

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

РЕШЕТНЕВ НИКИТА ЯРОСЛАВОВИЧ

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ АВТОМАТИЗАЦИИ РАБОТЫ СО СПРАВОЧНИКАМИ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОЕКТА «ТОРГОВАЯ ПЛОЩАДКА ИГРЫ»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Фундаментальные структуры данных и алгоритмы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 - Программная инженерия

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студент гр. Б9120-09.03.04прогин | | |
|  |  |  | | Н.Я. Решетнев |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | | | | | | |
| Защищен с оценкой | | | | | | |  |  | Руководитель | | |
|  | | | | | | |  |  | ст. преподаватель | | |
|  | | | |  |  | |  |  | ученая степень, должность |  | О.А. Крестникова |
| (подпись) | | | |  | (И.О. Фамилия) | |  |  | (подпись) |  | (И.О. Фамилия) |
| « |  | » |  | | | 2022 г. |  |  |  | | |
|  | | | | | | | | | | | |

г. Владивосток

2022

# Оглавление

[Оглавление 2](#_Toc107563053)

[Введение 3](#_Toc107563054)

[1 Анализ предметной области 4](#_Toc107563055)

[1.1 Объекты предметной области 4](#_Toc107563056)

[1.2 Законы ПО 6](#_Toc107563057)

[1.3 Постановка задачи «Формирование отчета «Магазин» 7](#_Toc107563058)

[2 Теоретическая часть 8](#_Toc107563059)

[2.1 Хеш-таблица 8](#_Toc107563060)

[2.1.1 Хеш-функция 9](#_Toc107563061)

[2.1.2 Разрешение коллизий методом открытой адресации(линейный) 12](#_Toc107563062)

[2.2 АВЛ дерево 14](#_Toc107563063)

[2.2.1 Элемент дерева – список 17](#_Toc107563064)

[3 Требования к информационной системе 18](#_Toc107563065)

[3.1 Требования к данным 18](#_Toc107563066)

[3.1.1 Требования к входным данным 18](#_Toc107563067)

[3.1.2 Требования к выходным данным 19](#_Toc107563068)

[3.2 Функциональные требования 20](#_Toc107563069)

[4 Реализация 22](#_Toc107563070)

[4.1 Спецификация структур данных 22](#_Toc107563071)

[4.2 Описание среды разработки 35](#_Toc107563072)

[4.3 Руководство пользователя 35](#_Toc107563073)

[4.4 Тестирование 44](#_Toc107563074)

[Заключение 47](#_Toc107563075)

[Список литературы 48](#_Toc107563076)

# Введение

В нынешнем мире каждая страна стремится к импортозамещению и своему производству. На данный момент есть много популярных торговых площадок разных игр, такие как Steam. Разработка и поддержание такого сервиса занимает огромное количество времени.

Целью курсового проекта является: разработка информационной системы для автоматизации работы со справочниками предметной области проекта «Торговая площадка игры».

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области «Торговая площадка игры».

2. Изучить теоретические основы методов построения справочников.

3. Определить требования к информационной системе.

4. Реализовать и провести тестирование.

# 1 Анализ предметной области

Требуется разработать информационную систему для предметной области (ПО) «Торговая площадка игры».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о продавце и о предмете;
2. позволять искать информацию продавца с заданным рейтингом и его продажи товара в заданной категории
3. формировать список записей продаж продавцов с заданным рейтингом и заданной категорией товаров

1.1 Объекты предметной области

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 «Продавцы» | Хранит информацию по каждому продавцу торговой площадки |
| Справочник «Продажи» | Хранит информацию о каждой продаже предмета |
| Отчет «Магазин» | Хранит информацию о предметах определенной категории и определенного рейтинга продавца  Для вывода хранится логин, количество продаж, рейтинг продавца из Справочника «Продавцы» и название предмета, стоимость и категория предмета из справочника «Продажи» |

Каждый продавец характеризуется следующими параметрами: Логин, количество продаж, рейтинг продавца.

**Логин** – является набором символов латинского алфавита, каждый из которых не может начинаться на цифру. Длина логина: от 5 до 25 символов включительно.

**Количество продаж** – целое число от 0 включительно.

**Рейтинг продавца** – десятичное число от 0 до 5 включительно с точностью в 2 знака после запятой

Пример справочника «Продавцы» представлен в таблице 2.

Таблица 2 – Справочник «Продавцы»

|  |  |  |
| --- | --- | --- |
| **Логин** | **Количество продаж** | **Рейтинг продавца** |
| J1233dav | 27124 | 4.25 |
| Kf23314 | 21 | 1.77 |
| dddeq | 300 | 4.12 |
| Vagner123 | 777 | 3.11 |
| krestnikova2022 | 1234 | 5.00 |
| leaderidnetop | 444 | 0.13 |
| dvfulutshiyvuz | 312 | 5.00 |
| hochu5pokursovoy | 290115 | 4.99 |
| kuplygara4 | 123321 | 3.71 |
| KolliziiNet2022 | 24 | 2.40 |

Справочник «Продажи» характеризуется следующими параметрами: Логин, название предмета, стоимость, категория.

**Логин** – является набором символов латинского алфавита, каждый из которых не может начинаться на число. Длина логина: от 5 до 25 символов включительно.

**Название предмета** – является набором символов. Начинается на заглавную букву. Может иметь пробелы. Длина названия: от 5 до 100 символов

**Стоимость предмета** - десятичное число от 0 не включительно с точностью в 2 знака после запятой

**Категория предмета** – является одним значением из набора категорий: {Красный, Оранжевый, Желтый, Зеленый, Голубой, Синий, Фиолетовый}

Пример справочника «Продажи» представлен в таблице 3.

Таблица 3 – Справочник «Продажи»

|  |  |  |  |
| --- | --- | --- | --- |
| **Логин** | **Название предмета** | **Стоимость предмета** | **Категория предмета** |
| J1233dav | Нож хороший | 2000.00 | Фиолетовый |
| Kf23314 | Green Knife | 1300.13 | Синий |
| dddeq | Летний автомат M4 | 20000.00 | Фиолетовый |
| Vagner123 | Синий пулемет A21 | 15.31 | Фиолетовый |
| krestnikova2022 | Killer Knife 123 | 13.41 | Фиолетовый |
| leaderidnetop | Leader 2022 | 11.52 | Зеленый |
| dvfulutshiyvuz | Кухонный нож | 14.13 | Фиолетовый |
| hochu5pokursovoy | Зимний автомат M4 | 13000.00 | Желтый |
| kuplygara4 | Красное солнце | 21.33 | Фиолетовый |
| KolliziiNet2022 | Левый и правый пистоль | 511.23 | Фиолетовый |

Пример отчета «Магазин» представлен в таблице 3

Таблица 4 – Отчет «Магазин»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Логин** | **Количество продаж** | **Рейтинг продавца** | **Название предмета** | **Стоимость предмета** | **Категория предмета** |
| J1233dav | 27124 | 4.25 | Нож хороший | 2000.00 | Фиолетовый |
| dddeq | 300 | 4.12 | Летний автомат M4 | 20000.00 | Фиолетовый |
| Vagner123 | 777 | 3.11 | Синий пулемет A21 | 15.31 | Фиолетовый |
| krestnikova2022 | 1234 | 5.00 | Killer Knife 123 | 13.41 | Фиолетовый |
| dvfulutshiyvuz | 312 | 5.00 | Кухонный нож | 14.13 | Фиолетовый |
| kuplygara4 | 123321 | 3.71 | Красное солнце | 21.33 | Фиолетовый |

1.2 Законы ПО

1. каждый логин в справочнике «Продавцы» уникален;
2. продавцы не обязательно должны иметь предметы на продаже;
3. не может быть предмет у несуществующего продавца;

1.3 Постановка задачи «Формирование отчета «Магазин»

Входные данные: Справочник «Продавцы», Справочник «Продажи», категория предмета, значение рейтинга от, значение рейтинга до.

Выходные данные: Отчет «Магазин».

Связь:

Значение рейтинга от <= Справочник «Продавцы».Рейтинг продавца & Справочник «Продавцы».Рейтинг продавца <= значение рейтинга до & Справочник «Продажи».категория\_предмета = категория предмета

# 2 Теоретическая часть

В рамках курсового проекта необходимо искать информацию о продавцах и их продажи. Для ускорения поиска используются такие структуры данных, например, как сбалансированные бинарные деревья поиска (АВЛ-деревья, красно-черные деревья, В-деревья и т.д.) и хеш-таблицы. АВЛ-дерево – это один из видов самобалансирующихся бинарных деревьев поиска, гарантирующий логарифмический показатель сложности для выполнения основных операция: добавление, удаление, поиск. Сбалансированность дерева достигается за счет введения дополнительного атрибута для узла дерева – «Высота». «Высота» принимает одно из трех значений – «-1», «0», «1».[2]

Хеш-таблицы часто применяются в базах данных а также в языковых процессорах типа компиляторов и ассемблеров, где они изящно обслуживают таблицы идентификаторов. В таких приложениях хеш-таблица – наилучшая структура данных

Для хранения данных из справочника «Продавцы» в курсовом проекте используется хеш-таблица, так как в данном справочнике имеются уникальные ключи. Так как количество записей не ограничено, то используется динамическая хеш-таблица.

2.1 Хеш-таблица

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией. Хеш-таблица содержит некоторый массив H, элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками).

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i=hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i]

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превысит 50% (если каждый элемент может равновероятно попасть в любую ячейку). Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

В некоторых специальных случаях удаётся избежать коллизий вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, использующие подобные хеш-функции, не нуждаются в механизме разрешения коллизий, и называются хеш-таблицами с прямой адресацией.

Число хранимых элементов, делённое на размер массива H (число возможных значений хеш-функции), называется коэффициентом заполнения хеш-таблицы (load factor) и является важным параметром, от которого зависит среднее время выполнения операций. [2]

2.1.1 Хеш-функция

Хеш-функция — это любая функция, которая может использоваться для сопоставления данных произвольного размера со значениями фиксированного размера. Значения, возвращаемые хеш-функцией, называются хеш-значениями, хеш-кодами, дайджестами или просто хешами. Эти значения обычно используются для индексации таблицы фиксированного размера, называемой хеш-таблицей. Использование хеш-функции для индексации хеш-таблицы называется хешированием или адресацией хранилища с разбросом.

Хеш-функции и связанные с ними хеш-таблицы используются в приложениях для хранения и извлечения данных для доступа к данным за небольшое и почти постоянное время на извлечение. Они требуют объема пространства для хранения, лишь немного превышающего общее пространство, необходимое для самих данных или записей. Хеширование — это эффективная с точки зрения вычислений и пространства хранения форма доступа к данным, которая позволяет избежать непостоянного времени доступа к упорядоченным и неупорядоченным спискам и структурированным деревьям, а также часто экспоненциальных требований к хранению при прямом доступе к пространствам состояний ключей большой или переменной длины.

Использование хеш-функций зависит от статистических свойств взаимодействия ключа и функции: поведение в наихудшем случае невыносимо плохое с исчезающе малой вероятностью, а поведение в среднем случае может быть почти оптимальным (минимальное столкновение).

Многочисленные тесты показали хорошую работу двух основных типов хеш-функций, один из которых основан на делении, а другой — на умножении.

Метод деления весьма прост; мы просто используем остаток от деления на М

h(К) = К mod М

В этом случае очевидно, что, например, при четном М значение h(К) будет четным при четном К и нечетным — при нечетном, что приведет к значительному смешению данных во многих файлах. Еще хуже обстоят дела, если М представляет собой степень основания счисления компьютера, поскольку при этом K mod M представляет собой несколько цифр числа К, расположенных справа, и не зависит от остальных цифр. Точно так же можно показать, что М не должно быть кратно трем, поскольку при буквенных ключах два из них, отличающиеся только перестановкой букв, могут давать числовые значения с разностью, кратной 3 (это происходит, поскольку 22n mod 3 = 1 и 10n mod 3 = 1). В целом, следует избегать значений М, делящих rk ± а, где k и а - небольшие числа, а r - "основание системы счисления" набора используемых алфавитно-цифровых символов (обычно r = 64, 256 или 100), так как остаток от деления по модулю на такие значения М зачастую представляет простую суперпозицию цифр ключа. Приведенные рассуждения приводят к мысли, что лучше всего использовать в качестве М простое число, такое, что rk ≠ а (по модулю М) при небольших k и а. В большинстве случаев подобный выбор вполне удовлетворителен [3]

Пример:

Входные данные:

J1233dav;27124;4,25

Kf23314;21;1,77

dddeq;300;4,12

Vagner123;777;3,11

krestnikova2022;1234;5,00

leaderidnetop;444;0,13

dvfulutshiyvuz;312;5,00

hochu5pokursovoy;290115;4,99

kuplygara4;123321;3,71

KolliziiNet2022;24;2,40

Размер таблицы = 20. Константа = 3;

1. Вставка 1 записи. Хеш функция = 10. Свободна. Вставляем.

2. Вставка 2 записи. Хеш функция = 10. Занята. Пересчитываем хеш-функцию. (10 + 1 \* 3) % 20 = 13. Свободна. Вставляем.

3. Вставка 3 записи. Хеш функция = 14. Свободна. Вставляем.

4. Вставка 4 записи. Хеш функция = 1. Свободна. Вставляем.

5. Вставка 5 записи. Хеш функция = 19. Свободна. Вставляем.

6. Вставка 6 записи. Хеш функция = 16. Свободна. Вставляем.

7. Вставка 7 записи. Хеш функция = 0. Свободна. Вставляем.

8. Вставка 8 записи. Хеш функция = 5. Свободна. Вставляем.

9. Вставка 9 записи. Хеш функция = 8. Свободна. Вставляем.

10. Вставка 10 записи. Хеш функция = 12. Свободна. Вставляем

При вставке записи 2 в хеш-таблицу хеш функция 2 записи была равна хеш-функции 1 записи. Данная ситуация называется коллизией. Для того, чтобы решить коллизию были придуманы методы разрешения коллизии(метод цепочек, метод открытой адресации и т.п.).

Для решения задачи коллизии был выбран метод линейного пробирования, так как как он не требует дополнительной памяти, в отличие от метода цепочек

2.1.2 Разрешение коллизий методом открытой адресации(линейный)

В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками.

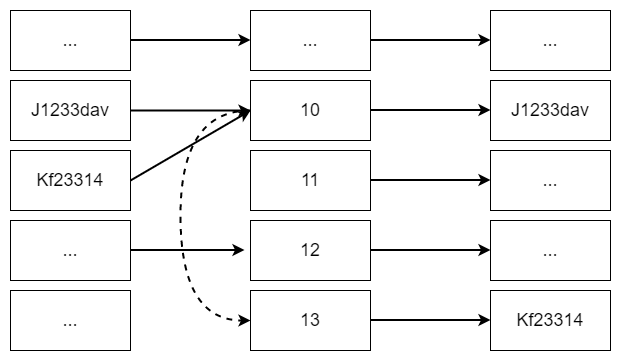
Последовательность, в которой просматриваются ячейки хеш-таблицы, называется последовательностью проб. В общем случае, она зависит только от ключа элемента, то есть это последовательность h0(x), h1(x), …, hn — 1(x), где x — ключ элемента, а hi(x) — произвольные функции, сопоставляющие каждому ключу ячейку в хеш-таблице. Первый элемент в последовательности, как правило, равен значению некоторой хеш-функции от ключа, а остальные считаются от него одним из приведённых ниже способов. Для успешной работы алгоритмов поиска последовательность проб должна быть такой, чтобы все ячейки хеш-таблицы оказались просмотренными ровно по одному разу.

Алгоритм поиска просматривает ячейки хеш-таблицы в том же самом порядке, что и при вставке, до тех пор, пока не найдется либо элемент с искомым ключом, либо свободная ячейка (что означает отсутствие элемента в хеш-таблице).

Удаление элементов в такой схеме несколько затруднено. Обычно поступают так: заводят булевый флаг для каждой ячейки, помечающий, удален элемент в ней или нет. Тогда удаление элемента состоит в установке этого флага для соответствующей ячейки хеш-таблицы, но при этом необходимо модифицировать процедуру поиска существующего элемента так, чтобы она считала удалённые ячейки занятыми, а процедуру добавления — чтобы она их считала свободными и сбрасывала значение флага при добавлении.

Линейное пробирование: ячейки хеш-таблицы последовательно просматриваются с некоторым фиксированным интервалом k между ячейками (обычно k = 1), то есть i-й элемент последовательности проб — это ячейка с номером (hash(x) + ik) mod N. Для того, чтобы все ячейки оказались просмотренными по одному разу, необходимо, чтобы k было взаимно-простым с размером хеш-таблицы.[2]

Для решения задачи коллизии был выбран метод линейного пробирования. Константа подбирается каждый раз, если она является делителем размерности таблицы. Ниже приведен пример хеш таблицы на данных, приведенных в главе 1.1 «Объекты предметной области» в таблице 2 - Справочник «Продавцы».

Рисунок 1 – Метод открытой адресации

2.2 АВЛ дерево

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ — аббревиатура, образованная первыми буквами создателей (советских учёных), Адельсон-Вельского Георгия Максимовича и Ландиса Евгения Михайловича.

Показатель сбалансированности в дальнейшем будем интерпретировать как разность между высотой левого и правого поддерева, а алгоритм будет основаться на типе TAVLTree, описанном выше. Непосредственно при вставке (листу) присваивается нулевой баланс. Процесс включения вершины состоит из трех частей (данный процесс описан Никлаусом Виртом в «Алгоритмы и структуры данных»):

Алгоритм добавления вершины:

Прохода по пути поиска, пока не убедимся, что ключа в дереве нет.

Включения новой вершины в дерево и определения результирующих показателей балансировки.

«Отступления» назад по пути поиска и проверки в каждой вершине показателя сбалансированности. Если необходимо — балансировка.

Будем возвращать в качестве результата функции, уменьшилась высота дерева или нет. Предположим, что процесс из левой ветви возвращается к родителю (рекурсия идет назад), тогда возможны три случая: {hl — высота левого поддерева, hr — высота правого поддерева} Включение вершины в левое поддерево приведет к

hl < hr: выравняется hl = hr. Ничего делать не нужно.

hl = hr: теперь левое поддерево будет больше на единицу, но балансировка пока не требуется.

hl > hr: теперь hl — hr = 2, — требуется балансировка.

В третьей ситуации требуется определить балансировку левого поддерева. Если левое поддерево этой вершины (Tree^.left^.left) выше правого (Tree^.left^.right), то требуется большое правое вращение, иначе хватит малого правого. Аналогичные (симметричные) рассуждения можно привести и для включение в правое поддерево.

Алгоритм удаления вершины:

Вершина — лист, удалим её и вызовем балансировку всех её предков в порядке от родителя к корню.

Докажем, что данный алгоритм сохраняет балансировку. Для этого докажем по индукции по высоте дерева, что после удаления некоторой вершины из дерева и последующей балансировки высота дерева уменьшается не более, чем на 1. База индукции: Для листа очевидно верно. Шаг индукции: Либо условие балансированности в корне (после удаления корень может изменится) не нарушилось, тогда высота данного дерева не изменилась, либо уменьшилось строго меньшее из поддеревьев => высота до балансировки не изменилась => после уменьшится не более чем на 1.

Очевидно, что в результате указанных действий процедура удаления вызывается не более 3 раз, так как у вершины, удаляемой по второму вызову, нет одного из поддеревьев. Но поиск ближайшего каждый раз требует O(N) операций. Становится очевидной возможность оптимизации: поиск ближайшей вершины может быть выполнен по краю поддерева, что сокращает сложность до O(log(N)).

Расстановка балансов при удалении:

Удаляемая вершина — лист, она удаляется, и обратный обход дерева происходит от родителя удалённого листа.

Если при переходе к родителю пришли слева — баланс увеличивается на 1, если же пришли справа — уменьшается на 1.

Это делается до тех пор, пока при изменении баланса он не станет равным −1 или 1: в данном случае такое изменение баланса будет гласить о неизменной дельта-высоте поддеревьев. Повороты происходят по тем же правилам, что и при вставке.[2]

Пример:

Входные данные:

Vagner123;Синий пулемет A21;15,31;Голубой

J1233dav;Нож хороший;2000;Красный

krestnikova2022;Killer Knife 123;13,41;Красный

leaderidnetop;Leader 2022;11,52;Зеленый

hochu5pokursovoy;Зимний автомат M4;13000;Желтый

Kf23314;Green Knife;1300,13;Синий

dvfulutshiyvuz;Кухонный нож;14,13;Синий

KolliziiNet2022;Левый и правый пистоль;511,23;Фиолетовый

KolliziiNet2022;Правый пистоль;51321;Фиолетовый

KolliziiNet2022;Левый пистоль;51321;Фиолетовый

KolliziiNet2022;Левый и правый пистоль;51321;Фиолетовый

kuplygara4;Красное солнце;21,33;Фиолетовый

dddeq;Летний автомат M4;20000;Оранжевый

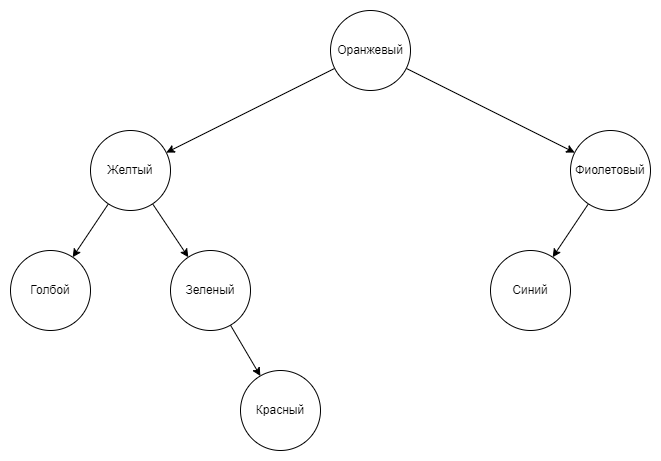


Рисунок 2 – Визуальное представление АВЛ-дерева

2.2.1 Элемент дерева – список

Линейные списки являются чрезвычайно гибкой структурой, так как их легко сделать большими или меньшими, и их элементы доступны для вставки или удаления в любой позиции списка. Списки также можно объединять или разбивать на меньшие списки.

Списки регулярно используются в приложениях, например, в программах информационного поиска, трансляторах программных языков или при моделировании различных процессов.

В математике список определяется как последовательность элементов определенного типа: X1, X2, … Xn , где n>=0. Количество элементов n называется длиной списка, X1 – первый элемент списка, Xn– последний элемент списка. В случае n=0, список пустой. Важное свойство списка заключается в том, что его элементы можно линейно упорядочить в соответствии с их позицией в списке, т.е., Xi предшествует Xi+1 и следует за Xi-1. Элемент Xi имеет позицию i.

Элементы списка могут быть разбросаны по памяти как угодно! Из-за этого теряется возможность быстро получить элемент по индексу, а также не представляется возможным быстро скопировать весь список, но получается довольно приятная штука — можно вставлять элементы за линейное время в любое место.[4]

# 3 Требования к информационной системе

3.1 Требования к данным

3.1.1 Требования к входным данным

Основываясь на анализе ПО, входными данными является:

* текстовый файл table1\_\*.txt, каждая строка файла содержит информацию об объекте «Продавцы», а именно: логин, количество продаж, рейтинг продавца, разделенные символом «;»

Пример текстового файла:

J1233dav;27124;4.25

Kf23314;21;1.77

dddeq;300;4.12

Vagner123;777;3.11

krestnikova2022;1234;5.00

leaderidnetop;444;0.13

dvfulutshiyvuz;312;5.00

hochu5pokursovoy;290115;4.99

kuplygara4;123321;3.71

KolliziiNet2022;24;2.40

* текстовый файл table2\_\*.txt, каждая строка файла содержит информацию об объекте «Продажи», а именно: логин, название предмета, стоимость предмета, категория предмета, разделенные символом «;»

Пример текстового файла:

J1233dav;Нож хороший;2000;Красный

Kf23314;Green Knife;1300,13;Синий

dddeq;Летний автомат M4;20000;Оранжевый

Vagner123;Синий пулемет A21;15,31;Голубой

krestnikova2022;Killer Knife 123;13,41;Красный

leaderidnetop;Leader 2022;11,52;Зелёный

dvfulutshiyvuz;Кухонный нож;14,13;Синий

hochu5pokursovoy;Зимний автомат M4;13000;Жёлтый

kuplygara4;Красное солнце;21,33;Фиолетовый

KolliziiNet2022;Левый и правый пистоль;511,23;Фиолетовый

* параметры, вводимые пользователем с клавиатуры:
* интервал рейтинга от
* интервал рейтинга до
* раскрывающийся список с выбором категории предмета
* поля для добавления со справочниками «Продавцы» и «Продажи»
* поля для удаления из справочников «Продавцы» и «Продажи»
* поля для ввода наименования файла для загрузки/ выгрузки справочников «Продавцы» и «Продажи»
* поле для ввода наименования файла для выгрузки отчета «Магазин»

3.1.2 Требования к выходным данным

Выходными данными являются:

* текстовый файл table1\_\*.txt, каждая строка файла содержит информацию об объекте «Продавцы», а именно: логин, количество продаж, рейтинг продавца, разделенные символом «;»
* текстовый файл table2\_\*.txt, каждая строка файла содержит информацию об объекте «Продажи», а именно: логин, название предмета, стоимость предмета, категория предмета, разделенные символом «;»
* текстовый файл report\_\*.txt, каждая строка файла содержит информацию из отчета «Магазин», а именно: логин, количество продаж, рейтинг продавца, название предмета, стоимость предмета, категория предмета, разделенные символом пробел.
* сообщения об ошибках:
* «Неправильно введённые данные»
* «Запись не найдена»
* «Загрузка прошла успешно»
* «Файл пуст»
* «Справочник сохранен»
* «Отчет сохранен»
* «Запись удалена»
* «Ошибка чтения файла»
* «С этим продавцом есть связанные записи»

3.2 Функциональные требования

Информационная система, должна позволять:

* Считывать справочники из файла(при считывании данных будем считать, что в файле все записи корректные)
* Сохранять справочники в файл
* Сохранять отчет в файл
* Выводить справочники на экран
* Строить отчет по заданным параметрам
* Добавлять/удалять записи из справочников
* Проверять данные на корректность:
* При добавлении записи в справочник «Продавцы» проверять логин на уникальность, корректность введенных данных
* При добавлении записи в справочник «Продажи» проверять нахождение логина в справочнике «Продавцы», корректность введенных данных
* При удалении записи из справочника «Продавцы» проверять, есть ли запись в этом справочнике и проверить на связанные записи в справочнике «Продажи»
* При удалении записи из справочника «Продажи» проверять, есть ли запись в этом справочнике
* При введении параметров с клавиатуры проверять, чтобы числа полей «рейтинг от» и «рейтинг до» находились в диапазоне от 0.01 до 5.00 и поле «рейтинг от» должно быть <= поля «рейтинг до»

# 4 Реализация

4.1 Спецификация структур данных

Класс **Avl** - класс, описывающий АВЛ-дерево.

Поля:

* List<Table2> Data – переменная, хранит данные в потомке;
* Node Left – указатель на левый потомок узла дерева;
* Node Right - указатель на правый потомок узла дерева;
* Node \_root – переменная, хранящая корень дерева.

Методы:

* void AvlToList(Node tree, List<Table2> mainList) – переводит все данные дерева в список;

Входные данные: экземпляр класса Node, экземпляр класса List;

Формальные параметры: экземпляр класса Node tree – узел, в котором на данный момент итерации рекурсии находится процедура, экземпляр класса List mainList – список, в который выгружаются элементы узла;

Выходные данные: экземпляр класса List.

* Add(Table2 data) – добавляет новый элемент в дерево. Если дерево пустое – добавляет в корень. Иначе – запускает RecursiveInsert;

Входные данные: экземпляр класса Node, элемент структуры Table2

Формальные параметры: элемент структуры Table2 data – элемент, который добавляют в дерево;

Выходные данные: экземпляр класса Node с добавленным элементом.

* Node RecursiveInsert(Node current, Node n) – рекурсивно ищет место вставки в дерево. При нахождении – вставляет элемент в список, находящийся в элементе дерева;

Входные данные: 2 экземпляра класса Node;

Формальные параметры: экземпляр класса Node current – узел, на котором сейчас выполняется процедура, экземпляр класса Node n – вспомогательный экземпляр;

Выходные данные: экземпляр класса Node со вставленным Node.

* Node balance\_tree(Node current) – балансирует дерево в заданном узле дерева;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node current – узел, в котором происходит балансирование;

Выходные данные: экземпляр класса Node с отбалансированными «весами».

* void Delete(Table2 target) – удаляет элемент из дерева. Если в узле остался 1 элемент – удаляет узел. Иначе – удаляет из списка, хранящегося в узле;

Входные данные: экземпляр класса Node, элемент структуры Table2;

Формальные параметры: элемент структуры Table2 target – элемент, который нужно удалить;

Выходные данные: экземпляр класса Node без элемента target.

* Node Delete(Node current, string target) – вспомогательная процедура для удаления элемента из дерева. Удаляет узел дерева, в котором хранится заданный ключ;

Входные данные: 2 экземпляра класса Node, переменная string;

Формальные параметры: экземпляр класса Node current – узел, в котором на данный момент проходит проверка, string target – ключ поиска;

Выходные данные: экземпляр класса Node с удалённым узлом.

* List<Table2> Find(string key) – возвращает список элементов, хранящийся в узле заданного ключа. Если узла по заданному ключу нет – то возвращает пустой список;

Входные данные: экземпляр класса Node, переменная string;

Формальные параметры: переменная string key – ключ поиска;

Выходные данные: список элементов структуры Table2.

* List<string> DisplayTree() – возвращает список с отформатированными строками. Одна строка – это один элемент в списке в узле дерева;

Входные данные: экземпляр класса Node;

Формальные параметры: отсутствуют;

Выходные данные: список, состоящий из элементов string.

* void InOrderDisplayTree(Node current, List<string> stringing) – запускает рекурсивно ЛКП обход в дереве;

Входные данные: экземпляр класса Node, список, состоящий из элементов string;

Формальные параметры: экземпляр класса Node current – узел, в котором в данный момент находится процедура, список, состоящий из элементов string stringing – список, в который выгружаются элементы узла, записанные строчкой;

Выходные данные: список, состоящий из элементов string – элементы дерева.

* Node Find(string target, Node current) – по заданному ключу рекурсивно ищет узел в дереве. Если узла по заданному ключу нет в дереве – то возвращает null;

Входные данные: переменная string, экземпляр класса Node;

Формальные параметры: переменная string target – ключ, по которому ищут узел, экземпляр класса Node – узел, в котором в данный момент находится процедура;

Выходные данные: экземпляр класса Node – найденный узел. Если узел не найден – null.

* int max(int l, int r) – возвращает максимальное между двумя числами

Входные данные: 2 переменные int;

Формальные параметры: переменная int l – число для сравнения, переменная int r – число для сравнения;

Выходные данные: l, если l > r, иначе r.

* int GetHeight(Node current) – возвращает максимальный коэффициент для узла;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node current - узел, в котором вычисляют максимальный коэффициент;

Выходные данные: максимальный коэффициент для узла.

* int balance\_factor(Node current) – возвращает коэффициент баланса в заданном узле;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node current – заданный узел, в котором ищется коэффициент;

Выходные данные: коэффициент баланса в заданном узле.

* Node RotateRR(Node parent) – выполняет правый поворот в заданном узле дерева;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node parent – узел, в котором происходит балансировка;

Выходные данные: - сбалансированный узел.

* Node RotateLL(Node parent) – выполняет левый поворот в заданном узле дерева;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node parent – узел, в котором происходит балансировка;

Выходные данные: - сбалансированный узел.

* Node RotateLR(Node parent) – выполняет лево-правый поворот в заданном узле дерева;

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node parent – узел, в котором происходит балансировка;

Выходные данные: - сбалансированный узел.

* Node RotateRL(Node parent) – выполняет право-левый поворот в заданном узле дерева.

Входные данные: экземпляр класса Node;

Формальные параметры: экземпляр класса Node parent – узел, в котором происходит балансировка;

Выходные данные: - сбалансированный узел.

Класс **Hash** - класс, описывающий хеш-таблицу

Поля:

* Int k – переменная, коэффициент для разрешения коллизии, который меняется при увеличении/уменьшении размера;
* Int Capacity – переменная, количество элементов, находящихся в таблице;
* Int \_sizeOfTable – переменная, текущая размерность таблицы;
* Int \_pop – переменная, количество попыток разрешения коллизии;
* Int mainSize – переменная, начальный размер таблицы;
* HashTableItem[] table – массив, хеш-таблица.

Методы:

* int Prime(int index) – возвращает наименьшее простое число к заданному.

Входные данные: переменная int index;

Формальные параметры: переменная int index – число, к которому ищется наименьшее простое число;

Выходные данные: целое число – простое число к заданному.

* Hash(int size) – конструктор, создаёт хеш-таблицу размерности size;

Входные данные: переменная int size;

Формальные параметры: переменная int size – размер хеш-таблицы;

Выходные данные: экземпляр класса Hash.

* int HashFunction(string pat) – функция, высчитывающая первичную хеш-функцию;

Входные данные: переменная string;

Формальные параметры: переменная string pat – ключ, для которого считается хеш-функция;

Выходные данные: число – хеш-функция заданного ключа.

Пример функции int HashFunction(string pat)

Предположим, что размер хеш-таблицы равен 20, а на вход поступает запись, содержащая информацию о продавце из модели предметной области (см п. 1.1), а именно:

J1233dav 27124 4,25

Hash(J1233dav) = (74+49+50+51+51+100+97+118+74+49+50+51+51+100+97+118) mod 20 = 10

Аналогично

Hash(Kf23314) = 10. Встретилась коллизия. Пересчёт с помощью функции int HashFunctuonTwo(10, 1) = 13

Hash(dddeq) = 14

Hash(Vagner123) = 1

Hash(krestnikova2022) = 19

Hash(leaderidnetop) = 16

Hash(dvfulutshiyvuz) = 0

Hash(hochu5pokursovoy) = 5

Hash(kuplygara4) = 8

Hash(KolliziiNet2022) = 12

* int HashFunctuonTwo(int hash1, int j) – функция, высчитывающая рекурсивно вторичную хеш-функцию;

Входные данные: переменные int, int;

Формальные параметры: переменная int hash1 – значение хеш-функции, на которой встретилась коллизия, переменная int j – номер попытки разрешения коллизии;

Выходные данные: число – вторичная хеш-функция заданного числа.

* HashTableItem[] Init(int size) – инициализация массива;

Входные данные: переменная int;

Формальные параметры: переменная int size – размер инициализированного массива;

Выходные данные: инициализированный массив элементов структуры HashTableItem.

* void Add(Table1 pat) – добавляет в хеш-таблицу экземпляр структуры Table1;

Входные данные: хеш-таблица, экземпляр структуры Table1;

Формальные параметры: экземпляр структуры Table1 pat – элемент для добавления в хеш-таблицу;

Выходные данные: хеш-таблица с добавленным элементом pat.

* void AddKolliz(Table1 pat, int pop, int hashKey) – рекурсивно добавляет в хеш-таблицу экземпляр структуры Table1, если при добавлении функцией add встретилась коллизия;

Входные данные: хеш-таблица, экземпляр структуры Table1, переменные int pop, int hashKey;

Формальные параметры: экземпляр структуры Table1 pat – элемент для добавления в хеш-таблицу, переменная int pop – номер попытки разрешения коллизии, переменная int hashKey – значение хеш-функции, на которой встретилась коллизия.;

Выходные данные: хеш-таблица с добавленным элементом pat.

* void Resize() – при достижении заполненности более 75% - увеличение размера таблицы в 2 раза;

Входные данные: хеш-таблица с количеством элементов > 75% от размерности хеш-таблицы;

Формальные параметры: отсутствуют;

Выходные данные: хеш-таблица с удвоенной размерностью хеш-таблицы.

* void changeCounts(string Login, int k) – подсчитывает количество связных записей во второй таблице;

Входные данные: переменные string Login, int k;

Формальные параметры: переменная string Login – ключ поиска для добавления/вычитания количества связных записей во второй таблице, переменная int k – позволяет различить, добавлять или вычитать в таблице;

Выходные данные: хеш-таблица с изменённым у элемента с ключом Login количеством связных записей.

* void Rehash() – при достижении заполненности менее 25% - уменьшение размера таблицы в 2 раза;

Входные данные: хеш-таблица с количеством элементов < 25% от размерности хеш-таблицы;

Формальные параметры: отсутствуют;

Выходные данные: хеш-таблица с уменьшенной в 2 раза размерностью хеш-таблицы.

* Table1 Searching(string key) – функция поиска элемента в хеш-таблице по заданному ключу. Если элемент не найден – возвращается null. Если найдено – возвращается экземпляр структуры Table1;

Входные данные: переменная string, хеш-таблица;

Формальные параметры: переменная string key – ключ поиска в хеш-таблице;

Выходные данные: экземпляр структуры Table1.

* int Search(string pat) – вспомогательная функция, которая возвращает индекс элемента, если он найден, и -1, если он не найден;

Входные данные: переменная string, хеш-таблица;

Формальные параметры: переменная string key – ключ поиска в хеш-таблице;

Выходные данные: число – индекс элемента с заданным ключом.

* void Delete(string pat) – удаляет элемент из хеш-таблицы по заданному ключу;

Входные данные: переменная string, хеш-таблица;

Формальные параметры: переменная string key – ключ поиска в хеш-таблице;

Выходные данные: хеш-таблица с удалённым из неё элементом с заданным ключом .

* int GetterSize() – возвращает количество элементов в хеш-таблице;

Входные данные: переменная int – количество элементов;

Формальные параметры: отсутствуют ;

Выходные данные: переменная int – количество элементов.

* Table1[] Getter() – возвращает массив значений хеш-таблицы;

Входные данные: хеш-таблица;

Формальные параметры: отсутствуют;

Выходные данные: массив экземпляров структур Table1 – значения из хеш-таблицы.

* HashTableItem[] GetterPublic() – возвращает массив элементов хеш-таблицы;

Входные данные: хеш-таблица;

Формальные параметры: отсутствуют;

Выходные данные: массив экземпляров структур HashTableItem – элементы хеш-таблицы.

Класс **List** - класс, описывающий список, созданный на массиве

Поля:

* T[] source – главный массив.

Методы:

* void Insert(int index, T x) – вставка элемента T по индексу index;

Входные данные: экземпляр класса List, переменные int, T;

Формальные параметры: переменная int index – индекс, куда нужно вставить элемент, T x – элемент, который вставляется в список;

Выходные данные: список с вставленным элементом.

* int Capacity() – возвращает количество элементов в списке;

Входные данные: экземпляр класса List;

Формальные параметры: отсутствуют;

Выходные данные: число – количество элементов в списке.

* List() – конструктор экземпляра класса;

Входные данные: отсутствуют;

Формальные параметры: отсутствуют;

Выходные данные: экземпляр класса List.

* string GetStringRepresentation() – представляет элементы списка в виде одной строки;

Входные данные: экземпляр класса List;

Формальные параметры: отсутствуют;

Выходные данные: строка, в которую записаны все элементы через запятую.

* List<T> CloneAs() – клонирование списка в другой список;

Входные данные: экземпляр класса List;

Формальные параметры: отсутствуют;

Выходные данные: экземпляр класса List – копия класса, в котором была вызвана функция.

* int IndexOf(T x) – возвращает индекс заданного элемента в списке. Если элемента нет – возвращает -1;

Входные данные: экземпляр класса List, переменная T;

Формальные параметры: переменная T x – элемент, который ищется в списке;

Выходные данные: число – индекс элемента в списке. -1, если элемента нет.

* void RemoveAt(int index) – удаляет элемент из списка по индексу;

Входные данные: экземпляр класса List, переменная int;

Формальные параметры: переменная int index – индекс элемента, который нужно удалить;

Выходные данные: экземпляр класса List с удалённым элементом индекса x.

* void AddListInList(List<T> target) – соединяет два списка между собой;

Входные данные: 2 экземпляра класса List;

Формальные параметры: экземпляра класса List<T> target – список, который добавляется в список, в котором вызвана функция;

Выходные данные: экземпляра класса List<T>, содержащий в себе target.

* T[] Getter() – возвращает массив элементов списка;

Входные данные: экземпляр класса List;

Формальные параметры: отсутствуют;

Выходные данные: массив T[] - массив элементов списка.

* void AddFirst(T x) – добавляет на первую позицию элемент;

Входные данные: экземпляр класса List, переменная T;

Формальные параметры: переменная T x – элемент, который добавляют в список;

Выходные данные: экземпляр класса List с добавленным элементом x в начале.

* void AddLast(T x) – добавляет на последнюю позицию элемент;

Входные данные: экземпляр класса List, переменная T;

Формальные параметры: переменная T x – элемент, который добавляют в список;

Выходные данные: экземпляр класса List с добавленным элементом x в конце.

* void Remove(T x) – удаляет элемент из списка;

Входные данные: экземпляр класса List, переменная T;

Формальные параметры: переменная T x – элемент, который удаляют из списка;

Выходные данные: экземпляр класса List с удалённым элементом x.

Структура **Table1** – структура, описывающая справочник «Продавцы»

Поля:

* string Login – логин продавца;
* double CountOfSell – количество продаж продавца;
* double Rating – рейтинг продавца.

Методы:

* static bool operator ==(Table1 obj1, Table1 obj2) – переопределённый оператор равенства двух элементов этой структуры;
* static bool operator !=(Table1 obj1, Table1 obj2) – переопределённый оператор неравенства двух элементов этой структуры.

Структура **Table2** - структура, описывающая справочник «Продажи»

Поля:

* string Login – логин продавца;
* string Naming – название предмета;
* string Category – категория предмета;
* double Price – цена предмета.

Методы:

* override string ToString() – функция, которая из элемента структуры делает строку;

Входные данные: экземпляр структуры Table2;

Формальные параметры: отсутствуют;

Выходные данные: строка – значение экземпляра структуры Table2.

* static bool operator ==(Table1 obj1, Table1 obj2) – переопределённый оператор равенства двух элементов этой структуры;
* static bool operator !=(Table1 obj1, Table1 obj2) – переопределённый оператор неравенства двух элементов этой структуры.

Структура **ReportStruct** - структура, описывающая отчёт «Магазин»

Поля:

* string Login - логин продавца;
* double CountOfSell – количество продаж продавца;
* double Rating – рейтинг продавца;
* string Naming – название предмета;
* string Category – категория предмета;
* double Price – цена предмета.

Методы:

* override string ToString() – функция, которая из элемента структуры делает строку;

Входные данные: экземпляр структуры ReportStruct;

Формальные параметры: отсутствуют;

Выходные данные: строка – значение экземпляра структуры ReportStruct.

Структура **HashTableItem** - структура, описывающая элемент хеш-таблицы

Поля:

* int Status – статус элемента;
* Table1 Value – значение хеш-таблицы;
* int CountOf2 – количество записей, связанных с элементом хеш-таблицы.

Методы:

* override string ToString() – функция, которая из элемента структуры делает строку;

Входные данные: экземпляр структуры HashTableItem;

Формальные параметры: отсутствуют;

Выходные данные: строка – значение экземпляра структуры HashTableItem.

4.2 Описание среды разработки

Для разработки десктопного приложения была выбрана среда разработки Rider от JetBrain, так как в составе Rider, в отличие от других IDE, есть внушительный набор для рефакторинга, проверки кода и контекстных действий для всех поддерживаемых им языков и технологий.

4.3 Руководство пользователя

При запуске программы встречает приветственное окно, изображенное на рисунке 3. Сверху находятся вкладки, из которых можно попасть в окна для работы со справочниками и для работы с отчетом

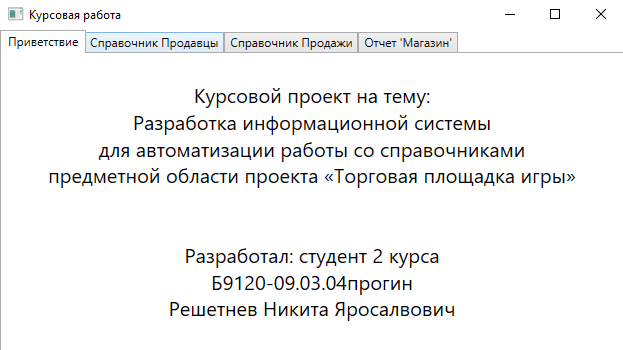


Рисунок 3 – «Окно «Приветствие»

Перейдя во вкладку «Справочник Продавцы», можно наблюдать функционал работы со справочником «Продавцы». На рисунке 4 изображено окно отображения справочника «Продавцы». При нажатии кнопки «Найти» на экран выведется весь справочник «Продавцы», который хранится в памяти программы.

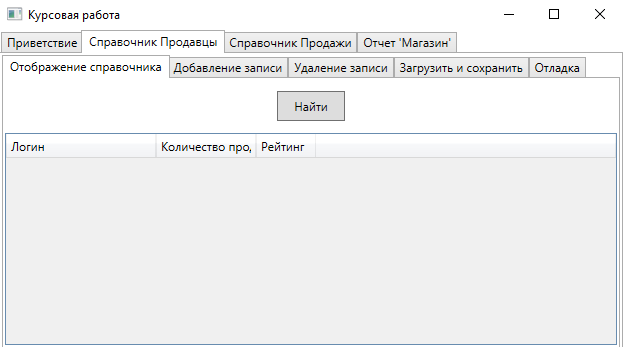


Рисунок 4 – «Окно «Отображение справочника» справочника «Продавцы»

При нажатии на вкладку «Добавление записи» открывается окно добавления записей в справочник «Продавцы». На рисунке 5 предоставлено окно добавления записи в справочник «Продавцы»

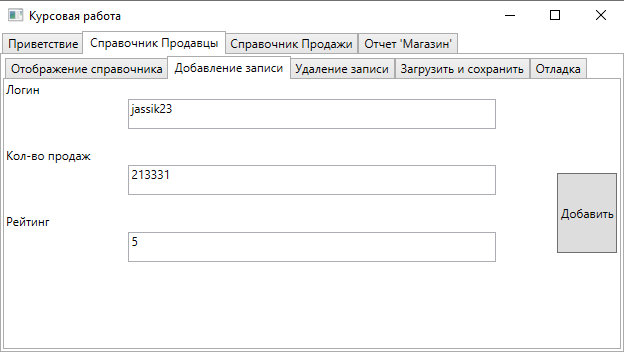


Рисунок 5 – «Окно «Добавление записи» справочника «Продавцы»

В текстовые поля «Логин», «Кол-во продаж», «Рейтинг» необходимо пользователю ввести соответствующие данные и затем нажать кнопку «Добавить». При некорректности данных всплывающее окно уведомит пользователя об этом.

При нажатии на вкладку «Удаление записи» открывается окно удаления записей из справочника «Продавцы». На рисунке 6 предоставлено окно удаления записи из справочника «Продавцы»

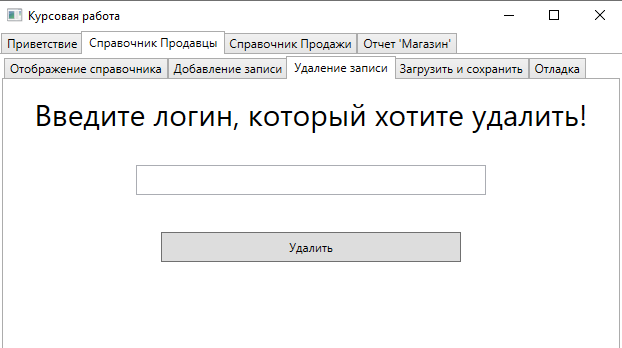


Рисунок 6 – «Окно «Удаление записи» справочника «Продавцы»

В текстовое поле «Введите логин, который хотите удалить!» необходимо ввести логин, который пользователь хочет удалить и нажать кнопку «Удалить». О некорректности данных, отсутствии логина в справочнике, наличии связанных записей в справочнике 2 пользователя уведомят соответствующие всплывающие окна.

При нажатии на вкладку «Загрузить и сохранить» открывается окно загрузки и сохранений записей из справочника «Продавцы». На рисунке 7 предоставлено окно загрузки и сохранений записей из справочника «Продавцы»

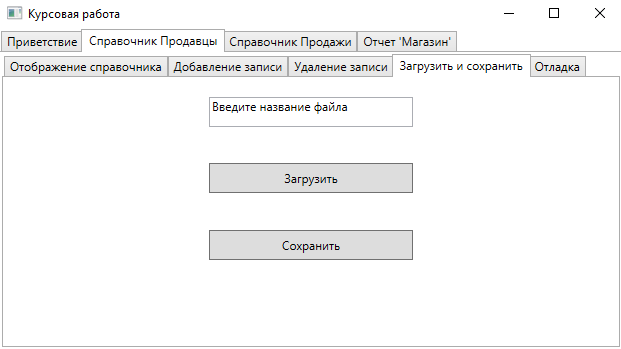


Рисунок 7 – «Окно «Загрузить и сохранить» справочника «Продавцы»

В текстовое поле «Введите название файла» пользователь должен ввести название файла, который хочет загрузить или в который хочет сохранить и нажать кнопку в зависимости от задачи (Загрузить или Сохранить). О некорректности названия файла пользователя уведомят соответствующее всплывающее окно.

При нажатии на вкладку «Отладка» открывается окно отладки хеш-таблицы. На рисунке 8 предоставлено окно отладки хеш-таблицы

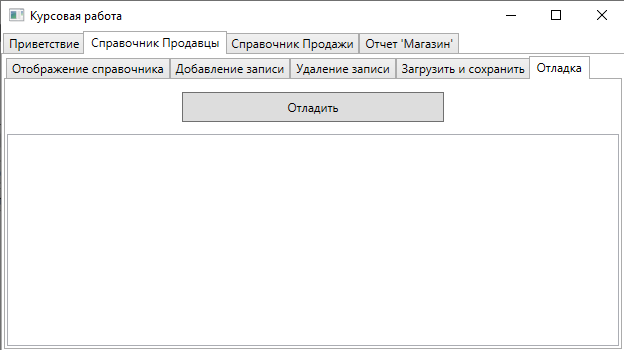


Рисунок 8 – «Окно «Отладка» справочника «Продавцы»

Нажав на кнопку «Отдалить» в окне отобразится хеш таблица со всеми пустыми полями, статусами и количеством связанных записей.

Перейдя во вкладку «Справочник Продажи», можно наблюдать функционал работы со справочником «Продажи». На рисунке 9 изображено окно отображения справочника «Продажи». При нажатии кнопки «Найти» на экран выведется весь справочник «Продажи», который хранится в памяти программы.

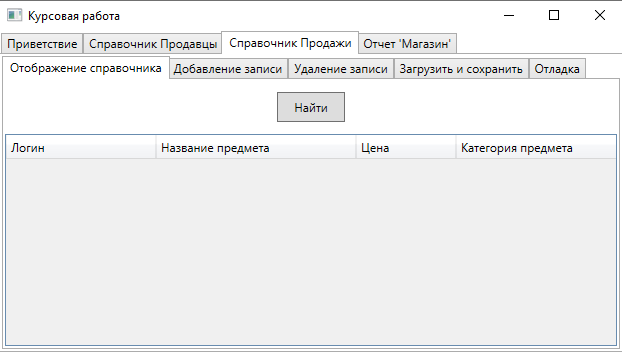


Рисунок 9 – «Окно «Отображение справочника» справочника «Продажи»

При нажатии на вкладку «Добавление записи» открывается окно добавления записей в справочник «Продажи». На рисунке 10 предоставлено окно добавления записи в справочник «Продажи»

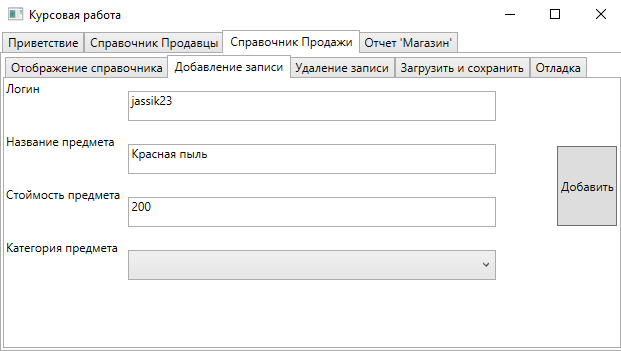


Рисунок 10 – «Окно «Добавление записи» справочника «Продажи»

В текстовые поля «Логин», «Название предмета», «Стоимость предмета» и в поле выбора «Категория предмета» необходимо пользователю ввести соответствующие данные и затем нажать кнопку «Добавить». При некорректности данных всплывающее окно уведомит пользователя об этом.

При нажатии на вкладку «Удаление записи» открывается окно удаления записей из справочника «Продажи». На рисунке 11 предоставлено окно удаления записи из справочника «Продажи»

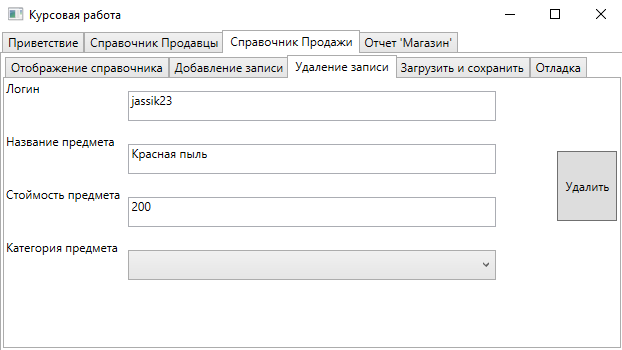


Рисунок 11 – «Окно «Удаление записи» справочника «Продажи»

В текстовые поля «Логин», «Название предмета», «Стоимость предмета» и в поле выбора «Категория предмета» необходимо пользователю ввести соответствующие данные и нажать кнопку «Удалить». О некорректности данных, отсутствии записи в справочнике пользователя уведомят соответствующие всплывающие окна.

При нажатии на вкладку «Загрузить и сохранить» открывается окно загрузки и сохранений записей из справочника «Продажи». На рисунке 12 предоставлено окно загрузки и сохранений записей из справочника «Продажи»

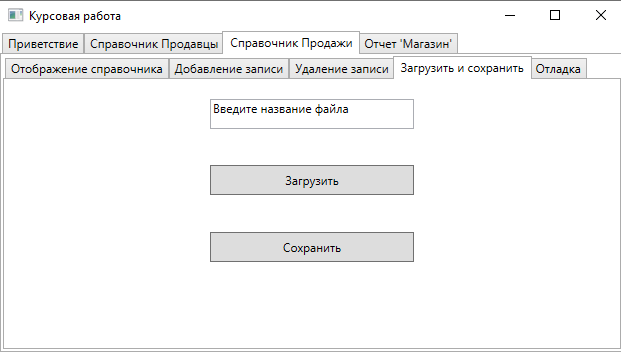


Рисунок 12 – «Окно «Загрузить и сохранить» справочника «Продажи»

В текстовое поле «Введите название файла» пользователь должен ввести название файла, который хочет загрузить или в который хочет сохранить и нажать кнопку в зависимости от задачи (Загрузить или Сохранить). О некорректности названия файла пользователя уведомят соответствующее всплывающее окно.

При нажатии на вкладку «Отладка» открывается окно отладки АВЛ дерева. На рисунке 13 предоставлено окно отладки АВЛ-дерева

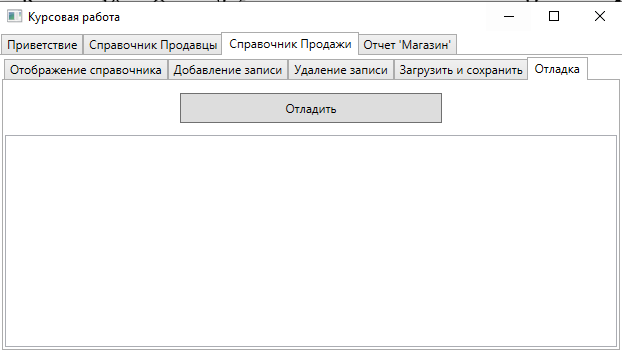


Рисунок 13 – «Окно «Отладка» справочника «Продажи»

Нажав на кнопку «Отдалить» в окне отобразится АВЛ-дерево, каждый узел которого заключен в квадратные скобки.

Перейдя во вкладку «Отчет Магазин», можно наблюдать функционал работы с отчетом «Магазин». На рисунке 14 изображено окно показа отчета «Магазин».

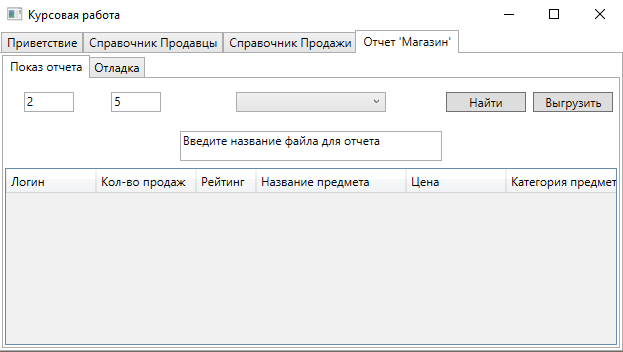


Рисунок 14 – «Окно «Показ отчета» отчета «Магазин»

В текстовые поля «Рейтинг от», «Рейтинг до» и в поле выбора «Категория предмета» необходимо пользователю ввести соответствующие данные и затем нажать кнопку «Найти», после чего построится отчет по заданным параметрам. О некорректности данных пользователя уведомят соответствующее всплывающее окно.

В текстовое поле «Введите название файла для отчета» пользователю необходимо ввести название файла, в который будет выгружен отчет. О некорректности данных пользователя уведомят соответствующее всплывающее окно.

При нажатии на вкладку «Отладка» открывается окно отладки структуры отчета «Магазин». На рисунке 15 предоставлено окно отладки структуры отчета «Магазин»

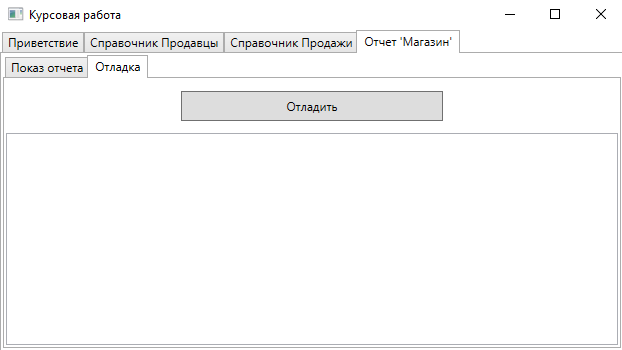


Рисунок 15 – «Окно «Отладка» отчета «Магазин»

При нажатии кнопка «Отладить» отображаются записи, хранящиеся в структуре отчета «Магазин».

4.4 Тестирование

Таблица 5 – Тестирование работы со справочником «Продавцы»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Хеш-таблица(метод открытой адресации) | Атрибуты(Логин, количество продаж,рейтинг) | Хеш-таблица(метод открытой адресации) | Атрибуты(Логин, количество продаж,рейтинг) |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | Jassik23, -1, 3 | - | «Неправильно введены данные» |
| - | Jassik23, 12, 7 | - | «Неправильно введены данные» |
| - | Jassik23, 12, -1 | - | «Неправильно введены данные» |
| - | Jassik23, 12, 0 | - | «Неправильно введены данные» |
| 2 | Добавление корректных данных | - | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | «Запись успешно сохранена» |
| 3 | Проверка уникальности ключа | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | «Такой логин уже есть в справочнике» |
| 4 | Добавление при коллизии | Jassik23, 12, 3.13 | Jassik32, 12, 3.13 | Jassik23, 12, 3.13 && Jassik32, 12, 3.13(Первичный хеш совпадает, вторичный различен) | «Запись успешно сохранена» |
| Поиск | | | | | |
| 5 | Запись не существует | - | Jassik23, 12, 3.13 | - | «Такой записи не существует» |
| 6 | Запись существует | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 |
| 7 | Поиск при коллизии | Jassik23, 12, 3.13 && Jassik32, 12, 3.13 | Jassik32, 12, 3.13 | Jassik23, 12, 3.13 && Jassik32, 12, 3.13 | Jassik32, 12, 3.13 |
| Удаление | | | | | |
| 8 | Запись не существует | - | Jassik23, 12, 3.13 | - | «Такой записи не существует» |
| 9 | Запись существует | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | - | «Запись успешно удалена» |
| 10 | Удаление при коллизии | Jassik23, 12, 3.13 && Jassik32, 12, 3.13 | Jassik32, 12, 3.13 | Jassik23, 12, 3.13 | «Запись успешно удалена» |
| 11 | Удаление при наличии связных записей в справочнике 2 | Jassik23, 12, 3.13(CountOf2 != 0) | Jassik23, 12, 3.13 | Jassik23, 12, 3.13 | «Есть связанная запись во втором справочнике. Удалите сначала её, прежде, чем удалять из этого справочника» |

Таблица 6 – Тестирование работы со справочником «Продавцы»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| АВЛ-дерево | Атрибуты(Логин, название предмета, стоимость предмета, категория предмета» | АВЛ-дерево | Атрибуты(Логин, название предмета, стоимость предмета, категория предмета» |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | Jassik23 Лось -1 Красный |  | «Неправильно введены данные» |
| - | Jassik23 Лось 0 Красный |  | Неправильно введены данные» |
| - | Jassik23 «Пусто» 13 Красный |  | Неправильно введены данные» |
| - | «Пусто» Лось 13 Красный |  | Неправильно введены данные» |
| 2 | Добавление корректных данных и существует связная запись | - | Jassik23 Лось 13 Красный | Jassik23 Лось 13 Красный | «Запись успешно сохранена» |
| 3 | Добавление корректных данных и не существует связная запись | - | Jassik23 Лось 13 Красный | - | «Невозможно добавить запись, так как нет связанного логина в 1 справочнике» |
| Поиск | | | | | |
| 5 | Запись не существует | - | Jassik23 Лось 13 Красный | - | «Запись не найдена» |
| 6 | Запись существует | Jassik23 Лось 13 Красный | Jassik23 Лось 13 Красный | - | Jassik23 Лось 13 Красный |
| Удаление | | | | | |
| 8 | Запись не существует | - | Jassik23 Лось 13 Красный | - | «Такой записи нет в справочнике 2» |
| 9 | Запись существует | Jassik23 Лось 13 Красный | Jassik23 Лось 13 Красный | - | «Запись успешно удалена» |

# Заключение

Целью курсового проекта было: разработка информационной системы для автоматизации работы со справочниками предметной области проекта «Торговая площадка игры».

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

1. Проведен анализ предметной области «Торговая площадка игры» и построена её модель;
2. Изучены теоретические основы методов построения справочников;
3. Определены требования к информационным системам;
4. Информационная система была реализована и спроектирована;
5. Был изучен язык C# версии 10
6. Изучена система для построения клиентских приложений Windows WPF
7. Изучен .NET Framework 4.7.2
8. Во время разработки JetBrains Rider была использована в качестве среды выполнения

# Список литературы

1. Седжвик Роберт. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. / Роберт Седжвик. – Киев : Издательство «ДиаСофт», 2001. – 688 с.
2. 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.
3. Кнут Д. Э. Искусство программирования, Том 3. Сортировка и поиск – 2-е изд. – Москва : Издательский дом «Вильямс», 2007. – 832 с.
4. Вирт Никлаус. Алгоритмы и структуры данных. / Никлаус Вирт – Москва : Издательство «Мир», 1989. – 360с