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 |  |  |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ИСПОЛЬЗОВАНИЕ ЗАДАННЫХ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ |
| по дисциплине: Структуры и алгоритмы обработки данных |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 | |  |  |  | Быстров М.Д. |
|  | номер группы | |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | |  | |  |  |  |

|  |  |
| --- | --- |
| Шифр ИНДО | 09.03.04 |

Санкт-Петербург 2023

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc135474197)

[ЗАДАНИЕ 3](#_Toc135474198)

[ВВЕДЕНИЕ 4](#_Toc135474199)

[1. Алгоритмы и структуры данных 5](#_Toc135474200)

[2. Описание программы 8](#_Toc135474201)

[3. Тестирование программы 12](#_Toc135474202)

[ЗАКЛЮЧЕНИЕ 28](#_Toc135474203)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 29](#_Toc135474204)

[ПРИЛОЖЕНИЕ 1 ИСХОДНЫЙ КОД ПРОГРАММЫ 30](#_Toc135474205)

# 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 предприятие, 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.

# Алгоритмы и структуры данных

1. Хеш-таблица с открытым хешированием

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

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

Хеш-таблица построена на свойстве детерминированности хеш-фунцкции, 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 О(1). При возникновении большого количества коллизий эффективность будет снижаться в худшую сторону вплоть до О(N).

1. 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).

1. АВЛ-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.

АВЛ-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.

1. 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.

1. Сортировка слиянием

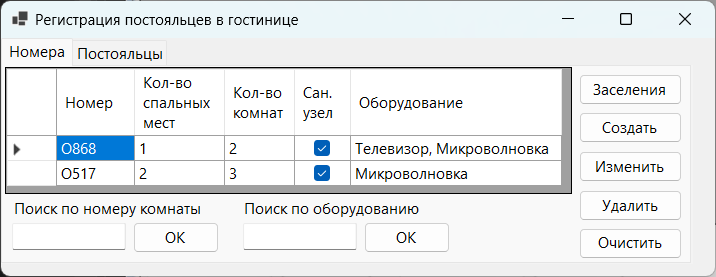
Сортировка слиянием – алгоритм сортировки, работающий за стабильное время *O(nlogn).* Принцип работы заключается в дроблении исходного массива данных на части до интервалов размером 1-2 элемента, их отдельной сортировке и итеративном слиянии получившихся массивов. Требует дополнительного объема памяти O(n).

# Описание программы

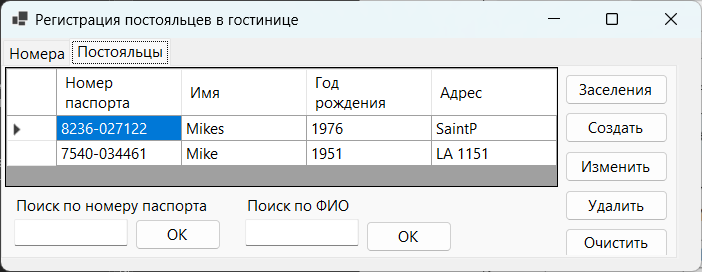
Для разработки программы был выбран язык C#. Программа имеет графический пользовательский интерфейс, построенный с помощью компонентов Windows Forms, входящих в состав платформы .NET.

В ходе работы с программой данные хранятся в определенных вариантом 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.



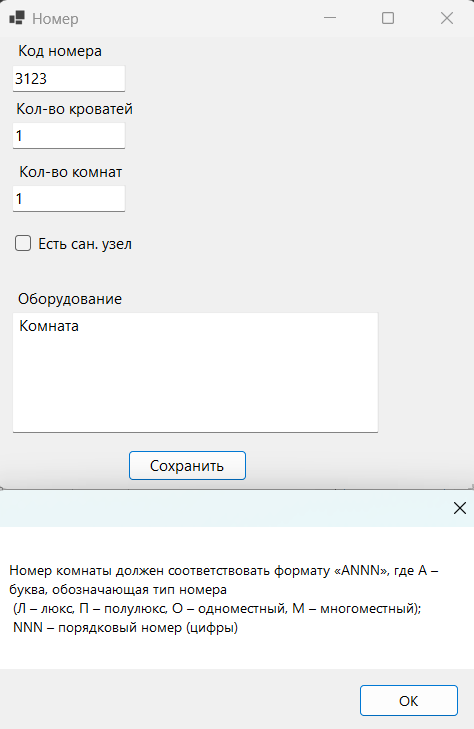
*Рисунок 1 Главный экран, вкладка «Номера»*



*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.

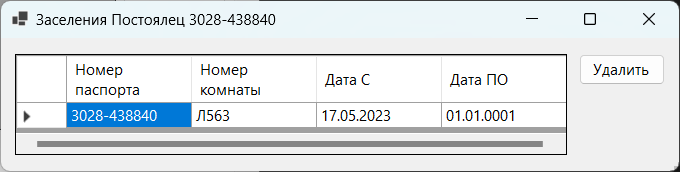
Для добавления данных используется отдельное диалоговое окно. Окно для добавления номера представлено на рисунке 3. При создании записи производится проверка пользовательского ввода.



*Рисунок 3 Создание номера*

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

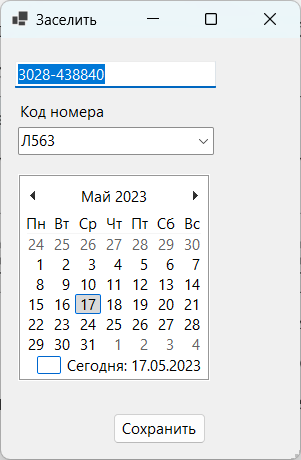
При нажатии на кнопку «Заселения» открывается форма с данными о заселениях. В зависимости от активной вкладки будут отображены данные о заселениях либо для номера, либо для постояльца. Внешний вид формы показан на рисунке 4.



*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.



*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.

# 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;
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;
15. **var found = TextSearch(text, search);**
17. Assert.AreEqual(0, found);
19. text = "qwerty";
20. **search = "qwe";**
22. found = TextSearch(text, search);
24. Assert.AreEqual(1, found);
26. search = "werty";
28. found = TextSearch(text, search);
30. **Assert.AreEqual(2, found);**
32. search = "y";
34. found = TextSearch(text, search);
36. Assert.AreEqual(6, found);
38. search = "qwey";
40. **found = TextSearch(text, search);**
42. Assert.AreEqual(0, found);
43. }
44. }
45. **}**

2. Тестирование односвязного циклического списка

1. **using** Microsoft.VisualStudio.TestPlatform.CrossPlatEngine.Client;
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;
11. **namespace** SaopCPTest
12. {
13. [TestClass]
14. **public** **class** CycledListTest
15. **{**
16. [TestMethod]
17. **public** **void** CreateAddTest()
18. {
19. **var** list = new OneWayCycledList<Lodger>();
21. Assert.IsNotNull(list);
22. Assert.IsTrue(list.Count == 0);
24. **var** lodgerArray = new Lodger[10];
26. **var** rand = new Random();
28. **for** (**int** i = 0; i < lodgerArray.Length; i++)
29. {
30. **lodgerArray[i] = new Lodger();**
32. lodgerArray[i].PassportId = Utils.GenerateRandomPassportId();
33. lodgerArray[i].Name = rand.GetHashCode().ToString();
34. }
36. **for** (**int** i = 0; i < lodgerArray.Length; i++)
37. {
38. list.**Add**(lodgerArray[i]);
39. }
41. Assert.AreEqual(lodgerArray.Length, list.Count);
43. **int** pos = 0;
45. **foreach (var lodger in list)**
46. {
47. Assert.AreEqual(lodger, lodgerArray[pos]);
48. Assert.IsTrue(list.Contains(lodger));
50. **pos++;**
51. }
53. Assert.IsTrue(list.Contains(lodgerArray[0]));
55. **list.Remove(lodgerArray[0]);**
57. Assert.IsFalse(list.Contains(lodgerArray[0]));
58. Assert.AreEqual(list.Count, lodgerArray.Length - 1);
60. **list.Clear();**
62. Assert.AreEqual(0, list.Count);
64. **var** found = **false**;
66. **foreach**(**var** lodger **in** list)
67. {
68. found = **true**;
69. }
71. Assert.IsFalse(found);
72. }
73. }
74. }

3. Тестирование операций заселения и выселения постояльца

1. **using** **static** SaodCP.Database.HostelContext;
3. **namespace** SaopCPTest
4. {
5. **[TestClass]**
6. **public** **class** HostelContextTest
7. {
8. [TestMethod]
9. **public** **void** PeriodAddTest()
10. **{**
11. InitTestData();
13. **var** lodger = Lodgers.First().**Value**;
14. **var** room = Apartments.First().**Value**;
16. **string** error = **string**.Empty;
18. **var** startDate = DateOnly.FromDateTime(DateTime.Now);
20. **var started = StartAccomodation(**
21. lodger.PassportId,
22. room.Number,
23. startDate,
24. **ref** error);
26. Assert.IsTrue(started);
27. Assert.AreEqual(Accommodations.Count, 1);
29. *// пытаемся повторно заселить того*
30. ***// же постояльца в то же место***
31. started = StartAccomodation(
32. lodger.PassportId,
33. room.Number,
34. startDate,
35. **ref error);**
37. Assert.IsFalse(started);
38. Assert.IsFalse(**string**.IsNullOrWhiteSpace(error));
40. ***// на дату позже***
41. started = StartAccomodation(
42. lodger.PassportId,
43. room.Number,
44. startDate.AddDays(5),
45. **ref error);**
47. Assert.IsFalse(started);
48. Assert.IsFalse(**string**.IsNullOrWhiteSpace(error));
50. ***// пытаемся выселить на день раньше чем заселили***
51. **var** ended = EndAccomodation(
52. lodger.PassportId,
53. room.Number,
54. startDate.AddDays(-1),
55. **ref error);**
57. Assert.IsFalse(ended);
58. Assert.IsFalse(**string**.IsNullOrWhiteSpace(error));
60. ***// выселяем успешно***
61. ended = EndAccomodation(
62. lodger.PassportId,
63. room.Number,
64. startDate.AddDays(5),
65. **ref error);**
67. Assert.IsTrue(ended);
68. Assert.IsTrue(**string**.IsNullOrWhiteSpace(error));
70. ***// выселяем повторно (должна быть ошибка)***
71. ended = EndAccomodation(
72. lodger.PassportId,
73. room.Number,
74. startDate.AddDays(5),
75. **ref error);**
77. Assert.IsFalse(ended);
78. Assert.IsFalse(**string**.IsNullOrWhiteSpace(error));
79. }
80. **}**
81. }

4. Тестирование создания постояльца и генерации ИД паспорта

1. **using** SaodCP.Models;
2. **using** SaodCP.Utils;
3. **using** System.Text.RegularExpressions;
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";
18. **for** (**int** i = 0; i < 1000; i++)
19. {
20. **var id = Utils.GenerateRandomPassportId();**
22. Assert.IsTrue(Regex.IsMatch(id, pattern));
23. }
24. }
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();
36. Assert.IsTrue(Utils.ValidateLodgerPassportId(id));
37. }
39. **var** passportId = **string**.Empty;
41. Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
43. passportId = "0000-00000A";
45. **Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));**
47. passportId = "0000000000";
49. Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
51. passportId = "0000-1231234";
53. Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
55. **passportId = "00001-231234";**
57. Assert.IsFalse(Utils.ValidateLodgerPassportId(passportId));
58. }
60. ***/// <summary>***
61. */// Тестирование создания постояльца и генерации ИД паспорта*
62. */// </summary>*
63. [TestMethod]
64. **public** **void** CreateLodgerTest()
65. **{**
66. **var** lodger = new Lodger();
68. Assert.IsNotNull(lodger);
70. **lodger.PassportId = Utils.GenerateRandomPassportId();**
72. Assert.IsTrue(Utils.ValidateLodgerPassportId(lodger.PassportId));
73. }
74. }
75. **}**
    1. Тестирование хеш-таблицы
76. **using** SaodCP.DataStructures;
78. **namespace** SaopCPTest
79. {
80. ***/// <summary>***
81. */// Тестирование хеш-таблицы*
82. */// </summary>*
83. [TestClass]
84. **public** **class** OpenHashTableTest
85. **{**
86. */// <summary>*
87. */// Создать, добавить, получить*
88. */// </summary>*
89. [TestMethod]
90. **public void CreateAddGetTest()**
91. {
92. OpenHashTable<**string**, **string**> hashTable = new();
94. Assert.AreEqual(0, hashTable.Count);
96. hashTable.**Add**("1", "2");
98. Assert.AreEqual(1, hashTable.Count);
100. **var value = hashTable["1"];**
102. Assert.AreEqual("2", **value**);
103. }
105. ***/// <summary>***
106. */// Нагрузочное тестирование - добавление 1000000 элементов*
107. */// Тестирование удаление элементов*
108. */// Тестирование перечисления элементов*
109. */// </summary>*
110. **[TestMethod]**
111. **public** **void** RebalanceRemoveTest()
112. {
113. Random rnd = new Random();
115. ***// на 1000000 отрабатывает за 6с на i7-10610U***
116. **int** cnt = 1000000;
118. Dictionary<**string**, **string**> pairs = new();
120. **for (int i = 0; i < cnt; i++)**
121. {
122. **var** key = rnd.Next().ToString();
124. **if** (!pairs.ContainsKey(key))
125. **{**
126. pairs.**Add**(key, rnd.Next().ToString());
127. }
128. }
130. **OpenHashTable<string, string> hashTable = new();**
132. **int** count = 0;
134. **foreach** (KeyValuePair<**string**, **string**> pair **in** pairs)
135. **{**
136. hashTable.**Add**(pair.Key, pair.**Value**);
138. count++;
140. **Assert.AreEqual(count, hashTable.Count);**
141. }
143. Console.WriteLine("Вставка пройдена");
145. **Assert.AreEqual(count, hashTable.Count);**
147. **foreach** (KeyValuePair<**string**, **string**> pair **in** pairs)
148. {
149. Assert.AreEqual(hashTable[pair.Key], pair.**Value**);
150. **}**
152. Console.WriteLine("Сверка элементов пройдена");
154. **foreach** (KeyValuePair<**string**, **string**> pair **in** pairs)
155. **{**
156. count--;
158. hashTable.**Remove**(pair.Key);
160. **Assert.AreEqual(hashTable.Count, count);**
161. }
163. Assert.AreEqual(hashTable.Count, 0);
165. **Console.WriteLine("Удаление элементов пройдено");**
166. }
168. */// <summary>*
169. */// Тестирование корректной работы с хеш-значениями*
170. ***/// </summary>***
171. [TestMethod]
172. **public** **void** CustomHashTest()
173. {
174. **var** hashTable = new OpenHashTable<HashString, **string**>();
176. **int** cnt = 10000;
178. Random rnd = new Random();
180. **Dictionary<HashString, string> pairs = new();**
182. **for** (**int** i = 0; i < cnt; i++)
183. {
184. **var** key = rnd.Next().ToString();
186. **if** (!pairs.ContainsKey(key))
187. {
188. pairs.**Add**(key, rnd.Next().ToString());
189. }
190. **}**
192. **foreach** (**var** (key, **value**) **in** pairs)
193. {
194. hashTable.**Add**(key, **value**);
195. **}**
197. **var** keys = new HashSet<HashString>();
199. **var** enumerator = hashTable.GetEnumerator();
201. **while** (enumerator.MoveNext())
202. {
203. keys.**Add**(enumerator.Current.Key);
204. }
206. *//Assert.AreEqual(pairs.Keys.ToHashSet(), keys.ToHashSet());*
208. **var** except = pairs.Keys.ToHashSet().Except(hashTable.Keys);
210. **Assert.AreEqual(except.Count(), 0);**
211. Assert.IsTrue(pairs.Values.ToHashSet().SetEquals(hashTable.Values.ToHashSet()));
212. Assert.AreEqual(pairs.Count, hashTable.Count);
213. }
214. }
215. **}**
     1. Тест алгоритма сортировки слиянием и тест автоматически сортируемого списка
216. **using** SaodCP.DataStructures;
217. **using** SaodCP.Utils;
219. **namespace** SaopCPTest
220. **{**
221. */// <summary>*
222. */// Тест сортировки*
223. */// </summary>*
224. [TestClass]
225. **public class SortTest**
226. {
227. */// <summary>*
228. */// Тест сортировки массива слиянием*
229. */// </summary>*
230. **[TestMethod]**
231. **public** **void** ArraySortTest()
232. {
233. Random randNum = new Random();
235. **int[] source = Enumerable**
236. .Repeat(0, 50)
237. .**Select**(i => randNum.Next(100))
238. .ToArray();
240. ***// сортировка по умолчанию (по возрастанию)***
241. **var** testDest = source.MergeSort();
243. **var** trustList = source.ToList();
245. **trustList.Sort();**
247. **var** trustDest = trustList.ToArray();
249. Assert.AreEqual(trustDest.Length, testDest.Length);
251. **for** (**int** i = 0; i < trustDest.Length; i++)
252. {
253. Assert.AreEqual(trustDest[i], testDest[i]);
254. }
256. *// сортировка по убыванию*
257. Comparison<**int**> comparison = (f, s) => -f.CompareTo(s);
259. testDest = source.MergeSort(comparison);
261. trustList = source.ToList();
263. trustList.Sort(comparison);
265. **trustDest = trustList.ToArray();**
267. Assert.AreEqual(trustDest.Length, testDest.Length);
269. **for** (**int** i = 0; i < trustDest.Length; i++)
270. **{**
271. Assert.AreEqual(trustDest[i], testDest[i]);
272. }
273. }
275. ***/// <summary>***
276. */// Тест автоматически сортируемого односвязного списка*
277. */// </summary>*
278. [TestMethod]
279. **public** **void** SortedListTest()
280. **{**
281. Random randNum = new Random();
283. **int**[] source = Enumerable
284. .Repeat(0, 50)
285. **.Select(i => randNum.Next(100))**
286. .ToArray();
288. **var** trustList = source.ToList();
290. **trustList.Sort();**
292. **var** trustDest = trustList.ToArray();
294. **var** sortedList = new SortedOneWayCycledList<**int**>();
296. **foreach** (**int** i **in** source)
297. {
298. sortedList.**Add**(i);
299. }
301. **int** cnt = 0;
303. **foreach** (**int** element **in** sortedList)
304. {
305. **Assert.AreEqual(element, trustDest[cnt]);**
307. cnt++;
308. }
309. }
310. **}**
311. }
     1. Тестирование АВЛ-дерева поиска
312. **using** SaodCP.DataStructures;
314. **namespace** SaopCPTest
315. {
316. ***/// <summary>***
317. */// Тесты для АВЛ-дерева поиска*
318. */// </summary>*
319. [TestClass]
320. **public** **class** TreeTest
321. **{**
322. */// <summary>*
323. */// Создание, добавление, удаление элементов*
324. */// </summary>*
325. [TestMethod]
326. **public void CreateAddRemoveTest()**
327. {
328. **var** tree = new Tree<**string**, **object**>();
330. tree.**Add**("1");
331. **tree.Add("2");**
332. tree.**Add**("3");
333. tree.**Add**("4");
334. tree.**Add**("5");
336. **Assert.AreEqual(5, tree.Count);**
338. **var** intTree = new Tree<**int**, **string**>();
340. **for** (**int** i = 0; i< 100; i++)
341. **{**
342. intTree.**Add**(i, i.ToString());
343. }
345. Assert.AreEqual(100, intTree.Count);
347. **var** treeSet = intTree
348. .**Select**(kv => kv.Key)
349. .ToHashSet();
351. **var rangeSet = Enumerable.Range(0, 100).ToHashSet();**
353. Assert.IsTrue(treeSet.SetEquals(rangeSet));
355. **var** count = intTree.Count;
357. *// проверка удаления*
358. **var** removed = intTree.**Remove**(100);
360. Assert.IsFalse(removed);
361. **Assert.AreEqual(intTree.Count, count);**
363. **foreach** (**var** i **in** treeSet)
364. {
365. removed = intTree.**Remove**(i);
367. *// удаление успешно*
368. Assert.IsTrue(removed);
370. *// элемента больше не содержится в дереве*
371. **Assert.IsFalse(intTree.Any(kv => kv.Key == i));**
373. *// проверка корректности счетчика*
374. Assert.AreEqual(intTree.Count, --count);
375. }
376. **}**
377. }
378. }

Результаты ручного тестирования представлены в таблице 1.

Таблица 1 Входные и выходные данные тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Тест** | **Входные данные** | **Результат** |
|  | Добавление заселения для постояльца 1 в номер 1 | Код номера, дата заселения | Постоялец заселен в номер 1 с указанной даты |
|  | Добавление пересекающегося периода проживания для постояльца 1 | Код номера, дата заселения | Сообщение о конфликте периодов заселения |
|  | Выселение постояльца 1 из номера 1 | Код номера, дата выселения | Постоялец выселен из номера 1 с указанной даты |
|  | Заселение в номер 1 (кол-во кроватей – 1) постояльца 2, период проживания пересекается с периодом проживания постояльца 1 в номере 1 | Код номера, дата заселения | Сообщение о конфликте максимального количества одновременно проживающих в комнате 1 |
|  | Изменение номера 1, увеличение количества спальных мест с 1 до 2 | Код номера, количество спальных мест | Данные номера изменены |
|  | Повтор п. 4, кол-во кроватей – 2 | Код номера, дата заселения | Постоялец 2 заселен в номер 1 с указанной даты |
|  | Удаление данных о номера 1 | Код номера | Сообщение о невозможности удаления данных ввиду наличия данных о заселении |
|  | Удаление данных о постояльце 1 | Номер паспорта | Сообщение о невозможности удаления данных ввиду наличия данных о заселении |
|  | Удаление данных о заселениях в номер 1 | Порядковый номер заселения в списке | Данные о заселениях удалены |
|  | Повтор пп. 8, 9 | Номер паспорта, Код номера | Данные удалены |
|  | Изменение данных пользователя; ввод некорректного формата номера паспорта | Номер паспорта | Сообщение о неправильном формате номера паспорта |
|  | Изменение данных номера; ввод некорректного формата кода номера | Код номера | Сообщение о неправильном формате кода номера |
|  | Поиск информации о постояльце по номеру паспорта; ввод существующего ключа | Номер паспорта | Вывод данных о постояльце, вывод данных о номере, в котором постоялец проживает на данный момент |
|  | Поиск информации о постояльце по номеру паспорта, ввод несуществующего ключа | Номер паспорта | Вывод сообщения об отсутствии совпадений по ключу |
|  | Поиск информации о постояльце по ФИО; ввод части имени, совпадающей у нескольких постояльцев | ФИО постояльца | Вывод списка постояльцев |
|  | Поиск информации о номере по коду номера; ввод существующего ключа | Код номера | Вывод информации о номере, вывод списка постояльцев, проживающих в номере |
|  | Поиск информации о номере по коду номера; ввод несуществующего ключа | Код номера | Вывод сообщения об отсутствии совпадений по ключу |
|  | 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 С#. Программирование на платформе Microsoft .NET Framework 4.0 на языке С#. 3-е изд. - СПб.: Питер, 201 2.- 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
2. **namespace** SaodCP.Algorithms
3. {
4. */// <summary>*
5. */// Алгоритм поиска подстроки в строке Боуэра и Мура*
6. ***/// </summary>***
7. **public** **static** **class** BMTextSearch
8. {
9. */// <summary>*
10. */// Поиск строки в тексте*
11. ***/// </summary>***
12. */// <param name="text">Текст, в котором производить поиск</param>*
13. */// <param name="search">Строка для поиска</param>*
14. */// <returns>Номер начала первого вхождения строки в 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. }
25. **int**[] shiftArray = FillShiftArray(search);
27. **int** startSearch = 0;
28. **int** foundPosition = 0;
30. */// поиск в строке, пока подстрока не выходит за её пределы*
31. **while (startSearch + search.Length <= text.Length)**
32. {
33. **bool** hasDifferences = **false**;
35. **int** textCharInSearchPosition = -1;
37. *// посимвольное сравнение с конца подстроки*
38. **for** (**int** i = search.Length - 1; i >= 0; i--)
39. {
40. **var** textIndex = startSearch + i;
42. **if** (search[i] != text[textIndex])
43. {
44. *// найдено расхождение*
45. hasDifferences = **true**;
47. *// поиск несовпадшего символа в подстроке*
48. **for** (**int** j = 0; j < search.Length; j++)
49. {
50. **if** (search[j] == text[textIndex])
51. **{**
52. *// индекс несовпадшего символа из строки в подстроке*
53. *// начиная с начала подстроки*
54. textCharInSearchPosition = j;
55. }
56. **}**
58. **break**;
59. }
60. }
62. **int** offset;
64. **if** (textCharInSearchPosition == -1)
65. {
66. ***// несовпадший символ не содержится в подстроке -***
67. *// сдвигаем подстроку на всю её длину вправо*
68. offset = search.Length;
69. }
70. **else**
71. **{**
72. *// количество символов для сдвига равно расстоянию от последнего*
73. *// вхождения несовпадшего символа в подстроку до конца подстроки*
74. offset = shiftArray[textCharInSearchPosition];
75. }
77. *// если найдено расхождение*
78. *// позиция поиска смещается, поиск повторяется*
79. **if** (hasDifferences == **true**)
80. {
81. **startSearch += offset;**
82. }
83. **else**
84. {
85. *// иначе - найдено вхождение подстроки, поиск завершен*
86. **foundPosition = startSearch + 1;**
88. **break**;
89. }
90. }
92. **return** foundPosition;
93. }
95. */// <summary>*
96. ***/// Заполнение сдвигов для символов в подстроке***
97. */// </summary>*
98. */// <param name="search">Подстрока</param>*
99. */// <returns></returns>*
100. **private** **static** **int**[] FillShiftArray(**string** search)
101. **{**
102. *// размер таблицы сдвигов равен длине подстроки*
103. **var** shiftArray = new **int**[search.Length];
105. *// для каждого символа в строке вычисляется расстояние от конца*
106. ***// подстроки до последнего вхождения аналогичного символа***
107. **for** (**int** i = 0; i < search.Length; i++)
108. {
109. **var** c = search[i];
111. **shiftArray[i] = search.Length;**
113. **for** (**int** j = search.Length - 1; j >= 0; j--)
114. {
115. **if** (c == search[j])
116. **{**
117. **int** lastOccurenceIdx = j;
118. shiftArray[i] = search.Length - j - 1;
120. **break**;
121. **}**
122. }
123. }
125. **return** shiftArray;
126. **}**
127. }
128. }
129. Файл «OpenHashTable.cs» - хеш-таблица.
130. **using** System.Collections;
131. **using** System.Diagnostics.CodeAnalysis;
133. **namespace** SaodCP.DataStructures
134. **{**
135. */// <summary>*
136. */// Хеш-таблица, открытое хеширование*
137. */// </summary>*
138. **public** **class** OpenHashTable<T, O> : IDictionary<T, O>
139. **{**
140. *// стартовое количество бакетов (размер таблицы)*
141. **protected** **static** **readonly** **int** START\_BUCKETS\_NUM = 2048;
143. *// таблица - массив связных списков, элементами*
144. ***// которых являются пары ключ-значение***
145. **protected** OneWayCycledList<KeyValuePair<T, O>>[] keyValuePairs =
146. new OneWayCycledList<KeyValuePair<T, O>>[START\_BUCKETS\_NUM];
148. **protected** **void** Init()
149. **{**
150. keyValuePairs = new OneWayCycledList<KeyValuePair<T, O>>[START\_BUCKETS\_NUM];
151. Count = 0;
152. }
154. ***/// <summary>***
155. */// Расширение хеш-корзины в 2 раза*
156. */// </summary>*
157. **protected** **void** Expand()
158. {
159. **var oldKeyValuePairs = keyValuePairs;**
161. keyValuePairs = new OneWayCycledList<KeyValuePair<T, O>>[keyValuePairs.Length \* 2];
162. Count = 0;
164. **foreach (var list in oldKeyValuePairs)**
165. {
166. **if** (list == **null**)
167. {
168. **continue**;
169. **}**
171. **foreach** (**var** pair **in** list)
172. {
173. **this**.**Add**(pair);
174. **}**
175. }
176. }
178. */// <summary>*
179. ***/// Расчет ячейки таблицы для ключа***
180. */// </summary>*
181. */// <param name="key"></param>*
182. */// <returns></returns>*
183. **protected** **int** GetBucketNumberByKey(T key)
184. **{**
185. **var** number = key?.GetHashCode() % keyValuePairs.Length
186. ?? **throw** new ArgumentNullException(nameof(key));
188. **return** number >= 0
189. **? number**
190. : -number;
191. }
193. **public** O **this**[T key]
194. **{**
195. **get**
196. {
197. **int** bucket = GetBucketNumberByKey(key);
199. **var list = keyValuePairs[bucket];**
201. **if** (list == **null**)
202. {
203. **return** **default**;
204. **}**
206. **foreach** (**var** pair **in** list)
207. {
208. **if** (pair.Key?.Equals(key) ?? **false**)
209. **{**
210. **return** pair.**Value**;
211. }
212. }
214. **return default;**
215. }
216. **set**
217. {
218. **Add**(key, **value**);
219. **}**
220. }
222. **public** ICollection<T> Keys => **this**.**Select**(kv => kv.Key).ToHashSet();
224. **public ICollection<O> Values => this.Select(kv => kv.Value).ToHashSet();**
226. **public** **int** Count { **get**; **protected** **set**; } = 0;
228. **public** **bool** IsReadOnly => **false**;
230. **public** **void** **Add**(T key, O **value**)
231. {
232. **int** bucket = GetBucketNumberByKey(key);
234. **if (keyValuePairs[bucket] == null)**
235. {
236. keyValuePairs[bucket] = new();
237. }
239. **var list = keyValuePairs[bucket];**
241. *// если длина списка превышает допустимое значение*
242. *// попытка учесть и плохое распределение, когда заполняется мало ячеек*
243. *// и наоборот хорошее, когда элементы равномерно "размазываются" по всем спискам*
244. **if (list.Count >= keyValuePairs.Length**
245. || Count > keyValuePairs.Length \* 10)
246. {
247. Expand();
249. **Add(key, value);**
251. **return**;
252. }
254. **KeyValuePair<T, O>? kv = null;**
256. **foreach** (**var** kvPair **in** list)
257. {
258. **if** (kvPair.Key?.Equals(key) ?? **false**)
259. **{**
260. kv = kvPair;
261. }
262. }
264. **if (kv != null)**
265. {
266. list.**Remove**((KeyValuePair<T, O>)kv);
267. }
269. **list.Add(new(key, value));**
271. Count++;
272. }
274. **public void Add(KeyValuePair<T, O> item)**
275. {
276. **Add**(item.Key, item.**Value**);
277. }
279. **public void Clear()**
280. {
281. Init();
282. }
284. **public bool Contains(KeyValuePair<T, O> item)**
285. {
286. **var** bucketNum = GetBucketNumberByKey(item.Key);
288. **var** list = keyValuePairs[bucketNum];
290. **if** (list == **null**)
291. {
292. **return** **false**;
293. }
295. **foreach** (**var** kv **in** list)
296. {
297. **if** (kv.Equals(item))
298. {
299. **return true;**
300. }
301. }
303. **return** **false**;
304. **}**
306. **public** **bool** ContainsKey(T key)
307. {
308. **var** bucketNum = GetBucketNumberByKey(key);
310. **var** list = keyValuePairs[bucketNum];
312. **if** (list == **null**)
313. {
314. **return false;**
315. }
317. **foreach** (**var** kv **in** list)
318. {
319. **if (kv.Key?.Equals(key) ?? false)**
320. {
321. **return** **true**;
322. }
323. }
325. **return** **false**;
326. }
328. **public** **void** CopyTo(KeyValuePair<T, O>[] array, **int** arrayIndex)
329. **{**
330. **foreach** (**var** kv **in** **this**)
331. {
332. array[arrayIndex++] = kv;
333. }
334. **}**
336. **public** IEnumerator<KeyValuePair<T, O>> GetEnumerator()
337. {
338. **return** new OpenHashTableEnumerator(**this**);
339. **}**
341. */// <summary>*
342. */// Удалить пару из таблицы по ключу*
343. */// </summary>*
344. ***/// <param name="key"></param>***
345. */// <returns></returns>*
346. **public** **bool** **Remove**(T key)
347. {
348. **var** bucket = GetBucketNumberByKey(key);
350. **var** list = keyValuePairs[bucket];
352. **if** (list == **null**)
353. {
354. **return false;**
355. }
357. KeyValuePair<T, O>? keyValue = **null**;
359. **foreach (var kv in list)**
360. {
361. **if** (kv.Key?.Equals(key) ?? **false**)
362. {
363. keyValue = kv;
364. **}**
365. }
367. **if** (keyValue != **null**)
368. {
369. **var ret = list.Remove((KeyValuePair<T, O>)keyValue);**
371. **if** (ret)
372. {
373. Count--;
374. **}**
376. **return** ret;
377. }
379. **return false;**
380. }
382. */// <summary>*
383. */// Удалить пару из таблицы*
384. ***/// </summary>***
385. */// <param name="item"></param>*
386. */// <returns></returns>*
387. **public** **bool** **Remove**(KeyValuePair<T, O> item)
388. {
389. **var bucket = GetBucketNumberByKey(item.Key);**
391. **var** list = keyValuePairs[bucket];
393. **if** (list == **null**)
394. **{**
395. **return** **false**;
396. }
398. KeyValuePair<T, O>? keyValue = **null**;
400. **foreach** (**var** kv **in** list)
401. {
402. **if** (kv.Equals(item))
403. {
404. **keyValue = kv;**
405. }
406. }
408. **if** (keyValue != **null**)
409. **{**
410. **var** ret = list.**Remove**((KeyValuePair<T, O>)keyValue);
412. **if** (ret)
413. {
414. **Count--;**
415. }
417. **return** ret;
418. }
420. **return** **false**;
421. }
423. */// <summary>*
424. ***/// Получить значение***
425. */// </summary>*
426. **public** **bool** TryGetValue(T key, [MaybeNullWhen(**false**)] **out** O **value**)
427. {
428. **if** (!ContainsKey(key))
429. **{**
430. **value** = **default**;
432. **return** **false**;
433. }
435. **value** = **this**[key];
437. **return** **true**;
438. }
440. IEnumerator IEnumerable.GetEnumerator()
441. {
442. **return** **this**.GetEnumerator();
443. }
445. **public** **class** OpenHashTableEnumerator : IEnumerator<KeyValuePair<T, O>>
446. {
447. **private** **readonly** OpenHashTable<T, O> \_table;
449. **private int \_currentBucket = -1;**
451. **private** IEnumerator<KeyValuePair<T, O>>? \_currentListEnumerator;
453. **private** **bool** \_isDisposed;
455. **public** OpenHashTableEnumerator(OpenHashTable<T, O> table)
456. {
457. \_table = table;
458. }
460. **public** KeyValuePair<T, O> Current => \_currentListEnumerator?.Current ?? **default**;
462. **object** IEnumerator.Current => Current;
464. **public void Dispose()**
465. {
466. **if** (\_isDisposed)
467. {
468. **throw** new ObjectDisposedException(GetType().FullName);
469. **}**
471. \_isDisposed = **true**;
472. }
474. **public bool MoveNext()**
475. {
476. *// работа с текущим итератором списка*
477. **if** (\_currentListEnumerator != **null**)
478. {
479. **if (\_currentListEnumerator.MoveNext())**
480. {
481. **return** **true**;
482. }
483. **else**
484. **{**
485. \_currentListEnumerator = **null**;
486. }
487. }
489. **OneWayCycledList<KeyValuePair<T, O>>? list = null;**
491. **while** (list == **null**
492. && \_currentBucket < \_table.keyValuePairs.Length - 1)
493. {
494. **list = \_table.keyValuePairs[++\_currentBucket];**
495. }
497. **if** (list == **null**)
498. {
499. **return false;**
500. }
501. **else**
502. {
503. \_currentListEnumerator = list.GetEnumerator();
505. **if** (\_currentListEnumerator != **null**)
506. {
507. **return** MoveNext();
508. }
509. **else**
510. {
511. **return** **false**;
512. }
513. }
514. **}**
516. **public** **void** Reset()
517. {
518. **throw** new NotImplementedException();
519. **}**
520. }
521. }
522. }
524. Файл «OneWayCycledList.cs» - циклический однонаправленный список
525. **using** System.Collections;
527. **namespace** SaodCP.DataStructures
528. {
529. ***/// <summary>***
530. */// Односвязный циклический список*
531. */// </summary>*
532. */// <typeparam name="T"></typeparam>*
533. **public** **class** OneWayCycledList<T> : ICollection<T>, IEnumerator<T>
534. **{**
535. **protected** OneWayLinkedListElement<T>? \_first;
536. **protected** OneWayLinkedListElement<T>? \_last;
537. **protected** OneWayLinkedListElement<T>? \_current;
539. **private bool \_isDisposed = false;**
541. **public** **int** Count { **get**; **protected** **set**; } = 0;
543. **public** **bool** IsReadOnly => **false**;
545. **public** T Current
546. {
547. **get**
548. {
549. **if (\_current == null)**
550. {
551. **throw** new NullReferenceException();
552. }
554. **return \_current.Value;**
555. }
556. }
558. **object** IEnumerator.Current => Current;
560. **public** **void** **Add**(T item)
561. {
562. **if** (\_last != **null**)
563. {
564. **\_last.Next = new (item);**
566. \_last = \_last.Next;
568. \_last.Next = \_first;
569. **}**
570. **else**
571. {
572. \_first = new(item);
573. \_last = \_first;
574. **\_first.Next = \_last;**
575. }
577. Count++;
578. }
580. **public** **void** Clear()
581. {
582. \_first = **null**;
583. \_last = **null**;
584. **Count = 0;**
585. }
587. **public** **bool** Contains(T item)
588. {
589. **var cnt = 0;**
591. **if** (\_first == **null**)
592. {
593. **return** **false**;
594. **}**
596. OneWayLinkedListElement<T> element = \_first;
598. **while** (cnt < Count)
599. **{**
600. **if** (element.**Value**?.Equals(item) ?? **false**)
601. {
602. **return** **true**;
603. }
605. cnt++;
607. **if** (element.Next == **null**)
608. {
609. **return false;**
610. }
612. element = element.Next;
613. }
615. **return** **false**;
616. }
618. **public** **void** CopyTo(T[] array, **int** arrayIndex)
619. **{**
620. **int** pos = arrayIndex;
622. **foreach**(**var** element **in** **this**)
623. {
624. **array[pos] = element;**
626. pos++;
627. }
628. }
630. **public** **void** Dispose()
631. {
632. **if** (\_isDisposed)
633. {
634. **throw new ObjectDisposedException(GetType().FullName);**
635. }
637. \_isDisposed = **true**;
638. }
640. **public** IEnumerator<T> GetEnumerator()
641. {
642. \_isDisposed = **false**;
643. \_current = **null**;
645. **return** **this**;
646. }
648. **public** **bool** MoveNext()
649. **{**
650. **if** (\_current == **null**)
651. {
652. \_current = \_first;
654. **return \_current != null;**
655. }
657. **if** (**object**.ReferenceEquals(\_current, \_last))
658. {
659. **return false;**
660. }
662. \_current = \_current.Next;
664. **return true;**
665. }
667. **public** **bool** **Remove**(T item)
668. {
669. **if (Count < 1)**
670. {
671. **return** **false**;
672. }
674. **if (\_first == null)**
675. {
676. **return** **false**;
677. }
679. **if (\_last == null)**
680. {
681. **return** **false**;
682. }
684. **var current = \_first;**
685. OneWayLinkedListElement<T> prev = \_last;
687. **do**
688. {
689. **if (current.Value?.Equals(item) ?? false)**
690. {
691. prev.Next = current.Next;
693. **if** (**object**.ReferenceEquals(\_first, current))
694. **{**
695. \_first = current.Next;
696. }
698. Count--;
700. **if** (Count < 1)
701. {
702. \_first = **null**;
703. \_last = **null**;
704. **}**
706. **return** **true**;
707. }
709. **prev = current;**
711. current = current.Next;
712. }
713. **while** (current != **null**);
715. **return** **false**;
716. }
718. **public** **void** Reset()
719. **{**
720. \_current = **null**;
721. }
723. IEnumerator IEnumerable.GetEnumerator()
724. **{**
725. **return** **this**.GetEnumerator();
726. }
728. */// <summary>*
729. ***/// Отсортировать элементы списка***
730. */// </summary>*
731. */// <param name="comparer"></param>*
732. **public** **void** Sort(Comparison<T>? comparer = **null**)
733. {
734. **if (\_first == null)**
735. {
736. **return**;
737. }
739. **if (\_first.Value == null)**
740. {
741. **return**;
742. }
744. **T[] source = new T[Count];**
746. CopyTo(source, 0);
748. **var** dest = Utils.Utils.MergeSort(source, comparer);
750. Clear();
752. **foreach** (**var** item **in** dest)
753. {
754. **Add(item);**
755. }
756. }
757. }
759. ***/// <summary>***
760. */// Элемент односвязного списка*
761. */// </summary>*
762. */// <typeparam name="T"></typeparam>*
763. **public** **class** OneWayLinkedListElement<T> : ICloneable
764. **{**
765. **private** **readonly** T \_value;
767. **public** OneWayLinkedListElement(T **value**)
768. {
769. **\_value = value;**
770. }
772. **public** T **Value** { **get** => \_value; }
774. **public OneWayLinkedListElement<T>? Next { get; set; }**
776. **public** **object** Clone()
777. {
778. **return** MemberwiseClone();
779. **}**
780. }
781. }
783. Файл «SortedOneWayCycledList.cs» - отсортированный односвязный список
784. **using** System;
785. **using** System.Collections.Generic;
786. **using** System.Linq;
787. **using** System.Text;
788. **using System.Threading.Tasks;**
790. **namespace** SaodCP.DataStructures
791. {
792. */// <summary>*
793. ***/// Отсортированный список***
794. */// </summary>*
795. */// <typeparam name="T"></typeparam>*
796. **public** **class** SortedOneWayCycledList<T> : OneWayCycledList<T>
797. {
798. **private Comparison<T> \_comparison;**
800. */// <summary>*
801. */// Делегат для сравнения элементов*
802. */// </summary>*
803. **public Comparison<T> Comparison**
804. {
805. **get** => \_comparison;
806. **set**
807. {
808. **\_comparison = value;**
810. **if** (Count > 1)
811. {
812. Sort();
813. **}**
814. }
815. }
817. */// <summary>*
818. ***/// Создает список для типа, являющегося IComparable***
819. */// </summary>*
820. **public** SortedOneWayCycledList()
821. {
822. **var** type = typeof(T);
824. **if** (!typeof(IComparable<T>).IsAssignableFrom(type))
825. {
826. **throw** new ApplicationException($"{nameof(T)} is not {nameof(IComparable)}"
827. + ", use constructor with a Comparison instead");
828. **}**
830. \_comparison = (f, s) =>
831. {
832. **if** (f != **null**)
833. **{**
834. **return** ((IComparable)f).CompareTo(s);
835. }
836. **else**
837. {
838. **throw new ArgumentNullException(nameof(f));**
839. }
840. };
841. }
843. **public SortedOneWayCycledList(Comparison<T> comparison)**
844. {
845. \_comparison = comparison;
846. }
848. ***/// <summary>***
849. */// Добавление элемента, затем сортировка списка*
850. */// Работает за O(n log(n))*
851. */// </summary>*
852. */// <param name="element"></param>*
853. **public new void Add(T element)**
854. {
855. **base**.**Add**(element);
857. Sort(\_comparison);
858. **}**
859. }
860. }
862. Файл «Tree.cs» - бинарное дерево поиска
863. **using** System.Drawing.Design;
864. **using** System.Net.Sockets;
866. **namespace** SaodCP.DataStructures
867. **{**
868. */// <summary>*
869. */// АВЛ-дерево поиска*
870. */// </summary>*
871. */// <typeparam name="T"></typeparam>*
872. ***/// <typeparam name="O"></typeparam>***
873. **public** **partial** **class** Tree<T, O>
874. {
875. **private** TreeNode<T, O>? head = **null**;
877. **private Comparison<T> \_comparison;**
879. **public** **int** Count { **get**; **private** **set**; } = 0;
881. **public** Tree()
882. **{**
883. **if** (!typeof(IComparable).IsAssignableFrom(typeof(T)))
884. {
885. **throw** new ApplicationException(
886. $"{nameof(T)} is not IComparable and Comparison<{nameof(T)}> was not provided");
887. **}**
889. \_comparison = (first, second) => ((IComparable)first).CompareTo(second);
890. }
892. **public Tree(Comparison<T> comparison)**
893. {
894. \_comparison = comparison;
895. }
897. ***/// <summary>***
898. */// Добавить пару ключ-значение*
899. */// </summary>*
900. */// <param name="key"></param>*
901. */// <param name="value"></param>*
902. **public void Add(T key, O? value = default)**
903. {
904. **var** node = new TreeNode<T, O>(key, **value**);
906. **if** (head == **null**)
907. **{**
908. head = node;
910. Count = 1;
912. **return;**
913. }
915. head = RecursiveAdd(head, node);
916. }
918. **private** TreeNode<T, O> RecursiveAdd(
919. TreeNode<T, O> parent,
920. TreeNode<T, O> addNode)
921. {
922. **var cmp = \_comparison(parent.Key, addNode.Key);**
924. **if** (cmp > 0)
925. {
926. **if** (parent.Left != **null**)
927. **{**
928. parent.Left = RecursiveAdd(parent.Left, addNode);
929. }
930. **else**
931. {
932. **parent.Left = addNode;**
934. Count++;
935. }
937. **parent.UpdateDimensions();**
939. **if** (parent.Diff >= 2
940. || parent.Diff <= -2)
941. {
942. **parent = Rebalance(parent);**
943. }
945. **return** parent;
946. }
947. **else if (cmp < 0)**
948. {
949. **if** (parent.Right != **null**)
950. {
951. parent.Right = RecursiveAdd(parent.Right, addNode);
952. **}**
953. **else**
954. {
955. parent.Right = addNode;
957. **Count++;**
958. }
960. parent.UpdateDimensions();
962. **if (parent.Diff >= 2**
963. || parent.Diff <= -2)
964. {
965. parent = Rebalance(parent);
966. }
968. **return** parent;
969. }
970. **else**
971. {
972. **parent.Value = addNode.Value;**
974. parent.UpdateDimensions();
976. **return** parent;
977. **}**
978. }
980. */// <summary>*
981. */// Удалить по ключу*
982. ***/// </summary>***
983. */// <param name="key"></param>*
984. */// <returns></returns>*
985. **public** **bool** **Remove**(T key)
986. {
987. **if (head == null)**
988. {
989. **return** **false**;
990. }
992. **bool ret = false;**
994. head = DeleteNode(head, key, **ref** ret);
996. **if** (ret)
997. **{**
998. Count--;
999. }
1001. **return** ret;
1002. **}**
1004. **private** TreeNode<T, O>? DeleteNode(
1005. TreeNode<T, O> currentNode,
1006. T itemValue)
1007. **{**
1008. **bool** stackBool = **false**;
1010. **return** DeleteNode(currentNode, itemValue, **ref** stackBool);
1011. }
1013. *// найти и удалить*
1014. **private** TreeNode<T,O>? DeleteNode(
1015. TreeNode<T,O> currentNode,
1016. T itemValue,
1017. **ref bool result)**
1018. {
1019. result = **true**;
1021. **var** cmp = \_comparison(currentNode.Key, itemValue);
1023. *// если нашли нужный элемент, начинаем процедуру удаления*
1024. **if** (cmp == 0)
1025. {
1026. *// обработка самого простого случая, вместо узла возвращается null*
1027. **if (currentNode.Left == null**
1028. && currentNode.Right == **null**)
1029. {
1030. **return** **null**;
1031. }
1033. *// обработка двух случаев, с только одним из поддеревьев*
1034. **if** (currentNode.Left == **null**)
1035. {
1036. **return** currentNode.Right;
1037. **}**
1039. **if** (currentNode.Right == **null**)
1040. {
1041. **return** currentNode.Left;
1042. **}**
1044. *// если у ноды есть оба потомка*
1045. **var** minNodeInRightSubtree = FindMinNode(currentNode.Right);
1046. *// заменили текущий элемент минимальным из правого поддерева*
1047. **currentNode.Value = minNodeInRightSubtree.Value;**
1048. currentNode.Key = minNodeInRightSubtree.Key;
1050. *// ищем в правом поддереве минимальный элемент,*
1051. *// значение которого уже вставлено на место текущего*
1052. **currentNode.Right = DeleteNode(**
1053. currentNode.Right,
1054. minNodeInRightSubtree.Key);
1056. **return** currentNode;
1057. **}**
1059. *// попадаем сюда, если элемент не был найден,*
1060. *// просто проваливаемся в дерево глубже и глубже*
1062. ***// производится рекурсивный вызов этой же функции***
1063. **if** (cmp > 0)
1064. {
1065. **if** (currentNode.Left == **null**)
1066. {
1067. **result = false;**
1069. **return** currentNode;
1070. }
1072. ***// проваливаемся в левое поддерево***
1073. currentNode.Left = DeleteNode(
1074. currentNode.Left,
1075. itemValue,
1076. **ref** result);
1078. *// если нового ребенка после удаления надо балансировать, балансируем*
1079. **if** (currentNode.Left != **null**)
1080. {
1081. **if** (currentNode.Left.Diff == 2
1082. **|| currentNode.Left.Diff == -2)**
1083. {
1084. currentNode.Left = Rebalance(currentNode.Left);
1085. }
1086. }
1088. *// обновление параметров*
1089. currentNode.UpdateDimensions();
1091. *// присваивание на рекурсивный уровень выше,*
1092. ***// может быть как в левое поддерево,так и в правое,***
1093. *// на текущем уровне мы не знаем какое поддерево обрабатываем*
1094. **return** currentNode;
1095. }
1097. ***// аналогичная обработка для правого поддерева***
1098. **if** (cmp < 0)
1099. {
1100. **if** (currentNode.Right == **null**)
1101. {
1102. **result = false;**
1104. **return** currentNode;
1105. }
1107. **currentNode.Right = DeleteNode(**
1108. currentNode.Right,
1109. itemValue,
1110. **ref** result);
1112. ***// если ребенка после удаления надо балансировать, балансируем***
1113. **if** (currentNode.Right != **null**)
1114. {
1115. **if** (currentNode.Right.Diff == 2
1116. || currentNode.Right.Diff == -2)
1117. **{**
1118. currentNode.Right = Rebalance(currentNode.Right);
1119. }
1120. }
1122. **currentNode.UpdateDimensions();**
1124. **return** currentNode;
1125. }
1127. **throw new ApplicationException("Delete Element error");**
1128. }
1130. **private** TreeNode<T,O> FindMinNode(TreeNode<T, O> node)
1131. {
1132. **if (node.Left == null)**
1133. {
1134. **return** node;
1135. }
1137. **return FindMinNode(node.Left);**
1138. }
1140. */// <summary>*
1141. */// Найти значение по ключу*
1142. ***/// </summary>***
1143. */// <param name="key"></param>*
1144. */// <returns></returns>*
1145. **public** O? Find(T key)
1146. {
1147. **if (head == null)**
1148. {
1149. **return** **default**;
1150. }
1152. **return RecursiveFind(head, key);**
1153. }
1155. **private** O? RecursiveFind(TreeNode<T,O> parent, T key)
1156. {
1157. **var cmp = \_comparison(parent.Key, key);**
1159. **if** (cmp == 0)
1160. {
1161. **return** parent.**Value**;
1162. **}**
1163. **else** **if** (cmp > 0)
1164. {
1165. **if** (parent.Left == **null**)
1166. {
1167. **return default;**
1168. }
1170. **return** RecursiveFind(parent.Left, key);
1171. }
1172. **else**
1173. {
1174. **if** (parent.Right == **null**)
1175. {
1176. **return** **default**;
1177. **}**
1179. **return** RecursiveFind(parent.Right, key);
1180. }
1181. }
1182. **}**
1183. }
1184. Файл «Tree\_Enumerable.cs» - АВЛ-дерево поиска, реализация интерфейса перечисляемого типа.
1185. **using** System.Collections;
1187. **namespace** SaodCP.DataStructures
1188. {
1189. ***/// <summary>***
1190. */// Реализация IEnumerable для Tree, чтобы было удобно использовать*
1191. */// Используется прямой обход дерева (вершина, левый, правый)*
1192. */// </summary>*
1193. */// <typeparam name="T"></typeparam>*
1194. ***/// <typeparam name="O"></typeparam>***
1195. **public** **partial** **class** Tree<T, O> : IEnumerable<KeyValuePair<T, O>>
1196. {
1197. */// <summary>*
1198. */// Получить Enumerator (обход дерева в прямом порядке)*
1199. ***/// </summary>***
1200. */// <returns></returns>*
1201. **public** IEnumerator<KeyValuePair<T, O>> GetEnumerator()
1202. {
1203. **var** pairs = new List<KeyValuePair<T, O>>();
1205. **var** node = head;
1207. AddToList(node);
1209. **void AddToList(TreeNode<T,O>? node)**
1210. {
1211. **if** (node == **null**)
1212. {
1213. **return**;
1214. **}**
1216. pairs.**Add**(new(node.Key, node.**Value**));
1218. AddToList(node.Left);
1219. **AddToList(node.Right);**
1220. }
1222. **return** pairs.GetEnumerator();
1223. }
1225. IEnumerator IEnumerable.GetEnumerator()
1226. {
1227. **return** GetEnumerator();
1228. }
1229. **}**
1230. }
1232. Файл «Tree\_Rebalance.cs» - операции по балансировке АВЛ-дерева поиска
1233. **namespace** SaodCP.DataStructures
1234. {
1235. */// <summary>*
1236. */// Операции по балансировке дерева*
1237. ***/// </summary>***
1238. **public** **partial** **class** Tree<T, O>
1239. {
1240. */// <summary>*
1241. */// Балансировка дерева*
1242. ***/// </summary>***
1243. */// <param name="head"></param>*
1244. */// <returns>Новая вершина поддерева</returns>*
1245. **private** TreeNode<T, O> Rebalance(TreeNode<T, O> head)
1246. {
1247. ***// надо ли вообще выполнять поворот***
1248. **if** (head.Diff != 2
1249. && head.Diff != -2)
1250. {
1251. **return** head;
1252. **}**
1254. **var** a = head;
1256. *// левый поворот*
1257. **if (a.Diff == 2)**
1258. {
1259. **var** b = head.Left;
1261. **if** (b == **null**)
1262. **{**
1263. **return** head;
1264. }
1266. **if** (b.Diff == 1
1267. **|| b.Diff == 0)**
1268. {
1269. **return** RotateLeft(head);
1270. }
1271. **else** **if** (b.Diff == -1)
1272. **{**
1274. **var** c = b.Right;
1276. **if** (c == **null**)
1277. **{**
1278. **return** head;
1279. }
1281. **return** BigRotateLeft(head);
1282. **}**
1283. }
1285. *// правый поворот*
1286. **if** (a.Diff == -2)
1287. **{**
1288. **var** b = head.Right;
1290. **if** (b == **null**)
1291. {
1292. **return head;**
1293. }
1295. **if** (b.Diff == -1
1296. || b.Diff == 0)
1297. **{**
1298. **return** RotateRight(head);
1299. }
1300. **else** **if** (b.Diff == 1)
1301. {
1302. **var c = b.Left;**
1304. **if** (c == **null**)
1305. {
1306. **return** head;
1307. **}**
1309. **return** BigRotateRight(head);
1310. }
1311. }
1313. **throw** new ApplicationException("Tree Internal error");
1314. }
1316. */// <summary>*
1317. ***/// левый поворот***
1318. */// </summary>*
1319. */// <param name="b"></param>*
1320. */// <returns></returns>*
1321. */// <exception cref="ArgumentNullException"></exception>*
1322. **private TreeNode<T, O> RotateLeft(TreeNode<T, O> b)**
1323. {
1324. TreeNode<T, O> a = b.Left ?? **throw** new NullReferenceException();
1326. b.Left = a.Right;
1328. a.Right = b;
1330. b.UpdateDimensions();
1331. a.UpdateDimensions();
1333. **return** a;
1334. }
1336. */// <summary>*
1337. ***/// правый поворот***
1338. */// </summary>*
1339. */// <param name="b"></param>*
1340. */// <returns></returns>*
1341. */// <exception cref="ArgumentNullException"></exception>*
1342. **private TreeNode<T, O> RotateRight(TreeNode<T, O> b)**
1343. {
1344. TreeNode<T, O> a = b.Right ?? **throw** new NullReferenceException();
1346. b.Right = a.Left;
1348. a.Left = b;
1350. b.UpdateDimensions();
1351. a.UpdateDimensions();
1353. **return** a;
1354. }
1356. */// <summary>*
1357. ***/// большой левый поворот***
1358. */// </summary>*
1359. */// <param name="a"></param>*
1360. */// <returns></returns>*
1361. */// <exception cref="NullReferenceException"></exception>*
1362. **private TreeNode<T, O> BigRotateLeft(TreeNode<T, O> a)**
1363. {
1364. a.Left = RotateRight(a.Left ?? **throw** new NullReferenceException());
1365. **return** RotateLeft(a);
1366. }
1368. */// <summary>*
1369. */// большой правый поворот*
1370. */// </summary>*
1371. */// <param name="a"></param>*
1372. ***/// <returns></returns>***
1373. */// <exception cref="NullReferenceException"></exception>*
1374. **private** TreeNode<T, O> BigRotateRight(TreeNode<T, O> a)
1375. {
1376. a.Right = RotateLeft(a.Right ?? **throw** new NullReferenceException());
1378. **return** RotateRight(a);
1379. }
1380. }
1381. }
1383. Файл «TreeNode.cs» - узел АВЛ-дерева
1384. **namespace** SaodCP.DataStructures
1385. {
1386. */// <summary>*
1387. */// Узел дерева*
1388. ***/// </summary>***
1389. */// <typeparam name="T"></typeparam>*
1390. **public** **class** TreeNode<T, O>
1391. {
1392. **public** T Key { **get**; **set**; }
1393. **public O? Value { get; set; }**
1395. **public** TreeNode(T key, O? **value**)
1396. {
1397. Key = key;
1398. **Value = value;**
1399. }
1401. **public** TreeNode<T, O>? Left { **get**; **set**; } = **null**;
1402. **public** TreeNode<T, O>? Right { **get**; **set**; } = **null**;
1404. */// <summary>*
1405. */// Разница между высотой левого и правого поддерева*
1406. */// </summary>*
1407. **public** **int** Diff { **get** => GetDiff(); }
1409. */// <summary>*
1410. */// Глубина поддерева*
1411. */// </summary>*
1412. **public** **int** Depth { **get**; **set**; }
1414. */// <summary>*
1415. */// Обновить глубину поддерева*
1416. */// </summary>*
1417. **public** **void** UpdateDepth()
1418. **{**
1419. **int** rightDepth = Right?.Depth + 1 ?? 0;
1420. **int** leftDepth = Left?.Depth + 1 ?? 0;
1422. **int** max = rightDepth > leftDepth ? rightDepth : leftDepth;
1424. Depth = max;
1425. }
1427. */// <summary>*
1428. ***/// Обновить разность***
1429. */// глубин поддеревьев*
1430. */// </summary>*
1431. **public** **int** GetDiff()
1432. {
1433. **int rightDepth = Right?.Depth + 1 ?? 0;**
1434. **int** leftDepth = Left?.Depth + 1 ?? 0;
1436. **return** leftDepth - rightDepth;
1437. }
1439. */// <summary>*
1440. */// Обновить измерения узла*
1441. */// </summary>*
1442. **public** **void** UpdateDimensions()
1443. **{**
1444. UpdateDepth();
1445. }
1446. }
1447. }
1448. Файл «Accommodation.cs» - Запись данных о заселении
1450. **using** System.ComponentModel;
1452. **namespace** SaodCP.Models
1453. **{**
1454. */// <summary>*
1455. */// Данные о периоде проживания постояльца в номере*
1456. */// </summary>*
1457. **public** **class** Accommodation : ICloneable
1458. **{**
1459. */// <summary>*
1460. */// Номер паспорта постояльца*
1461. */// </summary>*
1462. [DisplayName("Номер паспорта")]
1463. **public string LodgerPassportId { get; set; } = string.Empty;**
1465. */// <summary>*
1466. */// Номер комнаты постояльца*
1467. */// </summary>*
1468. **[DisplayName("Номер комнаты")]**
1469. **public** **string** ApartmentNumber { **get**; **set**; } = **string**.Empty;
1471. */// <summary>*
1472. */// Дата заселения постояльца*
1473. ***/// </summary>***
1474. [DisplayName("Дата С")]
1475. **public** DateOnly FromDate { **get**; **set**; }
1477. */// <summary>*
1478. ***/// Дата выезда постояльца***
1479. */// </summary>*
1480. [DisplayName("Дата ПО")]
1481. **public** DateOnly ToDate { **get**; **set**; }
1483. **public object Clone()**
1484. {
1485. **return** MemberwiseClone();
1486. }
1488. **public override string? ToString()**
1489. {
1490. **return** $"Заселение постояльца " +
1491. $"{this.LodgerPassportId} в номер " +
1492. $"{this.ApartmentNumber} " +
1493. **$"с {this.FromDate} по {this.ToDate}";**
1494. }
1495. }
1496. }
1498. Файл «Lodger.cs» - данные постояльца
1499. **using** SaodCP.DataStructures;
1500. **using** System.ComponentModel;
1502. **namespace** SaodCP.Models
1503. **{**
1504. */// <summary>*
1505. */// Постоялец*
1506. */// </summary>*
1507. **public** **class** Lodger
1508. **{**
1509. **public** **static** **readonly** **string** PassportIdPattern = "NNNN-NNNNNN";
1511. **private** HashString passportId = **string**.Empty;
1513. **[DisplayName("Номер паспорта")]**
1514. **public** **string** PassportId
1515. {
1516. **get** => passportId;
1517. **set**
1518. **{**
1519. **if** (Utils.Utils.ValidateLodgerPassportId(**value**))
1520. {
1521. passportId = **value**;
1522. }
1523. **else**
1524. {
1525. **throw** new FormatException("Неверный формат номера паспорта!");
1526. }
1527. }
1528. **}**
1530. [DisplayName("Имя")]
1531. **public** **string** Name { **get**; **set**; } = **string**.Empty;
1533. **[DisplayName("Год рождения")]**
1534. **public** **int** BirthYear { **get**; **set**; }
1536. [DisplayName("Адрес")]
1537. **public** **string** Address { **get**; **set**; } = **string**.Empty;
1538. **}**
1539. }
1541. Файл «Apartment.cs» - запись с данными о гостиничном номере
1542. **using** System.ComponentModel;
1543. **using** System.ComponentModel.DataAnnotations;
1545. **namespace** SaodCP.Models
1546. **{**
1547. */// <summary>*
1548. */// Номер в гостинице*
1549. */// </summary>*
1550. **public** **class** Apartment
1551. **{**
1552. **private** **string** \_number;
1554. #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.
1555. **public** Apartment(**string** number)
1556. **#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.**
1557. {
1558. Number = number;
1559. }
1561. ***/// <summary>***
1562. */// Номер апартаментов*
1563. */// </summary>*
1564. [DisplayName("Номер")]
1565. **public** **string** Number
1566. **{**
1567. **get** => \_number;
1568. **set**
1569. {
1570. **if** (!Utils.Utils.ValidateRoomNumberId(**value**))
1571. **{**
1572. **throw** new ArgumentException(**null**, nameof(**value**));
1573. }
1575. \_number = **value**;
1576. **}**
1577. }
1579. */// <summary>*
1580. */// Кол-во спальных мест*
1581. ***/// </summary>***
1582. [DisplayName("Кол-во спальных мест")]
1583. **public** **int** BedsNumber { **get**; **set**; }
1585. */// <summary>*
1586. ***/// Кол-во комнат***
1587. */// </summary>*
1588. [DisplayName("Кол-во комнат")]
1589. **public** **int** RoomNumber { **get**; **set**; }
1591. ***/// <summary>***
1592. */// Есть ли сан.узел*
1593. */// </summary>*
1594. [DisplayName("Сан. узел")]
1595. **public** **bool** HasToilet { **get**; **set**; }
1597. */// <summary>*
1598. */// Список оборудования*
1599. */// </summary>*
1600. [StringLength(200)]
1601. **[DisplayName("Оборудование")]**
1602. **public** **string** Equipment { **get**; **set**; } = **string**.Empty;
1603. }
1604. }
1606. Файл «Utils.cs» - инструменты для работы с первичными ключами, реализация сортировки слиянием, генерация случайных первичных ключей
1607. **using** SaodCP.Models;
1608. **namespace** SaodCP.Utils
1609. {
1610. **public static class Utils**
1611. {
1612. **static** **readonly** **char**[] NUMBER\_CHARS = new **char**[] { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };
1614. **static** **readonly** **char**[] ROOM\_NUMBER\_FIRST\_LITERAL = new **char**[] { 'Л', 'П', 'О', 'М' };
1616. */// <summary>*
1617. */// Проверка формата номера паспорта постояльца*
1618. */// </summary>*
1619. **public** **static** **bool** ValidateLodgerPassportId(
1620. **string? passportId)**
1621. {
1622. **var** refString = **string**.Empty;
1624. **return** ValidateLodgerPassportId(passportId, **ref** refString);
1625. **}**
1627. */// <summary>*
1628. */// Проверка формата номера паспорта*
1629. */// </summary>*
1630. ***/// <param name="passportId">Номер паспорта</param>***
1631. */// <param name="error">Описание ошибки</param>*
1632. **public** **static** **bool** ValidateLodgerPassportId(
1633. **string**? passportId,
1634. **ref** **string**? error)
1635. **{**
1636. **if** (**string**.IsNullOrWhiteSpace(passportId))
1637. {
1638. error = "Номер паспорта должен быть заполнен";
1640. **return false;**
1641. }
1643. **var** ret = **true**;
1645. **if (passportId.Length != 11)**
1646. {
1647. ret = **false**;
1648. }
1650. **var allowedCharactersArray = new char[]**
1651. {
1652. '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
1653. };
1655. **var delimiter = '-';**
1657. **if** (ret)
1658. {
1659. **for** (**var** i = 0; i < passportId.Length; i++)
1660. **{**
1661. **var** c = passportId[i];
1663. **if** (i == 4)
1664. {
1665. **if (c == delimiter)**
1666. {
1667. **continue**;
1668. }
1669. **else**
1670. **{**
1671. ret = **false**;
1673. **break**;
1674. }
1675. **}**
1677. **var** found = **false**;
1679. **for** (**var** j = 0; j < allowedCharactersArray.Length; j++)
1680. **{**
1681. **if** (c == allowedCharactersArray[j])
1682. {
1683. found = **true**;
1685. **break;**
1686. }
1687. }
1689. **if** (!found)
1690. **{**
1691. ret = **false**;
1693. **break**;
1694. }
1695. **}**
1696. }
1698. **if** (!ret)
1699. {
1700. **error = $"Номер паспорта должен соответствовать шаблону {Lodger.PassportIdPattern}";**
1701. }
1703. **return** ret;
1704. }
1706. */// <summary>*
1707. */// Проверка формата номера гостичничного номера*
1708. */// </summary>*
1709. **public** **static** **bool** ValidateRoomNumberId(
1710. **string? roomNumber)**
1711. {
1712. **var** str = **string**.Empty;
1714. **return** ValidateRoomNumberId(
1715. **roomNumber,**
1716. **ref** str);
1717. }
1719. */// <summary>*
1720. ***/// Проверка формата номера комнаты***
1721. */// </summary>*
1722. */// <param name="passportId">Номер комнаты</param>*
1723. */// <param name="error">Описание ошибки</param>*
1724. **public** **static** **bool** ValidateRoomNumberId(
1725. **string? roomNumber,**
1726. **ref** **string**? error)
1727. {
1728. **bool** ret = **true**;
1730. **var description = "Номер комнаты должен соответствовать формату «ANNN», " +**
1731. "где A –**\r\n**буква, обозначающая тип номера**\r\n** " +
1732. "(Л – люкс, П – полулюкс, О – одноместный, М – многоместный);**\r\n** " +
1733. "NNN – порядковый номер (цифры)";
1735. **if (string.IsNullOrWhiteSpace(roomNumber))**
1736. {
1737. ret = **false**;
1738. }
1740. **string number = roomNumber ?? string.Empty;**
1742. **if** (number.Length != 4)
1743. {
1744. ret = **false**;
1746. error = description;
1748. **return** ret;
1749. }
1751. **var** firstLiteral = number[0];
1753. **for** (**int** i = 0; i < ROOM\_NUMBER\_FIRST\_LITERAL.Length; i++)
1754. {
1755. **if (firstLiteral == ROOM\_NUMBER\_FIRST\_LITERAL[i])**
1756. {
1757. **break**;
1758. }
1760. **if (i == ROOM\_NUMBER\_FIRST\_LITERAL.Length - 1)**
1761. {
1762. ret = **false**;
1763. }
1764. }
1766. **for** (**int** i = 1; i < number.Length; i++)
1767. {
1768. **for** (**int** j = 0; j < NUMBER\_CHARS.Length; j++)
1769. {
1770. **if (number[i] == NUMBER\_CHARS[j])**
1771. {
1772. **break**;
1773. }
1775. **if (j == NUMBER\_CHARS.Length - 1)**
1776. {
1777. ret = **false**;
1778. }
1779. }
1781. **if** (!ret)
1782. {
1783. **break**;
1784. }
1785. **}**
1787. **if** (!ret)
1788. {
1789. error = description;
1790. **}**
1792. **return** ret;
1793. }
1795. ***/// <summary>***
1796. */// Генерация случайного ИД паспорта*
1797. */// </summary>*
1798. **public** **static** **string** GenerateRandomPassportId()
1799. {
1800. **var passportIdCharArray = new char[11];**
1802. **var** random = new Random();
1804. **for** (**int** i = 0; i < passportIdCharArray.Length; i++)
1805. **{**
1806. **if** (i == 4)
1807. {
1808. passportIdCharArray[i] = '-';
1810. **continue;**
1811. }
1813. passportIdCharArray[i] = NUMBER\_CHARS[random.Next(NUMBER\_CHARS.Length - 1)];
1814. }
1816. **return** new **string**(passportIdCharArray);
1817. }
1819. */// <summary>*
1820. ***/// Генерация случайного номера комнаты***
1821. */// </summary>*
1822. **public** **static** **string** GenerateRandomRoomNumber()
1823. {
1824. **var** random = new Random();
1826. **char**[] ret = new **char**[4];
1828. **for** (**int** i = 0; i < 4; i++)
1829. {
1830. **if (i == 0)**
1831. {
1832. ret[i] = ROOM\_NUMBER\_FIRST\_LITERAL[random.Next(ROOM\_NUMBER\_FIRST\_LITERAL.Length - 1)];
1833. }
1834. **else**
1835. **{**
1836. ret[i] = NUMBER\_CHARS[random.Next(NUMBER\_CHARS.Length - 1)];
1837. }
1838. }
1840. **return new string(ret);**
1841. }
1843. */// <summary>*
1844. */// Получить следующий по порядку номер комнаты*
1845. ***/// </summary>***
1846. **public** **static** **string** GetNextRoomNumber(**string** roomNumber)
1847. {
1848. **if** (!ValidateRoomNumberId(roomNumber))
1849. {
1850. **throw new ArgumentException(null, nameof(roomNumber));**
1851. }
1853. **int** number = 0;
1855. **int multiplexor = 1;**
1857. **for** (**int** i = 3; i > 0; i--)
1858. {
1859. number += NumCharToInt(roomNumber[i]) \* multiplexor;
1861. multiplexor \*= 10;
1862. }
1864. **char**[] ret = new **char**[4];
1866. ret[0] = roomNumber[0];
1868. **var** newNumberString = **string**.Format("{0}", number);
1870. **ret[1] = newNumberString[0];**
1871. ret[2] = newNumberString[1];
1872. ret[3] = newNumberString[2];
1874. **return** new **string**(ret);
1875. **}**
1877. */// <summary>*
1878. */// Перевод символа цифры в численное представление*
1879. */// </summary>*
1880. **public static int NumCharToInt(char c)**
1881. {
1882. **for** (**int** i = 0; i < NUMBER\_CHARS.Length; i++)
1883. {
1884. **if** (c == NUMBER\_CHARS[i])
1885. **{**
1886. **return** i;
1887. }
1888. }
1890. **throw new ArgumentOutOfRangeException(nameof(c));**
1891. }
1893. */// <summary>*
1894. */// The hash code for a String object is computed as*
1895. ***// s[0]\*31^(n-1) + s[1]\*31^(n-2) + ... + s[n-1]***
1896. *// using int arithmetic,*
1897. *// where s[i] is the i-th character of the string,*
1898. *// n is the length of the string,*
1899. *// and ^ indicates exponentiation.*
1900. ***// (The hash value of the empty string is zero.)***
1901. */// </summary>*
1902. */// <param name="\_key"></param>*
1903. */// <returns></returns>*
1904. **public** **static** **int** HashCode(**this** **string** \_key)
1905. **{**
1906. **int** h = 0;
1908. **if** (\_key.Length > 0)
1909. {
1910. **char[] val = \_key.ToCharArray();**
1912. **for** (**int** i = 0; i < \_key.Length; i++)
1913. {
1914. h = 31 \* h + val[i];
1915. **}**
1916. }
1918. **return** (h);
1919. }
1921. */// <summary>*
1922. */// Сортировка слиянием*
1923. */// </summary>*
1924. */// <typeparam name="T"></typeparam>*
1925. ***/// <param name="array">Исходный массив</param>***
1926. */// <param name="comparer">Сравнение элементов</param>*
1927. */// <returns></returns>*
1928. */// <exception cref="ApplicationException"></exception>*
1929. */// <exception cref="ArgumentNullException"></exception>*
1930. **public static T[] MergeSort<T>(**
1931. **this** T[] array,
1932. Comparison<T>? comparer = **null**)
1933. {
1934. **if** (array.Length == 0
1935. **|| array.Length == 1)**
1936. {
1937. **return** (T[])array.Clone();
1938. }
1940. **Comparison<T> comp;**
1942. **if** (comparer == **null**)
1943. {
1944. **if** (array[0] is not IComparable)
1945. **{**
1946. **throw** new ApplicationException(
1947. $"Comparator is not found and type {nameof(T)} is not IComparable");
1948. }
1950. **comp = (T f, T s) =>**
1951. {
1952. **if** (f != **null**)
1953. {
1954. **return** ((IComparable)f).CompareTo(s);
1955. **}**
1956. **else**
1957. {
1958. **throw** new ArgumentNullException(nameof(f));
1959. }
1960. **};**
1961. }
1962. **else**
1963. {
1964. comp = comparer;
1965. **}**
1967. T[] ret = (T[])array.Clone();
1969. ret = SortArray(array, 0, array.Length - 1);
1971. **return** ret;
1973. T[] SortArray(T[] array, **int** left, **int** right)
1974. {
1975. **if (left < right)**
1976. {
1977. **int** middle = left + (right - left) / 2;
1978. SortArray(array, left, middle);
1979. SortArray(array, middle + 1, right);
1980. **MergeArray(array, left, middle, right);**
1981. }
1983. **return** array;
1984. }
1986. **void** MergeArray(T[] array, **int** left, **int** middle, **int** right)
1987. {
1988. **var** leftArrayLength = middle - left + 1;
1989. **var** rightArrayLength = right - middle;
1990. **var leftTempArray = new T[leftArrayLength];**
1991. **var** rightTempArray = new T[rightArrayLength];
1992. **int** i, j;
1994. **for** (i = 0; i < leftArrayLength; i++)
1995. **{**
1996. leftTempArray[i] = array[left + i];
1997. }
1998. **for** (j = 0; j < rightArrayLength; j++)
1999. {
2000. **rightTempArray[j] = array[middle + 1 + j];**
2001. }
2003. i = 0;
2004. j = 0;
2005. **int k = left;**
2007. **while** (i < leftArrayLength
2008. && j < rightArrayLength)
2009. {
2010. **var comparison = comp(leftTempArray[i], rightTempArray[j]);**
2012. **if** (comparison <= 0)
2013. {
2014. array[k++] = leftTempArray[i++];
2015. **}**
2016. **else**
2017. {
2018. array[k++] = rightTempArray[j++];
2019. }
2020. **}**
2022. **while** (i < leftArrayLength)
2023. {
2024. array[k++] = leftTempArray[i++];
2025. **}**
2027. **while** (j < rightArrayLength)
2028. {
2029. array[k++] = rightTempArray[j++];
2030. **}**
2031. }
2032. }
2033. }
2034. }
2036. Файл «HostelContext.cs» - хранение данных, обработка заселений
2037. **using** SaodCP.DataStructures;
2038. **using** SaodCP.Models;
2040. **namespace** SaodCP.Database
2041. **{**
2042. **public** **static** **class** HostelContext
2043. {
2044. */// <summary>*
2045. */// Постояльцы - хеш-таблица*
2046. ***/// </summary>***
2047. **public** **static** OpenHashTable<**string**, Lodger> Lodgers { **get**; **set**; } = new();
2049. */// <summary>*
2050. */// Гостиничные номера - АВЛ-дерево*
2051. ***/// </summary>***
2052. **public** **static** Tree<**string**, Apartment> Apartments { **get**; **set**; } = new();
2054. */// <summary>*
2055. */// Данные о заселении - сортированный односвязный циклический список*
2056. ***/// </summary>***
2057. **public** **static** SortedOneWayCycledList<Accommodation> Accommodations { **get**; **set**; } = new(
2058. (a1, a2) => a1.ApartmentNumber.CompareTo(a2.ApartmentNumber));
2060. */// <summary>*
2061. ***/// Сериализация данных для сохранения***
2062. */// </summary>*
2063. */// <returns></returns>*
2064. **public** **static** DataSerializationScheme ToDataSheme()
2065. {
2066. **var data = new DataSerializationScheme();**
2068. data.Accommodations.AddRange(Accommodations);
2069. data.Lodgers.AddRange(Lodgers.Values);
2070. data.Apartments.AddRange(Apartments.**Select**(kv => kv.**Value**));
2072. **return** data;
2073. }
2075. */// <summary>*
2076. ***/// Десериализация данных***
2077. */// </summary>*
2078. */// <param name="data"></param>*
2079. **public** **static** **void** FromDataScheme(DataSerializationScheme data)
2080. {
2081. **Lodgers.Clear();**
2083. **foreach** (**var** lodger **in** data.Lodgers)
2084. {
2085. Lodgers.**Add**(lodger.PassportId, lodger);
2086. **}**
2088. Accommodations.Clear();
2090. **foreach** (**var** accommodation **in** data.Accommodations)
2091. **{**
2092. Accommodations.**Add**(accommodation);
2093. }
2095. Apartments = new();
2097. **foreach** (**var** apartment **in** data.Apartments)
2098. {
2099. Apartments.**Add**(apartment.Number, apartment);
2100. }
2101. **}**
2103. */// <summary>*
2104. */// Заполнить БД тестовыми данными*
2105. */// </summary>*
2106. **public static void InitTestData()**
2107. {
2108. **var** data = new DataSerializationScheme();
2110. data.Apartments.**Add**(new Apartment(Utils.Utils.GenerateRandomRoomNumber())
2111. **{**
2112. BedsNumber = 1,
2113. Equipment = "Телевизор, Микроволновка",
2114. HasToilet = **true**,
2115. RoomNumber = 2
2116. **});**
2118. data.Apartments.**Add**(new Apartment(Utils.Utils.GenerateRandomRoomNumber())
2119. {
2120. BedsNumber = 2,
2121. **Equipment = "Микроволновка",**
2122. HasToilet = **true**,
2123. RoomNumber = 3
2124. });
2126. **data.Lodgers.Add(new Lodger()**
2127. {
2128. PassportId = Utils.Utils.GenerateRandomPassportId(),
2129. Address = "LA 1151",
2130. BirthYear = 1951,
2131. **Name = "Mike"**
2132. });
2134. data.Lodgers.**Add**(new Lodger()
2135. {
2136. **PassportId = Utils.Utils.GenerateRandomPassportId(),**
2137. Address = "SaintP",
2138. BirthYear = 1976,
2139. Name = "Mikes"
2140. });
2142. FromDataScheme(data);
2143. }
2145. */// <summary>*
2146. ***/// Получить сущность определенного типа по идентификатору***
2147. */// </summary>*
2148. */// <typeparam name="T">Тип сущности для поиска</typeparam>*
2149. */// <param name="id">Идентификатор</param>*
2150. */// <returns></returns>*
2151. **public static T? FindById<T>(string id)**
2152. {
2153. **if** (typeof(T) == typeof(Lodger))
2154. {
2155. Lodger? lodger = Lodgers[id];
2157. **if** (lodger == **null**)
2158. {
2159. **return** **default**;
2160. }
2162. **return** (T?)(**object**?)lodger;
2163. }
2164. **else** **if** (typeof(T) == typeof(Apartment))
2165. {
2166. **Apartment? apartment = Apartments.Find(id);**
2168. **if** (apartment == **null**)
2169. {
2170. **return** **default**;
2171. **}**
2173. **return** (T?)(**object**?)apartment;
2174. }
2175. **else**
2176. **{**
2177. **return** **default**;
2178. }
2179. }
2181. ***/// <summary>***
2182. */// Добавление нового элемента*
2183. */// </summary>*
2184. */// <typeparam name="T">Тип элемента</typeparam>*
2185. */// <param name="entry">Новый элемент</param>*
2186. **public static void Add<T>(T entry)**
2187. {
2188. **if** (entry is Lodger lodger)
2189. {
2190. Lodgers.**Add**(lodger.PassportId, lodger);
2191. **}**
2192. **else** **if** (entry is Apartment apartment)
2193. {
2194. Apartments.**Add**(apartment.Number, apartment);
2195. }
2196. **else if (entry is Accommodation accommodation)**
2197. {
2198. Accommodations.**Add**(accommodation);
2199. }
2200. }
2202. */// <summary>*
2203. */// Выселить постояльца*
2204. */// </summary>*
2205. */// <param name="passportId">Номер паспорта постояльца</param>*
2206. ***/// <param name="roomNumber">Номер комнаты</param>***
2207. */// <param name="endDate">Дата выселения</param>*
2208. */// <param name="error">Текст ошибки</param>*
2209. */// <returns>Успех/неуспех</returns>*
2210. **public** **static** **bool** EndAccomodation(
2211. **string passportId,**
2212. **string** roomNumber,
2213. DateOnly endDate,
2214. **ref** **string** error)
2215. {
2216. **Accommodation? acc = null;**
2218. **foreach** (**var** accommodation **in** Accommodations)
2219. {
2220. **var** fromDate = accommodation.FromDate;
2221. **var toDate = accommodation.ToDate == DateOnly.MinValue**
2222. ? DateOnly.MaxValue
2223. : accommodation.ToDate;
2225. **if** (endDate >= fromDate
2226. **&& endDate <= toDate)**
2227. {
2228. acc = accommodation;
2230. **break**;
2231. **}**
2233. }
2235. **if** (acc == **null**)
2236. **{**
2237. error = $"Период проживания на дату {endDate} " +
2238. $"для постояльца {passportId} не найден";
2240. **return** **false**;
2241. **}**
2242. **else** **if** (acc.ToDate != DateOnly.MinValue)
2243. {
2244. error = $"{acc} " +
2245. $"для постояльца {passportId} не является открытым. " +
2246. **$"Выселение невозможно";**
2248. **return** **false**;
2249. }
2251. ***// с помощью копии объекта проверяем возможность записи***
2252. **var** copy = (Accommodation)acc.Clone();
2254. copy.ToDate = endDate;
2256. **var ret = ValidateAccomodationWrite(copy, out error, true);**
2258. **if** (!ret)
2259. {
2260. **return** ret;
2261. **}**
2263. *// все хорошо, меняем у оригинальной копии дату*
2264. acc.ToDate = endDate;
2266. **return ret;**
2267. }
2269. */// <summary>*
2270. */// Заселить постояльца с даты*
2271. ***/// </summary>***
2272. */// <param name="passportId">Номер паспорта постояльца</param>*
2273. */// <param name="roomNumber">Номер комнаты</param>*
2274. */// <param name="fromDate">Дата заселения</param>*
2275. */// <param name="error">Текст ошибки</param>*
2276. ***/// <returns></returns>***
2277. **public** **static** **bool** StartAccomodation(
2278. **string** passportId,
2279. **string** roomNumber,
2280. DateOnly fromDate,
2281. **ref string error)**
2282. {
2283. **var** acc = new Accommodation()
2284. {
2285. ApartmentNumber = roomNumber,
2286. **LodgerPassportId = passportId,**
2287. FromDate = fromDate
2288. };
2290. **bool** ret = ValidateAccomodationWrite(acc, **out** error);
2292. **if** (!ret)
2293. {
2294. **return** ret;
2295. }
2297. Accommodations.**Add**(acc);
2299. **return** **true**;
2300. }
2302. */// <summary>*
2303. */// Проверка периода проживания для записи в БД*
2304. */// </summary>*
2305. */// <param name="acc"></param>*
2306. ***/// <param name="error"></param>***
2307. */// <returns></returns>*
2308. **public** **static** **bool** ValidateAccomodationWrite(
2309. Accommodation acc,
2310. **out** **string** error,
2311. **bool existed = false)**
2312. {
2313. **bool** ret = **true**;
2314. error = **string**.Empty;
2316. ***// клиент и комната, для которых создаем период проживания***
2317. **var** lodger = FindById<Lodger>(acc.LodgerPassportId);
2318. **var** room = FindById<Apartment>(acc.ApartmentNumber);
2320. **if** (lodger == **null**)
2321. **{**
2322. error = $"Постоялец с номером {acc.LodgerPassportId}" +
2323. $" не найден";
2325. **return** **false**;
2326. **}**
2328. **if** (room == **null**)
2329. {
2330. error = $"Комната с номером {acc.ApartmentNumber}" +
2331. **$" не найдена";**
2333. **return** **false**;
2334. }
2336. **var fromDate = acc.FromDate;**
2337. **var** toDate = acc.ToDate;
2339. **if** (toDate == DateOnly.MinValue)
2340. {
2341. **toDate = DateOnly.MaxValue;**
2342. }
2344. **var** periodLength = toDate.DayNumber - fromDate.DayNumber;
2346. **if (periodLength < 1)**
2347. {
2348. ret = **false**;
2350. error = $"Период проживания {acc} имеет " +
2351. **$"некорректную продолжительность {periodLength} дн.";**
2353. **return** ret;
2354. }
2356. ***// проверка на пересечение периодов проживания для клиента***
2357. **foreach** (**var** accommodation **in** Accommodations)
2358. {
2359. **var** checkFromDate = accommodation.FromDate;
2360. **var** checkToDate = accommodation.ToDate;
2362. **if** (checkToDate == DateOnly.MinValue)
2363. {
2364. checkToDate = DateOnly.MaxValue;
2365. }
2367. *// если заселение уже существует,*
2368. *// проверяем по первичному ключу*
2369. **if** (existed
2370. && accommodation.FromDate == acc.FromDate
2371. **&& accommodation.LodgerPassportId == acc.LodgerPassportId**
2372. && accommodation.ApartmentNumber == acc.ApartmentNumber)
2373. {
2374. **continue**;
2375. }
2377. **if** (accommodation.LodgerPassportId == acc.LodgerPassportId
2378. && checkFromDate <= toDate
2379. && checkToDate >= fromDate)
2380. {
2381. **ret = false;**
2383. error = acc.ToString()
2384. + "конфликтует с "
2385. + accommodation.ToString();
2387. **break**;
2388. }
2389. }
2391. **if (!ret)**
2392. {
2393. **return** ret;
2394. }
2396. **byte[] dateLogdersNumArray = new byte[periodLength];**
2398. *// проверка на кол-во проживающих в комнате*
2399. **foreach** (**var** accommodation **in** Accommodations)
2400. {
2401. **var checkFromDate = accommodation.FromDate;**
2402. **var** checkToDate = accommodation.ToDate;
2404. **if** (checkToDate == DateOnly.MinValue)
2405. {
2406. **checkToDate = DateOnly.MaxValue;**
2407. }
2409. *// если заселение уже существует,*
2410. *// проверяем по первичному ключу*
2411. **if (existed**
2412. && accommodation.FromDate == acc.FromDate
2413. && accommodation.LodgerPassportId == acc.LodgerPassportId
2414. && accommodation.ApartmentNumber == acc.ApartmentNumber)
2415. {
2416. **continue;**
2417. }
2419. *// если заселение по комнате не пересекается, пропускаем*
2420. **if** (accommodation.ApartmentNumber != acc.ApartmentNumber
2421. **|| checkFromDate > toDate**
2422. || checkToDate < fromDate)
2423. {
2424. **continue**;
2425. }
2427. *// период пересечения найденного заселения с проверяемым*
2428. **var** periodFromDate = checkFromDate < fromDate
2429. ? fromDate :
2430. checkFromDate;
2432. **var** periodToDate = checkToDate > toDate
2433. ? toDate
2434. : checkToDate;
2436. ***// стартовая дата***
2437. **uint** startIdx = (**uint**)(periodFromDate.DayNumber - fromDate.DayNumber);
2439. *//длина периода*
2440. **uint** length = (**uint**)(periodToDate.DayNumber - periodFromDate.DayNumber);
2442. **if** (startIdx + length > dateLogdersNumArray.Length)
2443. {
2444. **throw** new ApplicationException("Checking internal error");
2445. }
2447. **for** (**uint** i = startIdx; i < startIdx + length; i++)
2448. {
2449. dateLogdersNumArray[i]++;
2451. ***// если кол-во уже заселенных + 1 (из нового заселения)***
2452. *// больше чем кол-во кроватей,*
2453. *// не разрешаем заселение*
2454. **if** (dateLogdersNumArray[i] + 1 > room.BedsNumber)
2455. {
2456. **error = $"На дату {fromDate.AddDays((int)i)} " +**
2457. $"кол-во заселенных постояльцев : {dateLogdersNumArray[i] + 1}. **\r\n**" +
2458. $"Макс. кол-во проживающих в номере {room.Number} : " +
2459. $"{room.BedsNumber}. **\r\n**Конфликт с {accommodation}";
2461. **ret = false;**
2463. **break**;
2464. }
2465. }
2467. **if** (!ret)
2468. {
2469. **break**;
2470. }
2471. **}**
2473. **return** ret;
2474. }
2475. }
2476. **}**
2477. Файл «DataSerializationScheme.cs» - модели для сохранения данных
2478. **using** SaodCP.Models;
2479. **using** System.Text.Json;
2481. **namespace** SaodCP.Database
2482. **{**
2483. */// <summary>*
2484. */// Схема для сериализации хранимых данных*
2485. */// </summary>*
2486. **public** **class** DataSerializationScheme
2487. **{**
2488. **public** List<Accommodation> Accommodations { **get**; **set**; } = new List<Accommodation>();
2490. **public** List<Apartment> Apartments { **get**; **set**; } = new List<Apartment>();
2492. **public List<Lodger> Lodgers { get; set; } = new List<Lodger>();**
2494. **public** **async** **static** Task<DataSerializationScheme?> FromStream(Stream stream)
2495. {
2496. **var** data = **await** JsonSerializer.DeserializeAsync<DataSerializationScheme>(stream);
2498. **return** data;
2499. }
2501. **public** **async** **static** Task<Stream?> ToStream(DataSerializationScheme data)
2502. **{**
2503. **var** stream = new MemoryStream();
2505. **await** JsonSerializer.SerializeAsync(stream, data);
2507. **return stream;**
2508. }
2509. }
2510. }
2511. Файл «Form1.cs» - главное окно программы, обработка пользовательского ввода
2512. **using** SaodCP.Database;
2513. **using** SaodCP.Models;
2514. **using** System.Text;
2516. **namespace SaodCP**
2517. {
2518. **public** **partial** **class** Hostel : Form
2519. {
2520. **public** Hostel()
2521. **{**
2522. InitializeComponent();
2523. }
2525. */// <summary>*
2526. ***/// Получение данных***
2527. */// </summary>*
2528. **private** **void** BindData()
2529. {
2530. **var** apartmentsArray = HostelContext
2531. **.Apartments**
2532. .**Select**(kv => kv.**Value**)
2533. .ToArray();
2535. **var** lodgersGrid = HostelContext
2536. **.Lodgers**
2537. .**Select**(kv => kv.**Value**)
2538. .ToArray();
2540. ApartmentsGrid.DataSource = apartmentsArray;
2541. **LodgersGrid.DataSource = lodgersGrid;**
2542. }
2544. **private** **void** Form1\_Load(**object** sender, EventArgs e)
2545. {
2546. **BindData();**
2547. }
2549. **private** **void** AccButton\_Click(**object** sender, EventArgs e)
2550. {
2551. **var current = ApartmentsGrid.CurrentRow?.DataBoundItem;**
2553. **if** (current == **null**)
2554. {
2555. MessageBox.Show("Данные не выбраны");
2556. **}**
2558. **if** (current is Apartment apartment)
2559. {
2560. new AccommodationForm(RoomNumber: apartment.Number).ShowDialog();
2561. **}**
2562. **else**
2563. {
2564. MessageBox.Show("Данные не выбраны");
2565. }
2566. **}**
2568. **private** **void** LodgerAccButton\_Click(**object** sender, EventArgs e)
2569. {
2570. **var** current = LodgersGrid.CurrentRow?.DataBoundItem;
2572. **if** (current == **null**)
2573. {
2574. MessageBox.Show("Данные не выбраны");
2575. }
2577. **if** (current is Lodger lodger)
2578. {
2579. new AccommodationForm(PassportId: lodger.PassportId)
2580. .ShowDialog();
2581. **}**
2582. **else**
2583. {
2584. MessageBox.Show("Данные не выбраны");
2585. }
2586. **}**
2588. */// <summary>*
2589. */// При каждой активации формы обновляем данные*
2590. */// </summary>*
2591. **private void Hostel\_Activated(object sender, EventArgs e)**
2592. {
2593. BindData();
2594. }
2596. **private void CreateLodgerButton\_Click(object sender, EventArgs e)**
2597. {
2598. new LodgerForm().ShowDialog();
2599. }
2601. **private void ChangeLodgerButton\_Click(object sender, EventArgs e)**
2602. {
2603. **var** item = LodgersGrid.CurrentRow?.DataBoundItem;
2605. **if** (item == **null**)
2606. **{**
2607. MessageBox.Show("Постоялец не выбран");
2609. **return**;
2610. }
2612. **if** (item is Lodger lodger)
2613. {
2614. new LodgerForm(lodger).ShowDialog();
2615. }
2616. **else**
2617. {
2618. **throw** new InvalidCastException();
2619. }
2620. }
2622. **private** **void** ApartmentButton\_Click(**object** sender, EventArgs e)
2623. {
2624. new ApartmentForm().ShowDialog();
2625. }
2627. **private** **void** ChangeApartmentButton\_Click(**object** sender, EventArgs e)
2628. {
2629. **var** item = LodgersGrid.CurrentRow?.DataBoundItem;
2631. **if (item == null)**
2632. {
2633. MessageBox.Show("Комната не выбрана");
2635. **return**;
2636. **}**
2638. **if** (item is Apartment lodger)
2639. {
2640. new ApartmentForm(lodger).ShowDialog();
2641. **}**
2642. **else**
2643. {
2644. **throw** new InvalidCastException();
2645. }
2646. **}**
2648. **private** **void** DeleteApartmentButton\_Click(**object** sender, EventArgs e)
2649. {
2650. **if** (ApartmentsGrid.CurrentRow == **null**)
2651. **{**
2652. MessageBox.Show("Номер не выбран");
2653. }
2654. **else**
2655. {
2656. **Apartment? apartment = ApartmentsGrid.CurrentRow.DataBoundItem as Apartment;**
2658. **if** (apartment == **null**)
2659. {
2660. MessageBox.Show("Номер не выбран");
2661. **}**
2662. **else**
2663. {
2664. **var** hasAcc = **false**;
2666. **foreach (var acc in HostelContext.Accommodations)**
2667. {
2668. **if** (acc.ApartmentNumber == apartment.Number)
2669. {
2670. hasAcc = **true**;
2672. **break**;
2673. }
2674. }
2676. **if (hasAcc)**
2677. {
2678. MessageBox.Show("Для комнаты существуют данные " +
2679. "о звселении. " +
2680. "Удаление невозможно");
2682. **return**;
2683. }
2685. **var** ret = HostelContext.Apartments.**Remove**(apartment.Number);
2687. **if** (!ret)
2688. {
2689. MessageBox.Show("Не удалось удалить номер");
2690. }
2691. **else**
2692. {
2693. BindData();
2694. }
2695. }
2696. **}**
2697. }
2699. **private** **void** DeleteLodgerButton\_Click(**object** sender, EventArgs e)
2700. {
2701. **if (LodgersGrid.CurrentRow == null)**
2702. {
2703. MessageBox.Show("Постоялец не выбран");
2704. }
2705. **else**
2706. **{**
2707. Lodger? lodger = LodgersGrid.CurrentRow.DataBoundItem **as** Lodger;
2709. **if** (lodger == **null**)
2710. {
2711. **MessageBox.Show("Постоялец не выбран");**
2712. }
2713. **else**
2714. {
2715. **var** hasAcc = **false**;
2717. **foreach** (**var** acc **in** HostelContext.Accommodations)
2718. {
2719. **if** (acc.LodgerPassportId == lodger.PassportId)
2720. {
2721. **hasAcc = true;**
2723. **break**;
2724. }
2725. }
2727. **if** (hasAcc)
2728. {
2729. MessageBox.Show("Для постояльца существуют данные " +
2730. "о звселении. " +
2731. **"Удаление невозможно");**
2733. **return**;
2734. }
2736. **var ret = HostelContext.Lodgers.Remove(lodger.PassportId);**
2738. **if** (!ret)
2739. {
2740. MessageBox.Show("Не удалось удалить постояльца");
2741. **}**
2742. **else**
2743. {
2744. BindData();
2745. }
2746. **}**
2747. }
2748. }
2750. */// <summary>*
2751. ***/// Поиск номера по коду номера***
2752. */// </summary>*
2753. **private** **void** ApartmentNumberFindButton\_Click(**object** sender, EventArgs e)
2754. {
2755. **var** apartmentNumber = ApartmentNumberFindTextBox.Text.Trim();
2757. **if** (**string**.IsNullOrWhiteSpace(apartmentNumber))
2758. {
2759. MessageBox.Show("Введен некорректный код номера");
2761. **return;**
2762. }
2764. **var** apartment = HostelContext.Apartments.Find(apartmentNumber);
2766. **if (apartment == null)**
2767. {
2768. MessageBox.Show($"Номер {apartmentNumber} не найден");
2770. **return**;
2771. **}**
2773. StringBuilder data = ApartmentToStringBuilder(apartment);
2775. **var** date = DateOnly.FromDateTime(DateTime.Today);
2777. data.AppendLine();
2779. data.AppendLine($"На дату {date} проживают:");
2781. **int cnt = 0;**
2783. **foreach** (**var** acc **in** HostelContext.Accommodations)
2784. {
2785. **if** (acc.ApartmentNumber != apartment.Number
2786. **|| acc.FromDate > date**
2787. || (acc.ToDate < date
2788. && acc.ToDate == DateOnly.MinValue))
2789. {
2790. **continue**;
2791. **}**
2793. **var** lodger = HostelContext.Lodgers[acc.LodgerPassportId];
2795. **if** (lodger == **null**)
2796. **{**
2797. **continue**;
2798. }
2800. cnt++;
2802. data.AppendLine($"{cnt}. {lodger.PassportId} - {lodger.Name}");
2803. }
2805. data.AppendLine($"Всего: {cnt} проживающих");
2807. MessageBox.Show(data.ToString());
2808. }
2810. */// <summary>*
2811. ***/// Преобразовать apartment в текстовое представление***
2812. */// </summary>*
2813. **private** **static** StringBuilder ApartmentToStringBuilder(Apartment apartment)
2814. {
2815. **var** data = new StringBuilder();
2817. data.AppendLine($"Номер {apartment.Number}");
2819. data.AppendLine($"Число комнат - {apartment.RoomNumber}");
2820. data.AppendLine($"Число кроватей - {apartment.BedsNumber}");
2821. **data.AppendLine($"Есть туалет - {apartment.HasToilet.ToString()}");**
2822. data.AppendLine($"Оборудование - {apartment.Equipment}");
2824. **return** data;
2825. }
2827. */// <summary>*
2828. */// Поиск номера по оборудованию*
2829. */// </summary>*
2830. **private** **void** ApartmentEquipmentFindButton\_Click(**object** sender, EventArgs e)
2831. **{**
2832. **var** search = ApartmentEquipmentFindTextBox.Text.Trim();
2834. **if** (**string**.IsNullOrEmpty(search))
2835. {
2836. **MessageBox.Show("Введите строку для поиска");**
2838. **return**;
2839. }
2841. **var data = new StringBuilder();**
2843. data.AppendLine($"Результаты поиска '{search}'");
2844. data.AppendLine();
2846. **int cnt = 0;**
2848. **foreach** (**var** (key, apartment) **in** HostelContext.Apartments)
2849. {
2850. *// поиск в тексте*
2851. **var found = Algorithms.BMTextSearch.TextSearch(**
2852. apartment.Equipment,
2853. search);
2855. **if** (found < 1)
2856. **{**
2857. **continue**;
2858. }
2860. cnt++;
2862. data.AppendLine($"{cnt}. **\n**{ApartmentToStringBuilder(apartment)}");
2863. data.AppendLine();
2864. }
2866. **data.AppendLine($"Всего: {cnt} номеров.");**
2868. MessageBox.Show(data.ToString());
2869. }
2871. ***/// <summary>***
2872. */// Поиск постояльца по номеру паспорта*
2873. */// </summary>*
2874. **private** **void** LodgerPassportFindButton\_Click(**object** sender, EventArgs e)
2875. {
2876. **var passportId = LodgerPassportFindTextBox.Text.Trim();**
2878. **if** (**string**.IsNullOrWhiteSpace(passportId))
2879. {
2880. MessageBox.Show("Введите номер паспорта");
2882. **return**;
2883. }
2885. **var** lodger = HostelContext.Lodgers[passportId];
2887. **if** (lodger == **null**)
2888. {
2889. MessageBox.Show($"Постоялец {passportId} не найден");
2891. **return;**
2892. }
2894. **var** lodgerData = new StringBuilder();
2896. **lodgerData.AppendLine($"Постоялец {lodger.PassportId} {lodger.Name}");**
2897. lodgerData.AppendLine();
2898. lodgerData.AppendLine($"Адрес - {lodger.Address}");
2899. lodgerData.AppendLine($"Год рождения - {lodger.BirthYear}");
2900. lodgerData.AppendLine();
2902. **var** date = DateOnly.FromDateTime(DateTime.Today);
2904. Accommodation? accommodation = **null**;
2906. **foreach (var acc in HostelContext.Accommodations)**
2907. {
2908. **if** (acc.FromDate < date
2909. && (acc.ToDate > date
2910. || acc.ToDate == DateOnly.MinValue)
2911. **&& acc.LodgerPassportId == lodger.PassportId)**
2912. {
2913. accommodation = acc;
2915. **break**;
2916. **}**
2917. }
2919. **if** (accommodation == **null**)
2920. {
2921. **lodgerData.AppendLine($"На {date} данных" +**
2922. $" о проживании не найдено");
2923. }
2924. **else**
2925. {
2926. **lodgerData.AppendLine($"На {date} проживает в " +**
2927. $"номере {accommodation.ApartmentNumber}");
2928. }
2930. MessageBox.Show(lodgerData.ToString());
2931. **}**
2933. **private** **void** LodgerNameFindButton\_Click(**object** sender, EventArgs e)
2934. {
2935. **var** name = LodgerNameFindTextBox.Text.Trim();
2937. **if** (**string**.IsNullOrWhiteSpace(name))
2938. {
2939. MessageBox.Show("Введите имя для поиска");
2941. **return;**
2942. }
2944. **var** data = new StringBuilder();
2946. **data.AppendLine($"Поиск '{name}'");**
2948. **int** cnt = 0;
2950. **foreach** (**var** (passportId, lodger) **in** HostelContext.Lodgers)
2951. **{**
2952. **if** (Algorithms.BMTextSearch.TextSearch(lodger.Name, name) > 0)
2953. {
2954. cnt++;
2956. **data.AppendLine($"{cnt}. {passportId} - {lodger.Name}");**
2957. }
2958. }
2960. MessageBox.Show(data.ToString());
2961. **}**
2963. */// <summary>*
2964. */// Очистить данные о проживающих*
2965. */// </summary>*
2966. ***/// <param name="sender"></param>***
2967. */// <param name="e"></param>*
2968. **private** **void** LodgerClearData\_Click(**object** sender, EventArgs e)
2969. {
2970. HostelContext.Lodgers.Clear();
2972. BindData();
2973. }
2975. **private** **void** ClearApartmentsDataButton\_Click(**object** sender, EventArgs e)
2976. **{**
2977. HostelContext.Apartments = new();
2979. BindData();
2980. }
2982. **private** **void** CreateLodgerAccButton\_Click(**object** sender, EventArgs e)
2983. {
2984. **var** item = LodgersGrid.CurrentRow?.DataBoundItem;
2986. **if (item == null)**
2987. {
2988. MessageBox.Show("Постоялец не выбран");
2990. **return**;
2991. **}**
2993. **if** (item is Lodger lodger)
2994. {
2995. new EditAccommodationForm(
2996. **type: AccommodationOperationType.OpenAccommodation,**
2997. lodgerPassportId: lodger.PassportId)
2998. .ShowDialog();
2999. }
3000. **else**
3001. **{**
3002. MessageBox.Show("Постоялец не выбран");
3003. }
3004. }
3006. **private void DeleteLodgerAccButton\_Click(object sender, EventArgs e)**
3007. {
3008. **var** item = LodgersGrid.CurrentRow?.DataBoundItem;
3010. **if** (item == **null**)
3011. **{**
3012. MessageBox.Show("Постоялец не выбран");
3014. **return**;
3015. }
3017. **if** (item is Lodger lodger)
3018. {
3019. new EditAccommodationForm(
3020. type: AccommodationOperationType.CloseAccomodation,
3021. **lodgerPassportId: lodger.PassportId)**
3022. .ShowDialog();
3023. }
3024. **else**
3025. {
3026. **MessageBox.Show("Постоялец не выбран");**
3027. }
3028. }
3029. }
3030. }
3031. Файл «LodgerForm.cs» - окно редактирования и создания постояльца
3032. **using** SaodCP.Database;
3033. **using** SaodCP.Models;
3035. **namespace** SaodCP
3036. **{**
3037. **public** **partial** **class** LodgerForm : Form
3038. {
3039. **public** LodgerForm()
3040. {
3041. **InitializeComponent();**
3042. }
3044. **public** Lodger? Lodger { **get**; **set**; }
3046. **public LodgerForm(Lodger lodger) : this()**
3047. {
3048. NameTextBox.Text = lodger.Name;
3049. BirhdayTextBox.Text = lodger.BirthYear.ToString();
3050. AddressTextBox.Text = lodger.Address;
3052. *// первичный ключ не редактируется*
3053. PassportTextBox.Text = lodger.PassportId;
3054. PassportTextBox.**ReadOnly** = **true**;
3056. **this.Lodger = lodger;**
3057. }
3059. **private** **void** SaveButton\_Click(**object** sender, EventArgs e)
3060. {
3061. **var passport = PassportTextBox.Text;**
3063. **string**? error = **null**;
3065. **if** (!Utils.Utils.ValidateLodgerPassportId(
3066. **passport,**
3067. **ref** error))
3068. {
3069. MessageBox.Show(error);
3071. **return;**
3072. }
3074. **var** year = BirhdayTextBox.Text;
3076. **int onlyYear;**
3078. **if** (!**int**.TryParse(year, **out** onlyYear))
3079. {
3080. MessageBox.Show("Неправильно введен год рождения");
3082. **return**;
3083. }
3084. **else** **if** (onlyYear < 0
3085. || onlyYear > DateOnly.FromDateTime(DateTime.Now).Year)
3086. **{**
3087. MessageBox.Show("Неправильно введен год рождения");
3089. **return**;
3090. }
3092. **var** address = AddressTextBox.Text;
3094. **var** name = NameTextBox.Text;
3096. **if (string.IsNullOrWhiteSpace(name))**
3097. {
3098. MessageBox.Show("Введите имя");
3100. **return**;
3101. **}**
3102. **else** **if** (name.Length < 3)
3103. {
3104. MessageBox.Show("Имя должно быть длинее 3х символов");
3106. **return;**
3107. }
3109. Lodger editLodger;
3111. **if (Lodger == null)**
3112. {
3113. editLodger = new Lodger();
3115. editLodger.PassportId = passport;
3116. **}**
3117. **else**
3118. {
3119. editLodger = Lodger;
3120. }
3122. editLodger.Address = address;
3123. editLodger.Name = name;
3124. editLodger.BirthYear = onlyYear;
3126. ***// если не было постояльца, значит создаем***
3127. **if** (Lodger == **null**)
3128. {
3129. **if** (HostelContext.Lodgers.ContainsKey(editLodger.PassportId))
3130. {
3131. **MessageBox.Show($"Постоялец с номером паспорта" +**
3132. $" {editLodger.PassportId} уже существует");
3134. **return**;
3135. }
3137. HostelContext.Lodgers.**Add**(passport, editLodger);
3138. }
3140. **this**.Close();
3141. **}**
3142. }
3143. }
3144. Файл «ApartmentForm.cs» - окно редактирования и создания готиничного номера
3145. **using** SaodCP.Database;
3146. **using** SaodCP.Models;
3148. **namespace** SaodCP
3149. **{**
3150. **public** **partial** **class** ApartmentForm : Form
3151. {
3152. **public** ApartmentForm()
3153. {
3154. **InitializeComponent();**
3155. }
3157. **public** Apartment? Apartment { **get**; **set**; }
3159. **public ApartmentForm(Apartment apartment) : this()**
3160. {
3161. **this**.Apartment = apartment;
3163. ApartmentNumberTextBox.Text = Apartment.Number;
3164. **ApartmentNumberTextBox.ReadOnly = true;**
3166. RoomNumberTextBox.Text = Apartment.RoomNumber.ToString();
3167. BedsNumberTextBox.Text = Apartment.BedsNumber.ToString();
3168. EquipmentTextBox.Text = Apartment.Equipment;
3170. HasToiletCheckBox.Checked = Apartment.HasToilet;
3171. }
3173. **private** **void** SaveButton\_Click(**object** sender, EventArgs e)
3174. **{**
3175. **var** apartmentNumber = ApartmentNumberTextBox.Text.Trim();
3177. **string**? error = **null**;
3179. **if (!Utils.Utils.ValidateRoomNumberId(**
3180. apartmentNumber,
3181. **ref** error))
3182. {
3183. MessageBox.Show(error);
3185. **return**;
3186. }
3188. **if** (!**int**.TryParse(
3189. **BedsNumberTextBox.Text.Trim(),**
3190. **out** **int** bedsNumber))
3191. {
3192. MessageBox.Show("Неправильный формат кол-ва кроватей");
3194. **return;**
3195. }
3197. **if** (!**int**.TryParse(
3198. BedsNumberTextBox.Text.Trim(),
3199. **out int roomNumber))**
3200. {
3201. MessageBox.Show("Неправильный формат кол-ва комнат");
3203. **return**;
3204. **}**
3206. **var** equipment = EquipmentTextBox.Text.Trim();
3208. **var** hasToilet = HasToiletCheckBox.Checked;
3210. Apartment apartment;
3212. **if** (Apartment == **null**)
3213. {
3214. **apartment = new(apartmentNumber);**
3215. }
3216. **else**
3217. {
3218. apartment = Apartment;
3219. **}**
3221. apartment.Equipment = equipment;
3222. apartment.RoomNumber = roomNumber;
3223. apartment.BedsNumber = bedsNumber;
3224. **apartment.HasToilet = hasToilet;**
3226. *// если создаем новый номер - сохраняем*
3227. **if** (Apartment == **null**)
3228. {
3229. **if (HostelContext.Apartments.Find(apartment.Number) != null)**
3230. {
3231. MessageBox.Show($"Номер {apartment.Number} уже существует");
3233. **return**;
3234. **}**
3236. HostelContext.Apartments.**Add**(
3237. apartment.Number,
3238. apartment);
3239. **}**
3241. Close();
3242. }
3243. }
3244. **}**
3245. Файл «AccomodationForm.cs» - окно просмотра заселений по номеру или постояльцу
3246. **using** SaodCP.Database;
3247. **using** SaodCP.Models;
3249. **namespace** SaodCP
3250. **{**
3251. **public** **partial** **class** AccommodationForm : Form
3252. {
3253. **public** **string** PassportId { **get**; **set**; } = **string**.Empty;
3255. **public string RoomNumber { get; set; } = string.Empty;**
3257. **public** AccommodationForm()
3258. {
3259. InitializeComponent();
3260. **}**
3262. **private** **void** SetLabel()
3263. {
3264. Text = "Заселения";
3266. **if** (!**string**.IsNullOrWhiteSpace(PassportId))
3267. {
3268. Text += $" Постоялец {PassportId}";
3269. }
3270. **else if (!string.IsNullOrWhiteSpace(RoomNumber))**
3271. {
3272. Text += $" Комната {RoomNumber}";
3273. }
3274. }
3276. **public** AccommodationForm(
3277. **string**? PassportId = **null**,
3278. **string**? RoomNumber = **null**) : **this**()
3279. {
3280. **if (!string.IsNullOrWhiteSpace(PassportId))**
3281. {
3282. **this**.PassportId = PassportId;
3283. }
3285. **if (!string.IsNullOrWhiteSpace(RoomNumber))**
3286. {
3287. **this**.RoomNumber = RoomNumber;
3288. }
3290. **SetLabel();**
3291. }
3293. **private** Accommodation[] FetchData()
3294. {
3295. **int cnt = 0;**
3297. **foreach** (**var** accommodation **in** HostelContext.Accommodations)
3298. {
3299. **if** ((**string**.IsNullOrEmpty(PassportId)
3300. **|| PassportId == accommodation.LodgerPassportId)**
3301. && (**string**.IsNullOrEmpty(RoomNumber)
3302. || RoomNumber == accommodation.ApartmentNumber))
3303. {
3304. cnt++;
3305. **}**
3306. }
3308. **var** ret = new Accommodation[cnt];
3310. **var i = 0;**
3312. **foreach** (**var** accommodation **in** HostelContext.Accommodations)
3313. {
3314. **if** ((**string**.IsNullOrEmpty(PassportId)
3315. **|| PassportId == accommodation.LodgerPassportId)**
3316. && (**string**.IsNullOrEmpty(RoomNumber)
3317. || RoomNumber == accommodation.ApartmentNumber))
3318. {
3319. **if** (i < ret.Length)
3320. **{**
3321. ret[i++] = accommodation;
3322. }
3323. }
3324. }
3326. **return** ret;
3327. }
3329. **private** **void** AccommodationForm\_Load(**object** sender, EventArgs e)
3330. **{**
3331. AccommodationGrid.DataSource = FetchData();
3332. }
3334. **private** **void** RemoveAccButton\_Click(**object** sender, EventArgs e)
3335. **{**
3336. **var** acc = AccommodationGrid.CurrentRow?.DataBoundItem;
3338. **if** (acc == **null**
3339. || acc is not Accommodation)
3340. **{**
3341. MessageBox.Show("Заселение не выбрано");
3343. **return**;
3344. }
3346. Accommodation accommodation = acc **as** Accommodation
3347. ?? **throw** new NullReferenceException();
3349. HostelContext.Accommodations.**Remove**(accommodation);
3351. AccommodationGrid.DataSource = FetchData();
3352. }
3353. }
3354. }
3355. Файл «EditAccomodationForm.cs» - окно создания заселений по постояльцу
3356. **using** SaodCP.Database;
3358. **namespace** SaodCP
3359. {
3360. **public partial class EditAccommodationForm : Form**
3361. {
3362. **private** **readonly** AccommodationOperationType \_type;
3363. **private** **readonly** **string**? \_lodgerPassportId;
3364. **private** **readonly** **string**? \_apartmentNumber;
3366. **public** EditAccommodationForm(
3367. AccommodationOperationType type,
3368. **string**? lodgerPassportId = **null**,
3369. **string**? apartmentNumber = **null**)
3370. **{**
3371. InitializeComponent();
3373. **var** array = new **string**[HostelContext.Apartments.Count];
3375. **int cnt = 0;**
3377. **foreach** (**var** (number, apartement) **in** HostelContext.Apartments)
3378. {
3379. array[cnt] = number;
3381. cnt++;
3382. }
3384. ApartmentNumberComboBox.DataSource = array;
3386. **if** (!**string**.IsNullOrEmpty(apartmentNumber))
3387. {
3388. ApartmentNumberComboBox.Text = apartmentNumber;
3389. }
3391. **if** (!**string**.IsNullOrWhiteSpace(lodgerPassportId))
3392. {
3393. LodgerPassportIdTextBox.Text = lodgerPassportId;
3394. }
3396. **switch** (type)
3397. {
3398. **case** AccommodationOperationType.OpenAccommodation:
3400. **Text = "Заселить";**
3402. **break**;
3404. **case** AccommodationOperationType.CloseAccomodation:
3406. Text = "Выселить";
3408. **break**;
3409. }
3411. **this**.\_type = type;
3412. **this**.\_lodgerPassportId = lodgerPassportId;
3413. **this**.\_apartmentNumber = apartmentNumber;
3414. }
3416. **private** **void** SaveAccButton\_Click(**object** sender, EventArgs e)
3417. {
3418. **var** passportId = LodgerPassportIdTextBox.Text.Trim();
3419. **var** apartmentNumber = ApartmentNumberComboBox.Text.Trim();
3420. **var onDate = DateOnly.FromDateTime(DateCalendar.SelectionRange.Start);**
3422. **if** (**string**.IsNullOrWhiteSpace(passportId)
3423. || **string**.IsNullOrWhiteSpace(apartmentNumber)
3424. || onDate == DateOnly.MinValue)
3425. **{**
3426. MessageBox.Show("Введены не все данные");
3428. **return**;
3429. }
3431. **string** error = **string**.Empty;
3432. **var** result = **false**;
3434. **switch** (\_type)
3435. **{**
3436. **case** AccommodationOperationType.OpenAccommodation:
3438. result = HostelContext.StartAccomodation(
3439. passportId,
3440. **apartmentNumber,**
3441. onDate,
3442. **ref** error);
3444. **break**;
3446. **case** AccommodationOperationType.CloseAccomodation:
3448. result = HostelContext.EndAccomodation(
3449. passportId,
3450. **apartmentNumber,**
3451. onDate,
3452. **ref** error);
3454. **break**;
3455. **}**
3457. **if** (!result)
3458. {
3459. MessageBox.Show(error);
3460. **}**
3461. **else**
3462. {
3463. Close();
3464. }
3465. **}**
3466. }
3468. **public** **enum** AccommodationOperationType
3469. {
3470. **None,**
3471. OpenAccommodation,
3472. CloseAccomodation
3473. }
3474. }
3475. Файл «Program.cs» - входная точка программы, инициализация тестовых данных
3476. **using** SaodCP.Database;
3478. **namespace** SaodCP
3479. {
3480. **internal static class Program**
3481. {
3482. */// <summary>*
3483. */// The main entry point for the application.*
3484. */// </summary>*
3485. **[STAThread]**
3486. **static** **void** Main()
3487. {
3488. *// To customize application configuration such as set high DPI settings or default font,*
3489. *// see https://aka.ms/applicationconfiguration.*
3490. **ApplicationConfiguration.Initialize();**
3492. *// инициализация тестовых данных*
3493. HostelContext.InitTestData();
3495. **Application.Run(new Hostel());**
3496. }
3497. }
3498. }