)
        {
            User createdUser = new User() {
                Login = user.Login,
                Password = _encryption.CreatePassword(user.Password),
            };
            _userDao.Add(createdUser);
        }

        // Проверка пароля по логину пользователя
        public bool CheckPassword(LoginUserRequest user)
        {
            User realUser = _userDao.GetByLogin(user.Login);
            if (realUser == null)
            {
                return false;
            }
            else
            {
                return _encryption.CheckPassword(realUser.Password, user.Password);
            }
```

```
37              }
38              // Проверка существует ли пользователь с переданным логином
39              public bool UserExists(CreateUserRequest user) {
40                  return _userDao.GetByLogin(user.Login) != null;
41              }
42          }
43      }
```

ПРИЛОЖЕНИЕ В

Уровень пользовательского интерфейса

Далее приводится описание основных классов-контроллеров.

```
1  namespace SimplePages.Controllers
2  {
3      public class GraphController : Controller
4      {
5          private readonly IAlgorithm _algorithmWork;
6          private readonly IGraphBL _graphBL;
7
8          public GraphController(IAlgorithm algorithmWork, IGraphBL graphBL) {
9              _algorithmWork = algorithmWork;
10             _graphBL = graphBL;
11         }
12
13         // Метод для получения страницы поиска вершины.
14         [HttpGet]
15         public ActionResult Index()
16         {
17             return View("FindCenter");
18         }
19         // Метод для нахождения центральных вершин в графе.
20         [HttpPost]
21         public ActionResult FindCentralVertex(HttpPostedFileBase upload)
22         {
23             if (upload == null)
24             {
25                 return View("~/Views/Shared/Error.cshtml", model: "Файл не был выбран");
26             }
27             try
28             {
29                 Graph graph = ReadGraph(upload);
30                 FindingVertexResponse algorithmResult = _algorithmWork.FindCentralVertex(
                       graph);
31                 return View("CalculationResult", algorithmResult);
32             }
33             catch (FormatException e)
34             {
35                 return View("~/Views/Shared/Error.cshtml", model: e.Message);
36             }
37         }
```

```
38          // Метод для возвращения страницы с добавлением графа.
39          [HttpGet]
40          [Authorize]
41          public ActionResult Add() {
42              return View("Add");
43          }
44          // Метод для добавления графа.
45          [HttpPost]
46          [Authorize]
47          public ActionResult Add(AddGraphRequest request) {
48              if (request.Upload == null) {
49                  return View("~/Views/Shared/Error.cshtml", model: "Выберите файл с
                        графом");
50              }
51              try
52              {
53                  Graph graph = ReadGraph(request.Upload);
54                  if (request.Name == null) {
55                      graph.Name = "Граф пользователя";
56                  }
57                  else {
58                      graph.Name = request.Name;
59                  }
60                  _graphBL.Add(graph);
61                  return Redirect("/");
62              }
63              catch (FormatException e) {
64                  return View("~/Views/Shared/Error.cshtml", model: e.Message);
65              }
66          }
67          // Чтение файла из потока данных.
68          private string [] ReadFile(Stream stream)
69          {
70              StreamReader reader = new StreamReader(stream);
71              List<string> lines = new List<string>();
72              string line;
73              while ((line = reader.ReadLine()) != null)
74              {
75                  lines.Add(line);
76              }
77              return lines.ToArray();
```

```
78              }
79              // Чтение графа из строк файла.
80              private Graph ReadGraph(HttpPostedFileBase upload)
81              {
82                  string [] fileLines = ReadFile(upload.InputStream);
83                  return GraphParser.ParseTxtFormat( fileLines );
84              }
85          }
86      }
```

```
1   namespace SimplePages.Controllers
2   {
3       public class HomeController : Controller
4       {
5           // Метод для получения домашней страницы.
6           public ActionResult Index()
7           {
8               return View("Index");
9           }
10      }
11  }
```

```
1   namespace SimplePages.Controllers
2   {
3       public class LoginController : Controller
4       {
5           private readonly IUserBL _userBL;
6           private readonly string signUpViewName = "SignUp";
7           private readonly string signInViewName = "SignIn";
8           private readonly string successSignUpViewName = "SignUpSuccess";
9
10          public LoginController(IUserBL userBL)
11          {
12              _userBL = userBL;
13          }
14          // Страница для входа в систему.
15          [HttpGet]
16          public ActionResult Index()
17          {
18              return View(signInViewName, new LoginUserRequest());
19          }
20          // Метод для входа в систему.
21          [HttpPost]
```

```
22          public ActionResult SignIn(LoginUserRequest user) {
23              if (!ModelState.IsValid) {
24                  return View(signInViewName, user);
25              }
26              if (_userBL.CheckPassword(user))
27              {
28                  FormsAuthentication.SetAuthCookie(user.Login, true);
29                  return Redirect("/");
30              }
31              else {
32                  ModelState.AddModelError("LOGIN_PASSWORD", "Неверный логин или
                        пароль");
33              }
34              return View(signInViewName, user);
35          }
36          // Страница для регистрации нового пользователя.
37          [HttpGet]
38          [Authorize]
39          public ActionResult SignUp() {
40              return View(signUpViewName);
41          }
42          // Метод для добавления нового пользователя.
43          [HttpPost]
44          [Authorize]
45          public ActionResult SignUp(CreateUserRequest creatingUser) {
46              if (!ModelState.IsValid) {
47                  return View(signUpViewName, creatingUser);
48              }
49
50              if (!_userBL.UserExists(creatingUser)) {
51                  _userBL.Add(creatingUser);
52                  return View(successSignUpViewName, creatingUser);
53              }
54              else {
55                  ModelState.AddModelError("LOGIN_PASSWORD", "Пользователь с таким
                        логином уже существует");
56              }
57              return View(signUpViewName, creatingUser);
58          }
59          // Метод для выхода из системы.
60          [HttpGet]
```

```
        public ActionResult SignOut() {
            FormsAuthentication.SignOut();
            return Redirect("/");
        }
    }
}
```

```
namespace SimplePages.Controllers
{
    public class ResearchController : Controller
    {
        private readonly IGraphBL _graphBL;
        private readonly IAlgorithm _algorithmWork;

        public ResearchController(IGraphBL graphBL, IAlgorithm algorithmWork)
        {
            _graphBL = graphBL;
            _algorithmWork = algorithmWork;
        }
        // Метод получения страницы для выбора параметров для запуска генетического
            алгоритма.
        [HttpGet]
        public ActionResult Index()
        {
            IEnumerable<GraphInfo> graphs = _graphBL.GetAllGraphInfo();
            return View("ResearchParametrs", model: graphs);
        }
        // Метод запуска алгоритма с выбранными параметрами.
        [HttpPost]
        public ActionResult ResearchAlgorithm(ResearchRequest request) {
            ResearchAlgorithmResponse response = _algorithmWork.ResearchAlgorithm(request
                );
            return View("ResearchResult", model: new AlgorithmResultResponse() {
                ResearchRequest = request,
                AlgorithmResponse = response,
            });
        }
    }
}
```

ПРИЛОЖЕНИЕ Г

Внедрение зависимостей

Далее приводится программный код, который отвечает за внедрение зависимостей в приложении:

```
namespace GeneticAlgorithm.IoC
{
    public class NinjectRegistrations : NinjectModule
    {
        public override void Load()
        {
            // Связывание интерфейсов с их реализвацией.
            Bind<IUserDao>().To<UserDao>();
            Bind<IGraphDao>().To<GraphDao>();
            Bind<IAlgorithm>().To<Algorithm>();
            Bind<IGraphBL>().To<GraphBL>();
            Bind<IUserBL>().To<UserBL>();
        }
    }
}
```

```
using GeneticAlgorithm.IoC;
using Ninject;
using Ninject.Modules;
using Ninject.Web.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace SimplePages
{
    public class WebApiApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            GlobalConfiguration.Configure(WebApiConfig.Register);
```

```
22                 FilterConfig . RegisterGlobalFilters ( GlobalFilters . Filters );
23                 RouteConfig.RegisterRoutes(RouteTable.Routes);
24                 BundleConfig.RegisterBundles(BundleTable.Bundles);
25                 // Регистрация класса для внедрения зависимостей
26                 NinjectModule registrations = new NinjectRegistrations();
27                 var kernel = new StandardKernel( registrations );
28                 DependencyResolver.SetResolver(new NinjectDependencyResolver(kernel));
29             }
30         }
31     }
```

ПРИЛОЖЕНИЕ Д
Генетический алгоритм

Классы, отвечающие за работу генетического алгоритма и работу с графами.

```
namespace GA
{
    public class ExactAlgorithmCore
    {
        // Реализация тривиального алгоритма.
        public int FindRadius(GraphContext graphContext) {
            int R = int.MaxValue;
            for (int v = 0; v < graphContext.N; v++) {
                R = Math.Min(R, graphContext. GetEccentricity (v));
            }
            return R;
        }
    }
}
```

```
namespace GA
{
    // Основной класс с реализацией генетического алгоритма.
    public class GeneticAlgorithmCore
    {
        // Ссылка на объект с графом.
        private Graph _graph;
        // Размер популяции.
        private int _populationSize;
        // Параметры для вероятности скрещивания и мутации.
        private double _pc, _pm;
        // Число итераций генетического алгоритма.
        private int _step;

        private Stopwatch _watch;

        private List<int> _population;
        // Обертка над графом в виде контектса.
        private GraphContext _graphContext;
        // Объект−декоратор для работы с генирацией случайных значений.
        private RandomWorker _rndWorker;

```

```
23          public GeneticAlgorithmCore(Graph graph, int populationSize , double pm, double pc
                )
24          {
25              _step = 20;
26              _graph = graph;
27              _pm = pm;
28              _pc = pc;
29              _populationSize = populationSize ;
30
31              _watch = new Stopwatch();
32              _rndWorker = new RandomWorker();
33          }
34          // Метод для старта алгоритма.
35          public FindingVertexResponse StartAlgorithm () {
36              // Начальная инициализация
37              Init () ;
38              // Запуск измерения времени работы.
39              _watch. Start () ;
40              for (int i = 0; i < _step ; i++)
41              {
42                  EvolutionStep () ;
43              }
44              // Окончание измерений.
45              _watch.Stop () ;
46
47              FindingVertexResponse res = new FindingVertexResponse() {
48                  Time = _watch.ElapsedMilliseconds / (double)1000,
49              };
50              GetBestResult( res ) ;
51              return res ;
52          }
53          // Начальная инициализация параметров перед стартом алгоритма.
54          private void Init () {
55              _graphContext = new GraphContext(_graph);
56              _population = new List<int>() ;
57              for (int i = 0; i < _populationSize ; i++) {
58                  _population . Add(_rndWorker.NextInt(_graph.N));
59              }
60          }
61
62          private void EvolutionStep () {
```

```
63              Crossing();
64              Mutation();
65              Selection();
66          }
67          // Этап мутации.
68          private void Mutation() {
69              for (int i = 0; i < _population.Count; i++)
70              {
71                  int v = _population[i];
72                  double condition = _rndWorker.NextDouble();
73                  if (condition < _pm) {
74                      int[] neighbors = _graphContext.GetNeighbors(v);
75                      int index = _rndWorker.NextInt(neighbors.Length);
76                      _population[i] = neighbors[index];
77                  }
78              }
79          }
80          // Этап скрещивания.
81          private void Crossing() {
82              List<int> crossed = new List<int>();
83              for (int i = 0; i < _population.Count; i++) {
84                  if (_rndWorker.NextDouble() <= _pc) {
85                      int ind1 = _rndWorker.NextInt(_population.Count);
86                      int ind2 = _rndWorker.NextInt(_population.Count);
87                      int x = _population[ind1], y = _population[ind2];
88                      int[] path = _graphContext.GetPath(x, y);
89                      crossed.Add(path[path.Length / 2]);
90                  }
91              }
92              _population.AddRange(crossed);
93          }
94          // Этап естественного отбора.
95          private void Selection()
96          {
97              List<int> e = new List<int>();
98              foreach (var v in _population)
99              {
100                 e.Add(_graphContext.GetEccentricity(v));
101             }
102             double[] prob = _rndWorker.InvertProb(e.ToArray());
103             List<int> selectedPopulation = new List<int>();
```

```
104                 while ( selectedPopulation .Count < _populationSize ) {
105                     selectedPopulation .Add(_rndWorker.Choice(_population.ToArray(), prob));
106                 }
107                 _population = selectedPopulation ;
108             }
109             // Поиск минимального эксцентриситета – центральных вершин,
110             // и радиуса графа.
111             private void GetBestResult(FindingVertexResponse res) {
112                 List<int> e = new List<int>();
113                 foreach (var v in _population)
114                 {
115                     e.Add(_graphContext.GetEccentricity(v));
116                 }
117                 int R = e.Min();
118                 HashSet<int> resVertex = new HashSet<int>();
119                 for (int i = 0; i < e.Count; i++) {
120                     if (e[i] == R) {
121                         resVertex .Add(_population[i]);
122                     }
123                 }
124                 res .Center = resVertex .ToArray();
125                 res .R = R;
126             }
127         }
128     }
```

```
1   namespace GeneticAlgorithm
2   {
3       public class GraphContext
4       {
5           private int _n;
6           private int _m;
7           // Массив для хранения флагов, показывающих
8           // был ли запущен алгоритм поиска эксцентриситета из вершины.
9           private bool[] _checked;
10          // Двумерная матрица для хранения расстояний между вершинами.
11          private int [,] _distance ;
12          // Список смежности для хранения графа.
13          private List<int>[] _adjacency;
14          // Двумераня матрица для хранения путей между вершинами.
15          private List<int >[,] _path;
16
```

```
17          public  GraphContext(Graph graph)
18          {
19              _n = graph.N;
20              _m = graph.M;
21
22              _checked = new bool[_n];
23              _distance  = new int[_n,  _n];
24              _adjacency  = new List<int>[_n];
25              _path  = new List<int>[_n,  _n];
26
27              for  (int  i  = 0;  i  < _n;  i++) {
28                  _adjacency[i]  = new List<int>();
29              }
30
31              foreach  (var  edge  in  graph.Edges)
32              {
33                  int  u = edge.V1, v = edge.V2;
34                  _adjacency[u].Add(v);
35                  _adjacency[v].Add(u);
36              }
37          }
38
39          public  int  N { get  => _n; }
40          public  int  M { get  => _m; }
41
42          // Метод получения эксцентриситета вершины.
43          // Запускает алгоритм поиска в ширину, после чего
44          // находит максималную длину пути от заданной вершины до всех остальных.
45          public  int  GetEccentricity (int  v)  {
46              if  (v < 0  ||  v >= _n)  {
47                  throw new ArgumentException("Номер вершины должен быть
                        положительным");
48              }
49              if  (!_checked[v])  {
50                  BFS(v);
51              }
52              int  max = 0;
53              for  (int  i  = 0;  i  < _n;  i++)  {
54                  max = Math.Max(max, _distance[v, i]);
55              }
56              return  max;
```

```
57          }
58          // Проверка связности графа.
59          // Запуск обхода из нулевой вершины, после чего проверяется
60          // длина пути до всех остальных вершин и если есть вершины, до которых не
                дошел алгоритм,
61          // то это означает, что граф не связный, иначе связный.
62          public bool CheckConnectivity() {
63              BFS(0);
64              for (int i = 0; i < _n; i++) {
65                  if (_path[0, i].Count == 0) {
66                      return false;
67                  }
68              }
69              return true;
70          }
71          // Получение пути между вершинами,
72          // если алгоритм поиска в ширину запускался из вершины не запускался,
73          // то запускается алгоритм поиска в ширину.
74          public int[] GetPath(int u, int v) {
75              if (!_checked[u]) {
76                  BFS(u);
77              }
78              return _path[u, v].ToArray();
79          }
80          // Получение расстояния между вершинами.
81          public int Distance(int x, int y) {
82              if (!_checked[x]) {
83                  BFS(x);
84              }
85              return _distance[x, y];
86          }
87          // Получение соседей вершины.
88          public int[] GetNeighbors(int v)
89          {
90              return _adjacency[v].ToArray();
91          }
92          // Классический алгоритм обхода в ширину.
93          // Сохраняет информацию о длинах найденных путей и сами пути.
94          // Также отмечает флагом посещенные вершины.
95          private void BFS(int x) {
96              _checked[x] = true;
```

```
97                bool[]  visited  = new bool[_n];
98
99                _path[x, x] = new List<int>() { x };
100               _distance[x, x] = 0;
101               visited[x] = true;
102
103               Queue<int> q = new Queue<int>();
104               q.Enqueue(x);
105
106               while (q.Count != 0) {
107                    int v = q.Dequeue();
108                    foreach (var u in _adjacency[v])
109                    {
110                         if (!visited[u])
111                         {
112                              visited[u] = true;
113
114                              List<int> path = _path[x, v];
115                              path.Add(u);
116                              _path[x, u] = path;
117                              _path[u, x] = path;
118
119                              _distance[x, u] = _distance[x, v] + 1;
120                              _distance[u, x] = _distance[x, v] + 1;
121                              q.Enqueue(u);
122                         }
123                    }
124               }
125          }
126     }
127 }
```

```
1  namespace GeneticAlgorithmWEB.BLL
2  {
3      public static class GraphParser
4      {
5          // Чтение графа из текстового формата файла.
6          // Проверяет соответствие текстового файла
7          // на соответствие верному формату.
8          public static Graph ParseTxtFormat(string[] lines)
9          {
10             if (lines.Length == 0) {
```

```
11                  throw new FormatException("Файл с графом пустрой");
12              }
13              List<Edge> edges = new List<Edge>();
14              foreach (var line in lines)
15              {
16                  int[] vertices = line.Split().Select(v => int.Parse(v)).ToArray();
17                  if (vertices.Length != 2)
18                  {
19                      throw new FormatException("Ребра должны быть записаны как пара
                            вершин");
20                  }
21                  Edge edge = new Edge()
22                  {
23                      V1 = vertices[0],
24                      V2 = vertices[1]
25                  };
26                  if (edge.V1 < 0 || edge.V2 < 0)
27                  {
28                      throw new FormatException("Индекс вершин должен быть
                            положительным");
29                  }
30                  edges.Add(edge);
31              }
32              int n = CountVerteces(edges);
33              return new Graph
34              {
35                  N = n,
36                  M = edges.Count,
37                  Edges = edges,
38              };
39          }
40          // Проверка факта, что нумерация вершин начинается с 0 и идет по порядку.
41          private static int CountVerteces(List<Edge> edges)
42          {
43              HashSet<int> uniqueLabel = new HashSet<int>();
44              foreach (var edge in edges)
45              {
46                  uniqueLabel.Add(edge.V1);
47                  uniqueLabel.Add(edge.V2);
48              }
49              int minValue = uniqueLabel.Min();
```

```
50                  if (minValue != 0) {
51                      throw new FormatException("Индексация вершин должна начинаться с нуля"
                            );
52                  }
53                  int n = uniqueLabel.Max() + 1;
54                  if (n != uniqueLabel.Count) {
55                      throw new FormatException("Индексация вершин должна идти по порядку");
56                  }
57                  return n;
58              }
59          }
60      }
```

ПРИЛОЖЕНИЕ Е

CD-диск с отчетом о выполненной работе

На приложенном диске можно ознакомиться со следующими файлами:

Папка `Pract` — LaTeX- вариант отчета о практике;

Папка `GAWeb` — Visual-Studio проект с полным кодом приложения;