

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

старший преподаватель
должность, уч. степень, звание

подпись, дата

С.А. Рогачев
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

ИСПОЛЬЗОВАНИЕ ЗАДАНЫХ СТРУКТУР ДАННЫХ И
АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
ИНФОРМАЦИОННОЙ СИСТЕМЫ

по дисциплине: Структуры и алгоритмы обработки данных

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

Z1431

номер группы

подпись, дата

Быстров М.Д.

инициалы, фамилия

Студенческий билет №

Шифр ИНДО

09.03.04

Санкт-Петербург 2023

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ.....	2
ЗАДАНИЕ	3
ВВЕДЕНИЕ.....	4
1. Алгоритмы и структуры данных	5
2. Описание программы.....	8
3. Тестирование программы.....	12
ЗАКЛЮЧЕНИЕ.....	28
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	29
ПРИЛОЖЕНИЕ 1 ИСХОДНЫЙ КОД ПРОГРАММЫ.....	30

ЗАДАНИЕ

Цель курсового проектирования: изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ. Задача курсового проекта: разработка информационной системы для заданной предметной области с использованием заданных структур данных и алгоритмов. Тема курсового проекта: «Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы».

Согласно алгоритму распределения вариантов определена комбинация переменных частей задания:

Предметная область – «Регистрация постояльцев в гостинице»;

Метод хеширования – открытое;

Метод сортировки – слиянием;

Вид списка – циклический однонаправленный;

Метод обхода дерева – прямой;

Алгоритм поиска слова в тексте – Боуера и Мура (БМ).

ВВЕДЕНИЕ

Предметная область – «Регистрация постояльцев в гостинице». Программное решение, созданное для работы в данной предметной области, позволит автоматизировать документооборот гостиничного обслуживания.

Предметная область, охватывающая взаимодействие с клиентами гостиничного сервиса, является благодатной почвой для автоматизации, поскольку в ходе работы непрерывно генерируются потоки новых данных (регистрация постояльцев, заселения и выселения). Автоматизация этих процессов позволит существенно сократить время на обслуживание конечных клиентов предприятия, что положительно отразится на экономических показателях предприятия.

Используя преимущества электронного документооборота, такие как скорость, надежность, защита от ошибок персонала, аналитические возможности, отчетные механизмы, предприятие получает возможность значительно повысить эффективность своей работы. Чем продуманнее и качественнее спроектировано, реализовано и протестировано ПО, тем больше потенциальные выгоды от его внедрения и использования.

В пояснительной записке будет описан процесс создания подобного решения.

1. Алгоритмы и структуры данных

1. Хеш-таблица с открытым хешированием

Хеш-таблица – структура данных, основанная на использовании для хранения данных хеш-функции.

Хеш-функция – это функция отображения множества значений из какого-либо множества A в множество B . В структурах данных и криптографии используются хеш-функции, множество B в реализации которых представляет из себя конечное множество значений фиксированной длины (в частном случае – множество целочисленных значений).

Хеш-таблица построена на свойстве детерминированности хеш-функции, т.е. неизменности выходных данных для одних и тех же входных данных. Если с помощью выходного значения хеш-функции возможно быстро рассчитать адрес в оперативной памяти, по которому можно получить доступ к данным, то это возможно использовать для построения структуры данных, позволяющей эффективно производить поиск данных по ключу.

В общем виде хеш-таблица представляет из себя массив фиксированного размера, элементами которого являются хранимые данные. Индексом элемента массива является значение, полученное при вычислении хеш-функции по значению ключа. Ключ также входит в состав данных, хранящихся в элементе массива.

Хеш-таблица с открытым хешированием является частным случаем хеш-таблицы. Её особенность состоит в том, что при возникновении коллизий (когда для разных ключей рассчитывается один и тот же хеш) происходит разрешение с помощью связного списка, т.е. в каждом элементе массива таблицы содержится ссылка на другие элементы, также находящиеся в хеш-таблице, но вставленные с разрешением коллизии. В таком виде каждый элемент таблицы является «головой» связного списка.

При использовании хеш-функции с хорошим распределением вставка и удаление элемента в таблице будет происходить за $O(1)$. При возникновении большого количества коллизий эффективность будет снижаться в худшую

сторону вплоть до $O(N)$.

2. Односвязный циклический список

Связный список – структура данных, представляющая из себя набор элементов, связанных между собой ссылками.

Односвязный список – такая разновидность связного списка, в котором каждый элемент содержит ссылку на последующий элемент (если он имеется).

Циклический список – такой список, в котором первый и последний элемент связаны ссылками. В случае с односвязным списком это означает, что в последнем элементе будет содержаться ссылка на первый. В том случае, если в списке всего один элемент, он будет содержать ссылку на самого себя.

Типичная программная реализация списка представляет из себя объект-оболочку, содержащий в себе ссылки на первый и последний элемент списка и позволяющий производить вставку и удаление данных. Элементами списка являются ссылочные типы либо структуры, под которые память динамически выделяется в куче. Полями-ссылками на соседние объекты являются указатели (в других терминологиях - ссылки).

3. AVL-дерево поиска с прямым обходом

Дерево представляет из себя граф с однонаправленными связями, в котором из каждого узла может исходить не более двух связей. Узел, в который не приходит связей, является вершиной. Узлы, из которых не исходит связей, называются листьями.

Дерево поиска – структура данных, представляющая из себя такое дерево, которое состоит из узлов, способных сравниваться между собой и сохранять упорядоченность с помощью этого свойства. Чаще всего реализация принимает следующий вид: все элементы в правом поддереве вершины превосходят вершину по значениям, в левом – уступают. Идея структуры состоит в вставке, поиске и удалении элементов за логарифмическое время.

AVL-дерево – разновидность дерева поиска, при котором сохраняется сбалансированность по высоте: для каждой вершины разность между высотами левого и правого поддерева не превышает 1. При обнаружении

факта утери баланса (при добавлении/удалении элементов) происходит балансировка дерева с помощью левых и правых поворотов, которые выполняются всегда за $O(1)$. Это позволяет сохранить стабильное время работы со структурой при любом порядке вставки данных, к примеру при упорядоченной вставке значений от 1 до 100. При использовании обычного дерева поиска структура дерева выродилась бы в односвязный список и время поиска значения составляло бы $O(N)$. При использовании АВЛ-дерева этот недостаток не будет проявлен благодаря восстановлению баланса.

4. Алгоритм Боуера и Мура

Алгоритм Боуера и Мура – модификация алгоритма прямого поиска фрагмента текста, основанная на использовании таблицы смещений. Если при прямом поиске после обнаружения несовпадения слово для поиска сдвигается на один символ вправо, при поиске Боуера и Мура слово сдвигается на количество символов, равное расстоянию от несовпадающего символа в тексте до конца слова поиска, если этот символ встречался в слове поиска. Если символ не встречался в слове поиска, слово поиска сдвигается на всю длину. Использование алгоритма позволяет уменьшить количество итераций, затрачиваемых на поиск фрагмента.

5. Сортировка слиянием

Сортировка слиянием – алгоритм сортировки, работающий за стабильное время $O(n \log n)$. Принцип работы заключается в дроблении исходного массива данных на части до интервалов размером 1-2 элемента, их отдельной сортировке и итеративном слиянии получившихся массивов. Требуется дополнительный объем памяти $O(n)$.

2. Описание программы

Для разработки программы был выбран язык C#. Программа имеет графический пользовательский интерфейс, построенный с помощью компонентов Windows Forms, входящих в состав платформы .NET.

В ходе работы с программой данные хранятся в определенных вариантах структурах данных. Данные о гостиничных номерах хранятся в АВЛ-дереве поиска, данные о постояльцах – в хеш-таблице, данные о проживании постояльцев в номерах – в упорядоченном списке. Применение алгоритмов при выполнении основных операций будет отмечено при описании взаимодействия с программой.

Главный экран программы представлен на рисунках 1 и 2.

The screenshot shows a Windows application window titled "Регистрация постояльцев в гостинице". It has two tabs: "Номера" (selected) and "Постояльцы". The "Номера" tab displays a table with columns: "Номер", "Кол-во спальных мест", "Кол-во комнат", "Сан. узел", and "Оборудование". The first row is highlighted with a blue selection bar. Below the table are two search fields: "Поиск по номеру комнаты" and "Поиск по оборудованию", each with an "OK" button. On the right side, there are five buttons: "Заселения", "Создать", "Изменить", "Удалить", and "Очистить".

	Номер	Кол-во спальных мест	Кол-во комнат	Сан. узел	Оборудование
▶	O868	1	2	<input checked="" type="checkbox"/>	Телевизор, Микроволновка
	O517	2	3	<input checked="" type="checkbox"/>	Микроволновка

Рисунок 1 Главный экран, вкладка «Номера»

The screenshot shows the same application window, but with the "Постояльцы" tab selected. The table has columns: "Номер паспорта", "Имя", "Год рождения", and "Адрес". The first row is highlighted with a blue selection bar. Below the table are two search fields: "Поиск по номеру паспорта" and "Поиск по ФИО", each with an "OK" button. On the right side, there are five buttons: "Заселения", "Создать", "Изменить", "Удалить", and "Очистить".

	Номер паспорта	Имя	Год рождения	Адрес
▶	8236-027122	Mikes	1976	SaintP
	7540-034461	Mike	1951	LA 1151

Рисунок 2 Главный экран, вкладка «Постояльцы»

На главном экране программы расположены две вкладки для управления

данными о номерах и постояльцев. Данные доступны для просмотра в двух таблицах, которые позволяют выделить необходимую строку. Для каждой строки таблиц доступны операции: вывод заселений, изменение записи, удаление записи. Вне контекста выбранной записи доступны операции: создание записи, очистка всех данных. Для номеров доступен поиск по номеру комнаты и по оборудованию. Для постояльцев доступен поиск по номеру паспорта и ФИО, а также функции заселения и выселения.

Для добавления данных используется отдельное диалоговое окно. Окно для добавления номера представлено на рисунке 3. При создании записи производится проверка пользовательского ввода.

Номер

Код номера
3123

Кол-во кроватей
1

Кол-во комнат
1

☐ Есть сан. узел

Оборудование
Комната

Сохранить

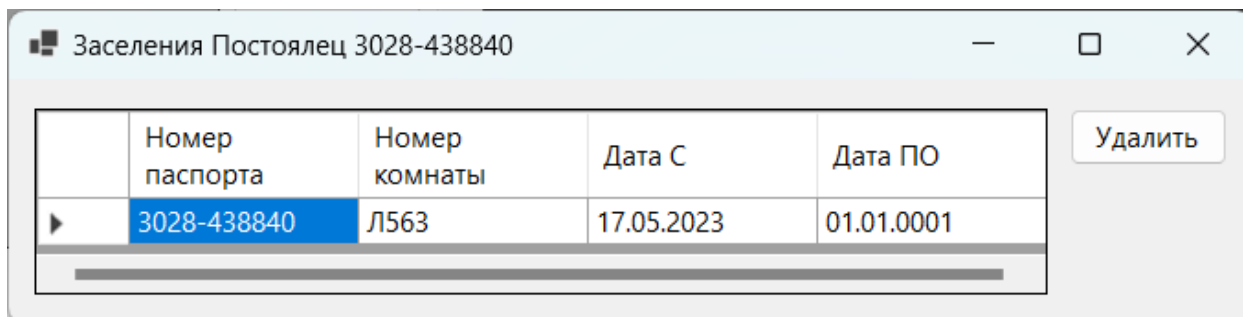
Номер комнаты должен соответствовать формату «ANNN», где А – буква, обозначающая тип номера (Л – люкс, П – полулюкс, О – одноместный, М – многоместный); NNN – порядковый номер (цифры)

ОК

Рисунок 3 Создание номера

Диалоговые окна редактирования данных номеров, создания и редактирования данных постояльцев выполнены аналогичным образом.

При нажатии на кнопку «Заселения» открывается форма с данными о заселениях. В зависимости от активной вкладки будут отображены данные о заселениях либо для номера, либо для постояльца. Внешний вид формы показан на рисунке 4.



The screenshot shows a window titled "Заселения Постоялец 3028-438840". It contains a table with the following data:

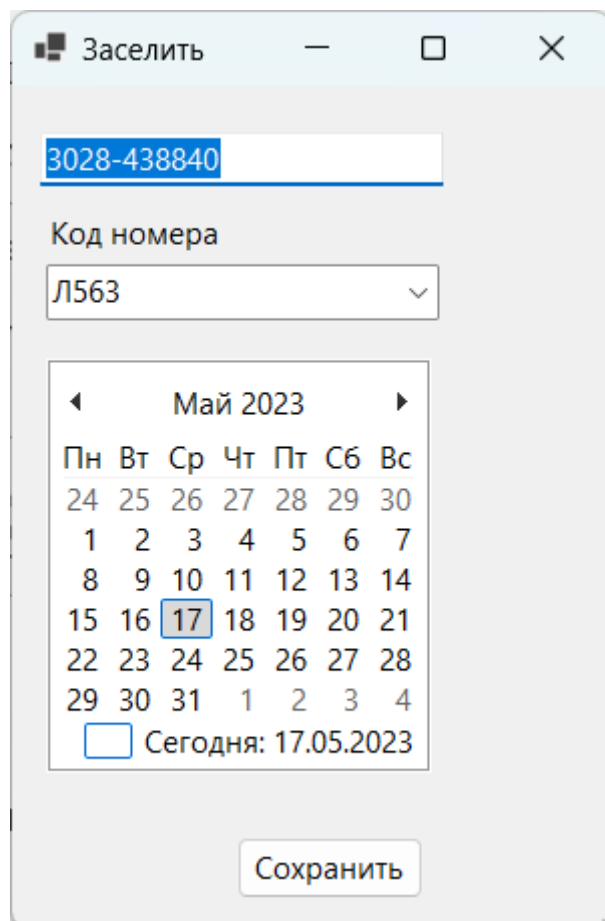
	Номер паспорта	Номер комнаты	Дата С	Дата ПО
▶	3028-438840	Л563	17.05.2023	01.01.0001

To the right of the table is a button labeled "Удалить".

Рисунок 4 Отображение списка заселений

Для удаления периода проживания должна быть использована кнопка «Удалить». Без удаления записей о проживании удаление данных номера или постояльца невозможно.

Для создания заселений используется отдельное диалоговое окно. Внешний вид окна создания заселения представлен на рисунке 5.



The screenshot shows a dialog window titled "Заселить". It contains the following elements:

- A text input field with the value "3028-438840".
- A label "Код номера" above a dropdown menu showing "Л563".
- A calendar widget for May 2023. The date 17 is selected. Below the calendar, it says "Сегодня: 17.05.2023".
- A "Сохранить" (Save) button at the bottom.

Рисунок 5 Заселение постояльца

Для заселения постояльца необходимо выбрать номер и дату, с которой необходимо заселить постояльца.

Для выселения постояльца используется аналогичная форма. При проведении выселения период проживания, ранее открытый при заселении, ограничивается датой, выбранной в диалоговом окне.

Исходный код программы с комментариями приведен в приложении 1.

3. Тестирование программы

Тестирование программы произведено в двух вариантах: модульное тестирование и ручное тестирование.

В рамках модульного тестирования были произведены проверки работы структур данных и алгоритмов, созданных в ходе выполнения курсового проекта. Входные данные и результаты модульного тестирования будут представлены в виде исходных файлов сценариев тестирования.

1. Тестирование алгоритма поиска строки в тексте Боуэра и Мура.

```
1. using SaodCP.Algorithms;
2. using static SaodCP.Algorithms.BMTextSearch;
3.
4. namespace SaopCPTest
5. {
6.     [TestClass]
7.     public class BMTextSearchTest
8.     {
9.         [TestMethod]
10.        public void SearchTest()
11.        {
12.            string text = string.Empty;
13.            string search = string.Empty;
14.
15.            var found = TextSearch(text, search);
16.
17.            Assert.AreEqual(0, found);
18.
19.            text = "qwerty";
20.            search = "qwe";
21.
22.            found = TextSearch(text, search);
23.
24.            Assert.AreEqual(1, found);
25.
26.            search = "werty";
27.
28.            found = TextSearch(text, search);
29.
30.            Assert.AreEqual(2, found);
31.
32.            search = "y";
33.
34.            found = TextSearch(text, search);
35.
36.            Assert.AreEqual(6, found);
```

```

37.
38.         search = "qwey";
39.
40.         found = TextSearch(text, search);
41.
42.         Assert.AreEqual(0, found);
43.     }
44. }
45. }

```

2. Тестирование односвязного циклического списка

```

1. using Microsoft.VisualStudio.TestTools.UnitTesting;
2. using SaodCP.DataStructures;
3. using SaodCP.Models;
4. using SaodCP.Utils;
5. using System;
6. using System.Collections.Generic;
7. using System.Linq;
8. using System.Text;
9. using System.Threading.Tasks;
10.
11. namespace SaodCPTest
12. {
13.     [TestClass]
14.     public class CycledListTest
15.     {
16.         [TestMethod]
17.         public void CreateAddTest()
18.         {
19.             var list = new OneWayCycledList<Lodger>();
20.
21.             Assert.IsNotNull(list);
22.             Assert.IsTrue(list.Count == 0);
23.
24.             var lodgerArray = new Lodger[10];
25.
26.             var rand = new Random();
27.
28.             for (int i = 0; i < lodgerArray.Length; i++)
29.             {
30.                 lodgerArray[i] = new Lodger();
31.
32.                 lodgerArray[i].PassportId =
33.                     Utils.GenerateRandomPassportId();
34.                 lodgerArray[i].Name = rand.GetHashCode().ToString();
35.             }
36.
37.             for (int i = 0; i < lodgerArray.Length; i++)

```

```

37.         {
38.             list.Add(lodgerArray[i]);
39.         }
40.
41.         Assert.AreEqual(lodgerArray.Length, list.Count);
42.
43.         int pos = 0;
44.
45.         foreach (var lodger in list)
46.         {
47.             Assert.AreEqual(lodger, lodgerArray[pos]);
48.             Assert.IsTrue(list.Contains(lodger));
49.
50.             pos++;
51.         }
52.
53.         Assert.IsTrue(list.Contains(lodgerArray[0]));
54.
55.         list.Remove(lodgerArray[0]);
56.
57.         Assert.IsFalse(list.Contains(lodgerArray[0]));
58.         Assert.AreEqual(list.Count, lodgerArray.Length - 1);
59.
60.         list.Clear();
61.
62.         Assert.AreEqual(0, list.Count);
63.
64.         var found = false;
65.
66.         foreach(var lodger in list)
67.         {
68.             found = true;
69.         }
70.
71.         Assert.IsFalse(found);
72.     }
73. }
74. }
75.

```

3. Тестирование операций заселения и выселения постояльца

```

1. using static SaodCP.Database.HostelContext;
2.
3. namespace SaopCPTest
4. {
5.     [TestClass]
6.     public class HostelContextTest
7.     {

```

```

8.         [TestMethod]
9.         public void PeriodAddTest()
10.        {
11.            InitTestData();
12.
13.            var lodger = Lodgers.First().Value;
14.            var room = Apartments.First().Value;
15.
16.            string error = string.Empty;
17.
18.            var startDate = DateOnly.FromDateTime(DateTime.Now);
19.
20.            var started = StartAccommodation(
21.                lodger.PassportId,
22.                room.Number,
23.                startDate,
24.                ref error);
25.
26.            Assert.IsTrue(started);
27.            Assert.AreEqual(Accommodations.Count, 1);
28.
29.            // пытаемся повторно заселить того
30.            // же постояльца в то же место
31.            started = StartAccommodation(
32.                lodger.PassportId,
33.                room.Number,
34.                startDate,
35.                ref error);
36.
37.            Assert.IsFalse(started);
38.            Assert.IsFalse(string.IsNullOrEmpty(error));
39.
40.            // на дату позже
41.            started = StartAccommodation(
42.                lodger.PassportId,
43.                room.Number,
44.                startDate.AddDays(5),
45.                ref error);
46.
47.            Assert.IsFalse(started);
48.            Assert.IsFalse(string.IsNullOrEmpty(error));
49.
50.            // пытаемся выселить на день раньше чем заселили
51.            var ended = EndAccommodation(
52.                lodger.PassportId,
53.                room.Number,
54.                startDate.AddDays(-1),
55.                ref error);
56.
57.            Assert.IsFalse(ended);

```

```

58.         Assert.IsFalse(string.IsNullOrEmpty(error));
59.
60.         // выселяем успешно
61.         ended = EndAccommodation(
62.             lodger.PassportId,
63.             room.Number,
64.             startDate.AddDays(5),
65.             ref error);
66.
67.         Assert.IsTrue(ended);
68.         Assert.IsTrue(string.IsNullOrEmpty(error));
69.
70.         // выселяем повторно (должна быть ошибка)
71.         ended = EndAccommodation(
72.             lodger.PassportId,
73.             room.Number,
74.             startDate.AddDays(5),
75.             ref error);
76.
77.         Assert.IsFalse(ended);
78.         Assert.IsFalse(string.IsNullOrEmpty(error));
79.     }
80. }
81. }

```

4. Тестирование создания постояльца и генерации ИД паспорта

```

1. using SaodCP.Models;
2. using SaodCP.Utills;
3. using System.Text.RegularExpressions;
4.
5. namespace SaopCPTest
6. {
7.     [TestClass]
8.     public class LodgerTest
9.     {
10.         /// <summary>
11.         /// Тестирование генерации случайного Id паспорта
12.         /// </summary>
13.         [TestMethod]
14.         public void RandomPassportIdTest()
15.         {
16.             var pattern = @"^d\d\d\d-d\d\d\d\d\d";
17.
18.             for (int i = 0; i < 1000; i++)
19.             {
20.                 var id = Utills.GenerateRandomPassportId();
21.
22.                 Assert.IsTrue(Regex.IsMatch(id, pattern));

```



```

23.         }
24.     }
25.
26.     /// <summary>
27.     /// Тестирование функции проверки формата Id паспорта
28.     /// </summary>
29.     [TestMethod]
30.     public void PassportIdValidationTest()
31.     {
32.         for (int i = 0; i < 1000; i++)
33.         {
34.             var id = Utils.GenerateRandomPassportId();
35.
36.             Assert.IsTrue(Utils.ValidateLodgerPassportId(id));
37.         }
38.
39.         var passportId = string.Empty;
40.
41.         Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
42.
43.         passportId = "0000-00000A";
44.
45.         Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
46.
47.         passportId = "00000000000";
48.
49.         Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
50.
51.         passportId = "0000-1231234";
52.
53.         Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
54.
55.         passportId = "00001-231234";
56.
57.         Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
58.     }
59.
60.     /// <summary>
61.     /// Тестирование создания постояльца и генерации ИД паспорта
62.     /// </summary>
63.     [TestMethod]
64.     public void CreateLodgerTest()
65.     {
66.         var lodger = new Lodger();
67.
68.         Assert.IsNotNull(lodger);
69.
70.         lodger.PassportId = Utils.GenerateRandomPassportId();
71.

```

```

72.
    Assert.IsTrue(Utils.ValidateLodgerPassportId(lodger.PassportId));
73.     }
74. }
75.}

```

5. Тестирование хеш-таблицы

```

1. using SaodCP.DataStructures;
2.
3. namespace SaopCPTest
4. {
5.     /// <summary>
6.     /// Тестирование хеш-таблицы
7.     /// </summary>
8.     [TestClass]
9.     public class OpenHashTableTest
10.    {
11.        /// <summary>
12.        /// Создать, добавить, получить
13.        /// </summary>
14.        [TestMethod]
15.        public void CreateAddGetTest()
16.        {
17.            OpenHashTable<string, string> hashTable = new();
18.
19.            Assert.AreEqual(0, hashTable.Count);
20.
21.            hashTable.Add("1", "2");
22.
23.            Assert.AreEqual(1, hashTable.Count);
24.
25.            var value = hashTable["1"];
26.
27.            Assert.AreEqual("2", value);
28.        }
29.
30.        /// <summary>
31.        /// Нагрузочное тестирование - добавление 1000000 элементов
32.        /// Тестирование удаление элементов
33.        /// Тестирование перечисления элементов
34.        /// </summary>
35.        [TestMethod]
36.        public void RebalanceRemoveTest()
37.        {
38.            Random rnd = new Random();
39.
40.            // на 1000000 отрабатывает за 6с на i7-10610U
41.            int cnt = 1000000;
42.

```

```

43. Dictionary<string, string> pairs = new();
44.
45. for (int i = 0; i < cnt; i++)
46. {
47.     var key = rnd.Next().ToString();
48.
49.     if (!pairs.ContainsKey(key))
50.     {
51.         pairs.Add(key, rnd.Next().ToString());
52.     }
53. }
54.
55. OpenHashTable<string, string> hashTable = new();
56.
57. int count = 0;
58.
59. foreach (KeyValuePair<string, string> pair in pairs)
60. {
61.     hashTable.Add(pair.Key, pair.Value);
62.
63.     count++;
64.
65.     Assert.AreEqual(count, hashTable.Count);
66. }
67.
68. Console.WriteLine("Вставка пройдена");
69.
70. Assert.AreEqual(count, hashTable.Count);
71.
72. foreach (KeyValuePair<string, string> pair in pairs)
73. {
74.     Assert.AreEqual(hashTable[pair.Key], pair.Value);
75. }
76.
77. Console.WriteLine("Сверка элементов пройдена");
78.
79. foreach (KeyValuePair<string, string> pair in pairs)
80. {
81.     count--;
82.
83.     hashTable.Remove(pair.Key);
84.
85.     Assert.AreEqual(hashTable.Count, count);
86. }
87.
88. Assert.AreEqual(hashTable.Count, 0);
89.
90. Console.WriteLine("Удаление элементов пройдено");
91. }
92.

```

```

93.         /// <summary>
94.         /// Тестирование корректной работы с хеш-значениями
95.         /// </summary>
96.         [TestMethod]
97.         public void CustomHashTest()
98.         {
99.             var hashTable = new OpenHashTable<HashString, string>();
100.
101.             int cnt = 10000;
102.
103.             Random rnd = new Random();
104.
105.             Dictionary<HashString, string> pairs = new();
106.
107.             for (int i = 0; i < cnt; i++)
108.             {
109.                 var key = rnd.Next().ToString();
110.
111.                 if (!pairs.ContainsKey(key))
112.                 {
113.                     pairs.Add(key, rnd.Next().ToString());
114.                 }
115.             }
116.
117.             foreach (var (key, value) in pairs)
118.             {
119.                 hashTable.Add(key, value);
120.             }
121.
122.             var keys = new HashSet<HashString>();
123.
124.             var enumerator = hashTable.GetEnumerator();
125.
126.             while (enumerator.MoveNext())
127.             {
128.                 keys.Add(enumerator.Current.Key);
129.             }
130.
131.             //Assert.AreEqual(pairs.Keys.ToHashSet(),
132.             keys.ToHashSet());
133.
134.             var except =
135.                 pairs.Keys.ToHashSet().Except(hashTable.Keys);
136.
137.             Assert.AreEqual(except.Count(), 0);
138.
139.             Assert.IsTrue(pairs.Values.ToHashSet().SetEquals(hashTable.Values.ToHashSet()));
140.
141.             Assert.AreEqual(pairs.Count, hashTable.Count);
142.         }

```

```

139.         }
140.     }

```

6. Тест алгоритма сортировки слиянием и тест автоматически сортируемого списка

```

1. using SaodCP.DataStructures;
2. using SaodCP.Utills;
3.
4. namespace SaopCPTest
5. {
6.     /// <summary>
7.     /// Тест сортировки
8.     /// </summary>
9.     [TestClass]
10.    public class SortTest
11.    {
12.        /// <summary>
13.        /// Тест сортировки массива слиянием
14.        /// </summary>
15.        [TestMethod]
16.        public void ArraySortTest()
17.        {
18.            Random randNum = new Random();
19.
20.            int[] source = Enumerable
21.                .Repeat(0, 50)
22.                .Select(i => randNum.Next(100))
23.                .ToArray();
24.
25.            // сортировка по умолчанию (по возрастанию)
26.            var testDest = source.MergeSort();
27.
28.            var trustList = source.ToList();
29.
30.            trustList.Sort();
31.
32.            var trustDest = trustList.ToArray();
33.
34.            Assert.AreEqual(trustDest.Length, testDest.Length);
35.
36.            for (int i = 0; i < trustDest.Length; i++)
37.            {
38.                Assert.AreEqual(trustDest[i], testDest[i]);
39.            }
40.
41.            // сортировка по убыванию
42.            Comparison<int> comparison = (f, s) => -f.CompareTo(s);

```

```

43.
44.         testDest = source.MergeSort(comparison);
45.
46.         trustList = source.ToList();
47.
48.         trustList.Sort(comparison);
49.
50.         trustDest = trustList.ToArray();
51.
52.         Assert.AreEqual(trustDest.Length, testDest.Length);
53.
54.         for (int i = 0; i < trustDest.Length; i++)
55.         {
56.             Assert.AreEqual(trustDest[i], testDest[i]);
57.         }
58.     }
59.
60.     /// <summary>
61.     /// Тест автоматически сортируемого односвязного списка
62.     /// </summary>
63.     [TestMethod]
64.     public void SortedListTest()
65.     {
66.         Random randNum = new Random();
67.
68.         int[] source = Enumerable
69.             .Repeat(0, 50)
70.             .Select(i => randNum.Next(100))
71.             .ToArray();
72.
73.         var trustList = source.ToList();
74.
75.         trustList.Sort();
76.
77.         var trustDest = trustList.ToArray();
78.
79.         var sortedList = new SortedOneWayCycledList<int>();
80.
81.         foreach (int i in source)
82.         {
83.             sortedList.Add(i);
84.         }
85.
86.         int cnt = 0;
87.
88.         foreach (int element in sortedList)
89.         {
90.             Assert.AreEqual(element, trustDest[cnt]);
91.
92.             cnt++;

```

```

93.     }
94. }
95. }
96.}

```

7. Тестирование AVL-дерева поиска

```

1. using SaodCP.DataStructures;
2.
3. namespace SaodCPTest
4. {
5.     /// <summary>
6.     /// Тесты для AVL-дерева поиска
7.     /// </summary>
8.     [TestClass]
9.     public class TreeTest
10.    {
11.        /// <summary>
12.        /// Создание, добавление, удаление элементов
13.        /// </summary>
14.        [TestMethod]
15.        public void CreateAddRemoveTest()
16.        {
17.            var tree = new Tree<string, object>();
18.
19.            tree.Add("1");
20.            tree.Add("2");
21.            tree.Add("3");
22.            tree.Add("4");
23.            tree.Add("5");
24.
25.            Assert.AreEqual(5, tree.Count);
26.
27.            var intTree = new Tree<int, string>();
28.
29.            for (int i = 0; i < 100; i++)
30.            {
31.                intTree.Add(i, i.ToString());
32.            }
33.
34.            Assert.AreEqual(100, intTree.Count);
35.
36.            var treeSet = intTree
37.                .Select(kv => kv.Key)
38.                .ToHashSet();
39.
40.            var rangeSet = Enumerable.Range(0, 100).ToHashSet();
41.
42.            Assert.IsTrue(treeSet.SetEquals(rangeSet));

```

```

43.
44.         var count = intTree.Count;
45.
46.         // проверка удаления
47.         var removed = intTree.Remove(100);
48.
49.         Assert.IsFalse(removed);
50.         Assert.AreEqual(intTree.Count, count);
51.
52.         foreach (var i in treeSet)
53.         {
54.             removed = intTree.Remove(i);
55.
56.             // удаление успешно
57.             Assert.IsTrue(removed);
58.
59.             // элемента больше не содержится в дереве
60.             Assert.IsFalse(intTree.Any(kv => kv.Key == i));
61.
62.             // проверка корректности счетчика
63.             Assert.AreEqual(intTree.Count, --count);
64.         }
65.     }
66. }
67. }

```

Результаты ручного тестирования представлены в таблице 1.

Таблица 1 Входные и выходные данные тестирования

№	Тест	Входные данные	Результат
1.	Добавление заселения для постояльца 1 в номер 1	Код номера, дата заселения	Постоялец заселен в номер 1 с указанной даты
2.	Добавление пересекающегося периода проживания для постояльца 1	Код номера, дата заселения	Сообщение о конфликте периодов заселения
3.	Выселение постояльца 1 из номера 1	Код номера, дата выселения	Постоялец выселен из номера 1 с указанной даты
4.	Заселение в номер 1 (кол-во	Код номера, дата	Сообщение о

	кроватей – 1) постояльца 2, период проживания пересекается с периодом проживания постояльца 1 в номере 1	заселения	конфликте максимального количества одновременно проживающих в комнате 1
5.	Изменение номера 1, увеличение количества спальных мест с 1 до 2	Код номера, количество спальных мест	Данные номера изменены
6.	Повтор п. 4, кол-во кроватей – 2	Код номера, дата заселения	Постоялец 2 заселен в номер 1 с указанной даты
7.	Удаление данных о номера 1	Код номера	Сообщение о невозможности удаления данных ввиду наличия данных о заселении
8.	Удаление данных о постояльце 1	Номер паспорта	Сообщение о невозможности удаления данных ввиду наличия данных о заселении
9.	Удаление данных о заселениях в номер 1	Порядковый номер заселения в списке	Данные о заселениях удалены
10.	Повтор пп. 8, 9	Номер паспорта, Код номера	Данные удалены

11.	Изменение данных пользователя; ввод некорректного формата номера паспорта	Номер паспорта	Сообщение о неправильном формате номера паспорта
12.	Изменение данных номера; ввод некорректного формата кода номера	Код номера	Сообщение о неправильном формате кода номера
13.	Поиск информации о постояльце по номеру паспорта; ввод существующего ключа	Номер паспорта	Вывод данных о постояльце, вывод данных о номере, в котором проживает на данный момент
14.	Поиск информации о постояльце по номеру паспорта, ввод несуществующего ключа	Номер паспорта	Вывод сообщения об отсутствии совпадений по ключу
15.	Поиск информации о постояльце по ФИО; ввод части имени, совпадающей у нескольких постояльцев	ФИО постояльца	Вывод списка постояльцев
16.	Поиск информации о номере по коду номера; ввод существующего ключа	Код номера	Вывод информации о номере, вывод списка постояльцев, проживающих в

			номере
17.	Поиск информации о номере по коду номера; ввод несуществующего ключа	Код номера	Вывод сообщения об отсутствии совпадений по ключу
18.	Поиск информации о номерах по фрагменту оборудования; ввод подстроки, содержащейся в данных об оборудовании нескольких номеров	Подстрока – фрагмент перечисления оборудования	Вывод списка номеров, содержащих необходимое оборудование

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана программа, использующая заданные индивидуальным вариантом структуры данных и алгоритмы.

Произведено изучение литературы по теме, приведены данные об эффективности и принципах работы использованных алгоритмов и структур данных.

Компоненты программы протестированы методами модульного и ручного тестирования, подтверждена корректная работа на наборе тестовых данных. Входные и выходные данные тестирований отражены в пояснительной записке.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Алгоритмы и структуры данных: учеб. пособие / В. А. Матьяш, С. А. Рогачев. – СПб.: ГУАП, 2021. – 71 с.
2. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. — СПб.: Питер, 2003. —461 с: ил.
3. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. 3-е изд. - СПб.: Питер, 2012.- 928 с.: ил.
4. Пышкин, Евгений Валерьевич. Основные концепции и механизмы объектно-ориентированного программирования : учеб. пособие для студентов вузов по направлению подгот. 533000 "Систем. анализ и упр." / Е. В. Пышкин. - СПб. : БХВ-Петербург, 2005. - 628 с. : ил.
5. Структуры и алгоритмы обработки данных: практическое руководство / А. Б. Демуськов, Т. Я. Каморникова ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель : ГГУ им. Ф. Скорины, 2017. – 34 с.

ПРИЛОЖЕНИЕ 1

ИСХОДНЫЙ КОД ПРОГРАММЫ

1. Файл BMTextSearch.cs – поиск подстроки в строке

```
1. namespace SaodCP.Algorithms
2. {
3.     /// <summary>
4.     /// Алгоритм поиска подстроки в строке Боуэра и Мура
5.     /// </summary>
6.     public static class BMTextSearch
7.     {
8.         /// <summary>
9.         /// Поиск строки в тексте
10.        /// </summary>
11.        /// <param name="text">Текст, в котором производить поиск</param>
12.        /// <param name="search">Строка для поиска</param>
13.        /// <returns>Номер начала первого вхождения строки в
14.            текст</returns>
15.        public static int TextSearch(
16.            string text,
17.            string search)
18.        {
19.            if (text.Length == 0
20.                || search.Length == 0)
21.            {
22.                return 0;
23.            }
24.
25.            int[] shiftArray = FillShiftArray(search);
26.
27.            int startSearch = 0;
28.            int foundPosition = 0;
29.
30.            /// поиск в строке, пока подстрока не выходит за её пределы
31.            while (startSearch + search.Length <= text.Length)
32.            {
33.                bool hasDifferences = false;
34.
35.                int textCharInSearchPosition = -1;
36.
37.                /// посимвольное сравнение с конца подстроки
38.                for (int i = search.Length - 1; i >= 0; i--)
39.                {
40.                    var textIndex = startSearch + i;
41.
42.                    if (search[i] != text[textIndex])
43.                    {
44.                        /// найдено расхождение
45.                        hasDifferences = true;
```

```

45.
46.         // поиск несовпавшего символа в подстроке
47.         for (int j = 0; j < search.Length; j++)
48.         {
49.             if (search[j] == text[textIndex])
50.             {
51.                 // индекс несовпавшего символа из строки в
подстроке
52.                 // начиная с начала подстроки
53.                 textCharInSearchPosition = j;
54.             }
55.         }
56.
57.         break;
58.     }
59. }
60.
61. int offset;
62.
63. if (textCharInSearchPosition == -1)
64. {
65.     // несовпавший символ не содержится в подстроке -
66.     // сдвигаем подстроку на всю её длину вправо
67.     offset = search.Length;
68. }
69. else
70. {
71.     // количество символов для сдвига равно расстоянию от
последнего
72.     // вхождения несовпавшего символа в подстроку до конца
подстроки
73.     offset = shiftArray[textCharInSearchPosition];
74. }
75.
76. // если найдено расхождение
77. // позиция поиска смещается, поиск повторяется
78. if (hasDifferences == true)
79. {
80.     startSearch += offset;
81. }
82. else
83. {
84.     // иначе - найдено вхождение подстроки, поиск завершен
85.     foundPosition = startSearch + 1;
86.
87.     break;
88. }
89. }
90.
91. return foundPosition;

```

```

92.         }
93.
94.         /// <summary>
95.         /// Заполнение сдвигов для символов в подстроке
96.         /// </summary>
97.         /// <param name="search">Подстрока</param>
98.         /// <returns></returns>
99.         private static int[] FillShiftArray(string search)
100.        {
101.            // размер таблицы сдвигов равен длине подстроки
102.            var shiftArray = new int[search.Length];
103.
104.            // для каждого символа в строке вычисляется расстояние от
            конца
105.            // подстроки до последнего вхождения аналогичного символа
106.            for (int i = 0; i < search.Length; i++)
107.            {
108.                var c = search[i];
109.
110.                shiftArray[i] = search.Length;
111.
112.                for (int j = search.Length - 1; j >= 0; j--)
113.                {
114.                    if (c == search[j])
115.                    {
116.                        int lastOccurenceIdx = j;
117.                        shiftArray[i] = search.Length - j - 1;
118.
119.                        break;
120.                    }
121.                }
122.            }
123.
124.            return shiftArray;
125.        }
126.    }
127. }

```

2. Файл «OpenHashTable.cs» - хеш-таблица.

```

1. using System.Collections;
2. using System.Diagnostics.CodeAnalysis;
3.
4. namespace SaodCP.DataStructures
5. {
6.     /// <summary>
7.     /// Хеш-таблица, открытое хеширование
8.     /// </summary>
9.     public class OpenHashTable<T, O> : IDictionary<T, O>
10.    {

```



```

11.         // стартовое количество бакетов (размер таблицы)
12.         protected static readonly int START_BUCKETS_NUM = 2048;
13.
14.         // таблица - массив СВЯЗНЫХ СПИСКОВ, элементами
15.         // которых являются пары ключ-значение
16.         protected OneWayCycledList<KeyValuePair<T, O>>[] keyValuePairs =
17.             new OneWayCycledList<KeyValuePair<T, O>>(START_BUCKETS_NUM);
18.
19.         protected void Init()
20.         {
21.             keyValuePairs = new OneWayCycledList<KeyValuePair<T,
22.             O>>(START_BUCKETS_NUM);
23.             Count = 0;
24.         }
25.         /// <summary>
26.         /// Расширение хеш-корзины в 2 раза
27.         /// </summary>
28.         protected void Expand()
29.         {
30.             var oldKeyValuePairs = keyValuePairs;
31.
32.             keyValuePairs = new OneWayCycledList<KeyValuePair<T,
33.             O>>(keyValuePairs.Length * 2);
34.             Count = 0;
35.
36.             foreach (var list in oldKeyValuePairs)
37.             {
38.                 if (list == null)
39.                 {
40.                     continue;
41.                 }
42.                 foreach (var pair in list)
43.                 {
44.                     this.Add(pair);
45.                 }
46.             }
47.         }
48.
49.         /// <summary>
50.         /// Расчет ячейки таблицы для ключа
51.         /// </summary>
52.         /// <param name="key"></param>
53.         /// <returns></returns>
54.         protected int GetBucketNumberByKey(T key)
55.         {
56.             var number = key?.GetHashCode() % keyValuePairs.Length
57.             ?? throw new ArgumentNullException(nameof(key));
58.

```

```

59.         return number >= 0
60.         ? number
61.         : -number;
62.     }
63.
64.     public O this[T key]
65.     {
66.         get
67.         {
68.             int bucket = GetBucketNumberByKey(key);
69.
70.             var list = keyValuePairs[bucket];
71.
72.             if (list == null)
73.             {
74.                 return default;
75.             }
76.
77.             foreach (var pair in list)
78.             {
79.                 if (pair.Key?.Equals(key) ?? false)
80.                 {
81.                     return pair.Value;
82.                 }
83.             }
84.
85.             return default;
86.         }
87.         set
88.         {
89.             Add(key, value);
90.         }
91.     }
92.
93.     public ICollection<T> Keys => this.Select(kv =>
kv.Key).ToHashSet();
94.
95.     public ICollection<O> Values => this.Select(kv =>
kv.Value).ToHashSet();
96.
97.     public int Count { get; protected set; } = 0;
98.
99.     public bool IsReadOnly => false;
100.
101.     public void Add(T key, O value)
102.     {
103.         int bucket = GetBucketNumberByKey(key);
104.
105.         if (keyValuePairs[bucket] == null)
106.         {

```

```

107.             keyValuePairs[bucket] = new();
108.         }
109.
110.         var list = keyValuePairs[bucket];
111.
112.         // если длина списка превышает допустимое значение
113.         // попытка учесть и плохое распределение, когда
        заполняется мало ячеек
114.         // и наоборот хорошее, когда элементы равномерно
        "размазываются" по всем спискам
115.         if (list.Count >= keyValuePairs.Length
116.             || Count > keyValuePairs.Length * 10)
117.         {
118.             Expand();
119.
120.             Add(key, value);
121.
122.             return;
123.         }
124.
125.         KeyValuePair<T, O>? kv = null;
126.
127.         foreach (var kvPair in list)
128.         {
129.             if (kvPair.Key?.Equals(key) ?? false)
130.             {
131.                 kv = kvPair;
132.             }
133.         }
134.
135.         if (kv != null)
136.         {
137.             list.Remove((KeyValuePair<T, O>) kv);
138.         }
139.
140.         list.Add(new(key, value));
141.
142.         Count++;
143.     }
144.
145.     public void Add(KeyValuePair<T, O> item)
146.     {
147.         Add(item.Key, item.Value);
148.     }
149.
150.     public void Clear()
151.     {
152.         Init();
153.     }
154.

```

```

155.         public bool Contains(KeyValuePair<T, O> item)
156.         {
157.             var bucketNum = GetBucketNumberByKey(item.Key);
158.
159.             var list = keyValuePairs[bucketNum];
160.
161.             if (list == null)
162.             {
163.                 return false;
164.             }
165.
166.             foreach (var kv in list)
167.             {
168.                 if (kv.Equals(item))
169.                 {
170.                     return true;
171.                 }
172.             }
173.
174.             return false;
175.         }
176.
177.         public bool ContainsKey(T key)
178.         {
179.             var bucketNum = GetBucketNumberByKey(key);
180.
181.             var list = keyValuePairs[bucketNum];
182.
183.             if (list == null)
184.             {
185.                 return false;
186.             }
187.
188.             foreach (var kv in list)
189.             {
190.                 if (kv.Key?.Equals(key) ?? false)
191.                 {
192.                     return true;
193.                 }
194.             }
195.
196.             return false;
197.         }
198.
199.         public void CopyTo(KeyValuePair<T, O>[] array, int
arrayIndex)
200.         {
201.             foreach (var kv in this)
202.             {
203.                 array[arrayIndex++] = kv;

```

```

204.         }
205.     }
206.
207.     public IEnumerator<KeyValuePair<T, O>> GetEnumerator()
208.     {
209.         return new OpenHashTableEnumerator(this);
210.     }
211.
212.     /// <summary>
213.     /// Удалить пару из таблицы по ключу
214.     /// </summary>
215.     /// <param name="key"></param>
216.     /// <returns></returns>
217.     public bool Remove(T key)
218.     {
219.         var bucket = GetBucketNumberByKey(key);
220.
221.         var list = keyValuePairs[bucket];
222.
223.         if (list == null)
224.         {
225.             return false;
226.         }
227.
228.         KeyValuePair<T, O>? keyValue = null;
229.
230.         foreach (var kv in list)
231.         {
232.             if (kv.Key?.Equals(key) ?? false)
233.             {
234.                 keyValue = kv;
235.             }
236.         }
237.
238.         if (keyValue != null)
239.         {
240.             var ret = list.Remove((KeyValuePair<T, O>)keyValue);
241.
242.             if (ret)
243.             {
244.                 Count--;
245.             }
246.
247.             return ret;
248.         }
249.
250.         return false;
251.     }
252.
253.     /// <summary>

```

```

254.         /// Удалить пару из таблицы
255.         /// </summary>
256.         /// <param name="item"></param>
257.         /// <returns></returns>
258.         public bool Remove(KeyValuePair<T, O> item)
259.         {
260.             var bucket = GetBucketNumberByKey(item.Key);
261.
262.             var list = keyValuePairs[bucket];
263.
264.             if (list == null)
265.             {
266.                 return false;
267.             }
268.
269.             KeyValuePair<T, O>? keyValue = null;
270.
271.             foreach (var kv in list)
272.             {
273.                 if (kv.Equals(item))
274.                 {
275.                     keyValue = kv;
276.                 }
277.             }
278.
279.             if (keyValue != null)
280.             {
281.                 var ret = list.Remove((KeyValuePair<T, O>)keyValue);
282.
283.                 if (ret)
284.                 {
285.                     Count--;
286.                 }
287.
288.                 return ret;
289.             }
290.
291.             return false;
292.         }
293.
294.         /// <summary>
295.         /// Получить значение
296.         /// </summary>
297.         public bool TryGetValue(T key, [MaybeNullWhen(false)] out O
298.             value)
299.         {
300.             if (!ContainsKey(key))
301.             {
302.                 value = default;

```

```

303.             return false;
304.         }
305.
306.         value = this[key];
307.
308.         return true;
309.     }
310.
311.     IEnumerator IEnumerable.GetEnumerator()
312.     {
313.         return this.GetEnumerator();
314.     }
315.
316.     public class OpenHashTableEnumerator :
        IEnumerator<KeyValuePair<T, O>>
317.     {
318.         private readonly OpenHashTable<T, O> _table;
319.
320.         private int _currentBucket = -1;
321.
322.         private IEnumerator<KeyValuePair<T, O>>?
            _currentListEnumerator;
323.
324.         private bool _isDisposed;
325.
326.         public OpenHashTableEnumerator(OpenHashTable<T, O> table)
327.         {
328.             _table = table;
329.         }
330.
331.         public KeyValuePair<T, O> Current =>
            _currentListEnumerator?.Current ?? default;
332.
333.         object IEnumerator.Current => Current;
334.
335.         public void Dispose()
336.         {
337.             if (_isDisposed)
338.             {
339.                 throw new
                    ObjectDisposedException(GetType().FullName);
340.             }
341.
342.             _isDisposed = true;
343.         }
344.
345.         public bool MoveNext()
346.         {
347.             // работа с текущим итератором списка
348.             if (_currentListEnumerator != null)

```

```

349.         {
350.             if (_currentListEnumerator.MoveNext())
351.             {
352.                 return true;
353.             }
354.             else
355.             {
356.                 _currentListEnumerator = null;
357.             }
358.         }
359.
360.         OneWayCycledList<KeyValuePair<T, O>>? list = null;
361.
362.         while (list == null
363.             && _currentBucket < _table.keyValuePairs.Length -
364.             1)
365.         {
366.             list = _table.keyValuePairs[++_currentBucket];
367.         }
368.
369.         if (list == null)
370.         {
371.             return false;
372.         }
373.         else
374.         {
375.             _currentListEnumerator = list.GetEnumerator();
376.
377.             if (_currentListEnumerator != null)
378.             {
379.                 return MoveNext();
380.             }
381.             else
382.             {
383.                 return false;
384.             }
385.         }
386.
387.         public void Reset()
388.         {
389.             throw new NotImplementedException();
390.         }
391.     }
392. }
393. }
394.

```

3. Файл «OneWayCycledList.cs» - циклический однонаправленный список


```

1. using System.Collections;
2.
3. namespace SaodCP.DataStructures
4. {
5.     /// <summary>
6.     /// Односвязный циклический список
7.     /// </summary>
8.     /// <typeparam name="T"></typeparam>
9.     public class OneWayCycledList<T> : ICollection<T>, IEnumerator<T>
10.    {
11.        protected OneWayLinkedListElement<T>? _first;
12.        protected OneWayLinkedListElement<T>? _last;
13.        protected OneWayLinkedListElement<T>? _current;
14.
15.        private bool _isDisposed = false;
16.
17.        public int Count { get; protected set; } = 0;
18.
19.        public bool IsReadOnly => false;
20.
21.        public T Current
22.        {
23.            get
24.            {
25.                if (_current == null)
26.                {
27.                    throw new NullReferenceException();
28.                }
29.
30.                return _current.Value;
31.            }
32.        }
33.
34.        object IEnumerator.Current => Current;
35.
36.        public void Add(T item)
37.        {
38.            if (_last != null)
39.            {
40.                _last.Next = new (item);
41.
42.                _last = _last.Next;
43.
44.                _last.Next = _first;
45.            }
46.            else
47.            {
48.                _first = new(item);
49.                _last = _first;
50.                _first.Next = _last;

```

```

51.         }
52.
53.         Count++;
54.     }
55.
56.     public void Clear()
57.     {
58.         _first = null;
59.         _last = null;
60.         Count = 0;
61.     }
62.
63.     public bool Contains(T item)
64.     {
65.         var cnt = 0;
66.
67.         if (_first == null)
68.         {
69.             return false;
70.         }
71.
72.         OneWayLinkedListElement<T> element = _first;
73.
74.         while (cnt < Count)
75.         {
76.             if (element.Value?.Equals(item) ?? false)
77.             {
78.                 return true;
79.             }
80.
81.             cnt++;
82.
83.             if (element.Next == null)
84.             {
85.                 return false;
86.             }
87.
88.             element = element.Next;
89.         }
90.
91.         return false;
92.     }
93.
94.     public void CopyTo(T[] array, int arrayIndex)
95.     {
96.         int pos = arrayIndex;
97.
98.         foreach(var element in this)
99.         {
100.             array[pos] = element;

```

```

101.
102.         pos++;
103.     }
104. }
105.
106.     public void Dispose()
107.     {
108.         if (_isDisposed)
109.         {
110.             throw new
ObjectDisposedException(GetType().FullName);
111.         }
112.
113.         _isDisposed = true;
114.     }
115.
116.     public IEnumerator<T> GetEnumerator()
117.     {
118.         _isDisposed = false;
119.         _current = null;
120.
121.         return this;
122.     }
123.
124.     public bool MoveNext()
125.     {
126.         if (_current == null)
127.         {
128.             _current = _first;
129.
130.             return _current != null;
131.         }
132.
133.         if (object.ReferenceEquals(_current, _last))
134.         {
135.             return false;
136.         }
137.
138.         _current = _current.Next;
139.
140.         return true;
141.     }
142.
143.     public bool Remove(T item)
144.     {
145.         if (Count < 1)
146.         {
147.             return false;
148.         }
149.

```

```

150.         if (_first == null)
151.         {
152.             return false;
153.         }
154.
155.         if (_last == null)
156.         {
157.             return false;
158.         }
159.
160.         var current = _first;
161.         OneWayLinkedListElement<T> prev = _last;
162.
163.         do
164.         {
165.             if (current.Value?.Equals(item) ?? false)
166.             {
167.                 prev.Next = current.Next;
168.
169.                 if (object.ReferenceEquals(_first, current))
170.                 {
171.                     _first = current.Next;
172.                 }
173.
174.                 Count--;
175.
176.                 if (Count < 1)
177.                 {
178.                     _first = null;
179.                     _last = null;
180.                 }
181.
182.                 return true;
183.             }
184.
185.             prev = current;
186.
187.             current = current.Next;
188.         }
189.         while (current != null);
190.
191.         return false;
192.     }
193.
194.     public void Reset()
195.     {
196.         _current = null;
197.     }
198.
199.     IEnumerator IEnumerable.GetEnumerator()

```

```

200.         {
201.             return this.GetEnumerator();
202.         }
203.
204.         /// <summary>
205.         /// Отсортировать элементы списка
206.         /// </summary>
207.         /// <param name="comparer"></param>
208.         public void Sort(Comparison<T>? comparer = null)
209.         {
210.             if (_first == null)
211.             {
212.                 return;
213.             }
214.
215.             if (_first.Value == null)
216.             {
217.                 return;
218.             }
219.
220.             T[] source = new T[Count];
221.
222.             CopyTo(source, 0);
223.
224.             var dest = Utils.Utils.MergeSort(source, comparer);
225.
226.             Clear();
227.
228.             foreach (var item in dest)
229.             {
230.                 Add(item);
231.             }
232.         }
233.     }
234.
235.     /// <summary>
236.     /// Элемент односвязного списка
237.     /// </summary>
238.     /// <typeparam name="T"></typeparam>
239.     public class OneWayLinkedListElement<T> : ICloneable
240.     {
241.         private readonly T _value;
242.
243.         public OneWayLinkedListElement(T value)
244.         {
245.             _value = value;
246.         }
247.
248.         public T Value { get => _value; }
249.

```

```

250.         public OneWayLinkedListElement<T>? Next { get; set; }
251.
252.         public object Clone()
253.         {
254.             return MemberwiseClone();
255.         }
256.     }
257. }
258.

```

4. Файл «SortedOneWayCycledList.cs» - отсортированный односвязный список

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.
7. namespace SaodCP.DataStructures
8. {
9.     /// <summary>
10.    /// Отсортированный список
11.    /// </summary>
12.    /// <typeparam name="T"></typeparam>
13.    public class SortedOneWayCycledList<T> : OneWayCycledList<T>
14.    {
15.        private Comparison<T> _comparison;
16.
17.        /// <summary>
18.        /// Делегат для сравнения элементов
19.        /// </summary>
20.        public Comparison<T> Comparison
21.        {
22.            get => _comparison;
23.            set
24.            {
25.                _comparison = value;
26.
27.                if (Count > 1)
28.                {
29.                    Sort();
30.                }
31.            }
32.        }
33.
34.        /// <summary>
35.        /// Создает список для типа, являющегося IComparable
36.        /// </summary>
37.        public SortedOneWayCycledList()
38.        {

```

```

39.         var type = typeof(T);
40.
41.         if (!typeof(Comparable<T>).IsAssignableFrom(type))
42.         {
43.             throw new ApplicationException($"{nameof(T)} is not
{nameof(Comparable)}")
44.                 + ", use constructor with a Comparison instead");
45.         }
46.
47.         _comparison = (f, s) =>
48.         {
49.             if (f != null)
50.             {
51.                 return ((Comparable)f).CompareTo(s);
52.             }
53.             else
54.             {
55.                 throw new ArgumentNullException(nameof(f));
56.             }
57.         };
58.     }
59.
60.     public SortedOneWayCycledList(Comparison<T> comparison)
61.     {
62.         _comparison = comparison;
63.     }
64.
65.     /// <summary>
66.     /// Добавление элемента, затем сортировка списка
67.     /// Работает за  $O(n \log(n))$ 
68.     /// </summary>
69.     /// <param name="element"></param>
70.     public new void Add(T element)
71.     {
72.         base.Add(element);
73.
74.         Sort(_comparison);
75.     }
76. }
77. }
78.

```

5. Файл «Tree.cs» - бинарное дерево поиска

```

1. using System.Drawing.Design;
2. using System.Net.Sockets;
3.
4. namespace SaodCP.DataStructures
5. {
6.     /// <summary>

```

```

7.      /// AVL-дерево поиска
8.      /// </summary>
9.      /// <typeparam name="T"></typeparam>
10.     /// <typeparam name="O"></typeparam>
11.     public partial class Tree<T, O>
12.     {
13.         private TreeNode<T, O>? head = null;
14.
15.         private Comparison<T> _comparison;
16.
17.         public int Count { get; private set; } = 0;
18.
19.         public Tree()
20.         {
21.             if (!typeof(Comparable).IsAssignableFrom(typeof(T)))
22.             {
23.                 throw new ApplicationException(
24.                     $"{nameof(T)} is not Comparable and
25.                     Comparison<{nameof(T)}> was not provided");
26.             }
27.             _comparison = (first, second) =>
28.                 ((Comparable)first).CompareTo(second);
29.
30.             public Tree(Comparison<T> comparison)
31.             {
32.                 _comparison = comparison;
33.             }
34.
35.             /// <summary>
36.             /// Добавить пару ключ-значение
37.             /// </summary>
38.             /// <param name="key"></param>
39.             /// <param name="value"></param>
40.             public void Add(T key, O? value = default)
41.             {
42.                 var node = new TreeNode<T, O>(key, value);
43.
44.                 if (head == null)
45.                 {
46.                     head = node;
47.
48.                     Count = 1;
49.
50.                     return;
51.                 }
52.
53.                 head = RecursiveAdd(head, node);
54.             }

```



```

55.
56.     private TreeNode<T, O> RecursiveAdd(
57.         TreeNode<T, O> parent,
58.         TreeNode<T, O> addNode)
59.     {
60.         var cmp = _comparison(parent.Key, addNode.Key);
61.
62.         if (cmp > 0)
63.         {
64.             if (parent.Left != null)
65.             {
66.                 parent.Left = RecursiveAdd(parent.Left, addNode);
67.             }
68.             else
69.             {
70.                 parent.Left = addNode;
71.
72.                 Count++;
73.             }
74.
75.             parent.UpdateDimensions();
76.
77.             if (parent.Diff >= 2
78.                 || parent.Diff <= -2)
79.             {
80.                 parent = Rebalance(parent);
81.             }
82.
83.             return parent;
84.         }
85.         else if (cmp < 0)
86.         {
87.             if (parent.Right != null)
88.             {
89.                 parent.Right = RecursiveAdd(parent.Right, addNode);
90.             }
91.             else
92.             {
93.                 parent.Right = addNode;
94.
95.                 Count++;
96.             }
97.
98.             parent.UpdateDimensions();
99.
100.             if (parent.Diff >= 2
101.                 || parent.Diff <= -2)
102.             {
103.                 parent = Rebalance(parent);
104.             }

```

```

105.
106.         return parent;
107.     }
108.     else
109.     {
110.         parent.Value = addNode.Value;
111.
112.         parent.UpdateDimensions();
113.
114.         return parent;
115.     }
116. }
117.
118. /// <summary>
119. /// Удалить по ключу
120. /// </summary>
121. /// <param name="key"></param>
122. /// <returns></returns>
123. public bool Remove(T key)
124. {
125.     if (head == null)
126.     {
127.         return false;
128.     }
129.
130.     bool ret = false;
131.
132.     head = DeleteNode(head, key, ref ret);
133.
134.     if (ret)
135.     {
136.         Count--;
137.     }
138.
139.     return ret;
140. }
141.
142. private TreeNode<T, O>? DeleteNode(
143.     TreeNode<T, O> currentNode,
144.     T itemValue)
145. {
146.     bool stackBool = false;
147.
148.     return DeleteNode(currentNode, itemValue, ref stackBool);
149. }
150.
151. // найти и удалить
152. private TreeNode<T,O>? DeleteNode(
153.     TreeNode<T,O> currentNode,
154.     T itemValue,

```

```

155.         ref bool result)
156.     {
157.         result = true;
158.
159.         var cmp = _comparison(currentNode.Key, itemValue);
160.
161.         // если нашли нужный элемент, начинаем процедуру удаления
162.         if (cmp == 0)
163.         {
164.             // обработка самого простого случая, вместо узла
возвращается null
165.             if (currentNode.Left == null
166.                 && currentNode.Right == null)
167.             {
168.                 return null;
169.             }
170.
171.             // обработка двух случаев, с только одним из
поддеревьев
172.             if (currentNode.Left == null)
173.             {
174.                 return currentNode.Right;
175.             }
176.
177.             if (currentNode.Right == null)
178.             {
179.                 return currentNode.Left;
180.             }
181.
182.             // если у ноды есть оба потомка
183.             var minNodeInRightSubtree =
FindMinNode(currentNode.Right);
184.             // заменили текущий элемент минимальным из правого
поддерева
185.             currentNode.Value = minNodeInRightSubtree.Value;
186.             currentNode.Key = minNodeInRightSubtree.Key;
187.
188.             // ищем в правом поддереве минимальный элемент,
// значение которого уже вставлено на место текущего
190.             currentNode.Right = DeleteNode(
191.                 currentNode.Right,
192.                 minNodeInRightSubtree.Key);
193.
194.             return currentNode;
195.         }
196.
197.         // попадаем сюда, если элемент не был найден,
// просто проваливаемся в дерево глубже и глубже
199.
200.         // производится рекурсивный вызов этой же функции

```

```

201.         if (cmp > 0)
202.         {
203.             if (currentNode.Left == null)
204.             {
205.                 result = false;
206.
207.                 return currentNode;
208.             }
209.
210.             // проваливаемся в левое поддерево
211.             currentNode.Left = DeleteNode(
212.                 currentNode.Left,
213.                 itemValue,
214.                 ref result);
215.
216.             // если нового ребенка после удаления надо
балансировать, балансируем
217.             if (currentNode.Left != null)
218.             {
219.                 if (currentNode.Left.Diff == 2
220.                     || currentNode.Left.Diff == -2)
221.                 {
222.                     currentNode.Left =
Rebalance(currentNode.Left);
223.                 }
224.             }
225.
226.             // обновление параметров
227.             currentNode.UpdateDimensions();
228.
229.             // присваивание на рекурсивный уровень выше,
230.             // может быть как в левое поддерево, так и в правое,
231.             // на текущем уровне мы не знаем какое поддерево
обрабатываем
232.             return currentNode;
233.         }
234.
235.         // аналогичная обработка для правого поддерева
236.         if (cmp < 0)
237.         {
238.             if (currentNode.Right == null)
239.             {
240.                 result = false;
241.
242.                 return currentNode;
243.             }
244.
245.             currentNode.Right = DeleteNode(
246.                 currentNode.Right,
247.                 itemValue,

```

```

248.             ref result);
249.
250.             // если ребенка после удаления надо балансировать,
балансируем
251.             if (currentNode.Right != null)
252.             {
253.                 if (currentNode.Right.Diff == 2
254.                     || currentNode.Right.Diff == -2)
255.                 {
256.                     currentNode.Right =
Rebalance(currentNode.Right);
257.                 }
258.             }
259.
260.             currentNode.UpdateDimensions();
261.
262.             return currentNode;
263.         }
264.
265.         throw new ApplicationException("Delete Element error");
266.     }
267.
268.     private TreeNode<T,O> FindMinNode(TreeNode<T, O> node)
269.     {
270.         if (node.Left == null)
271.         {
272.             return node;
273.         }
274.
275.         return FindMinNode(node.Left);
276.     }
277.
278.     /// <summary>
279.     /// Найти значение по ключу
280.     /// </summary>
281.     /// <param name="key"></param>
282.     /// <returns></returns>
283.     public O? Find(T key)
284.     {
285.         if (head == null)
286.         {
287.             return default;
288.         }
289.
290.         return RecursiveFind(head, key);
291.     }
292.
293.     private O? RecursiveFind(TreeNode<T,O> parent, T key)
294.     {
295.         var cmp = _comparison(parent.Key, key);

```

```

296.
297.         if (cmp == 0)
298.         {
299.             return parent.Value;
300.         }
301.         else if (cmp > 0)
302.         {
303.             if (parent.Left == null)
304.             {
305.                 return default;
306.             }
307.
308.             return RecursiveFind(parent.Left, key);
309.         }
310.         else
311.         {
312.             if (parent.Right == null)
313.             {
314.                 return default;
315.             }
316.
317.             return RecursiveFind(parent.Right, key);
318.         }
319.     }
320. }
321.

```

6. Файл «Tree_Enumerable.cs» - АВЛ-дерево поиска, реализация интерфейса перечисляемого типа.

```

1. using System.Collections;
2.
3. namespace SaodCP.DataStructures
4. {
5.     /// <summary>
6.     /// Реализация IEnumerable для Tree, чтобы было удобно использовать
7.     /// Используется прямой обход дерева (вершина, левый, правый)
8.     /// </summary>
9.     /// <typeparam name="T"></typeparam>
10.    /// <typeparam name="O"></typeparam>
11.    public partial class Tree<T, O> : IEnumerable<KeyValuePair<T, O>>
12.    {
13.        /// <summary>
14.        /// Получить Enumerator (обход дерева в прямом порядке)
15.        /// </summary>
16.        /// <returns></returns>
17.        public IEnumerator<KeyValuePair<T, O>> GetEnumerator()
18.        {
19.            var pairs = new List<KeyValuePair<T, O>>();
20.

```

```

21.         var node = head;
22.
23.         AddToList (node);
24.
25.         void AddToList (TreeNode<T,O>? node)
26.         {
27.             if (node == null)
28.             {
29.                 return;
30.             }
31.
32.             pairs.Add (new (node.Key, node.Value));
33.
34.             AddToList (node.Left);
35.             AddToList (node.Right);
36.         }
37.
38.         return pairs.GetEnumerator();
39.     }
40.
41.     IEnumerator IEnumerable.GetEnumerator()
42.     {
43.         return GetEnumerator();
44.     }
45. }
46. }
47.

```

7. Файл «Tree_Rebalance.cs» - операции по балансировке AVL-дерева поиска

```

1. namespace SaodCP.DataStructures
2. {
3.     /// <summary>
4.     /// Операции по балансировке дерева
5.     /// </summary>
6.     public partial class Tree<T, O>
7.     {
8.         /// <summary>
9.         /// Балансировка дерева
10.        /// </summary>
11.        /// <param name="head"></param>
12.        /// <returns>Новая вершина поддерева</returns>
13.        private TreeNode<T, O> Rebalance (TreeNode<T, O> head)
14.        {
15.            // надо ли вообще выполнять поворот
16.            if (head.Diff != 2
17.                && head.Diff != -2)
18.            {
19.                return head;
20.            }

```

```

21.
22.     var a = head;
23.
24.     // левый поворот
25.     if (a.Diff == 2)
26.     {
27.         var b = head.Left;
28.
29.         if (b == null)
30.         {
31.             return head;
32.         }
33.
34.         if (b.Diff == 1
35.             || b.Diff == 0)
36.         {
37.             return RotateLeft(head);
38.         }
39.         else if (b.Diff == -1)
40.         {
41.
42.             var c = b.Right;
43.
44.             if (c == null)
45.             {
46.                 return head;
47.             }
48.
49.             return BigRotateLeft(head);
50.         }
51.     }
52.
53.     // правый поворот
54.     if (a.Diff == -2)
55.     {
56.         var b = head.Right;
57.
58.         if (b == null)
59.         {
60.             return head;
61.         }
62.
63.         if (b.Diff == -1
64.             || b.Diff == 0)
65.         {
66.             return RotateRight(head);
67.         }
68.         else if (b.Diff == 1)
69.         {
70.             var c = b.Left;

```



```

71.
72.             if (c == null)
73.             {
74.                 return head;
75.             }
76.
77.             return BigRotateRight(head);
78.         }
79.     }
80.
81.     throw new ApplicationException("Tree Internal error");
82. }
83.
84.     /// <summary>
85.     /// левый поворот
86.     /// </summary>
87.     /// <param name="b"></param>
88.     /// <returns></returns>
89.     /// <exception cref="ArgumentNullException"></exception>
90.     private TreeNode<T, O> RotateLeft(TreeNode<T, O> b)
91.     {
92.         TreeNode<T, O> a = b.Left ?? throw new
NullReferenceException();
93.
94.         b.Left = a.Right;
95.
96.         a.Right = b;
97.
98.         b.UpdateDimensions();
99.         a.UpdateDimensions();
100.
101.         return a;
102.     }
103.
104.     /// <summary>
105.     /// правый поворот
106.     /// </summary>
107.     /// <param name="b"></param>
108.     /// <returns></returns>
109.     /// <exception cref="ArgumentNullException"></exception>
110.     private TreeNode<T, O> RotateRight(TreeNode<T, O> b)
111.     {
112.         TreeNode<T, O> a = b.Right ?? throw new
NullReferenceException();
113.
114.         b.Right = a.Left;
115.
116.         a.Left = b;
117.
118.         b.UpdateDimensions();

```

```

119.             a.UpdateDimensions();
120.
121.             return a;
122.         }
123.
124.         /// <summary>
125.         /// Большой левый поворот
126.         /// </summary>
127.         /// <param name="a"></param>
128.         /// <returns></returns>
129.         /// <exception cref="NullReferenceException"></exception>
130.         private TreeNode<T, O> BigRotateLeft(TreeNode<T, O> a)
131.         {
132.             a.Left = RotateRight(a.Left ?? throw new
NullReferenceException());
133.             return RotateLeft(a);
134.         }
135.
136.         /// <summary>
137.         /// Большой правый поворот
138.         /// </summary>
139.         /// <param name="a"></param>
140.         /// <returns></returns>
141.         /// <exception cref="NullReferenceException"></exception>
142.         private TreeNode<T, O> BigRotateRight(TreeNode<T, O> a)
143.         {
144.             a.Right = RotateLeft(a.Right ?? throw new
NullReferenceException());
145.
146.             return RotateRight(a);
147.         }
148.     }
149. }
150.

```

8. Файл «TreeNode.cs» - узел AVL-дерева

```

1. namespace SaodCP.DataStructures
2. {
3.     /// <summary>
4.     /// Узел дерева
5.     /// </summary>
6.     /// <typeparam name="T"></typeparam>
7.     public class TreeNode<T, O>
8.     {
9.         public T Key { get; set; }
10.        public O? Value { get; set; }
11.
12.        public TreeNode(T key, O? value)
13.        {

```

```

14.         Key = key;
15.         Value = value;
16.     }
17.
18.     public TreeNode<T, O>? Left { get; set; } = null;
19.     public TreeNode<T, O>? Right { get; set; } = null;
20.
21.     /// <summary>
22.     /// Разница между высотой левого и правого поддерева
23.     /// </summary>
24.     public int Diff { get => GetDiff(); }
25.
26.     /// <summary>
27.     /// Глубина поддерева
28.     /// </summary>
29.     public int Depth { get; set; }
30.
31.     /// <summary>
32.     /// Обновить глубину поддерева
33.     /// </summary>
34.     public void UpdateDepth()
35.     {
36.         int rightDepth = Right?.Depth + 1 ?? 0;
37.         int leftDepth = Left?.Depth + 1 ?? 0;
38.
39.         int max = rightDepth > leftDepth ? rightDepth : leftDepth;
40.
41.         Depth = max;
42.     }
43.
44.     /// <summary>
45.     /// Обновить разность
46.     /// глубин поддеревьев
47.     /// </summary>
48.     public int GetDiff()
49.     {
50.         int rightDepth = Right?.Depth + 1 ?? 0;
51.         int leftDepth = Left?.Depth + 1 ?? 0;
52.
53.         return leftDepth - rightDepth;
54.     }
55.
56.     /// <summary>
57.     /// Обновить измерения узла
58.     /// </summary>
59.     public void UpdateDimensions()
60.     {
61.         UpdateDepth();
62.     }
63. }

```

64. }

9. Файл «Accommodation.cs» - Запись данных о заселении

```
1.
2. using System.ComponentModel;
3.
4. namespace SaodCP.Models
5. {
6.     /// <summary>
7.     /// Данные о периоде проживания постояльца в номере
8.     /// </summary>
9.     public class Accommodation : ICloneable
10.    {
11.        /// <summary>
12.        /// Номер паспорта постояльца
13.        /// </summary>
14.        [DisplayName("Номер паспорта")]
15.        public string LodgerPassportId { get; set; } = string.Empty;
16.
17.        /// <summary>
18.        /// Номер комнаты постояльца
19.        /// </summary>
20.        [DisplayName("Номер комнаты")]
21.        public string ApartmentNumber { get; set; } = string.Empty;
22.
23.        /// <summary>
24.        /// Дата заселения постояльца
25.        /// </summary>
26.        [DisplayName("Дата С")]
27.        public DateOnly FromDate { get; set; }
28.
29.        /// <summary>
30.        /// Дата выезда постояльца
31.        /// </summary>
32.        [DisplayName("Дата ПО")]
33.        public DateOnly ToDate { get; set; }
34.
35.        public object Clone()
36.        {
37.            return MemberwiseClone();
38.        }
39.
40.        public override string? ToString()
41.        {
42.            return $"Заселение постояльца " +
43.                $"{this.LodgerPassportId} в номер " +
44.                $"{this.ApartmentNumber} " +
45.                $"с {this.FromDate} по {this.ToDate}";
46.        }
```

```
47.     }
48. }
49.
```

10. Файл «Lodger.cs» - данные постояльца

```
1. using SaodCP.DataStructures;
2. using System.ComponentModel;
3.
4. namespace SaodCP.Models
5. {
6.     /// <summary>
7.     /// Постоялец
8.     /// </summary>
9.     public class Lodger
10.    {
11.        public static readonly string PassportIdPattern = "NNNN-NNNNNN";
12.
13.        private HashString passportId = string.Empty;
14.
15.        [DisplayName("Номер паспорта")]
16.        public string PassportId
17.        {
18.            get => passportId;
19.            set
20.            {
21.                if (Utils.Utils.ValidateLodgerPassportId(value))
22.                {
23.                    passportId = value;
24.                }
25.                else
26.                {
27.                    throw new FormatException("Неверный формат номера
                паспорта!");
28.                }
29.            }
30.        }
31.
32.        [DisplayName("Имя")]
33.        public string Name { get; set; } = string.Empty;
34.
35.        [DisplayName("Год рождения")]
36.        public int BirthYear { get; set; }
37.
38.        [DisplayName("Адрес")]
39.        public string Address { get; set; } = string.Empty;
40.    }
41. }
42.
```

11. Файл «Apartment.cs» - запись с данными о гостиничном номере

```
1. using System.ComponentModel;
2. using System.ComponentModel.DataAnnotations;
3.
4. namespace SaodCP.Models
5. {
6.     /// <summary>
7.     /// Номер в гостинице
8.     /// </summary>
9.     public class Apartment
10.    {
11.        private string _number;
12.
13.        #pragma warning disable CS8618 // Поле, не допускающее значения NULL,
            должно содержать значение, отличное от NULL, при выходе из конструктора.
            Возможно, стоит объявить поле как допускающее значения NULL.
14.        public Apartment(string number)
15.        #pragma warning restore CS8618 // Поле, не допускающее значения NULL,
            должно содержать значение, отличное от NULL, при выходе из конструктора.
            Возможно, стоит объявить поле как допускающее значения NULL.
16.        {
17.            Number = number;
18.        }
19.
20.        /// <summary>
21.        /// Номер апартаментов
22.        /// </summary>
23.        [DisplayName("Номер")]
24.        public string Number
25.        {
26.            get => _number;
27.            set
28.            {
29.                if (!Utils.Utils.ValidateRoomNumberId(value))
30.                {
31.                    throw new ArgumentException(null, nameof(value));
32.                }
33.
34.                _number = value;
35.            }
36.        }
37.
38.        /// <summary>
39.        /// Кол-во спальных мест
40.        /// </summary>
41.        [DisplayName("Кол-во спальных мест")]
42.        public int BedsNumber { get; set; }
43.
44.        /// <summary>
```

```

45.         /// Кол-во комнат
46.         /// </summary>
47.         [DisplayName("Кол-во комнат")]
48.         public int RoomNumber { get; set; }
49.
50.         /// <summary>
51.         /// Есть ли сан.узел
52.         /// </summary>
53.         [DisplayName("Сан. узел")]
54.         public bool HasToilet { get; set; }
55.
56.         /// <summary>
57.         /// Список оборудования
58.         /// </summary>
59.         [StringLength(200)]
60.         [DisplayName("Оборудование")]
61.         public string Equipment { get; set; } = string.Empty;
62.     }
63. }
64.

```

12. Файл «Utils.cs» - инструменты для работы с первичными ключами, реализация сортировки слиянием, генерация случайных первичных ключей

```

1. using SaodCP.Models;
2. namespace SaodCP.Utils
3. {
4.     public static class Utils
5.     {
6.         static readonly char[] NUMBER_CHARS = new char[] { '0', '1', '2',
7.             '3', '4', '5', '6', '7', '8', '9' };
8.
9.         static readonly char[] ROOM_NUMBER_FIRST_LITERAL = new char[] {
10.             'Л', 'П', 'О', 'М' };
11.
12.         /// <summary>
13.         /// Проверка формата номера паспорта постояльца
14.         /// </summary>
15.         public static bool ValidateLodgerPassportId(
16.             string? passportId)
17.         {
18.             var refString = string.Empty;
19.
20.             return ValidateLodgerPassportId(passportId, ref refString);
21.         }
22.
23.         /// <summary>
24.         /// Проверка формата номера паспорта
25.         /// </summary>
26.         public static bool ValidatePassportId(
27.             string? passportId)
28.         {
29.             var refString = string.Empty;
30.
31.             return ValidatePassportId(passportId, ref refString);
32.         }
33.     }
34. }

```

```

24.      /// <param name="passportId">Номер паспорта</param>
25.      /// <param name="error">Описание ошибки</param>
26.      public static bool ValidateLodgerPassportId(
27.          string? passportId,
28.          ref string? error)
29.      {
30.          if (string.IsNullOrEmpty(passportId))
31.          {
32.              error = "Номер паспорта должен быть заполнен";
33.
34.              return false;
35.          }
36.
37.          var ret = true;
38.
39.          if (passportId.Length != 11)
40.          {
41.              ret = false;
42.          }
43.
44.          var allowedCharactersArray = new char[]
45.          {
46.              '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
47.          };
48.
49.          var delimiter = '-';
50.
51.          if (ret)
52.          {
53.              for (var i = 0; i < passportId.Length; i++)
54.              {
55.                  var c = passportId[i];
56.
57.                  if (i == 4)
58.                  {
59.                      if (c == delimiter)
60.                      {
61.                          continue;
62.                      }
63.                      else
64.                      {
65.                          ret = false;
66.
67.                          break;
68.                      }
69.                  }
70.
71.                  var found = false;
72.
73.                  for (var j = 0; j < allowedCharactersArray.Length; j++)

```



```

74.         {
75.             if (c == allowedCharactersArray[j])
76.             {
77.                 found = true;
78.
79.                 break;
80.             }
81.         }
82.
83.         if (!found)
84.         {
85.             ret = false;
86.
87.             break;
88.         }
89.     }
90. }
91.
92. if (!ret)
93. {
94.     error = $"Номер паспорта должен соответствовать шаблону
        {Lodger.PassportIdPattern}";
95. }
96.
97. return ret;
98. }
99.
100.     /// <summary>
101.     /// Проверка формата номера гостиничного номера
102.     /// </summary>
103.     public static bool ValidateRoomNumberId(
104.         string? roomNumber)
105.     {
106.         var str = string.Empty;
107.
108.         return ValidateRoomNumberId(
109.             roomNumber,
110.             ref str);
111.     }
112.
113.     /// <summary>
114.     /// Проверка формата номера комнаты
115.     /// </summary>
116.     /// <param name="passportId">Номер комнаты</param>
117.     /// <param name="error">Описание ошибки</param>
118.     public static bool ValidateRoomNumberId(
119.         string? roomNumber,
120.         ref string? error)
121.     {
122.         bool ret = true;

```

```

123.
124.         var description = "Номер комнаты должен соответствовать
           формату «ANNN», " +
125.             "где А -\r\nбуква, обозначающая тип номера\r\n " +
126.             "(Л - люкс, П - полулюкс, О - одноместный, М -
           многоместный);\r\n " +
127.             "NNN - порядковый номер (цифры)";
128.
129.         if (string.IsNullOrEmpty(roomNumber))
130.         {
131.             ret = false;
132.         }
133.
134.         string number = roomNumber ?? string.Empty;
135.
136.         if (number.Length != 4)
137.         {
138.             ret = false;
139.
140.             error = description;
141.
142.             return ret;
143.         }
144.
145.         var firstLiteral = number[0];
146.
147.         for (int i = 0; i < ROOM_NUMBER_FIRST_LITERAL.Length;
           i++)
148.         {
149.             if (firstLiteral == ROOM_NUMBER_FIRST_LITERAL[i])
150.             {
151.                 break;
152.             }
153.
154.             if (i == ROOM_NUMBER_FIRST_LITERAL.Length - 1)
155.             {
156.                 ret = false;
157.             }
158.         }
159.
160.         for (int i = 1; i < number.Length; i++)
161.         {
162.             for (int j = 0; j < NUMBER_CHARS.Length; j++)
163.             {
164.                 if (number[i] == NUMBER_CHARS[j])
165.                 {
166.                     break;
167.                 }
168.
169.                 if (j == NUMBER_CHARS.Length - 1)

```

```

170.         {
171.             ret = false;
172.         }
173.     }
174.
175.     if (!ret)
176.     {
177.         break;
178.     }
179. }
180.
181. if (!ret)
182. {
183.     error = description;
184. }
185.
186. return ret;
187. }
188.
189. /// <summary>
190. /// Генерация случайного ИД паспорта
191. /// </summary>
192. public static string GenerateRandomPassportId()
193. {
194.     var passportIdCharArray = new char[11];
195.
196.     var random = new Random();
197.
198.     for (int i = 0; i < passportIdCharArray.Length; i++)
199.     {
200.         if (i == 4)
201.         {
202.             passportIdCharArray[i] = '-';
203.
204.             continue;
205.         }
206.
207.         passportIdCharArray[i] =
208.             NUMBER_CHARS[random.Next(NUMBER_CHARS.Length - 1)];
209.     }
210.
211.     return new string(passportIdCharArray);
212. }
213.
214. /// <summary>
215. /// Генерация случайного номера комнаты
216. /// </summary>
217. public static string GenerateRandomRoomNumber()
218. {
219.     var random = new Random();

```

```

219.
220.         char[] ret = new char[4];
221.
222.         for (int i = 0; i < 4; i++)
223.         {
224.             if (i == 0)
225.             {
226.                 ret[i] =
ROOM_NUMBER_FIRST_LITERAL[random.Next(ROOM_NUMBER_FIRST_LITERAL.Length -
1)];
227.             }
228.             else
229.             {
230.                 ret[i] =
NUMBER_CHARS[random.Next(NUMBER_CHARS.Length - 1)];
231.             }
232.         }
233.
234.         return new string(ret);
235.     }
236.
237.     /// <summary>
238.     /// Получить следующий по порядку номер комнаты
239.     /// </summary>
240.     public static string GetNextRoomNumber(string roomNumber)
241.     {
242.         if (!ValidateRoomNumberId(roomNumber))
243.         {
244.             throw new ArgumentException(null,
nameof(roomNumber));
245.         }
246.
247.         int number = 0;
248.
249.         int multiplexor = 1;
250.
251.         for (int i = 3; i > 0; i--)
252.         {
253.             number += NumCharToInt(roomNumber[i]) * multiplexor;
254.
255.             multiplexor *= 10;
256.         }
257.
258.         char[] ret = new char[4];
259.
260.         ret[0] = roomNumber[0];
261.
262.         var newNumberString = string.Format("{0}", number);
263.
264.         ret[1] = newNumberString[0];

```

```

265.         ret[2] = newNumberString[1];
266.         ret[3] = newNumberString[2];
267.
268.         return new string(ret);
269.     }
270.
271.     /// <summary>
272.     /// Перевод символа цифры в численное представление
273.     /// </summary>
274.     public static int NumCharToInt(char c)
275.     {
276.         for (int i = 0; i < NUMBER_CHARS.Length; i++)
277.         {
278.             if (c == NUMBER_CHARS[i])
279.             {
280.                 return i;
281.             }
282.         }
283.
284.         throw new ArgumentOutOfRangeException(nameof(c));
285.     }
286.
287.     /// <summary>
288.     /// The hash code for a String object is computed as
289.     /// s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
290.     /// using int arithmetic,
291.     /// where s[i] is the i-th character of the string,
292.     /// n is the length of the string,
293.     /// and ^ indicates exponentiation.
294.     /// (The hash value of the empty string is zero.)
295.     /// </summary>
296.     /// <param name="_key"></param>
297.     /// <returns></returns>
298.     public static int GetHashCode(this string _key)
299.     {
300.         int h = 0;
301.
302.         if (_key.Length > 0)
303.         {
304.             char[] val = _key.ToCharArray();
305.
306.             for (int i = 0; i < _key.Length; i++)
307.             {
308.                 h = 31 * h + val[i];
309.             }
310.         }
311.
312.         return (h);
313.     }
314.

```

```

315.         /// <summary>
316.         /// Сортировка слиянием
317.         /// </summary>
318.         /// <typeparam name="T"></typeparam>
319.         /// <param name="array">Исходный массив</param>
320.         /// <param name="comparer">Сравнение элементов</param>
321.         /// <returns></returns>
322.         /// <exception cref="ApplicationException"></exception>
323.         /// <exception cref="ArgumentNullException"></exception>
324.         public static T[] MergeSort<T>(
325.             this T[] array,
326.             Comparison<T>? comparer = null)
327.         {
328.             if (array.Length == 0
329.                 || array.Length == 1)
330.             {
331.                 return (T[])array.Clone();
332.             }
333.
334.             Comparison<T> comp;
335.
336.             if (comparer == null)
337.             {
338.                 if (array[0] is not IComparable)
339.                 {
340.                     throw new ApplicationException(
341.                         $"Comparator is not found and type
342. {nameof(T)} is not IComparable");
343.                 }
344.
345.                 comp = (T f, T s) =>
346.                 {
347.                     if (f != null)
348.                     {
349.                         return ((IComparable)f).CompareTo(s);
350.                     }
351.                     else
352.                     {
353.                         throw new ArgumentNullException(nameof(f));
354.                     }
355.                 };
356.             }
357.             else
358.             {
359.                 comp = comparer;
360.
361.                 T[] ret = (T[])array.Clone();
362.
363.                 ret = SortArray(array, 0, array.Length - 1);

```

```

364.
365.         return ret;
366.
367.     T[] SortArray(T[] array, int left, int right)
368.     {
369.         if (left < right)
370.         {
371.             int middle = left + (right - left) / 2;
372.             SortArray(array, left, middle);
373.             SortArray(array, middle + 1, right);
374.             MergeArray(array, left, middle, right);
375.         }
376.
377.         return array;
378.     }
379.
380.     void MergeArray(T[] array, int left, int middle, int
    right)
381.     {
382.         var leftArrayLength = middle - left + 1;
383.         var rightArrayLength = right - middle;
384.         var leftTempArray = new T[leftArrayLength];
385.         var rightTempArray = new T[rightArrayLength];
386.         int i, j;
387.
388.         for (i = 0; i < leftArrayLength; i++)
389.         {
390.             leftTempArray[i] = array[left + i];
391.         }
392.         for (j = 0; j < rightArrayLength; j++)
393.         {
394.             rightTempArray[j] = array[middle + 1 + j];
395.         }
396.
397.         i = 0;
398.         j = 0;
399.         int k = left;
400.
401.         while (i < leftArrayLength
402.             && j < rightArrayLength)
403.         {
404.             var comparison = comp(leftTempArray[i],
    rightTempArray[j]);
405.
406.             if (comparison <= 0)
407.             {
408.                 array[k++] = leftTempArray[i++];
409.             }
410.             else
411.             {

```

```

412.             array[k++] = rightTempArray[j++];
413.         }
414.     }
415.
416.     while (i < leftArrayLength)
417.     {
418.         array[k++] = leftTempArray[i++];
419.     }
420.
421.     while (j < rightArrayLength)
422.     {
423.         array[k++] = rightTempArray[j++];
424.     }
425.     }
426. }
427. }
428. }
429.

```

13. Файл «HostelContext.cs» - хранение данных, обработка заселений

```

1. using SaodCP.DataStructures;
2. using SaodCP.Models;
3.
4. namespace SaodCP.Database
5. {
6.     public static class HostelContext
7.     {
8.         /// <summary>
9.         /// Постояльцы - хеш-таблица
10.        /// </summary>
11.        public static OpenHashTable<string, Lodger> Lodgers { get; set; } =
            new();
12.
13.        /// <summary>
14.        /// Гостиничные номера - AVL-дерево
15.        /// </summary>
16.        public static Tree<string, Apartment> Apartments { get; set; } =
            new();
17.
18.        /// <summary>
19.        /// Данные о заселении - сортированный односвязный циклический
            СПИСОК
20.        /// </summary>
21.        public static SortedOneWayCycledList<Accommodation> Accommodations
            { get; set; } = new(
22.            (a1, a2) => a1.ApartmentNumber.CompareTo(a2.ApartmentNumber));
23.
24.        /// <summary>
25.        /// Сериализация данных для сохранения

```



```

26.         /// </summary>
27.         /// <returns></returns>
28.         public static DataSerializationScheme ToDataSheme()
29.         {
30.             var data = new DataSerializationScheme();
31.
32.             data.Accommodations.AddRange(Accommodations);
33.             data.Lodgers.AddRange(Lodgers.Values);
34.             dataApartments.AddRange(Apartments.Select(kv => kv.Value));
35.
36.             return data;
37.         }
38.
39.         /// <summary>
40.         /// Десериализация данных
41.         /// </summary>
42.         /// <param name="data"></param>
43.         public static void FromDataScheme(DataSerializationScheme data)
44.         {
45.             Lodgers.Clear();
46.
47.             foreach (var lodger in data.Lodgers)
48.             {
49.                 Lodgers.Add(lodger.PassportId, lodger);
50.             }
51.
52.             Accommodations.Clear();
53.
54.             foreach (var accommodation in data.Accommodations)
55.             {
56.                 Accommodations.Add(accommodation);
57.             }
58.
59.             Apartments = new();
60.
61.             foreach (var apartment in data.Apartments)
62.             {
63.                 Apartments.Add(apartment.Number, apartment);
64.             }
65.         }
66.
67.         /// <summary>
68.         /// Заполнить БД тестовыми данными
69.         /// </summary>
70.         public static void InitTestData()
71.         {
72.             var data = new DataSerializationScheme();
73.
74.             data.Apartments.Add(new
                Apartment(Utils.Utils.GenerateRandomRoomNumber())

```

```

75.         {
76.             BedsNumber = 1,
77.             Equipment = "Телевизор, Микроволновка",
78.             HasToilet = true,
79.             RoomNumber = 2
80.         });
81.
82.         data.Apartments.Add(new
            Apartment(Utils.Utils.GenerateRandomRoomNumber())
83.         {
84.             BedsNumber = 2,
85.             Equipment = "Микроволновка",
86.             HasToilet = true,
87.             RoomNumber = 3
88.         });
89.
90.         data.Lodgers.Add(new Lodger()
91.         {
92.             PassportId = Utils.Utils.GenerateRandomPassportId(),
93.             Address = "LA 1151",
94.             BirthYear = 1951,
95.             Name = "Mike"
96.         });
97.
98.         data.Lodgers.Add(new Lodger()
99.         {
100.             PassportId = Utils.Utils.GenerateRandomPassportId(),
101.             Address = "SaintP",
102.             BirthYear = 1976,
103.             Name = "Mikes"
104.         });
105.
106.         FromDataScheme(data);
107.     }
108.
109.     /// <summary>
110.     /// Получить сущность определенного типа по идентификатору
111.     /// </summary>
112.     /// <typeparam name="T">Тип сущности для поиска</typeparam>
113.     /// <param name="id">Идентификатор</param>
114.     /// <returns></returns>
115.     public static T? FindById<T>(string id)
116.     {
117.         if (typeof(T) == typeof(Lodger))
118.         {
119.             Lodger? lodger = Lodgers[id];
120.
121.             if (lodger == null)
122.             {
123.                 return default;

```

```

124.         }
125.
126.         return (T?) (object?) lodger;
127.     }
128.     else if (typeof(T) == typeof(Apartment))
129.     {
130.         Apartment? apartment = Apartments.Find(id);
131.
132.         if (apartment == null)
133.         {
134.             return default;
135.         }
136.
137.         return (T?) (object?) apartment;
138.     }
139.     else
140.     {
141.         return default;
142.     }
143. }
144.
145. /// <summary>
146. /// Добавление нового элемента
147. /// </summary>
148. /// <typeparam name="T">Тип элемента</typeparam>
149. /// <param name="entry">Новый элемент</param>
150. public static void Add<T>(T entry)
151. {
152.     if (entry is Lodger lodger)
153.     {
154.         Lodgers.Add(lodger.PassportId, lodger);
155.     }
156.     else if (entry is Apartment apartment)
157.     {
158.         Apartments.Add(apartment.Number, apartment);
159.     }
160.     else if (entry is Accommodation accommodation)
161.     {
162.         Accommodations.Add(accommodation);
163.     }
164. }
165.
166. /// <summary>
167. /// Выселить постояльца
168. /// </summary>
169. /// <param name="passportId">Номер паспорта
    постояльца</param>
170. /// <param name="roomNumber">Номер комнаты</param>
171. /// <param name="endDate">Дата выселения</param>
172. /// <param name="error">Текст ошибки</param>

```

```

173.         /// <returns>Успех/неуспех</returns>
174.     public static bool EndAccommodation(
175.         string passportId,
176.         string roomNumber,
177.         DateOnly endDate,
178.         ref string error)
179.     {
180.         Accommodation? acc = null;
181.
182.         foreach (var accommodation in Accommodations)
183.         {
184.             var fromDate = accommodation.FromDate;
185.             var toDate = accommodation.ToDate ==
                DateOnly.MinValue
186.                 ? DateOnly.MaxValue
187.                 : accommodation.ToDate;
188.
189.             if (endDate >= fromDate
190.                 && endDate <= toDate)
191.             {
192.                 acc = accommodation;
193.
194.                 break;
195.             }
196.
197.         }
198.
199.         if (acc == null)
200.         {
201.             error = $"Период проживания на дату {endDate} " +
202.                 $"для постояльца {passportId} не найден";
203.
204.             return false;
205.         }
206.         else if (acc.ToDate != DateOnly.MinValue)
207.         {
208.             error = $"{acc} " +
209.                 $"для постояльца {passportId} не является
                открытым. " +
210.                 $"Выселение невозможно";
211.
212.             return false;
213.         }
214.
215.         // с помощью копии объекта проверяем возможность записи
216.         var copy = (Accommodation)acc.Clone();
217.
218.         copy.ToDate = endDate;
219.

```

```

220.         var ret = ValidateAccommodationWrite(copy, out error,
           true);
221.
222.         if (!ret)
223.         {
224.             return ret;
225.         }
226.
227.         // все хорошо, меняем у оригинальной копии дату
228.         acc.ToDate = endDate;
229.
230.         return ret;
231.     }
232.
233.     /// <summary>
234.     /// Заселить постояльца с даты
235.     /// </summary>
236.     /// <param name="passportId">Номер паспорта
        постояльца</param>
237.     /// <param name="roomNumber">Номер комнаты</param>
238.     /// <param name="fromDate">Дата заселения</param>
239.     /// <param name="error">Текст ошибки</param>
240.     /// <returns></returns>
241.     public static bool StartAccommodation(
242.         string passportId,
243.         string roomNumber,
244.         DateOnly fromDate,
245.         ref string error)
246.     {
247.         var acc = new Accommodation()
248.         {
249.             ApartmentNumber = roomNumber,
250.             LodgerPassportId = passportId,
251.             FromDate = fromDate
252.         };
253.
254.         bool ret = ValidateAccommodationWrite(acc, out error);
255.
256.         if (!ret)
257.         {
258.             return ret;
259.         }
260.
261.         Accommodations.Add(acc);
262.
263.         return true;
264.     }
265.
266.     /// <summary>
267.     /// Проверка периода проживания для записи в БД

```

```

268.      /// </summary>
269.      /// <param name="acc"></param>
270.      /// <param name="error"></param>
271.      /// <returns></returns>
272.      public static bool ValidateAccommodationWrite(
273.          Accommodation acc,
274.          out string error,
275.          bool existed = false)
276.      {
277.          bool ret = true;
278.          error = string.Empty;
279.
280.          // клиент и комната, для которых создаем период
проживания
281.          var lodger = FindById<Lodger>(acc.LodgerPassportId);
282.          var room = FindById<Apartment>(acc.ApartmentNumber);
283.
284.          if (lodger == null)
285.          {
286.              error = $"Постоялец с номером {acc.LodgerPassportId}"
287.                  +
288.                  $" не найден";
289.
290.              return false;
291.          }
292.          if (room == null)
293.          {
294.              error = $"Комната с номером {acc.ApartmentNumber}" +
295.                  $" не найдена";
296.
297.              return false;
298.          }
299.
300.          var fromDate = acc.FromDate;
301.          var toDate = acc.ToDate;
302.
303.          if (toDate == DateOnly.MinValue)
304.          {
305.              toDate = DateOnly.MaxValue;
306.          }
307.
308.          var periodLength = toDate.DayNumber - fromDate.DayNumber;
309.
310.          if (periodLength < 1)
311.          {
312.              ret = false;
313.
314.              error = $"Период проживания {acc} имеет " +

```

```

315.                $"некорректную продолжительность {periodLength}
                дн.";
316.
317.                return ret;
318.            }
319.
320.            // проверка на пересечение периодов проживания для
            клиента
321.            foreach (var accommodation in Accommodations)
322.            {
323.                var checkFromDate = accommodation.FromDate;
324.                var checkToDate = accommodation.ToDate;
325.
326.                if (checkToDate == DateOnly.MinValue)
327.                {
328.                    checkToDate = DateOnly.MaxValue;
329.                }
330.
331.                // если заселение уже существует,
332.                // проверяем по первичному ключу
333.                if (existed
334.                    && accommodation.FromDate == acc.FromDate
335.                    && accommodation.LodgerPassportId ==
            acc.LodgerPassportId
336.                    && accommodation.ApartmentNumber ==
            acc.ApartmentNumber)
337.                {
338.                    continue;
339.                }
340.
341.                if (accommodation.LodgerPassportId ==
            acc.LodgerPassportId
342.                    && checkFromDate <= toDate
343.                    && checkToDate >= fromDate)
344.                {
345.                    ret = false;
346.
347.                    error = acc.ToString()
348.                        + "конфликтует с "
349.                        + accommodation.ToString();
350.
351.                    break;
352.                }
353.            }
354.
355.            if (!ret)
356.            {
357.                return ret;
358.            }
359.

```

```

360.         byte[] dateLogdersNumArray = new byte[periodLength];
361.
362.         // проверка на кол-во проживающих в комнате
363.         foreach (var accommodation in Accommodations)
364.         {
365.             var checkFromDate = accommodation.FromDate;
366.             var checkToDate = accommodation.ToDate;
367.
368.             if (checkToDate == DateOnly.MinValue)
369.             {
370.                 checkToDate = DateOnly.MaxValue;
371.             }
372.
373.             // если заселение уже существует,
374.             // проверяем по первичному ключу
375.             if (existed
376.                 && accommodation.FromDate == acc.FromDate
377.                 && accommodation.LodgerPassportId ==
378.                 acc.LodgerPassportId
379.                 && accommodation.ApartmentNumber ==
380.                 acc.ApartmentNumber)
381.             {
382.                 continue;
383.             }
384.
385.             // если заселение по комнате не пересекается,
386.             пропускаем
387.             if (accommodation.ApartmentNumber !=
388.                 acc.ApartmentNumber
389.                 || checkFromDate > toDate
390.                 || checkToDate < fromDate)
391.             {
392.                 continue;
393.             }
394.
395.             // период пересечения найденного заселения с
396.             проверяемым
397.             var periodFromDate = checkFromDate < fromDate
398.                 ? fromDate :
399.                 checkFromDate;
400.
401.             var periodToDate = checkToDate > toDate
402.                 ? toDate :
403.                 checkToDate;
404.
405.             // стартовая дата
406.             uint startIdx = (uint)(periodFromDate.DayNumber -
407.                 fromDate.DayNumber);
408.
409.             //длина периода

```



```

404.                uint length = (uint) (periodToDate.DayNumber -
                periodFromDate.DayNumber);
405.
406.                if (startIdx + length > dateLogdersNumArray.Length)
407.                {
408.                    throw new ApplicationException("Checking internal
                error");
409.                }
410.
411.                for (uint i = startIdx; i < startIdx + length; i++)
412.                {
413.                    dateLogdersNumArray[i]++;
414.
415.                    // если кол-во уже заселенных + 1 (из нового
                заселения)
416.                    // больше чем кол-во кроватей,
417.                    // не разрешаем заселение
418.                    if (dateLogdersNumArray[i] + 1 > room.BedsNumber)
419.                    {
420.                        error = $"На дату {fromDate.AddDays((int)i)}
                " +
421.                            $"кол-во заселенных постояльцев :
                {dateLogdersNumArray[i] + 1}. \r\n" +
422.                            $"Макс. кол-во проживающих в номере
                {room.Number} : " +
423.                            $"{{{room.BedsNumber}}}. \r\nКонфликт с
                {accommodation}";
424.
425.                        ret = false;
426.
427.                        break;
428.                    }
429.                }
430.
431.                if (!ret)
432.                {
433.                    break;
434.                }
435.            }
436.
437.            return ret;
438.        }
439.    }
440. }

```

14. Файл «DataSerializationScheme.cs» - модели для сохранения данных

```

1. using SaodCP.Models;
2. using System.Text.Json;
3.

```

```

4. namespace SaodCP.Database
5. {
6.     /// <summary>
7.     /// Схема для сериализации хранимых данных
8.     /// </summary>
9.     public class DataSerializationScheme
10.    {
11.        public List<Accommodation> Accommodations { get; set; } = new
            List<Accommodation>();
12.
13.        public List<Apartment> Apartments { get; set; } = new
            List<Apartment>();
14.
15.        public List<Lodger> Lodgers { get; set; } = new List<Lodger>();
16.
17.        public async static Task<DataSerializationScheme?>
            FromStream(Stream stream)
18.        {
19.            var data = await
                JsonSerializer.DeserializeAsync<DataSerializationScheme>(stream);
20.
21.            return data;
22.        }
23.
24.        public async static Task<Stream?> ToStream(DataSerializationScheme
            data)
25.        {
26.            var stream = new MemoryStream();
27.
28.            await JsonSerializer.SerializeAsync(stream, data);
29.
30.            return stream;
31.        }
32.    }
33.}

```

15. Файл «Form1.cs» - главное окно программы, обработка пользовательского ввода

```

1. using SaodCP.Database;
2. using SaodCP.Models;
3. using System.Text;
4.
5. namespace SaodCP
6. {
7.     public partial class Hostel : Form
8.     {
9.         public Hostel()
10.        {
11.            InitializeComponent();

```

```

12.         }
13.
14.         /// <summary>
15.         /// Получение данных
16.         /// </summary>
17.         private void BindData()
18.         {
19.             var apartmentsArray = HostelContext
20.                 .Apartments
21.                 .Select(kv => kv.Value)
22.                 .ToArray();
23.
24.             var lodgersGrid = HostelContext
25.                 .Lodgers
26.                 .Select(kv => kv.Value)
27.                 .ToArray();
28.
29.             ApartmentsGrid.DataSource = apartmentsArray;
30.             LodgersGrid.DataSource = lodgersGrid;
31.         }
32.
33.         private void Form1_Load(object sender, EventArgs e)
34.         {
35.             BindData();
36.         }
37.
38.         private void AccButton_Click(object sender, EventArgs e)
39.         {
40.             var current = ApartmentsGrid.CurrentRow?.DataBoundItem;
41.
42.             if (current == null)
43.             {
44.                 MessageBox.Show("Данные не выбраны");
45.             }
46.
47.             if (current is Apartment apartment)
48.             {
49.                 new AccommodationForm(RoomNumber:
50.                     apartment.Number).ShowDialog();
51.             }
52.             else
53.             {
54.                 MessageBox.Show("Данные не выбраны");
55.             }
56.
57.         private void LodgerAccButton_Click(object sender, EventArgs e)
58.         {
59.             var current = LodgersGrid.CurrentRow?.DataBoundItem;
60.

```

```

61.         if (current == null)
62.         {
63.             MessageBox.Show("Данные не выбраны");
64.         }
65.
66.         if (current is Lodger lodger)
67.         {
68.             new AccommodationForm(PassportId: lodger.PassportId)
69.                 .ShowDialog();
70.         }
71.         else
72.         {
73.             MessageBox.Show("Данные не выбраны");
74.         }
75.     }
76.
77.     /// <summary>
78.     /// При каждой активации формы обновляем данные
79.     /// </summary>
80.     private void Hostel_Activated(object sender, EventArgs e)
81.     {
82.         BindData();
83.     }
84.
85.     private void CreateLodgerButton_Click(object sender, EventArgs e)
86.     {
87.         new LodgerForm().ShowDialog();
88.     }
89.
90.     private void ChangeLodgerButton_Click(object sender, EventArgs e)
91.     {
92.         var item = LodgersGrid.CurrentRow?.DataBoundItem;
93.
94.         if (item == null)
95.         {
96.             MessageBox.Show("Постоялец не выбран");
97.
98.             return;
99.         }
100.
101.         if (item is Lodger lodger)
102.         {
103.             new LodgerForm(lodger).ShowDialog();
104.         }
105.         else
106.         {
107.             throw new InvalidCastException();
108.         }
109.     }
110.

```

```

111.         private void ApartmentButton_Click(object sender, EventArgs
           e)
112.         {
113.             new ApartmentForm().ShowDialog();
114.         }
115.
116.         private void ChangeApartmentButton_Click(object sender,
           EventArgs e)
117.         {
118.             var item = LodgersGrid.CurrentRow?.DataBoundItem;
119.
120.             if (item == null)
121.             {
122.                 MessageBox.Show("Комната не выбрана");
123.
124.                 return;
125.             }
126.
127.             if (item is Apartment lodger)
128.             {
129.                 new ApartmentForm(lodger).ShowDialog();
130.             }
131.             else
132.             {
133.                 throw new InvalidCastException();
134.             }
135.         }
136.
137.         private void DeleteApartmentButton_Click(object sender,
           EventArgs e)
138.         {
139.             if (ApartmentsGrid.CurrentRow == null)
140.             {
141.                 MessageBox.Show("Номер не выбран");
142.             }
143.             else
144.             {
145.                 Apartment? apartment =
                   ApartmentsGrid.CurrentRow.DataBoundItem as Apartment;
146.
147.                 if (apartment == null)
148.                 {
149.                     MessageBox.Show("Номер не выбран");
150.                 }
151.                 else
152.                 {
153.                     var hasAcc = false;
154.
155.                     foreach (var acc in HostelContext.Accommodations)
156.                     {

```

```

157.             if (acc.ApartmentNumber == apartment.Number)
158.             {
159.                 hasAcc = true;
160.
161.                 break;
162.             }
163.         }
164.
165.         if (hasAcc)
166.         {
167.             MessageBox.Show("Для комнаты существуют
даннные " +
168.
169.                 "о звселении. " +
170.                 "Удаление невозможно");
171.
172.             return;
173.
174.             var ret =
HostelContext.Apartments.Remove(apartment.Number);
175.
176.             if (!ret)
177.             {
178.                 MessageBox.Show("Не удалось удалить номер");
179.             }
180.             else
181.             {
182.                 BindData();
183.             }
184.         }
185.     }
186. }
187.
188. private void DeleteLodgerButton_Click(object sender,
EventArgs e)
189. {
190.     if (LodgersGrid.CurrentRow == null)
191.     {
192.         MessageBox.Show("Постоялец не выбран");
193.     }
194.     else
195.     {
196.         Lodger? lodger = LodgersGrid.CurrentRow.DataBoundItem
as Lodger;
197.
198.         if (lodger == null)
199.         {
200.             MessageBox.Show("Постоялец не выбран");
201.         }
202.         else

```

```

203.            {
204.                var hasAcc = false;
205.
206.                foreach (var acc in HostelContext.Accommodations)
207.                {
208.                    if (acc.LodgerPassportId ==
lodger.PassportId)
209.                    {
210.                        hasAcc = true;
211.
212.                        break;
213.                    }
214.                }
215.
216.                if (hasAcc)
217.                {
218.                    MessageBox.Show("Для постояльца существуют
данные " +
219.
"о заселении. " +
220.
"Удаление невозможно");
221.
222.                    return;
223.                }
224.
225.                var ret =
HostelContext.Lodgers.Remove(lodger.PassportId);
226.
227.                if (!ret)
228.                {
229.                    MessageBox.Show("Не удалось удалить
постояльца");
230.                }
231.                else
232.                {
233.                    BindData();
234.                }
235.            }
236.        }
237.    }
238.
239.    /// <summary>
240.    /// Поиск номера по коду номера
241.    /// </summary>
242.    private void ApartmentNumberFindButton_Click(object sender,
EventArgs e)
243.    {
244.        var apartmentNumber =
ApartmentNumberFindTextBox.Text.Trim();
245.
246.        if (string.IsNullOrEmpty(apartmentNumber))

```

```

247.         {
248.             MessageBox.Show("Введен некорректный код номера");
249.
250.             return;
251.         }
252.
253.         var apartment =
HostelContextApartments.Find(apartmentNumber);
254.
255.         if (apartment == null)
256.         {
257.             MessageBox.Show($"Номер {apartmentNumber} не
найден");
258.
259.             return;
260.         }
261.
262.         StringBuilder data = ApartmentToStringBuilder(apartment);
263.
264.         var date = DateOnly.FromDateTime(DateTime.Today);
265.
266.         data.AppendLine();
267.
268.         data.AppendLine($"На дату {date} проживают:");
269.
270.         int cnt = 0;
271.
272.         foreach (var acc in HostelContext.Accommodations)
273.         {
274.             if (acc.ApartmentNumber != apartment.Number
275.                 || acc.FromDate > date
276.                 || (acc.ToDate < date
277.                     && acc.ToDate == DateOnly.MinValue))
278.             {
279.                 continue;
280.             }
281.
282.             var lodger =
HostelContextLodgers[acc.LodgerPassportId];
283.
284.             if (lodger == null)
285.             {
286.                 continue;
287.             }
288.
289.             cnt++;
290.
291.             data.AppendLine($"{{cnt}}. {{lodger.PassportId}} -
{{lodger.Name}}");
292.         }

```



```

293.
294.         data.AppendLine($"Всего: {cnt} проживающих");
295.
296.         MessageBox.Show(data.ToString());
297.     }
298.
299.     /// <summary>
300.     /// Преобразовать apartment в текстовое представление
301.     /// </summary>
302.     private static StringBuilder
        ApartmentToStringBuilder(Apartment apartment)
303.     {
304.         var data = new StringBuilder();
305.
306.         data.AppendLine($"Номер {apartment.Number}");
307.
308.         data.AppendLine($"Число комнат -
            {apartment.RoomNumber}");
309.         data.AppendLine($"Число кроватей -
            {apartment.BedsNumber}");
310.         data.AppendLine($"Есть туалет -
            {apartment.HasToilet.ToString()}");
311.         data.AppendLine($"Оборудование - {apartment.Equipment}");
312.
313.         return data;
314.     }
315.
316.     /// <summary>
317.     /// Поиск номера по оборудованию
318.     /// </summary>
319.     private void ApartmentEquipmentFindButton_Click(object
        sender, EventArgs e)
320.     {
321.         var search = ApartmentEquipmentFindTextBox.Text.Trim();
322.
323.         if (string.IsNullOrEmpty(search))
324.         {
325.             MessageBox.Show("Введите строку для поиска");
326.
327.             return;
328.         }
329.
330.         var data = new StringBuilder();
331.
332.         data.AppendLine($"Результаты поиска  '{search}'");
333.         data.AppendLine();
334.
335.         int cnt = 0;
336.

```

```

337.         foreach (var (key, apartment) in
HostelContextApartments)
338.         {
339.             // поиск в тексте
340.             var found = Algorithms.BMTextSearch.TextSearch(
341.                 apartment.Equipment,
342.                 search);
343.
344.             if (found < 1)
345.             {
346.                 continue;
347.             }
348.
349.             cnt++;
350.
351.             data.AppendLine($"{cnt}.
\n{ApartmentToStringBuilder(apartment)}");
352.             data.AppendLine();
353.         }
354.
355.         data.AppendLine($"Всего: {cnt} номеров.");
356.
357.         MessageBox.Show(data.ToString());
358.     }
359.
360.     /// <summary>
361.     /// Поиск постояльца по номеру паспорта
362.     /// </summary>
363.     private void LodgerPassportFindButton_Click(object sender,
EventArgs e)
364.     {
365.         var passportId = LodgerPassportFindTextBox.Text.Trim();
366.
367.         if (string.IsNullOrEmpty(passportId))
368.         {
369.             MessageBox.Show("Введите номер паспорта");
370.
371.             return;
372.         }
373.
374.         var lodger = HostelContext.Lodgers[passportId];
375.
376.         if (lodger == null)
377.         {
378.             MessageBox.Show($"Постоялец {passportId} не найден");
379.
380.             return;
381.         }
382.
383.         var lodgerData = new StringBuilder();

```

```

384.
385.         lodgerData.AppendLine($"Постоялец {lodger.PassportId}
           {lodger.Name}");
386.         lodgerData.AppendLine();
387.         lodgerData.AppendLine($"Адрес - {lodger.Address}");
388.         lodgerData.AppendLine($"Год рождения -
           {lodger.BirthYear}");
389.         lodgerData.AppendLine();
390.
391.         var date = DateOnly.FromDateTime(DateTime.Today);
392.
393.         Accommodation? accommodation = null;
394.
395.         foreach (var acc in HostelContext.Accommodations)
396.         {
397.             if (acc.FromDate < date
398.                 && (acc.ToDate > date
399.                     || acc.ToDate == DateOnly.MinValue)
400.                 && acc.LodgerPassportId == lodger.PassportId)
401.             {
402.                 accommodation = acc;
403.
404.                 break;
405.             }
406.         }
407.
408.         if (accommodation == null)
409.         {
410.             lodgerData.AppendLine($"На {date} данных " +
411.                 $" о проживании не найдено");
412.         }
413.         else
414.         {
415.             lodgerData.AppendLine($"На {date} проживает в " +
416.                 $"номере {accommodation.ApartmentNumber}");
417.         }
418.
419.         MessageBox.Show(lodgerData.ToString());
420.     }
421.
422.     private void LodgerNameFindButton_Click(object sender,
           EventArgs e)
423.     {
424.         var name = LodgerNameFindTextBox.Text.Trim();
425.
426.         if (string.IsNullOrWhiteSpace(name))
427.         {
428.             MessageBox.Show("Введите имя для поиска");
429.
430.             return;

```

```

431.         }
432.
433.         var data = new StringBuilder();
434.
435.         data.AppendLine($"Поиск '{name}'");
436.
437.         int cnt = 0;
438.
439.         foreach (var (passportId, lodger) in
HostelContext.Lodgers)
440.         {
441.             if (Algorithms.BMTextSearch.TextSearch(lodger.Name,
name) > 0)
442.             {
443.                 cnt++;
444.
445.                 data.AppendLine($" {cnt}. {passportId} -
{lodger.Name}");
446.             }
447.         }
448.
449.         MessageBox.Show(data.ToString());
450.     }
451.
452.     /// <summary>
453.     /// Очистить данные о проживающих
454.     /// </summary>
455.     /// <param name="sender"></param>
456.     /// <param name="e"></param>
457.     private void LodgerClearData_Click(object sender, EventArgs
e)
458.     {
459.         HostelContext.Lodgers.Clear();
460.
461.         BindData();
462.     }
463.
464.     private void ClearApartmentsDataButton_Click(object sender,
EventArgs e)
465.     {
466.         HostelContext.Apartments = new();
467.
468.         BindData();
469.     }
470.
471.     private void CreateLodgerAccButton_Click(object sender,
EventArgs e)
472.     {
473.         var item = LodgersGrid.CurrentRow?.DataBoundItem;
474.

```

```

475.         if (item == null)
476.         {
477.             MessageBox.Show("Постоялец не выбран");
478.
479.             return;
480.         }
481.
482.         if (item is Lodger lodger)
483.         {
484.             new EditAccommodationForm(
485.                 type:
486.                 AccommodationOperationType.OpenAccommodation,
487.                 lodgerPassportId: lodger.PassportId)
488.                 .ShowDialog();
489.         }
490.         else
491.         {
492.             MessageBox.Show("Постоялец не выбран");
493.         }
494.
495.     private void DeleteLodgerAccButton_Click(object sender,
496.         EventArgs e)
497.     {
498.         var item = LodgersGrid.CurrentRow?.DataBoundItem;
499.
500.         if (item == null)
501.         {
502.             MessageBox.Show("Постоялец не выбран");
503.
504.             return;
505.         }
506.
507.         if (item is Lodger lodger)
508.         {
509.             new EditAccommodationForm(
510.                 type:
511.                 AccommodationOperationType.CloseAccommodation,
512.                 lodgerPassportId: lodger.PassportId)
513.                 .ShowDialog();
514.         }
515.         else
516.         {
517.             MessageBox.Show("Постоялец не выбран");
518.         }
519.     }

```

16. Файл «LodgerForm.cs» - окно редактирования и создания постояльца

```
1. using SaodCP.Database;
2. using SaodCP.Models;
3.
4. namespace SaodCP
5. {
6.     public partial class LodgerForm : Form
7.     {
8.         public LodgerForm()
9.         {
10.             InitializeComponent();
11.         }
12.
13.         public Lodger? Lodger { get; set; }
14.
15.         public LodgerForm(Lodger lodger) : this()
16.         {
17.             NameTextBox.Text = lodger.Name;
18.             BirthdayTextBox.Text = lodger.BirthYear.ToString();
19.             AddressTextBox.Text = lodger.Address;
20.
21.             // первичный ключ не редактируется
22.             PassportTextBox.Text = lodger.PassportId;
23.             PassportTextBox.ReadOnly = true;
24.
25.             this.Lodger = lodger;
26.         }
27.
28.         private void SaveButton_Click(object sender, EventArgs e)
29.         {
30.             var passport = PassportTextBox.Text;
31.
32.             string? error = null;
33.
34.             if (!Utils.Utils.ValidateLodgerPassportId(
35.                 passport,
36.                 ref error))
37.             {
38.                 MessageBox.Show(error);
39.
40.                 return;
41.             }
42.
43.             var year = BirthdayTextBox.Text;
44.
45.             int onlyYear;
46.
47.             if (!int.TryParse(year, out onlyYear))
48.             {
```

```

49.         MessageBox.Show("Неправильно введен год рождения");
50.
51.         return;
52.     }
53.     else if (onlyYear < 0
54.         || onlyYear > DateOnly.FromDateTime(DateTime.Now).Year)
55.     {
56.         MessageBox.Show("Неправильно введен год рождения");
57.
58.         return;
59.     }
60.
61.     var address = AddressTextBox.Text;
62.
63.     var name = NameTextBox.Text;
64.
65.     if (string.IsNullOrEmpty(name))
66.     {
67.         MessageBox.Show("Введите имя");
68.
69.         return;
70.     }
71.     else if (name.Length < 3)
72.     {
73.         MessageBox.Show("Имя должно быть длинее 3х символов");
74.
75.         return;
76.     }
77.
78.     Lodger editLodger;
79.
80.     if (Lodger == null)
81.     {
82.         editLodger = new Lodger();
83.
84.         editLodger.PassportId = passport;
85.     }
86.     else
87.     {
88.         editLodger = Lodger;
89.     }
90.
91.     editLodger.Address = address;
92.     editLodger.Name = name;
93.     editLodger.BirthYear = onlyYear;
94.
95.     // если не было постояльца, значит создаем
96.     if (Lodger == null)
97.     {

```

```

98.         if
(HostelContext.Lodgers.ContainsKey(editLodger.PassportId))
99.         {
100.             MessageBox.Show($"Постоялец с номером паспорта" +
101.                 $" {editLodger.PassportId} уже существует");
102.
103.             return;
104.         }
105.
106.         HostelContext.Lodgers.Add(passport, editLodger);
107.     }
108.
109.     this.Close();
110. }
111. }
112. }

```

17. Файл «ApartmentForm.cs» - окно редактирования и создания гостиничного номера

```

1. using SaodCP.Database;
2. using SaodCP.Models;
3.
4. namespace SaodCP
5. {
6.     public partial class ApartmentForm : Form
7.     {
8.         public ApartmentForm()
9.         {
10.             InitializeComponent();
11.         }
12.
13.         public Apartment? Apartment { get; set; }
14.
15.         public ApartmentForm(Apartment apartment) : this()
16.         {
17.             this.Apartment = apartment;
18.
19.             ApartmentNumberTextBox.Text = Apartment.Number;
20.             ApartmentNumberTextBox.ReadOnly = true;
21.
22.             RoomNumberTextBox.Text = Apartment.RoomNumber.ToString();
23.             BedsNumberTextBox.Text = Apartment.BedsNumber.ToString();
24.             EquipmentTextBox.Text = Apartment.Equipment;
25.
26.             HasToiletCheckBox.Checked = Apartment.HasToilet;
27.         }
28.
29.         private void SaveButton_Click(object sender, EventArgs e)
30.         {

```



```

31.         var apartmentNumber = ApartmentNumberTextBox.Text.Trim();
32.
33.         string? error = null;
34.
35.         if (!Utils.Utils.ValidateRoomNumberId(
36.             apartmentNumber,
37.             ref error))
38.         {
39.             MessageBox.Show(error);
40.
41.             return;
42.         }
43.
44.         if (!int.TryParse(
45.             BedsNumberTextBox.Text.Trim(),
46.             out int bedsNumber))
47.         {
48.             MessageBox.Show("Неправильный формат кол-ва кроватей");
49.
50.             return;
51.         }
52.
53.         if (!int.TryParse(
54.             BedsNumberTextBox.Text.Trim(),
55.             out int roomNumber))
56.         {
57.             MessageBox.Show("Неправильный формат кол-ва комнат");
58.
59.             return;
60.         }
61.
62.         var equipment = EquipmentTextBox.Text.Trim();
63.
64.         var hasToilet = HasToiletCheckBox.Checked;
65.
66.         Apartment apartment;
67.
68.         if (Apartment == null)
69.         {
70.             apartment = new(apartmentNumber);
71.         }
72.         else
73.         {
74.             apartment = Apartment;
75.         }
76.
77.         apartment.Equipment = equipment;
78.         apartment.RoomNumber = roomNumber;
79.         apartment.BedsNumber = bedsNumber;
80.         apartment.HasToilet = hasToilet;

```

```

81.
82.         // если создаем новый номер - сохраняем
83.         if (Apartment == null)
84.         {
85.             if (HostelContext.Apartments.Find(apartment.Number) !=
null)
86.             {
87.                 MessageBox.Show($"Номер {apartment.Number} уже
существует");
88.
89.                 return;
90.             }
91.
92.             HostelContext.Apartments.Add(
93.                 apartment.Number,
94.                 apartment);
95.         }
96.
97.         Close();
98.     }
99. }
100. }

```

18. Файл «AccommodationForm.cs» - окно просмотра заселений по номеру или постояльцу

```

1. using SaodCP.Database;
2. using SaodCP.Models;
3.
4. namespace SaodCP
5. {
6.     public partial class AccommodationForm : Form
7.     {
8.         public string PassportId { get; set; } = string.Empty;
9.
10.        public string RoomNumber { get; set; } = string.Empty;
11.
12.        public AccommodationForm()
13.        {
14.            InitializeComponent();
15.        }
16.
17.        private void SetLabel()
18.        {
19.            Text = "Заселения";
20.
21.            if (!string.IsNullOrEmpty(PassportId))
22.            {

```

```

23.         Text += $" Постоялец {PassportId}";
24.     }
25.     else if (!string.IsNullOrEmpty(RoomNumber))
26.     {
27.         Text += $" Комната {RoomNumber}";
28.     }
29. }
30.
31. public AccommodationForm(
32.     string? PassportId = null,
33.     string? RoomNumber = null) : this()
34. {
35.     if (!string.IsNullOrEmpty(PassportId))
36.     {
37.         this.PassportId = PassportId;
38.     }
39.
40.     if (!string.IsNullOrEmpty(RoomNumber))
41.     {
42.         this.RoomNumber = RoomNumber;
43.     }
44.
45.     SetLabel();
46. }
47.
48. private Accommodation[] FetchData()
49. {
50.     int cnt = 0;
51.
52.     foreach (var accommodation in HostelContext.Accommodations)
53.     {
54.         if ((string.IsNullOrEmpty(PassportId)
55.             || PassportId == accommodation.LodgerPassportId)
56.             && (string.IsNullOrEmpty(RoomNumber)
57.             || RoomNumber == accommodation.ApartmentNumber))
58.         {
59.             cnt++;
60.         }
61.     }
62.
63.     var ret = new Accommodation[cnt];
64.
65.     var i = 0;
66.
67.     foreach (var accommodation in HostelContext.Accommodations)
68.     {
69.         if ((string.IsNullOrEmpty(PassportId)
70.             || PassportId == accommodation.LodgerPassportId)
71.             && (string.IsNullOrEmpty(RoomNumber)
72.             || RoomNumber == accommodation.ApartmentNumber))

```

```

73.         {
74.             if (i < ret.Length)
75.             {
76.                 ret[i++] = accommodation;
77.             }
78.         }
79.     }
80.
81.     return ret;
82. }
83.
84. private void AccommodationForm_Load(object sender, EventArgs e)
85. {
86.     AccommodationGrid.DataSource = FetchData();
87. }
88.
89. private void RemoveAccButton_Click(object sender, EventArgs e)
90. {
91.     var acc = AccommodationGrid.CurrentRow?.DataBoundItem;
92.
93.     if (acc == null
94.         || acc is not Accommodation)
95.     {
96.         MessageBox.Show("Заселение не выбрано");
97.
98.         return;
99.     }
100.
101.     Accommodation accommodation = acc as Accommodation
102.         ?? throw new NullReferenceException();
103.
104.     HostelContext.Accommodations.Remove(accommodation);
105.
106.     AccommodationGrid.DataSource = FetchData();
107. }
108. }
109. }

```

19. Файл «EditAccomodationForm.cs» - окно создания заселений по постояльцу

```

1. using SaodCP.Database;
2.
3. namespace SaodCP
4. {
5.     public partial class EditAccommodationForm : Form
6.     {
7.         private readonly AccommodationOperationType _type;
8.         private readonly string? _lodgerPassportId;
9.         private readonly string? _apartmentNumber;

```

```

10.
11.     public EditAccommodationForm(
12.         AccommodationOperationType type,
13.         string? lodgerPassportId = null,
14.         string? apartmentNumber = null)
15.     {
16.         InitializeComponent();
17.
18.         var array = new string[HostelContext.Apartments.Count];
19.
20.         int cnt = 0;
21.
22.         foreach (var (number, apartment) in HostelContext.Apartments)
23.         {
24.             array[cnt] = number;
25.
26.             cnt++;
27.         }
28.
29.         ApartmentNumberComboBox.DataSource = array;
30.
31.         if (!string.IsNullOrEmpty(apartmentNumber))
32.         {
33.             ApartmentNumberComboBox.Text = apartmentNumber;
34.         }
35.
36.         if (!string.IsNullOrEmpty(lodgerPassportId))
37.         {
38.             LodgerPassportIdTextBox.Text = lodgerPassportId;
39.         }
40.
41.         switch (type)
42.         {
43.             case AccommodationOperationType.OpenAccommodation:
44.
45.                 Text = "Заселить";
46.
47.                 break;
48.
49.             case AccommodationOperationType.CloseAccommodation:
50.
51.                 Text = "Выселить";
52.
53.                 break;
54.         }
55.
56.         this._type = type;
57.         this._lodgerPassportId = lodgerPassportId;
58.         this._apartmentNumber = apartmentNumber;
59.     }

```

```

60.
61.     private void SaveAccButton_Click(object sender, EventArgs e)
62.     {
63.         var passportId = LodgerPassportIdTextBox.Text.Trim();
64.         var apartmentNumber = ApartmentNumberComboBox.Text.Trim();
65.         var onDate =
            DateOnly.FromDateTime(DateCalendar.SelectionRange.Start);
66.
67.         if (string.IsNullOrEmpty(passportId)
68.             || string.IsNullOrEmpty(apartmentNumber)
69.             || onDate == DateOnly.MinValue)
70.         {
71.             MessageBox.Show("Введены не все данные");
72.
73.             return;
74.         }
75.
76.         string error = string.Empty;
77.         var result = false;
78.
79.         switch (_type)
80.         {
81.             case AccommodationOperationType.OpenAccommodation:
82.
83.                 result = HostelContext.StartAccommodation(
84.                     passportId,
85.                     apartmentNumber,
86.                     onDate,
87.                     ref error);
88.
89.                 break;
90.
91.             case AccommodationOperationType.CloseAccommodation:
92.
93.                 result = HostelContext.EndAccommodation(
94.                     passportId,
95.                     apartmentNumber,
96.                     onDate,
97.                     ref error);
98.
99.                 break;
100.        }
101.
102.        if (!result)
103.        {
104.            MessageBox.Show(error);
105.        }
106.        else
107.        {
108.            Close();

```

```

109.         }
110.     }
111. }
112.
113.     public enum AccommodationOperationType
114.     {
115.         None,
116.         OpenAccommodation,
117.         CloseAccommodation
118.     }
119. }

```

20. Файл «Program.cs» - входная точка программы, инициализация тестовых данных

```

1. using SaodCP.Database;
2.
3. namespace SaodCP
4. {
5.     internal static class Program
6.     {
7.         /// <summary>
8.         /// The main entry point for the application.
9.         /// </summary>
10.        [STAThread]
11.        static void Main()
12.        {
13.            // To customize application configuration such as set high DPI
settings or default font,
14.            // see https://aka.ms/applicationconfiguration.
15.            ApplicationConfiguration.Initialize();
16.
17.            // инициализация тестовых данных
18.            HostelContext.InitTestData();
19.
20.            Application.Run(new Hostel());
21.        }
22.    }
23. }

```