

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**(ШКОЛА)**

**Департамент программной инженерии и искусственного интеллекта**

ШЕВЕЛЕВ РОМАН ВАДИМОВИЧ

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ РАБОТЫ СО СПРАВОЧНИКАМИ ПРЕДМЕТНОЙ ОБЛАСТИ «ОНЛАЙН БИБЛИОТЕКА»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Фундаментальные структуры данных и алгоритмы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 «Программная инженерия», профиль «Программная инженерия»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студент группы Б9121-09.03.04 | | | | | | | |
|  |  |  | | | | Шевелев Р.В. | | | |
|  | | | | | | |  |  | (подпись) | | | |  | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | | | | | | |
|  | | | | | | |  |  | ст. преподаватель ДПИиИИ | | | | | | | |
|  | | | | | | |  |  | ученая степень, должность | | | | |  | Крестникова О.А. | |
|  | | | | | | |  |  | (подпись) | | | | |  | (ФИО) | |
|  | | | | | | | | | | | | | | | | |
| Регистрационный № | | | | | | |  |  | Защищен с оценкой | | | | | | | |
|  | | | | | | |  |  |  | | | | | | | |
|  | | | |  |  | |  |  | « |  | » |  |  | | | 2023 г. |
| (подпись) | | | |  | (ФИО) | |  |  |  | | | | | | | |
| « |  | » |  | | | 2023 г. |  |  |  | | | | | | | |

г. Владивосток

2023

# Оглавление

[Оглавление 2](#_Toc150788863)

[Введение 3](#_Toc150788864)

[1 Анализ предметной области 4](#_Toc150788865)

[1.1 Объекты предметной области 4](#_Toc150788866)

[1.2 Законы ПО 5](#_Toc150788867)

[2 Теоретическая часть 6](#_Toc150788868)

[2.1 Хеш-таблица 6](#_Toc150788869)

[2.1.1 Хеш-функция 7](#_Toc150788870)

[2.1.2 Разрешение коллизий методом открытой адресации 10](#_Toc150788871)

[2.2 Красно-чёрное дерево 12](#_Toc150788872)

[2.2.1 Вставка 15](#_Toc150788873)

[2.2.2 Удаление 17](#_Toc150788874)

[2.2.3 Элемент дерева двусвязный кольцевой список 20](#_Toc150788875)

[3 Требования к информационной системе 22](#_Toc150788876)

[3.1 Требования к данным 22](#_Toc150788877)

[3.1.1 Требования к входным данным 22](#_Toc150788878)

[3.1.2 Требования к выходным данным 23](#_Toc150788879)

[3.2 Функциональные требования 24](#_Toc150788880)

[3.2.1 Общие требования 24](#_Toc150788881)

[3.2.2 Требования для работы со справочником «Книги» 24](#_Toc150788882)

[4 Реализация 25](#_Toc150788883)

[4.1 Спецификация структур данных 25](#_Toc150788884)

[4.2 Описание среды разработки 37](#_Toc150788885)

[4.3 Руководство пользователя 37](#_Toc150788886)

[4.3.1 Работа со справочником «Книги» 38](#_Toc150788887)

[4.4 Тестирование 41](#_Toc150788888)

[Заключение 49](#_Toc150788889)

[Список литературы 50](#_Toc150788890)

[Приложение 51](#_Toc150788891)

# Введение

В сфере онлайн библиотек, где слова и истории важны не менее, чем квадратные метры недвижимости, эффективность и конкурентоспособность также становятся решающими факторами. На фоне постоянно меняющихся требований читателей и стремительно развивающихся технологий, онлайн библиотекам необходимо постоянно совершенствоваться, чтобы удовлетворить жажду знаний своих пользователей.

Одной из ключевых составляющих успешной онлайн библиотеки является разработка системы, обеспечивающей оптимальную ликвидность. В данном контексте ликвидность означает способность системы эффективно удовлетворять потребности читателей и библиотеки в мире литературы. Такая система должна предоставлять возможность автоматизировать и оптимизировать ключевые процессы, управлять информацией о книгах, а также обеспечивать высокий уровень обслуживания и удовлетворение потребностей читателей.

Целью курсового проекта является: разработка информационной системы «Электронная библиотека». Подсистема работы со справочниками, содержащими информацию о книгах и покупках.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области них

2. Изучить теоретические основы методов построения справочников.

3. Определить требования к информационной системе.

4. Реализовать информационную систему и провести тестирование.

# 1 Анализ предметной области

Требуется разработать информационную систему для работы со справочниками предметной области (ПО) «Электронная библиотека».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о книгах;
2. позволять искать информацию о книгах по автору и названию книги;
3. проверять целостность информации, представленной в справочниках.

1.1 Объекты предметной области

На примере ПО «Электронная библиотека».

Исходя из задач, которые должна решать информационная система, выделены объекты ПО, представленные в Таблице 1.

Таблица 1 – Объекты ПО

|  |  |
| --- | --- |
| Тип и название объекта | Описание объекта |
| Справочник «Книга» | Хранит информацию по каждой книге |

Каждая книга характеризуется следующими параметрами: название книги, автор (ФИО), год, жанр.

**Название книги** –является словом или сочетанием слов, состоящими из букв английского алфавита и/или цифр, первая буква первого слова – обязательно заглавная (или цифра). Слова (если их количество больше одного) разделены одним пробелом.

**ФИО** – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв английского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

**Год** – целое число от 0 до 2023.

**Жанр** – является словом или сочетанием слов, состоящими из букв английского алфавита. Первая буква первого слова – заглавная, остальные – строчные. Слова (если их количество больше одного) разделены одним пробелом.

Пример справочника «Книга» представлен в таблице 2.

Таблица 2 – Справочник «Книга»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название книги** | **Автор** | **Год** | **Жанр** |
| 1984 | Eric Artur Bler | 1949 | Fantastika |
| White nights | Fyodor Mikhailovich Dostoevskiy | 1848 | Povest |
| Voina i mir | Lev Nikolaevich Tolstoy | 1869 | Roman |
| Prestuplenie i nakazanie | Fyodor Mikhailovich Dostoevskiy | 1866 | Drama |
| Master i Margarita | Mikhail Afanasyevich Bulgakov | 1967 | Mistika |
| Kazaki | Lev Nikolaevich Tolstoy | 1863 | Povest |
| Garri Potter i filosofskiy kamen | Joan Ketlin Rowling | 1997 | Fentezi |
| Anna Karenina | Lev Nikolaevich Tolstoy | 1877 | Roman |
| Alisa v strane chudes | Charles Lyutvidzh Dodgson | 1865 | Fantastika |
| Idiot | Fyodor Mikhailovich Dostoevskiy | 1869 | Drama |

1.2 Законы ПО

1. Количество книг у авторов неограниченно;
2. У каждой книги есть ровно один жанр;
3. В один год может быть выпущено несколько книг;
4. ФИО автора неуникально;
5. Название книги – неуникально;
6. В совокупности ФИО + Название книги – уникально;

# 2 Теоретическая часть

Для работы со справочником «Книги», а именно с добавлением в него записей, требуется проверка на уникальность добавляемого ключа - названия книги и автора. Поскольку ключ уникален, то для этого поиска используется хеш-таблица.

В рамках курсового проекта в качестве хеш-функции была выбрана функция свёртки по 2. Хеш-таблица является динамической по размеру, поскольку количество записей в справочнике не ограничено. Разрешение коллизий происходит методом открытой адресации, так как он занимает меньше памяти, относительно метода цепочек [2].

При добавлении записи в справочнике «Книги» могут встретиться повторяющиеся название, ФИО, жанр, год. Для поиска, добавления и удаления неуникальных ключей название, ФИО, жанр, год, было решено использовать красно-чёрное дерево поиска, элементы которого представлены в виде структуры двусвязного цикличного списка (содержащий индексы добавляемых элементов), а при удалении узла дерева он заменяется на максимальный слева [1].

2.1 Хеш-таблица

Хеш-таблица — это специальный вид массива T [0...m-1], где каждый ключ из множества уникальных ключей U преобразуется в индекс ячейки массива с использованием хеш-функции. Существует два основных вида хеш-таблиц: с цепочками и с открытой адресацией. В хеш-таблице элементы представлены в виде пар (в случае открытой адресации) или списков пар (в случае цепочек).

Для выполнения операций, таких как добавление, удаление и поиск, сначала вычисляется хеш-значение для заданного ключа. Это хеш-значение i=hash(key) используется в качестве индекса для доступа к соответствующей ячейке массива. Операция затем выполняется над объектом, хранящимся в этой ячейке.

Иногда возникают ситуации, когда разные ключи приводят к одному и тому же хеш-значению. Это называется коллизией. Коллизии довольно распространены, особенно в больших хеш-таблицах. Поэтому важно иметь механизм разрешения коллизий.

Если все ключи заранее известны, можно избежать коллизий, выбрав такую идеальную хеш-функцию, которая равномерно распределит ключи по ячейкам, без коллизий. Однако это довольно редкое исключение. Хеш-таблицы, которые используют такие идеальные хеш-функции и не требуют механизма разрешения коллизий, называются хеш-таблицами с прямой адресацией.

2.1.1 Хеш-функция

Хеш-функция принимает в качестве аргумента ключ из множества U и возвращает соответствующий ему номер ячейки таблицы T [0...m-1]. Другими словами, хеш-функция отображает совокупность ключей U на множество ячеек хеш-таблицы T [0...m-1]: [1].

Хеш-функция «свёртка по 2» представляет собой метод хеширования, который используется для преобразования данных, чаще всего текстовых строк, в целочисленное значение. Этот метод основан на разбиении входных данных на непересекающиеся группы символов, каждая из которых представляет собой пару символов (байтов). Затем каждая из этих пар символов преобразуется в числовое значение, и эти значения комбинируются в единое целое число, которое используется в качестве хеш-кода. Процесс «свёртка по 2» можно разделить на следующие шаги:

1. Разбиение данных: Входные данные, такие как текстовая строка, разбиваются на непересекающиеся группы по два символа (байта). Например, строка "Hello, World!" будет разделена на следующие пары: "He", "ll", "o,", " W", "or", "ld", "!". Эти пары символов служат "блоками" для хеширования.
2. Преобразование в числовые значения: Каждая пара символов (байтов) преобразуется в числовое значение, используя, например, кодировку ASCII или Unicode. Это может быть выполнено путем преобразования символов в их числовые коды, которые представляют собой целые числа.
3. Комбинирование числовых значений: Полученные числовые значения комбинируются в единое целое число, например, с помощью операций побитового сдвига, побитового ИЛИ или сложения. Это число становится хеш-кодом для входных данных.
4. Модуль: Полученное целое число может быть подвергнуто операции деления по модулю на размерность хеш-таблицы (m), чтобы ограничить диапазон значений и получить индекс для хеш-таблицы.

Преимущества и особенности хеш-функции «свёртка по 2» включают в себя:

* Простоту реализации: Этот метод хеширования относительно прост в реализации и не требует сложных вычислений.
* Эффективность: Хеш-функция «свёртка по 2» может обеспечивать равномерное распределение хеш-кодов для разных входных данных, что помогает уменьшить вероятность коллизий.
* Быстроту вычислений: Операции, выполняемые в этом методе, обычно выполняются быстро, что делает его подходящим для хеширования в реальном времени.
* Использование хеш-функции «свёртка по 2» в справочнике «Книги» позволит эффективно хранить и быстро обрабатывать информацию о книгах, обеспечивая быстрый доступ к необходимым ячейкам. Хеш-функция применяется на совокупность атрибутов (уникальный ключ) «название книги + ФИО автора».

Предположим, что размер хеш-таблицы равен 20, на вход поступают записи Книги из нашей предметной области. Приведем 10 примеров:

1984:Eric Artur Bler:1949:Fantastika

White nights:Fyodor Mikhailovich Dostoevskiy:1848:Povest

Voina i mir:Lev Nikolaevich Tolstoy:1869:Roman

Prestuplenie i nakazanie:Fyodor Mikhailovich Dostoevskiy:1866:Drama

Master i Margarita:Mikhail Afanasyevich Bulgakov:1967:Mistika

Kazaki:Lev Nikolaevich Tolstoy:1863:Povest

Garri Potter i filosofskiy kamen:Joan Ketlin Rowling:1997:Fentezi

Anna Karenina:Lev Nikolaevich Tolstoy:1877:Roman

Alisa v strane chudes:Charles Lyutvidzh Dodgson:1865:Fantastika

Idiot:Fyodor Mikhailovich Dostoevskiy:1869:Drama

Посчитаем хеш:

h(1984EricArturBler) = h(19 + 84 + Er + ic + Ar + tu + rB + le + r) => 106 + 108 + 183 + 204 + 179 + 233 + 180 + 209 + 114 => 1516 % 20 = 16

h(WhitenightsFyodorMikhailovichDostoevskiy) = h(Wh + it + en + ig + ht + sF + yo + do + rM + ik + ha + il + ov + ic + hD + os + to + ev + sk + iy) => 191 + 221 + 211 + 208 + 220 + 185 + 232 + 211 + 191 + 212 + 201 + 213 + 229 + 204 + 172 + 226 + 227 + 219 + 222 + 226 => 4221 % 20 = 1

h(VoinaimirLevNikolaevichTolstoy) = h(Vo + in + ai + mi + rL + ev + Ni + ko + la + ev + ic + hT + ol + st + oy) => 197 + 215 + 202 + 214 + 190 + 219 + 183 + 218 + 205 + 219 + 204 + 188 + 219 + 231 + 232 => 3136 % 20 = 16

h(PrestuplenieinakazanieFyodorMikhailovichDostoevskiy) = h(Pr + es + tu + pl + en + ie + in + ak + az + an + ie + Fy + od + or + Mi + kh + ai + lo + vi + ch + Do + st + oe + vs + ki + y) => 194 + 216 + 233 + 220 + 211 + 206 + 215 + 204 + 219 + 207 + 206 + 191 + 211 + 225 + 182 + 211 + 202 + 219 + 223 + 203 + 179 + 231 + 212 + 233 + 212 + 121 => 5386 % 20 = 6

h(MasteriMargaritaMikhailAfanasyevichBulgakov) = h(Ma + st + er + iM + ar + ga + ri + ta + Mi + kh + ai + lA + fa + na + sy + ev + ic + hB + ul + ga + ko + v) => 174 + 231 + 215 + 182 + 211 + 200 + 219 + 213 + 182 + 211 + 202 + 173 + 199 + 207 + 236 + 219 + 204 + 170 + 225 + 200 + 218 + 118 => 4409 % 20 = 9

h(KazakiLevNikolaevichTolstoy) = h(Ka + za + ki + Le + vN + ik + ol + ae + vi + ch + To + ls + to + y) => 172 + 219 + 212 + 177 + 196 + 212 + 219 + 198 + 223 + 203 + 195 + 223 + 227 + 121 => 2797 % 20 = 17

h(GarriPotterifilosofskiykamenJoanKetlinRowling) = h(Ga + rr + iP + ot + te + ri + fi + lo + so + fs + ki + yk + am + en + Jo + an + Ke + tl + in + Ro + wl + in + g) => 168 + 228 + 185 + 227 + 217 + 219 + 207 + 219 + 226 + 217 + 212 + 228 + 206 + 211 + 185 + 207 + 176 + 224 + 215 + 193 + 227 + 215 + 103 => 4715 % 20 = 15

h(AnnaKareninaLevNikolaevichTolstoy) = h(An + na + Ka + re + ni + na + Le + vN + ik + ol + ae + vi + ch + To + ls + to + y) => 175 + 207 + 172 + 215 + 215 + 207 + 177 + 196 + 212 + 219 + 198 + 223 + 203 + 195 + 223 + 227 + 121 => 3385 % 20 = 5

h(AlisavstranechudesCharlesLyutvidzhDodgson) = h(Al + is + av + st + ra + ne + ch + ud + es + Ch + ar + le + sL + yu + tv + id + zh + Do + dg + so + n) => 173 + 220 + 215 + 231 + 211 + 211 + 203 + 217 + 216 + 171 + 211 + 209 + 191 + 238 + 234 + 205 + 226 + 179 + 203 + 226 + 110 => 4300 % 20 = 0

h(IdiotFyodorMikhailovichDostoevskiy) = h(Id + io + tF + yo + do + rM + ik + ha + il + ov + ic + hD + os + to + ev + sk + iy) => 173 + 216 + 186 + 232 + 211 + 191 + 212 + 201 + 213 + 229 + 204 + 172 + 226 + 227 + 219 + 222 + 226 => 3560 % 20 = 0

2.1.2 Разрешение коллизий методом открытой адресации

В рамках курсового проекта рассматривается метод линейного пробирования для разрешения коллизий в хеш-таблице. Хеш-таблица представлена массивом T, и при попытке вставки значения в ячейку, мы используем алгоритм линейного пробирования, который определяет, в какую ячейку вставить значение [1].

При линейном пробировании используется формула *t*=ℎ(𝑘+𝑐⋅𝑖), где *t* - адрес ячейки, 𝑘 - ключ, 𝑖 - номер текущей попытки при разрешении коллизии, а 𝑐 - фиксированный шаг. Это означает, что мы последовательно перебираем ячейки с увеличивающимся на каждой итерации шагом, чтобы найти свободную ячейку для вставки значения.

Таким образом, линейное пробирование представляет собой метод разрешения коллизий, основанный на последовательном переборе ячеек с постоянным шагом.

Пример работы метода разрешения коллизии методом открытой адресации представлен на рисунке 1.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Рисунок 1 – «Разрешение коллизии методом открытой адресации»

Однако по мере заполнения хеш-таблицы можно достигнуть её полного переполнения что приведёт к невозможности вставки нового элемента. Для решения этой проблемы необходимо проводить рехеширование – увеличение размеров таблицы с последующим её повторным заполнением. В рамках курсового проекта рехеширование происходит при 75% и 25% заполненности хеш-таблицы [3].

На рисунке 2 представлен пример хеш-таблицы на записях предметной области, приведенных в главе 1.1 «Объекты предметной области» в таблице 2 - Справочник «Книги».

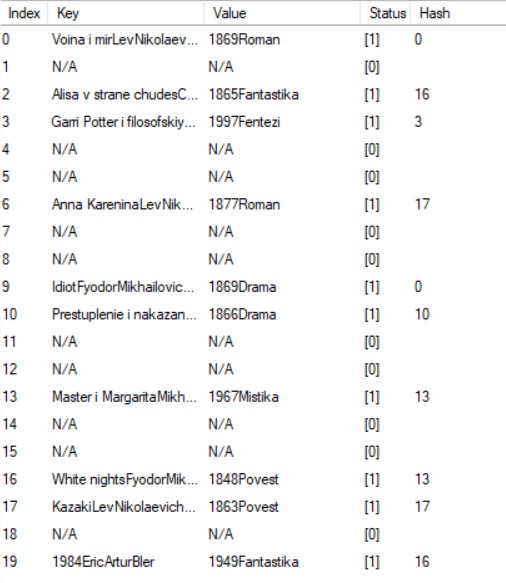


Рисунок 2 – «Хеш таблица «Книги» из модели предметной области»

2.2 Красно-чёрное дерево

Бинарное дерево поиска (БДП) представляет собой структуру данных, в которой каждый узел имеет не более двух потомков и удовлетворяет следующим условиям:

* Оба поддерева, левое и правое, также являются бинарными деревьями поиска.
* Значения ключей данных у всех узлов левого поддерева любого узла X меньше значения ключа данных самого узла X.
* Значения ключей данных у всех узлов правого поддерева любого узла X больше значения ключа данных самого узла X [1].

Каждый узел представляет собой запись вида: {left, data, right}, где left - ссылка на левое поддерево, right - ссылка на правое поддерево, data - данные, содержащиеся в узле.

Бинарное Дерево Поиска (БДП) требует определения операции сравнения "меньше" для ключей. Эта операция определяется следующим образом:

1. Если ключ вещественное число, то интерпретируем ее как операцию сравнения на числах;
2. Если ключ является строкой, то, пусть s1 и s2 строки: говорят, что строка s1 меньше строки s2, если при последовательной проверке символов (т.е. первый символ строки s1 проверятся с первым символом s2, и т.д.) нашелся такой символ в строке, что код символа меньше кода символа строки s2, несмотря на регистр символа. Например, «"Lev" < "leVa"».

В БДП определены следующие операции: добавление по ключу и значению, поиск по ключу (ключу, значению), удаление по ключу и значению, последующей заменой удаляемого узла либо на максимальный ключ его левого поддерева, либо на минимальный ключ его правого поддерева. В рамках курсового проекта при удалении будем заменять на минимальный ключ правого поддерева. Среднее время выполнение данных операций составляет О(lg(n)) [1].

Но возможен такой случай, что при добавлении дерево превратиться в список. Например, добавлялись последовательно такие элементы, что они были меньше (больше) предыдущих. Тогда производительность операций приближено равна производительности операций на списке, поэтому вместо обычных БДП лучше использовать самобалансирующиеся БДП. Случай, когда ключи совпадают, будет рассмотрен позже [1].

Самобалансирующиеся БДП – это БДП с балансировкой высоты, которые автоматически сохраняют высоту как можно меньше, когда операции вставки и удаления выполняются над деревом. Типичная операция, выполняемая деревьями – это вращение. Пусть дан узел B с дочерними узлами A и C. Правым поворотом в дереве с корнем B называется такой процесс перестановки связей так, чтобы узел B стал правым потомком узла A, а левым поворотом – B есть левый потомок узла C. Существует несколько видов самобалансирующихся деревьев: АВЛ, красно-черное, B-дерево, но для курсового проекта будет рассмотрено красно-черное дерево [1].

Красно-чёрное дерево – один из видов самобалансирующихся БДП, в котором каждый узел имеет атрибут цвета, принимающий либо «черный», либо «красный». При этом:

1. Узел может быть либо красным, либо чёрным и имеет двух потомков;
2. Корень дерева– чёрный узел;
3. Все листья, не содержащие данных – чёрные.
4. Оба потомка каждого красного узла – чёрные.
5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов [1].

Благодаря этим ограничениям путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего, и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям быть более эффективными в худшем случае, чем обычные БДП [1].

В случае совпадения ключей при добавлении, но различии значений, мы изменяем структуру узла. Вместо пары <ключ - значение>, узел теперь содержит <ключ - список значений>. Если узел уже содержит данный ключ, мы просто добавляем новое значение в соответствующий список значений узла.

Для реализации этой структуры данных мы используем двусвязный кольцевой список, позволяя эффективно добавлять новые значения в конец списка. Каждый узел содержит пару <ключ - список значений>, где список значений представлен двусвязным кольцевым списком.

Это позволяет эффективно управлять множеством значений для одного ключа в узле бинарного дерева поиска (БДП) и обеспечивает быстрый доступ к списку значений для конкретного ключа.

2.2.1 Вставка

Вставка в красно-черном дереве идет также, как и в БДП по следующему алгоритму:

1. Если корень пуст, то:
   1. добавляемый элемент становится корнем.
2. Иначе:
   1. если ключ добавляемого элемента меньше ключа корня, то:
      1. переход на шаг 1 для левого поддерева корня.
   2. Иначе если ключ добавляемого элемента больше ключа корня, то:
      1. переход на шаг 1 для правого поддерева корня.
   3. Иначе если ключ добавляемого элемента совпадает с ключом корня, то:
      1. добавляем в список значений корня значения добавляемого элемента [1].

Однако, данный алгоритм может нарушать свойства красно-черного дерева, например, если добавляемый узел красный, то нарушается либо свойства 2, либо 4, а если черный – 5. Чтобы восстановить свойства дерева, нужно балансировать дерево, и это достигается сменой цветов у узлов и поворотами [1].

Условимся, что вставляемый элемент имеет красный цвет, т.е. в зависимости какое свойство красно-черного дерева нарушено (2 или 4) будем их восстанавливать [1].

Пусть узел [1848] – вставляемый узел (рисунок 3), узел [1866] – родитель



Рисунок 3 – Вставляемый узел

узла [1848], узел [1869] – родитель узла [1866], т.е. дедушка узла [1848], [1949] – потомок узла [1869] такой, что [1949] != [1866], также условимся, что перед вставкой ни одно из свойств красно-черного дерева не нарушено [1] (рисунок 4).

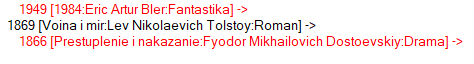


Рисунок 4 – Дерево до вставки элемента [1848]

Первый случай: [1949] – красный. Тогда меняем цвета у узлов [1866], [1949] и [1869] на противоположные. После проверяем первый и второй случай для [1949] (рисунок 5).

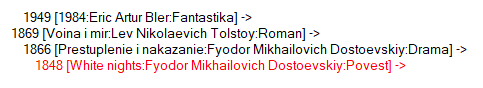


Рисунок 5 – Пример балансировки при вставке в первом случае

Второй случай: [1949] – черный (в конкретном примере это будет нулевой лист от [1869] не содержит данных) (рисунок 6). Тогда рассмотрим две возможных ситуации:

1. Если [1866] – левый (правый) потомок [1869], а [1848] – левый (правый) потомок [1869], то перекрашиваем [1866] и [1869] в противоположные цвета и поворачиваем узел [1869] направо (налево) (рисунок 7) [1].



Рисунок 6 – Дерево до балансировки

Изображение выглядит как текст, Шрифт, линия

Автоматически созданное описание

Рисунок 7 – Дерево после балансировки

1. Если [1866] – левый (правый) потомок [1869], а [1848] – правый (левый) потомок [1869], то поворачиваем узел [1866] налево (направо) (рисунок 8). После действуем по первой ситуации (рисунок 6, 7) [1].

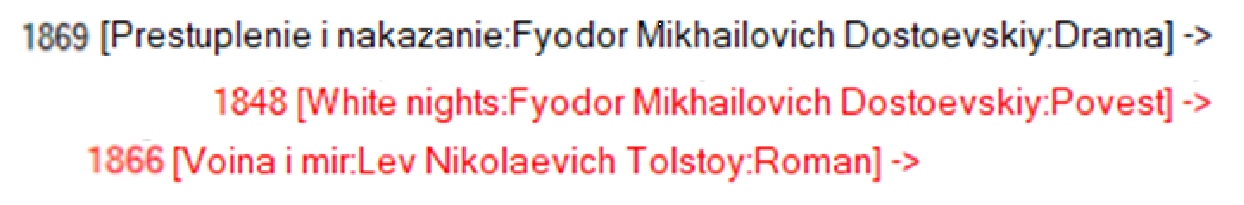


Рисунок 8 – Дерево до балансировки

2.2.2 Удаление

Удаление, как и в БДП, происходит по следующему алгоритму:

Пусть delete – удаляемая пара <ключ - значение>, current – узел, на котором находимся. Изначально current – корень дерева. Ищем узел, имеющий ключ такой же, как и у delete:

1. Если ключ current меньше delete, идем в правое поддерево, иначе в левое.
2. Если ключи совпали, то удаляем значение из списка узла.
3. Если у вершины нет детей, то изменяем указатель на неё у родителя на nil (нулевой лист).
4. Если у неё только один ребёнок, то делаем у родителя ссылку на него вместо этой вершины.
5. Если же имеются оба ребёнка, то находим вершину со следующим значением ключа. У такой вершины нет левого ребёнка (так как такая вершина находится в правом поддереве исходной вершины, и она самая левая в нем, иначе бы мы взяли ее левого ребенка. Сначала мы переходим в правое поддерево, а после спускаемся в левое до тех пор, пока у вершины есть левый ребенок).
6. Удаляем уже эту вершину описанным во втором пункте способом, скопировав её ключ в изначальную вершину [1].

Данный алгоритм не будет нарушать свойства красно-черного дерева, если удаляемый цвет узла красный. Но если удаляемый узел черный, то будут нарушены 2, 4 и 5 свойства.

Если брат этого ребёнка красный, то делаем вращение вокруг ребра между отцом и братом, тогда брат становится родителем отца. Красим его в чёрный, а отца — в красный цвет, сохраняя таким образом черную высоту дерева (рисунок 9). Хотя все пути по-прежнему содержат одинаковое количество чёрных узлов, сейчас *x* имеет чёрного брата и красного отца. Таким образом, мы можем перейти к следующему шагу.

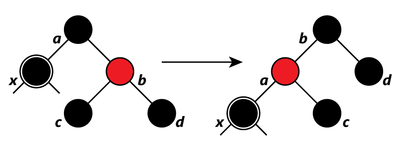


Рисунок 9 – Брат ребенка красный

Если брат текущей вершины был чёрным, то получаем три случая:

1. Оба ребёнка у брата чёрные. Красим брата в красный цвет и рассматриваем далее отца вершины. Делаем его черным, это не повлияет на количество чёрных узлов на путях, проходящих через b, но добавит один к числу чёрных узлов на путях, проходящих через x, восстанавливая тем самым влияние удаленного чёрного узла (рисунок 10). Таким образом, после удаления вершины черная глубина от отца этой вершины до всех листьев в этом поддереве будет одинаковой.

Изображение выглядит как круг, мультфильм, зарисовка, иллюстрация

Автоматически созданное описание

Рисунок 10 – Оба ребенка брата чёрные

1. Если у брата правый ребёнок чёрный, а левый красный, то перекрашиваем брата и его левого сына и делаем вращение (рисунок 11). Все пути по-прежнему содержат одинаковое количество чёрных узлов, но теперь у x есть чёрный брат с красным правым потомком, и мы переходим к следующему случаю. Ни x, ни его отец не влияют на эту трансформацию.

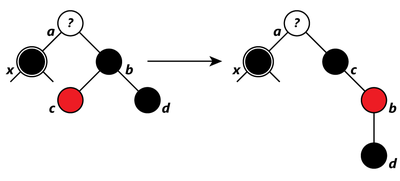


Рисунок 11 – Правый ребенок брата чёрный

1. Если у брата правый ребёнок красный, то перекрашиваем брата в цвет отца, его ребёнка и отца — в чёрный, делаем вращение (рисунок 12). Поддерево по-прежнему имеет тот же цвет корня, поэтому свойство 3 и 4 не нарушаются. Но у x теперь появился дополнительный чёрный предок: либо a стал чёрным, или он и был чёрным и b был добавлен в качестве чёрного дедушки. Таким образом, проходящие через x пути проходят через один дополнительный чёрный узел. Выходим из алгоритма.

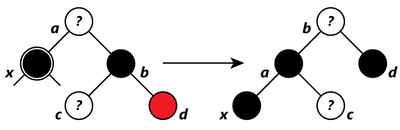


Рисунок 12 – Правый ребенок брата красный

Продолжаем тот же алгоритм, пока текущая вершина чёрная и мы не дошли до корня дерева. Но данные балансировки для случаев не всегда восстанавливают 2 свойство. Чтобы его восстановить после балансировок, красим корень в черный [1]. Из рассмотренных случаев ясно, что при удалении выполняется не более трёх вращений.

2.2.3 Элемент дерева двусвязный кольцевой список

Линейные списки являются чрезвычайно гибкой структурой, так как их размерность можно запросто уменьшить или увеличить, а их элементы доступны для вставки или удаления в любой позиции списка. Списки также можно объединять или разбивать на меньшие списки [4].

Элементы списка могут быть хаотично распределены по памяти. Из-за этого теряется возможность быстро получить элемент по индексу, а также не представляется возможным быстро скопировать весь список, однако, есть возможность вставки элементов за линейное время в любое место [4].

В качестве списка в рамках курсового проекта был выбран двусвязный кольцевой список с добавлением в конец.

Каждый узел двусвязного (двунаправленного) списка содержит два поля указателей — на следующий и предыдущий узел. При добавлении нового элемента, он добавляется в конец. Удаление элемента происходит по значению с последующим переопределением связей [4]. Пример красно-чёрного дерева, элементом которого является двусвязный кольцевой список представлен на рисунке 13. В качестве ключа было выбрано поле ФИО автора.

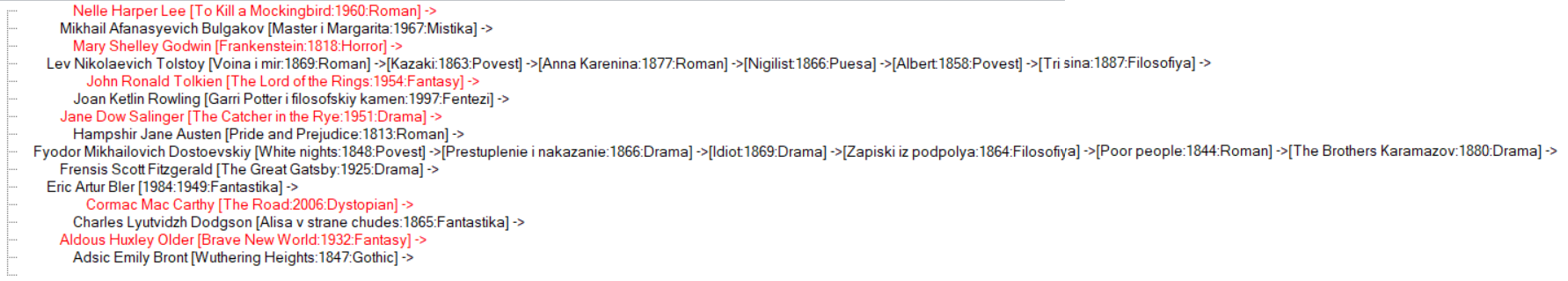


Рисунок 13 – Пример красно-чёрного дерева на примере записей справочника «Книги» по полю «ФИО»

# 3 Требования к информационной системе

В данной главе описываются требования к информационной системе для работы со справочниками, а именно: требования к входным данным системы, требования к ее выходным данным и требования к ее функционалу.

* 1. Требования к данным

3.1.1 Требования к входным данным

Основываясь на анализе ПО, входными данными является:

* текстовый файл books.txt, каждая строка файла содержит информацию об объекте «Книги», а именно название, ФИО, год, жанр, разделенный символом “:”;

Пример текстового файла:

1984:Eric Artur Bler:1949:Fantastika

White nights:Fyodor Mikhailovich Dostoevskiy:1848:Povest

Voina i mir:Lev Nikolaevich Tolstoy:1869:Roman

Prestuplenie i nakazanie:Fyodor Mikhailovich Dostoevskiy:1866:Drama

Master i Margarita:Mikhail Afanasyevich Bulgakov:1967:Mistika

Kazaki:Lev Nikolaevich Tolstoy:1863:Povest

Garri Potter i filosofskiy kamen:Joan Ketlin Rowling:1997:Fentezi

Anna Karenina:Lev Nikolaevich Tolstoy:1877:Roman

Alisa v strane chudes:Charles Lyutvidzh Dodgson:1865:Fantastika

Idiot:Fyodor Mikhailovich Dostoevskiy:1869:Drama

* Поля, вводимые пользователем: название книги, номер ФИО, жанр, год, для поиска в справочниках, добавления в справочники, удаления из справочников, (см. п. 1.1). Пользователь может ввести только то, что описано для каждого поля, (см. п. 1.1), другие данные не будут вводиться в поля ввода.

3.1.2 Требования к выходным данным

Выходными данными являются:

* текстовый файл books.txt, каждая строка файла содержит информацию об объекте «Книги», а именно название книги, ФИО автора, год, жанр, разделенный символом “:”.

Сообщения об ошибках:

* «Загрузите файл!»
* «Проверьте файл/данные!»
* «Данные успешно загружены!»
* «Запись успешно добавлена!»
* «Данная книга уже существует в справочнике.»
* «Запись успешно удалена!»
* «Книга не найдена!»
* «Записи не найдены.»
* «Пожалуйста, введите корректное название книги с первой заглавной буквой (или цифрой) и буквами английского алфавита, разделенное пробелом.»
* «Пожалуйста, введите корректное ФИО, где каждое слово начинается с заглавной буквы, и слова разделены пробелами.»
* «Пожалуйста, введите допустимый год в формате от 0 до 2023.»
* «Пожалуйста, введите корректный жанр с первой заглавной буквой и буквами английского алфавита, разделенный одним пробелом.»
* «Пожалуйста, введите допустимые года в формате от 0 до 2023.»
* «Год 'с' не может быть больше года 'до'.»
* «Загрузите файл перед записью!»
* «Не удалось открыть файл для записи.»
* «Данные успешно записаны в файл.»
  1. Функциональные требования

3.2.1 Общие требования

Информационная система должна позволять:

1. считывать данные справочников из текстовых файлов (см п 3.1.2);
2. добавлять информацию в справочник «Книги» по значениям: название, ФИО, год, жанр;
3. проверять добавляемые значения в справочник «Книги» на корректность;
4. проверять на уникальность совокупность ключей ФИО + название книги, введенное пользователем, при добавлении в справочник «Книги», и выводить соответствующую ошибку при не уникальности ключа;
5. удалять информацию из справочника «Книги» по уникальному ключу ФИО + название книги;
6. сохранять все внесённые изменения в справочник «Книги» в файлы (см. п. 3.1.2);

3.2.2 Требования для работы со справочником «Книги»

1. Считывать и сохранять данные о книгах;

2. Добавлять информацию о новых книгах. Запись о книге должна содержать следующую информацию: название книги, автор, год, жанр. При некорректном вводе данных во время добавления новой записи программа должна вывести сообщение о некорректности вводимых данных. При добавлении существующей записи программа должна вывести сообщение о невозможности добавления неуникальной записи;

3. Удалять информацию о книгах, имеющихся в справочнике.

4. Искать информацию об услуге по каждому её полю: название книги, автор, год, жанр, а также по комбинации полей название книги + ФИО автора.

# 4 Реализация

В данной главе содержится спецификация реализованных структур данных, описание графического интерфейса, а также приведены результаты тестирования.

4.1 Спецификация структур данных

**Класс Database** - класс, описывающий базу данных.

Поля:

std::vector<Book> books: вектор для хранения экземпляров класса Book.

Методы:

Конструктор Database ( ) – инициализирует экземпляр класса Database.

bool loadFromDatabaseFile(const std::string& filename) – Метод для загрузки данных о книгах из файла базы данных. Открывает указанный файл и считывает каждую строку. Разбирает каждую строку, используя разделитель ':' для извлечения информации о книге. Проверяет валидность названия книги, информации об авторе (FIO), годе и жанре. Если данные валидны, создает новый экземпляр класса Book и добавляет его в вектор books.

Входные параметры: filename - имя файла базы данных для загрузки.

Выходные параметры: возвращает «true», если загрузка прошла успешно, иначе «false».

std::vector<Book> getAllBooks() – Метод для получения всех книг, хранящихся в базе данных.

Входные параметры: нет.

Выходные параметры: Возвращает вектор, содержащий все книги в базе данных типа std::vector<Book>.

**Файл Algo.h** - файл, описывающий используемые алгоритмы и функции.

int gcd(int a, int b) – Метод для вычисления наибольшего общего делителя двух чисел.

Входные параметры: a - первое целое число, b - второе целое число.

Выходные параметры: Наибольший общий делитель чисел a и b.

int findCoprime(int n) – Находит взаимно простое число с заданным числом n.

Входные параметры: n - целое число.

Выходные параметры: возвращает взаимно простое число с n или -1, если таковое не найдено.

bool isNumeric(const std::string& str) – Проверяет, является ли строка числовой.

Входные параметры: str - строка для проверки.

Выходные параметры: возвращает true, если строка является числовой, иначе false.

System::Boolean IsValidFIO(System::String^ text) – Проверяет корректность введенных данных для ФИО.

Входные данные: System::String^ text - строка для проверки.

Выходные данные: true, если ФИО корректно, иначе выводит сообщение об ошибке и возвращает false.

System::Boolean IsValidYear(System::String^ text) – Проверяет корректность введенного года.

Входные данные: System::String^ text - строка для проверки.

Выходные данные: true, если год корректен, иначе false.

System::Boolean IsValidBookTitle(System::String^ text) – Проверяет корректность введенного названия книги.

Входные данные: System::String^ text - строка для проверки.

Выходные данные: true, если название книги корректно, иначе выводит сообщение об ошибке и возвращает false.

System::Boolean IsValidGenre(System::String^ text) – Проверяет корректность введенного жанра.

Входные данные: System::String^ text - строка для проверки.

Выходные данные: true, если жанр корректен, иначе выводит сообщение об ошибке и возвращает false.

**Файл List.h** - содержит объявление структуры list\_elem и функции для работы с двусвязным кольцевым списком.

struct list\_elem: структура, представляющая элемент двусвязного списка.

Поля:

int key - ключ (значение) элемента списка.

list\_elem\* next - указатель на следующий элемент списка.

list\_elem\* prev - указатель на предыдущий элемент списка.

Методы:

list\_elem\* list\_init () – инициализирует пустой список и возвращает указатель на корень списка.

Входные данные: нет.

Выходные данные: Указатель на корень списка.

void list\_insert(list\_elem\*& head, int value) – вставляет новый элемент с указанным значением в конец списка.

Входные данные: list\_elem\*& head - указатель на корень списка, int value - значение для вставки.

Выходные данные: нет.

void remove(list\_elem\*& head, int key) – удаляет элемент с указанным ключом из списка.

Входные данные: list\_elem\*& head - указатель на корень списка, int key - ключ элемента для удаления.

Выходные данные: нет.

void list\_erase(list\_elem\*& head) – очищает память, освобождая все элементы списка.

Входные данные: list\_elem\*& head - указатель на корень списка.

Выходные данные: нет.

std::string print\_list(list\_elem\* root, bool color) – формирует строку, представляющую значения элементов списка.

Входные данные: list\_elem\* root - указатель на корень списка, bool color - флаг цвета

Выходные данные: Строка, представляющая значения элементов списка.

**Файл Book.h** - файл, описывающий структуру книги Book.

struct Book

Поля:

std::string bookname; Название книги

FIO fio; - информация об авторе (ФИО)

int year; - год издания

std::string genre; - жанр

int index; - индекс книги

struct FIO

Поля:

std::string surname; - фамилия автора

std::string name; - имя автора

std::string patronymic; - отчество автора

bool operator==(const Book& other) const

Перегрузка оператора == для сравнения книг. Возвращает true, если книги равны по всем полям, иначе false.

**Class HashTable** - класс, описывающий хэш-таблицу.

Поля:

int initialSize: начальный размер таблицы.

int size: текущий размер таблицы.

int count: количество элементов в таблице.

int k: значение для линейного пробинга.

std::vector<HashEntry> table: вектор для хранения записей в таблице.

Методы:

hash(const std::string& key) – вычисляет хэш для заданного ключа.

Входные данные: "const std::string& key" - ключ для вычисления хэша.

Выходные данные: возвращает вычисленный хэш.

Детали реализации: данный метод использует простой алгоритм хэширования. Он итерирует по ключу, беря два символа за шаг, и добавляет их ASCII-значения к значению хэша. Затем возвращает остаток от деления хэша на размер таблицы.

Пример просчёта первичной хеш-функции, для данных, описанных в главе 3.1.1 «Требования к входным файлам»:

h(1984EricArturBler) = h(19 + 84 + Er + ic + Ar + tu + rB + le + r) => 106 + 108 + 183 + 204 + 179 + 233 + 180 + 209 + 114 => 1516 % 20 = 16

h(WhitenightsFyodorMikhailovichDostoevskiy) = h(Wh + it + en + ig + ht + sF + yo + do + rM + ik + ha + il + ov + ic + hD + os + to + ev + sk + iy) => 191 + 221 + 211 + 208 + 220 + 185 + 232 + 211 + 191 + 212 + 201 + 213 + 229 + 204 + 172 + 226 + 227 + 219 + 222 + 226 => 4221 % 20 = 1

h(VoinaimirLevNikolaevichTolstoy) = h(Vo + in + ai + mi + rL + ev + Ni + ko + la + ev + ic + hT + ol + st + oy) => 197 + 215 + 202 + 214 + 190 + 219 + 183 + 218 + 205 + 219 + 204 + 188 + 219 + 231 + 232 => 3136 % 20 = 16

h(PrestuplenieinakazanieFyodorMikhailovichDostoevskiy) = h(Pr + es + tu + pl + en + ie + in + ak + az + an + ie + Fy + od + or + Mi + kh + ai + lo + vi + ch + Do + st + oe + vs + ki + y) => 194 + 216 + 233 + 220 + 211 + 206 + 215 + 204 + 219 + 207 + 206 + 191 + 211 + 225 + 182 + 211 + 202 + 219 + 223 + 203 + 179 + 231 + 212 + 233 + 212 + 121 => 5386 % 20 = 6

h(MasteriMargaritaMikhailAfanasyevichBulgakov) = h(Ma + st + er + iM + ar + ga + ri + ta + Mi + kh + ai + lA + fa + na + sy + ev + ic + hB + ul + ga + ko + v) => 174 + 231 + 215 + 182 + 211 + 200 + 219 + 213 + 182 + 211 + 202 + 173 + 199 + 207 + 236 + 219 + 204 + 170 + 225 + 200 + 218 + 118 => 4409 % 20 = 9

h(KazakiLevNikolaevichTolstoy) = h(Ka + za + ki + Le + vN + ik + ol + ae + vi + ch + To + ls + to + y) => 172 + 219 + 212 + 177 + 196 + 212 + 219 + 198 + 223 + 203 + 195 + 223 + 227 + 121 => 2797 % 20 = 17

h(GarriPotterifilosofskiykamenJoanKetlinRowling) = h(Ga + rr + iP + ot + te + ri + fi + lo + so + fs + ki + yk + am + en + Jo + an + Ke + tl + in + Ro + wl + in + g) => 168 + 228 + 185 + 227 + 217 + 219 + 207 + 219 + 226 + 217 + 212 + 228 + 206 + 211 + 185 + 207 + 176 + 224 + 215 + 193 + 227 + 215 + 103 => 4715 % 20 = 15

h(AnnaKareninaLevNikolaevichTolstoy) = h(An + na + Ka + re + ni + na + Le + vN + ik + ol + ae + vi + ch + To + ls + to + y) => 175 + 207 + 172 + 215 + 215 + 207 + 177 + 196 + 212 + 219 + 198 + 223 + 203 + 195 + 223 + 227 + 121 => 3385 % 20 = 5

h(AlisavstranechudesCharlesLyutvidzhDodgson) = h(Al + is + av + st + ra + ne + ch + ud + es + Ch + ar + le + sL + yu + tv + id + zh + Do + dg + so + n) => 173 + 220 + 215 + 231 + 211 + 211 + 203 + 217 + 216 + 171 + 211 + 209 + 191 + 238 + 234 + 205 + 226 + 179 + 203 + 226 + 110 => 4300 % 20 = 0

h(IdiotFyodorMikhailovichDostoevskiy) = h(Id + io + tF + yo + do + rM + ik + ha + il + ov + ic + hD + os + to + ev + sk + iy) => 173 + 216 + 186 + 232 + 211 + 191 + 212 + 201 + 213 + 229 + 204 + 172 + 226 + 227 + 219 + 222 + 226 => 3560 % 20 = 0

linearProbe(int index, int attempt, int step) – используется для линейного пробинга в случае коллизий. Рассчитывает новый индекс, добавляя к оригинальному индексу произведение номера попытки на шаг, затем берет остаток от деления результата на размер таблицы.

Входные данные: "int index" - текущий индекс, "int attempt" - номер попытки, "int step" - шаг.

Выходные данные: возвращает новый индекс для разрешения коллизии.

resolveCollision(int index, const std::string& key, const std::vector<HashEntry>& table) – разрешает коллизии, используя линейный пробинг. Повторяет вызов метода linearProbe до тех пор, пока не будет найден незанятый слот. Использует метод getKey для получения ключа текущей книги по индексу.

Входные данные: "int index" - текущий индекс, "const std::string& key" - ключ, "const std::vector<HashEntry>& table" - текущая таблица.

Выходные данные: возвращает новый индекс для разрешения коллизии.

resizeTable(int newSize) – изменяет размер таблицы на новый указанный размер. Пересчитывает значение k с использованием функции findCoprime и затем перехэширует все существующие записи в новую таблицу, решая коллизии с использованием линейного пробинга.

Входные данные: "int newSize" - новый размер таблицы.

Выходные данные: нет.

getKey(Book book) – генерирует ключ для заданной книги, конкатенируя ее название, фамилию, имя и отчество

Входные данные: "Book book" - экземпляр класса Book для генерации ключа.

Выходные данные: Сгенерированный ключ в виде строки.

addAfterRemoval(int index) – вызывается после удаления записи и используется для заполнения пустого слота, оставшегося после удаления. Ищет следующий доступный слот с использованием линейного пробинга и перемещает запись в новое место.

Входные данные: "int index" - индекс, откуда была удалена запись.

Выходные данные: нет.

Конструктор HashTable(int initialSize) – инициализирует экземпляр класса HashTable.

Входные данные: "int initialSize" - начальный размер таблицы.

Выходные данные: нет.

Деструктор ~HashTable() – освобождает ресурсы, используемые экземпляром класса HashTable.

Входные данные: нет.

Выходные данные: нет.

void clear() – очищает таблицу, устанавливая все элементы в неактивное состояние и восстанавливает исходный размер.

Входные данные: нет.

Выходные данные: нет.

void add(const std::string& bookname, const std::string& surname, const std::string& name, const std::string& patronymic, int year, const std::string& genre, int num)

Входные данные: "const std::string& bookname", "const std::string& surname", "const std::string& name", "const std::string& patronymic", "int year", "const std::string& genre", "int num" - данные о книге.

Выходные данные: нет.

void remove(const std::string& bookname, const std::string& surname, const std::string& name, const std::string& patronymic) – удаляет книгу из таблицы.

Входные данные: "const std::string& bookname", "const std::string& surname", "const std::string& name", "const std::string& patronymic" - данные о книге, которую нужно удалить из таблицы.

Выходные данные: нет.

const Book search(const std::string& surname, const std::string& name, const std::string& patronymic, const std::string& bookname) – ищет книгу в таблице по заданным параметрам.

Входные данные: "const std::string& surname", "const std::string& name", "const std::string& patronymic", "const std::string& bookname" - данные для поиска книги в таблице.

Выходные данные: возвращает экземпляр класса Book.

std::vector<Book> returnTable() – возвращает вектор, содержащий все книги в таблице.

Входные данные: нет.

Выходные данные: возвращает вектор экземпляров класса Book.

void updateIndex(int newIndex) – обновляет индекс книг в таблице.

Входные данные: "int newIndex" - новый индекс для обновления.

Выходные данные: нет.

int hashF(const std::string& key) – дубликат функции hash, но описанный в разделе public

int returnK() – возвращает значение k (используемое для линейного пробинга).

Входные данные: нет.

Выходные данные: возвращает значение k.

int returnSize() – возвращает текущий размер таблицы.

Входные данные: нет.

Выходные данные: возвращает текущий размер таблицы.

**Файл RBTree.h** - содержит объявление структуры tree\_elem\_generic и функции для работы с красно-черным деревом.

tree\_elem\_generic<T>\* root\_init\_generic(tree\_elem\_generic<T>\* nullnode) – Инициализирует корень дерева пустым узлом.

Входные данные: Указатель на пустой узел (nullnode).

Выходные данные: Указатель на корень дерева.

tree\_elem\_generic<T>\* null\_init\_generic() – Инициализирует пустой узел.

Входные данные: нет.

Выходные данные: Указатель на новый пустой узел.

tree\_elem\_generic<T>\* search(tree\_elem\_generic<T>\* root, tree\_elem\_generic<T>\* nullnode, T data) – Выполняет поиск узла с заданными данными в дереве.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), данные для поиска (data).

Выходные данные: Указатель на найденный узел или пустой узел, если узел с заданными данными не найден.

void rightRotate(tree\_elem\_generic<T>\* element, tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode) – Выполняет правый поворот вокруг заданного узла.

Входные данные: Указатель на узел для поворота (element), указатель на корень дерева, указатель на пустой узел (nullnode).

Выходные данные: нет.

void leftRotate(tree\_elem\_generic<T>\* element, tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode) – Выполняет левый поворот вокруг заданного узла.

Входные данные: Указатель на узел для поворота (element), указатель на корень дерева, указатель на пустой узел (nullnode).

Выходные данные: нет.

void insert\_balance(tree\_elem\_generic<T>\*& element, tree\_elem\_generic<T> \* & root, tree\_elem\_generic<T>\* nullnode) – Выполняет балансировку дерева после вставки нового узла.

Входные данные: Указатель на вставленный узел (element), указатель на корень дерева, указатель на пустой узел (nullnode).

Выходные данные: нет.

tree\_elem\_generic<T>\* insert(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode, T data, int index) – Вставляет новый узел с заданными данными и индексом в дерево.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), данные для вставки (data), индекс для связанного списка (index).

Выходные данные: Указатель на корень дерева после вставки.

void replace\_node(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode, tree\_elem\_generic<T>\* n, tree\_elem\_generic<T>\* child) – Заменяет узел в дереве другим узлом.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), узел для замены (n), новый узел (child).

Выходные данные: нет.

void erase\_balance(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode, tree\_elem\_generic<T>\* element) – Выполняет балансировку дерева после удаления узла.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), узел, который был удален (element).

Выходные данные: нет.

tree\_elem\_generic<T>\* left\_max(tree\_elem\_generic<T>\* element, tree\_elem\_generic<T>\* nullnode) – Находит максимальный узел в левом поддереве заданного узла.

Входные данные: Указатель на узел, для которого выполняется поиск максимального узла (element), указатель на пустой узел (nullnode).

Выходные данные: Указатель на максимальный узел в левом поддереве.

void search\_between(tree\_elem\_generic<T>\* root, tree\_elem\_generic<T>\* nullnode, int year1, int year2, list\_elem\*& result\_list) – Выполняет поиск всех индексов в дереве, связанных с узлами, чьи данные находятся в заданном диапазоне.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), начало и конец диапазона данных (year1, year2), список для сохранения результатов (result\_list).

Выходные данные: нет.

tree\_elem\_generic<T>\* erase(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode, T data, int number) – Удаляет узел с заданными данными и номером индекса из дерева.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), данные для удаления (data), номер индекса для удаления из списка (number).

Выходные данные: Указатель на корень дерева после удаления.

void delete\_list(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode) – Удаляет связанный список в узле и освобождает память.

Входные данные: Указатель на узел, для которого выполняется удаление списка (root), указатель на пустой узел (nullnode).

Выходные данные: нет.

void delete\_node(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode) – Удаляет узел, включая связанный список в узле и освобождает память.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode).

Выходные данные: нет.

tree\_elem\_generic<T>\* delete\_tree(tree\_elem\_generic<T>\*& root, tree\_elem\_generic<T>\* nullnode) – Удаляет все узлы дерева, включая связанные списки и освобождает память.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode).

Выходные данные: Указатель на пустой узел (nullnode) после удаления дерева.

void index\_update(tree\_elem\_generic<T>\* root, tree\_elem\_generic<T>\* nullnode, int index, int cur\_size) – Обновляет индексы в связанных списках дерева после удаления индекса из списка.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), новый индекс (index), текущий размер списка (cur\_size).

Выходные данные: нет.

std::string print(tree\_elem\_generic<T>\* root, tree\_elem\_generic<T>\* nullnode, int h, int ln) – Возвращает строковое представление дерева в виде отформатированной строки.

Входные данные: Указатель на корень дерева, указатель на пустой узел (nullnode), отступ для форматирования (h), количество новых строк для разделения уровней (ln).

Выходные данные: Строковое представление дерева.

4.2 Описание среды разработки

Для разработки была выбрана среда разработки WinForms, входящая в пакет Visual Studio — мощную и удобную интегрированную среду разработки от Microsoft. WinForms предоставляет средства для создания графических пользовательских интерфейсов (GUI) приложений на языке программирования C++. Среда включает в себя графический дизайнер, который облегчает создание и редактирование интерфейса, а также интегрированный отладчик для обеспечения эффективной отладки кода.

Язык программирования C++ был выбран для реализации приложения, обеспечивая удобство разработки и возможность использования различных библиотек и фреймворков. WinForms предоставляет доступ к богатому набору элементов управления, что упрощает создание интерфейса и взаимодействие с пользователем.

Одной из особенностей WinForms является возможность изменять графический интерфейс в режиме выполнения программы или в режиме отладки. Это облегчает настройку интерфейса под конкретные задачи пользователя и обеспечивает удобство при дебаге приложения.

4.3 Руководство пользователя

При запуске программы появляется окно Файлы, в котором пользователь может импортировать и экспортировать данные для справочников (рисунок 14). Сверху находятся вкладки, из которых можно попасть в окна для работы со справочниками и просмотра структур данных.

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 14 – «Окно файлы»

4.3.1 Работа со справочником «Книги»

Перейдя во вкладку «Книги», можно наблюдать функционал работы со справочником «Книги». На рисунке 15 изображено окно отображения справочника «Книги».

Для выполнения поиска по справочнику «Книги» необходимо ввести одно из четырёх полей, а затем нажать одну из кнопок «Найти». Для сброса поиска необходимо нажать кнопку «Очистить поиск». Поля для поиска в справочнике «Книги» приведены на рисунке 16.

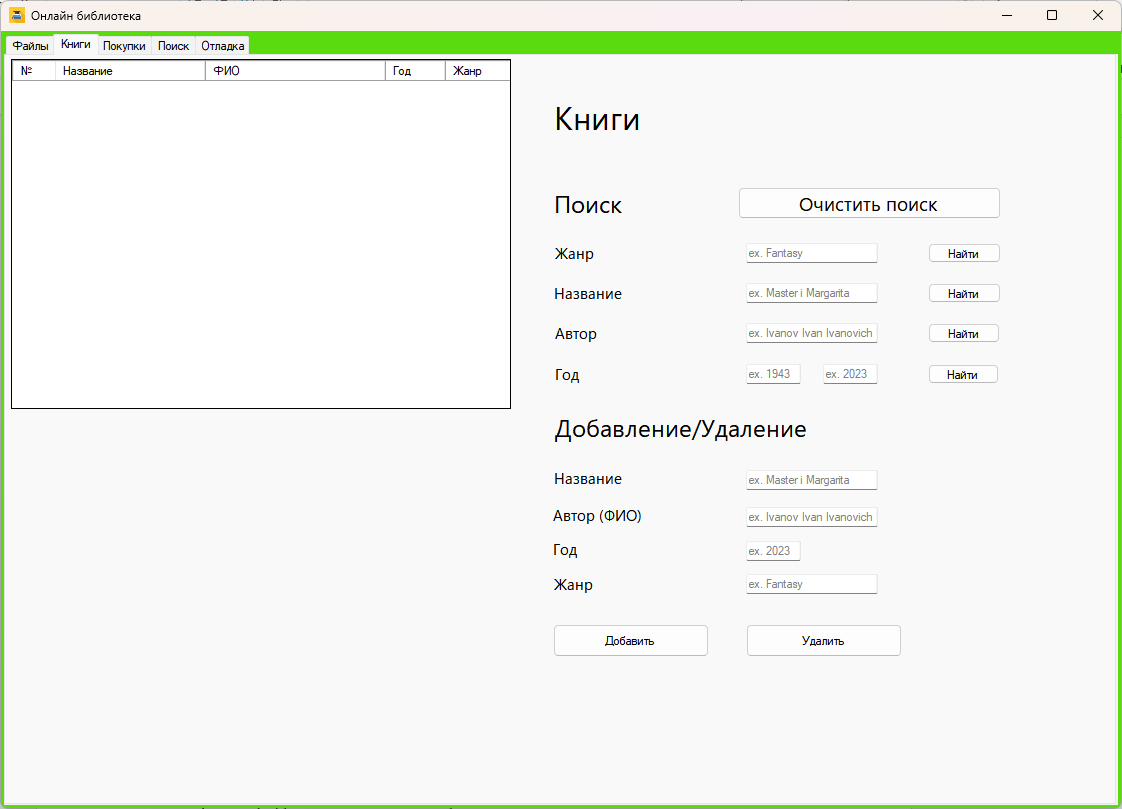


Рисунок 15 – «Окно Книги»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 16 – «Поиск в справочнике Книги»

Для добавления в / удаления из справочник(-а) «Книги» необходимо ввести поля для добавления / удаления и нажать на кнопку «Добавить» / «Удалить». Пример добавления / удаления приведён на рисунке 17.



Рисунок 17 – «Добавление/удаление в справочнике Книги»

Для выполнения поиска по двум параметрам (Название книги и автор книги) необходимо перейти во вкладку «Поиск» (рисунок 18).

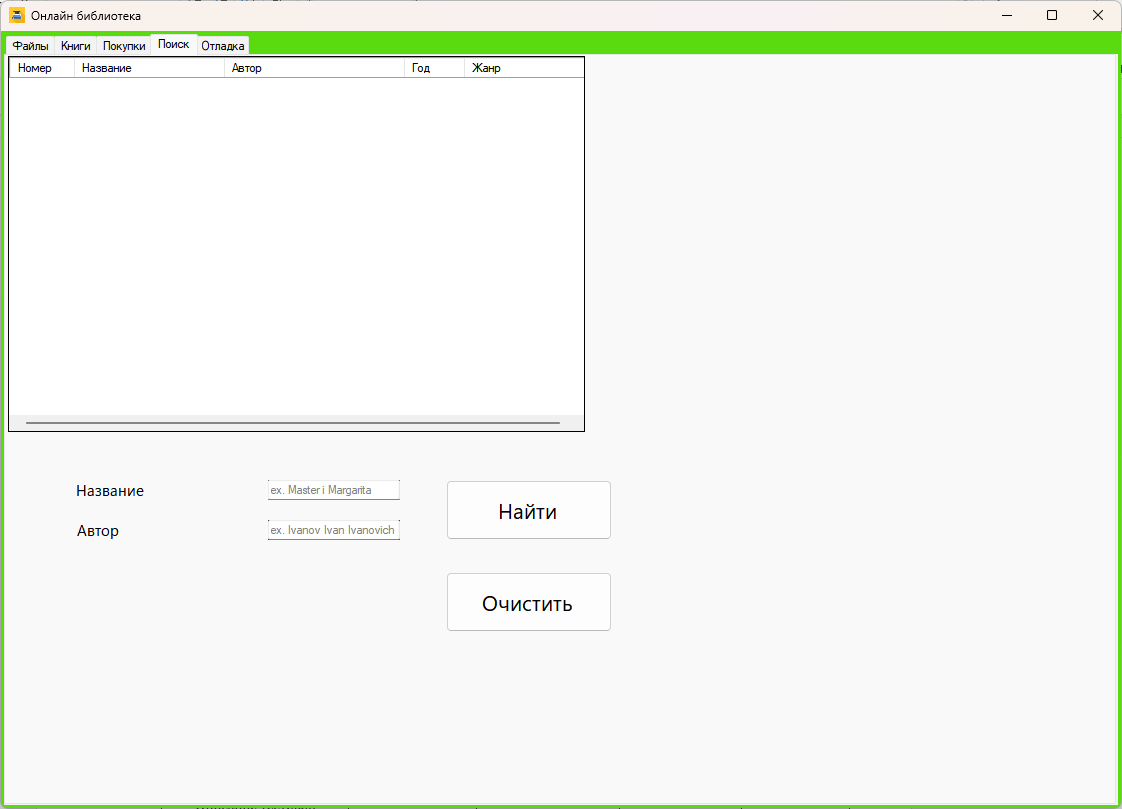


Рисунок 18 – «Окно Поиск»

Чтобы найти книгу необходимо ввести название и автора в соответствующие поля и нажать кнопку «Найти». Для очистки результатов поиска необходимо нажать кнопку «Очистить».

Переходя в окно «Отладка» можем наблюдать функционал для вывода структур данных, где каждая кнопка отвечает за вывод соответствующей ей структуре. Окно «Отладка» представлено на рисунке 19.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 19 – «Окно Отладка»

4.4 Тестирование

Тестирование проводилось методом черного ящика, результаты которого представлены в Таблицах 3, 4, 5 (ключевые поля выделены полужирным шрифтом).

Таблица 3 – Тестирование работы со справочником «Книги»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Вектор записей с данными о книге (название, автор, год, жанр) | Запись о книге (название, автор, год, жанр) | Вектор записей с данными о книге (название, автор, год, жанр) | Сообщение |
| Работа с файлом | | | | | |
| 1 | Попытка открыть несуществующий файл | - | - | - | Рисунок 20 |
| 2 | Нажатие кнопки экспортировать до загрузки файла | - | - | - | Рисунок 21 |
| 3 | Нажатие кнопки Очистить поиск до загрузки файла | - | - | - | Рисунок 22 |

*Продолжение таблицы 3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Вектор записей с данными о книге (название, автор, год, жанр) | Запись о книге (название, автор, год, жанр) | Вектор записей с данными о книге (название, автор, год, жанр) | Сообщение |
| 4 | Нажатие кнопки Найти в окне Поиск до загрузки файла | - | - | - | Рисунок 22 |
| 5 | Вывод структур деревьев до загрузки файла в окне Debug | - | - | - | Tree is empty. |
| 6 | Вывод структуры хеш-таблицы до загрузки файла в окне Debug | - | - | - | - |
| 7 | Нажатие кнопок Найти/Добавить/Удалить в справочнике Книги до загрузки файла | - | - | - | Рисунок 22 |
| 8 | Добавление существующего файла | - | - | - | Рисунок 23 |
| Добавление | | | | | |
| 9 | Добавление книги, которой еще нет в справочнике | - | Voina i mir:Lev Nikolaevich Tolstoy:1869:  Roman | Voina i mir:Lev Nikolaevich Tolstoy:1869:  Roman | Рисунок 24 |
| 10 | Добавление существующей книги | The Road:Cormac Mac Carthy:2006:Dystopian | The Road:Cormac Mac Carthy:2006:Dystopian | The Road:Cormac Mac Carthy:2006:Dystopian | Рисунок 25 |
| 11 | Добавление книги с некорректным названием | - | **the road**:Cormac Mac Carthy:2006:Dystopian | - | Рисунок 26 |
| 12 | Добавление книги с некорректным ФИО автора | - | The Road:**cormac mac carthy**:2006:Dystopian | - | Рисунок 27 |
| 13 | Добавление книги с некорректным ФИО автора | - | The Road:**CormacMacCarthy**:2006:Dystopian | - | Рисунок 27 |
| 14 | Добавление книги с некорректным годом | - | The Road:Cormac Mac Carthy:**200600**:Dystopian | - | Рисунок 28 |
| 15 | Добавление книги с некорректным годом | - | The Road:Cormac Mac Carthy: **AA2006**:Dystopian | - | Рисунок 28 |
| 16 | Добавление книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**dystopian** | - | Рисунок 29 |

*Продолжение таблицы 3*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Вектор записей с данными о книге (название, автор, год, жанр) | Запись о книге (название, автор, год, жанр) | Вектор записей с данными о книге (название, автор, год, жанр) | Сообщение |
| 17 | Добавление книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**Dystopian11** | - | Рисунок 29 |
| Поиск | | | | | |
| 18 | Запись не существует | - | The Road:Cormac Mac Carthy:2006:Dystopian | - | Рисунок 30 |
| 19 | Запись существует | The Road:Cormac Mac Carthy:2006:Dystopian | The Road:Cormac Mac Carthy:2006:Dystopian | The Road:Cormac Mac Carthy:2006:Dystopian | Рисунок 31 |
| 20 | Поиск книги с некорректным названием | - | **the road**:Cormac Mac Carthy:2006:Dystopian | - | Рисунок 26 |
| 21 | Поиск книги с некорректным ФИО автора | - | The Road:**cormac mac carthy**:2006:Dystopian | - | Рисунок 27 |
| 22 | Поиск книги с некорректным ФИО автора |  | The Road:**CormacMacCarthy**:2006:Dystopian | - | Рисунок 27 |
| 23 | Поиск книги с некорректным годом | - | Год1: 200060 Год2: 1900 | - | Рисунок 35 |
| 24 | Поиск книги с некорректным годом | - | Год1: 2000 Год2: 1900 | - | Рисунок 34 |
| 25 | Поиск книги с некорректным годом | - | Год1: 1899 Год2: a1900 | - | Рисунок 35 |
| 26 | Поиск книги с некорректным годом |  | Год1: AA1899 Год2: 1900 | - | Рисунок 35 |
| 27 | Поиск книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**dystopian** | - | Рисунок 29 |
| 28 | Поиск книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**Dystopian11** | - | Рисунок 29 |
| вв | | | | | |
| 29 | Удаление существующей книги | The Road:Cormac Mac Carthy:2006:Dystopian | The Road:Cormac Mac Carthy:2006:Dystopian | - | Рисунок 32 |
| 30 | Удаление несуществующей книги | The Road:Cormac Mac Carthy:2006:Dystopian | Voina i mir:Lev Nikolaevich Tolstoy:1869:  Roman | The Road:Cormac Mac Carthy:2006:Dystopian | Рисунок 33 |
| 31 | Добавление книги с некорректным названием | - | **the road**:Cormac Mac Carthy:2006:Dystopian | - | Рисунок 26 |

*Окончание таблицы 3*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | | |
| Вектор записей с данными о книге (название, автор, год, жанр) | Запись о книге (название, автор, год, жанр) | Вектор записей с данными о книге (название, автор, год, жанр) | | Сообщение |
| 32 | Удаление книги с некорректным ФИО автора | - | The Road:**cormac mac carthy**:2006:Dystopian | - | Рисунок 27 | |
| 33 | Удаление книги с некорректным ФИО автора | - | The Road:**CormacMacCarthy**:2006:Dystopian | - | Рисунок 27 | |
| 34 | Удаление книги с некорректным годом | - | The Road:Cormac Mac Carthy:**200600**:Dystopian | - | Рисунок 28 | |
| 35 | Удаление книги с некорректным годом | - | The Road:Cormac Mac Carthy: **AA2006**:Dystopian | - | Рисунок 28 | |
| 36 | Удаление книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**dystopian** | - | Рисунок 29 | |
| 37 | Удаление книги с некорректным жанром | - | The Road:Cormac Mac Carthy:2006:**Dystopian11** | - | Рисунок 29 | |

Таблица 4 – Тестирование красно-черного дерева

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| КЧ-дерево | Название или ФИО автора или год или жанр | КЧ-дерево | Сообщение |
| Добавление | | | | | |
| 1 | Добавление в пустое дерево | - | 1984:Eric Artur Bler:**1949**:Fantastika | Рисунок 36 | Рисунок 24 |
| 2 | Добавление в непустое дерево | Рисунок 36 | White nights:Fyodor Mikhailovich Dostoevskiy:**1848**:Povest | Рисунок 37 | Рисунок 24 |
| 3 | Добавление повторяющегося элемента | Рисунок 37 | Voina:Fyodor Mikhailovich Dostoevskiy:**1848**:Povest | Рисунок 38 | Рисунок 24 |
| 4 | Добавление повторяющегося элемента в корень | Рисунок 37 | Elka:Eric Artur Bler:**1949**:Fantastika | Рисунок 39 | Рисунок 24 |
| 5 | Добавление с левым поворотом | Рисунок 40 | Elka:Eric Artur Bler:**2000**:Fantastika | Рисунок 41 | Рисунок 24 |
| 6 | Добавление с правым поворотом | Рисунок 37 | White nights:Fyodor Mikhailovich Dostoevskiy:**1800**:Povest | Рисунок 42 | Рисунок 24 |
| 7 | Добавление с право-левым поворотом | Рисунок 40 | Elka:Eric Artur Bler:**1900**:Fantastika | Рисунок 43 | Рисунок 24 |

*Продолжение таблицы 4*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| КЧ-дерево | Название или ФИО автора или год или жанр | КЧ-дерево | Сообщение |
| 8 | Добавление с лево-правым поворотом | Рисунок 37 | Elka:Eric Artur Bler:**1900**:Fantastika | Рисунок 43 | Рисунок 24 |
| 9 | Добавление узла в левое поддерево.  Дядя красный. | Рисунок 37 | Elka:Eric Artur Bler:**1800**:Fantastika | Рисунок 44 | Рисунок 24 |
| 10 | Добавление узла в правое поддерево. Дядя чёрный, менялось направление вставки. | Рисунок 44 | OK:Eric Artur Bler:**1820**:Fantastika | Рисунок 45 | Рисунок 24 |
| 11 | Добавление узла в правое поддерево. Дядя чёрный, не менялось направление вставки. | Рисунок 46 | White nights:Fyodor Mikhailovich Dostoevskiy:**1000**:Povest | Рисунок 47 | Рисунок 24 |
| 12 | Добавление узла в правое поддерево.  Дядя красный. | Рисунок 48 | 1003:Eric Artur Bler:**1003**:Fantastika | Рисунок 49 | Рисунок 24 |
| 13 | Добавление узла в левое поддерево. Дядя чёрный, менялось направление вставки. | Рисунок 50 | 1005:Eric Artur Bler:**1005**:Fantastika | Рисунок 51 | Рисунок 24 |
| 14 | Добавление узла в левое поддерево. Дядя чёрный, не менялось направление вставки. | Рисунок 50 | 1008:Eric Artur Bler:**1008**:Fantastika | Рисунок 52 | Рисунок 24 |
| Удаление | | | | | |
| 15 | Удаление элемента из пустого дерева | - | 1984:Eric Artur Bler:**1949**:Fantastika | - | Рисунок 32 |
| 16 | Удаление элемента без потомков | Рисунок 53 | 1015 [19] | Рисунок 54 | Рисунок 33 |
| 17 | Удаление элемента с одним потомком | Рисунок 53 | 1014 [18] | Рисунок 55 | Рисунок 33 |
| 18 | Удаление элемента с двумя потомками | Рисунок 53 | 1013 [17] | Рисунок 56 | Рисунок 33 |
| 19 | Удаление повторяющегося элемента | Рисунок 56 | 1004 [7] | Рисунок 57 | Рисунок 33 |
| 20 | Удаление элемента, которого нет в дереве | Рисунок 53 | 1030 [20] | Рисунок 53 | Рисунок 32 |

*Окончание таблицы 4*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | | | | Входные данные | | Выходные данные | |
| КЧ-дерево | Название или ФИО автора или год или жанр | КЧ-дерево | Сообщение |
| 21 | Брат красный  (удаление слева) (1) | | Рисунок 58 | | | 1004 [6], 1004 [7], 1004 [8] | Рисунок 59 | Рисунок 33 |
| 22 | Брат красный (удаление справа) (1) | | Рисунок 60 | | | 1013 [1] | Рисунок 61 | Рисунок 33 |
| 23 | Брат черный. Оба потомка брата чёрные. (2) (Удаление слева) | | Рисунок 61 | | | 1008 [6] | Рисунок 62 | Рисунок 33 |
| 24 | Брат черный. Оба потомка брата чёрные. (2) (Удаление справа) | | Рисунок 62 | | | 1012 [2] | Рисунок 63 | Рисунок 33 |
| 25 | Брат чёрный. Внешний потомок брата чёрный. (3) (Удаление слева) | | Рисунок 64 | | | 1009 [5] | Рисунок 65 | Рисунок 33 |
| 26 | Брат чёрный. Внешний потомок брата чёрный. (3) (Удаление справа) | | Рисунок 63 | | | 1011 [3] | Рисунок 66 | Рисунок 33 |
| 27 | Брат чёрный. Внешний потомок брата – красный (4) (Удаление слева) | | Рисунок 68 | | | 1006 [8] | Рисунок 69 | Рисунок 33 |
| 28 | Брат чёрный. Внешний потомок брата – красный (4) (Удаление справа) | | Рисунок 65 | | | 1014 [0] | Рисунок 67 | Рисунок 33 |
| Поиск | | | | | | | | |
| 29 | | Поиск в пустом дереве | | - | | 1000 [0] | - | Рисунок 30 |
| 30 | | Поиск элемента в дереве, где присутствует заданный элемент | | Рисунок 60 | | 1014 [0] | Рисунок 60 | Рисунок 31 |
| 31 | | Поиск элемента в дереве, где отсутствует заданный элемент. | | Рисунок 60 | | 1015 [19] | Рисунок 60 | Рисунок 30 |

Таблица 5 – Тестирование хеш-таблицы (динамическая, статус 0/1, ХФ-свёртка по 2, линейное пробирование с k>1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные |
| Хеш-таблица (динамическая, статус 0/1, ХФ-свёртка по 2, линейное пробирование с k>1) | Книга (Название, ФИО автора, год, жанр) | Хеш-таблица (динамическая, статус 0/1, ХФ-свёртка по 2, линейное пробирование с k>1) |
| 1 | Добавление в пустую хеш-таблицу | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | 1 |  | 0 | | … |  | 0 | | 13 |  | 0 | | 14 |  | 0 | | Poor people:  Fyodor Mikhailovich Dostoevskiy:1844:Roman | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 5 |  | 0 | | 6 | Poor people…. | 1 | | … |  | 0 | | 14 |  | 0 | |
| 2 | Добавление в наполненную хеш-таблицу | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | 1 |  | 0 | | 2 |  | 0 | | 3 |  | 0 | | 4 |  | 0 | | 5 |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 |  | 0 | | 9 |  | 0 | | 10 |  | 0 | | 11 |  | 0 | | 12 |  | 0 | | 13 |  | 0 | | 14 |  | 0 | | Voina i mir:Lev Nikolaevich Tolstoy:1869:Roman | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | 1 |  | 0 | | 2 |  | 0 | | 3 |  | 0 | | 4 |  | 0 | | 5 |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 | Voina i mir….. | 1 | | 9 |  | 0 | | 10 |  | 0 | | 11 |  | 0 | | 12 |  | 0 | | 13 |  | 0 | | 14 |  | 0 | |
| 3 | Добавление повторяющегося элемента | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | 1 |  | 0 | | 2 |  | 0 | | 3 |  | 0 | | 4 |  | 0 | | 5 |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 |  | 0 | | 9 |  | 0 | | 10 |  | 0 | | 11 |  | 0 | | 12 |  | 0 | | 13 |  | 0 | | 14 |  | 0 | | Poor people:  Fyodor Mikhailovich Dostoevskiy:1844:Roman | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | 1 |  | 0 | | 2 |  | 0 | | 3 |  | 0 | | 4 |  | 0 | | 5 |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 |  | 0 | | 9 |  | 0 | | 10 |  | 0 | | 11 |  | 0 | | 12 |  | 0 | | 13 |  | 0 | | 14 |  | 0 | |

*Окончание таблицы 5*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | | Входные данные | | Выходные данные |
| Хеш-таблица (динамическая, статус 0/1, ХФ-свёртка по 2, линейное пробирование с k>1) | Книга (Название, ФИО автора, год, жанр) | Хеш-таблица (динамическая, статус 0/1, ХФ-свёртка по 2, линейное пробирование с k>1) |
| 4 | Увеличение ХТ | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 |  | 0 | | | The Catcher in the Rye:Jane Dow Salinger:1014:Drama  Pride and Prejudice:Hampshir Jane Austen:1013:Roman  To Kill a Mockingbird:Nelle Harper Lee:1012:Roman  The Great Gatsby:Frensis Scott Fitzgerald:1011:Drama  Poor people:Fyodor Mikhailovich Dostoevskiy:1010:Roman | |  |  |  | | --- | --- | --- | | 0 | The Catcher… | 1 | | 1 |  | 0 | | 2 | Poor people… | 1 | | 3 |  | 0 | | … |  | 0 | | 10 | The Great Gatsby | 1 | | … |  | 0 | | 14 | Poor people… | 1 | | 15 |  | 0 | | 16 | To Kill… | 1 | |
| 5 | Уменьшение ХТ | |  |  |  | | --- | --- | --- | | 0 | The Catcher… | 1 | | 1 |  | 0 | | 2 | Poor people… | 1 | | 3 |  | 0 | | … |  | 0 | | 10 | The Great Gatsby | 1 | | … |  | 0 | | 14 | Poor people… | 1 | | 15 |  | 0 | | 16 | To Kill… | 1 | | | The Catcher in the Rye:Jane Dow Salinger:1014:Drama  Pride and Prejudice:Hampshir Jane Austen:1013:Roman  To Kill a Mockingbird:Nelle Harper Lee:1012:Roman  The Great Gatsby:Frensis Scott Fitzgerald:1011:Drama  Poor people:Fyodor Mikhailovich Dostoevskiy:1010:Roman | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | 7 |  | 0 | | 8 |  | 0 | |
| 6 | Добавление с таким же значением первичного хеша | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | … |  | 0 | | 10 |  | 0 | | | The Great Gatsby:Frensis Scott Fitzgerald:1011:Drama | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | … |  | 0 | | 9 | The Great Gatsby | 0 | | 10 |  | 0 | |
| 7 | Удаление существующего элемента | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | … |  | 0 | | 10 |  | 0 | | | Poor people:  Fyodor Mikhailovich Dostoevskiy:1844:Roman | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 |  | 0 | | … |  | 0 | | 10 |  | 0 | |
| 8 | Удаление с коллизией | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | … |  | 0 | | 9 | The Great Gatsby | 0 | | 10 |  | 0 | | | The Great Gatsby:Frensis Scott Fitzgerald:1011:Drama | |  |  |  | | --- | --- | --- | | 0 |  | 0 | | … |  | 0 | | 6 | Poor people… | 1 | | … |  | 0 | | 10 |  | 0 | |

# Заключение

Целью курсового проекта было: разработать информационную систему для работы со справочниками предметной области «Электронная библиотека».

Цель достигнута. Для достижения поставленной цели были выполнены следующие задачи:

1. Проведен анализ предметной области «Электронная библиотека» и построена её модель;
2. Изучены теоретические основы методов построения справочников;
3. Определены требования к информационным системам;
4. Реализована информационная система и проведено её тестирование;
5. Был изучен язык C++ версии 20
6. Изучен модуль для разработки программного интерфейса WinForms

# Список литературы

1. Кормен Т. Х. Алгоритмы: построение и анализ. / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, Клиффорд Штайн – 2-е изд. – Москва : Издательский дом «Вильямс», 2005. – 1296 с.
2. Кнут Д. Э. Искусство программирования, Том 3. Сортировка и поиск – 2-е изд. – Москва : Издательский дом «Вильямс», 2007. – 832 с.
3. Вирт Н. Алгоритмы и структуры данных. / Никлаус Вирт – Москва : Издательство «Мир», 1989. – 360 с.
4. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. / Роберт Седжвик. – Киев: Издательство «ДиаСофт», 2001. – 688 с.
5. Красно-чёрное дерево / [Электронный ресурс] // Викиконспекты ИТМО: [сайт]. — URL: <https://neerc.ifmo.ru/wiki/index.php?title=Красно-черное_дерево> (дата обращения: 10.11.2023).

# Приложение

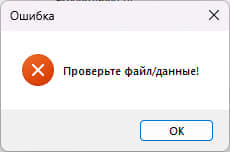


Рисунок 20

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 21

Изображение выглядит как текст, снимок экрана, Шрифт, мультимедиа

Автоматически созданное описание

Рисунок 22

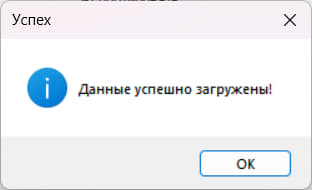


Рисунок 23

Изображение выглядит как текст, снимок экрана, мультимедиа, Шрифт

Автоматически созданное описание

Рисунок 24

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 25

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 26

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 27

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 28

Изображение выглядит как текст, Шрифт, программное обеспечение, снимок экрана

Автоматически созданное описание

Рисунок 29

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 30

Изображение выглядит как текст, снимок экрана, Шрифт, мультимедиа

Автоматически созданное описание

Рисунок 31

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 32

Изображение выглядит как текст, снимок экрана, мультимедиа, Шрифт

Автоматически созданное описание

Рисунок 33

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 34

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 35



Рисунок 36

Изображение выглядит как текст, Шрифт, белый, типография

Автоматически созданное описание

Рисунок 37



Рисунок 38



Рисунок 39

Изображение выглядит как текст, Шрифт, белый, типография

Автоматически созданное описание

Рисунок 40

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание  
Рисунок 41

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 42

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 43

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 44

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 45

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 46

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 47

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 48

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 49

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 50

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 51

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

Рисунок 52

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 53

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 54

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 55

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 56

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 57

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание  
Рисунок 58

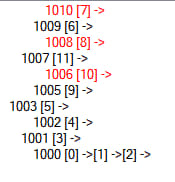


Рисунок 59

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 60

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание

Рисунок 61

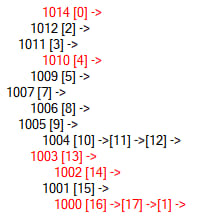


Рисунок 62

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание  
Рисунок 63

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 64

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание

Рисунок 65

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 66

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание

Рисунок 67

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание

Рисунок 68

Изображение выглядит как текст, Шрифт, дизайн

Автоматически созданное описание

Рисунок 69