|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Хеширование и организация быстрого поиска данных»** | |
|  | |
| Выполнил студент группы ИНБО-02-20 | Лукьяненко Д.В. |
| Принял преподаватель | Сорокин А.В. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2021

**Содержание**

[Цель работы 3](#_Toc83678661)

[Ответы на вопросы 3](#_Toc83678662)

[Постановка задачи 4](#_Toc83678663)

[Подход к решению 5](#_Toc83678670)

[Алгоритмы операций на псевдокоде 5](#_Toc83678671)

[Код приложения 6](#_Toc83678672)

[Скриншот результатов выполнения операций с хеш-таблицей 12](#_Toc83678673)

[Скриншоты содержания файла и хеш-таблицы 15](#_Toc83678674)

[Вывод 16](#_Toc83678675)

# **Цель работы**

Получить навыки по разработке хеш-таблиц и их применении.

# **Ответы на вопросы**

1. Расскажите о назначении хеш-фунции.

Хэш-функции – это функции, предназначенные для «сжатия» произвольного сообщения или набора данных, записанных, как правило, в двоичном алфавите, в некоторую битовую комбинацию фиксированной длины, называемую сверткой.

1. Что такое коллизия?

Коллизия хеш-функции — это равенство значений хеш-функции на двух различных блоках данных.

1. Что такое «открытый адрес» по отношению к хеш-таблице?

Эта методика предполагает, что каждая ячейка таблицы помечена как незанятая. Поэтому при добавлении нового ключа всегда можно определить, занята ли данная ячейка таблицы или нет. Если да, алгоритм осуществляет перебор по кругу, пока не встретится «открытый адрес» (свободное место).

1. Как в хеш-таблице с открытым адресом реализуется коллизия?

Пары ключ-значение хранятся непосредственно в хеш-таблице. А алгоритм вставки проверяет ячейки в некотором порядке, пока не будет найдена пустая ячейка. Порядок вычисляется на лету

1. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?

При удалении требуемое свойство - отсутствия пустот - может нарушиться

1. Что определяет коэффициент нагрузки в хеш-таблице?

Определяет на сколько сложна хеш-таблица, а значит операции вставки, доступа и удаления будут занимать много времени. При превышении коэффициента нагрузки (по умолчанию 0,75) производится рехеширование таблицы.

1. Что такое «первичный кластер» в таблице с открытым адресом?

Первичным кластером является первичный ключ хеш-таблицы с открытым адресом

1. Как реализуется двойное хеширование?

Двойное хеширование использует идею применения второй хеш-функции к ключу при возникновении коллизии. Результатом второй хеш-функции будет количество позиций от точки столкновения для вставки.

# **Постановка задачи**

Разработайте приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла, структура записи которого приведена в варианте.

Дано.

Хеш-таблица для реализации коллизий по методу *С открытой адресацией (двойное хеширование)*.

Файл двоичный с записями фиксированной длины.

Структура записи файла *номер телефона – последовательность символов, Адрес*

Результат.

Приложение, выполняющее операции

Управление хеш-таблицей: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу

Управление файлом:

1) посредством хеш-таблицы: считать запись из файла при поиске записи

2) добавить запись в файл, удалить запись из файла, прочитать запись файла по заданному номеру записи.

Вариант №3. Условие задания:

|  |  |  |
| --- | --- | --- |
| № | Метод хеширования (способ реализации коллизий) | Структура записи файла (ключ – подчеркнутое поле) |
| 3 | С открытой адресацией (двойное хеширование) | Владелец телефона: номер телефона – последовательность символов, Адрес |

# **Подход к решению**

Сразу отмечу, что в процессе решения данного практического задания мною был использован язык программирования Java. Это связано с тем, что я собираюсь пользоваться этим языком в будущем и тему хеш-таблиц мне показалось важным узнать именно на нем.

1. Хеш-таблица - класс
2. Структура элемента цепной хеш-таблицы: *массив элементов*
3. Структура элемента линейного однонаправленного списка: *номер телефона, адрес (в моем случае город)*
4. Методы хеш-таблицы: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу, вывод хеш-таблицы в консоль.
5. Файл двоичный содержащий хеш-таблицу.

Структура записи файла:

* Номер телефона – строка типа String
* Адрес - строка типа String

Операции по управлению файлом:

- добавить запись в файл: занести запись в файл, вставить сведения о записи в хеш-таблицу

- удалить запись из файла: удалить из хеш-таблицы и из файла

# **Алгоритмы операций на псевдокоде**

Алгоритм вставки в таблицу

|  |
| --- |
| FUNCTION insert(String telefonNumber, String adress):  IF size >= hashSize: rehash();  INT hashing1 ← hasingOne(telefonNumber);  INT hashing2 ← hashingTwo(telefonNumber);  WHILE table[hashing1] != null:  hashing1 = hashing2 + hashing1;  hashing1 = hashing1 % hashSize;  table[hashing1] ← new value(telefonNumber, adress);  size = size + 1;  CALL save(); |

Поиск записи по ключу в таблице и возвращение ссылки на запись в файле

|  |
| --- |
| FUNCTION getTelefoNumber(String telefonNumber):  INT hash1 ← hasingOne(telefonNumber);  INT hash2 ← hashingTwo(telefonNumber);  WHILE table[hash1] ≠ null  AND !table[hash1].telefonNumber.equals(telefonNumber):  hash1 = hash2 + hash1;  hash1 = hash1 % hashSize;  RETURN table[hash1]; |

Удаление элемента из хеш-таблицы

|  |
| --- |
| FUNCTION removeFunc(String telefonNumber):  INT hash1 ← hasingOne(telefonNumber);  INT hash2 ← hashingTwo(telefonNumber);  WHILE table[hash1] ≠ null  AND !table[hash1].telefonNumber.equals(telefonNumber):  hash1 = hash2 + hash1;  hash1 = hash1 % hashSize;  table[hash1] ← null;  size = size - 1;  CALL save(); |

# **Код приложения**

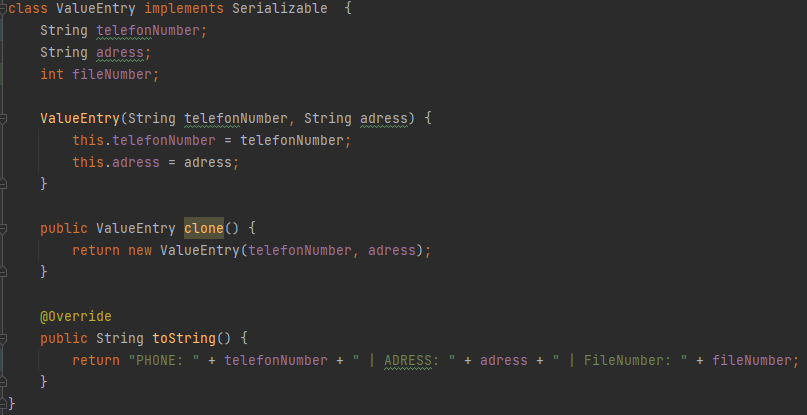


Рисунок 1 – Класс создания переменных

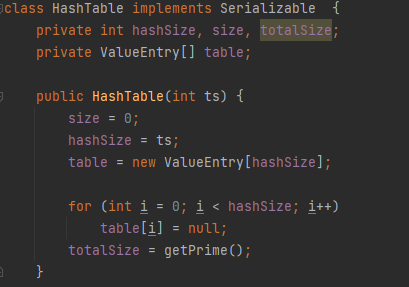


Рисунок 2 – Конструктор хеш-таблицы

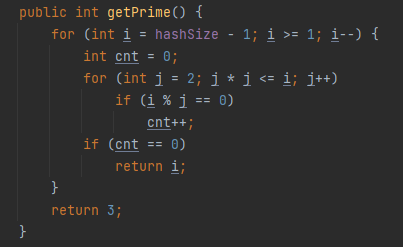


Рисунок 3 – Метод возращения простого числа

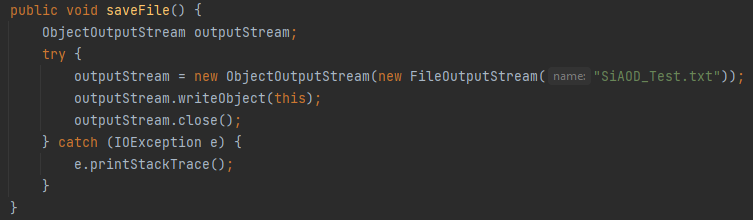


Рисунок 4 – Метод сохранения файла

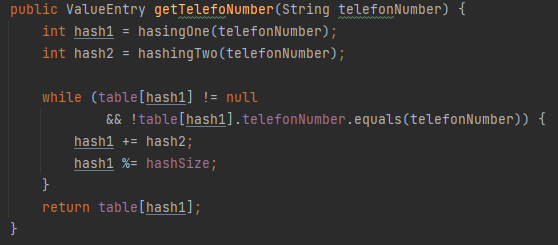


Рисунок 5 – Метод получения номера телефона (ключа)

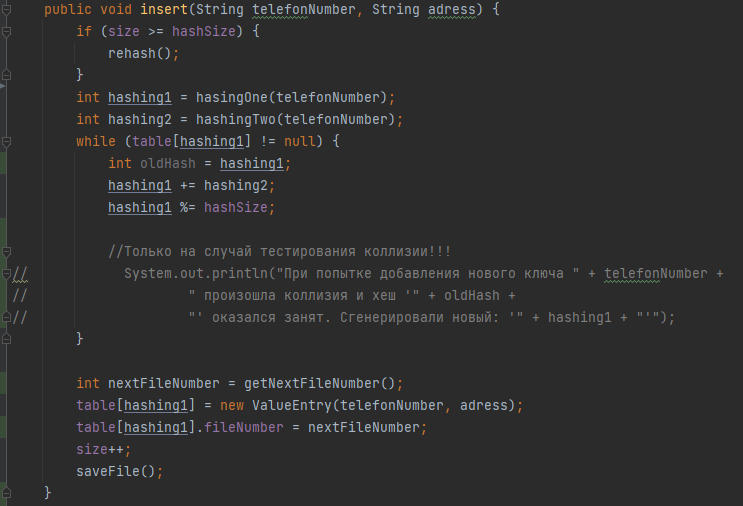


Рисунок 6 – Метод вставки элемента в хеш-таблицу

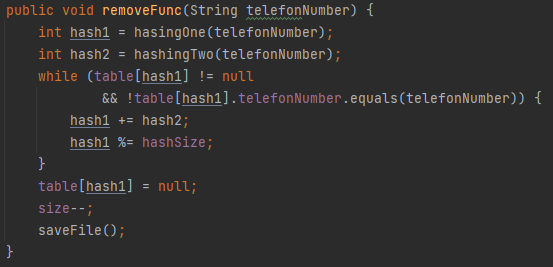


Рисунок 7 – Метод удаления элемента из хеш-таблицы

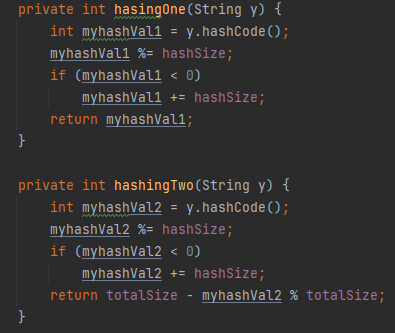


Рисунок 8 – Хеш-функции

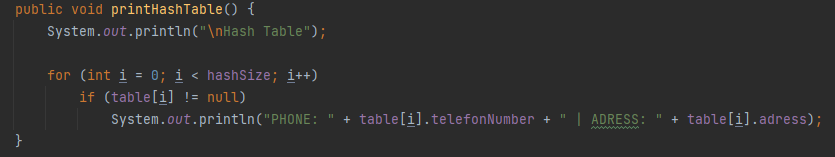


Рисунок 9 – Вывод всей хеш-таблицы

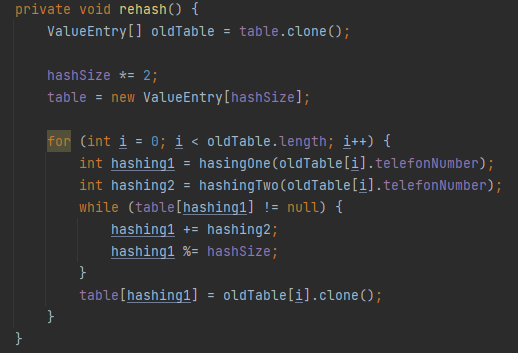


Рисунок 10 – Метод рехеширования

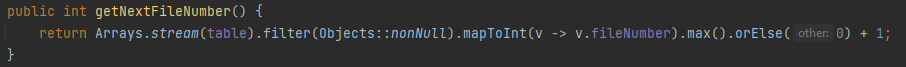


Рисунок 11 – Метод получения следующего номера файла

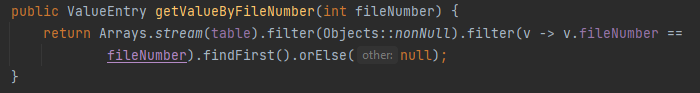


Рисунок 12 – Метод получения значения по номеру в файле

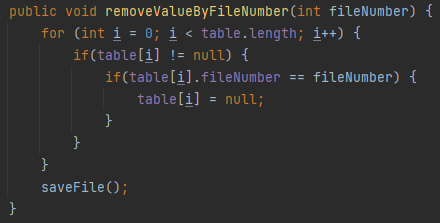


Рисунок 13 – Метод удаления значения по номеру в файле



Рисунок 14 – Создание таблицы

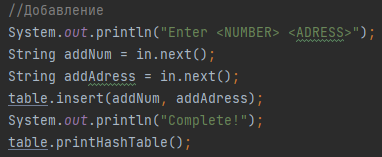


Рисунок 15 – Добавление нового элемента в таблицу

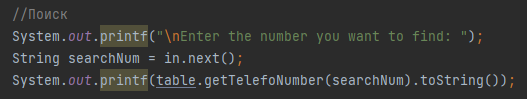


Рисунок 16 – Поиск элемента в таблице

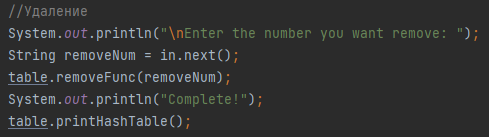


Рисунок 17 – Удаление элемента из таблицы

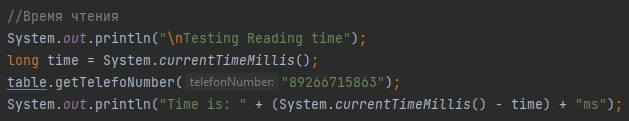


Рисунок 18 – Время поиска элемента в таблице

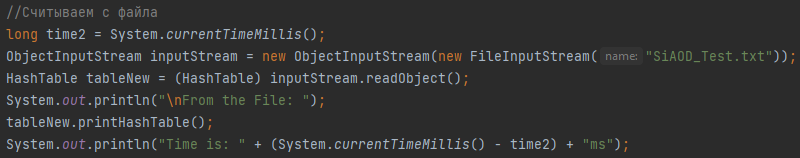


Рисунок 19 – Считывание с файла (с временем чтения)

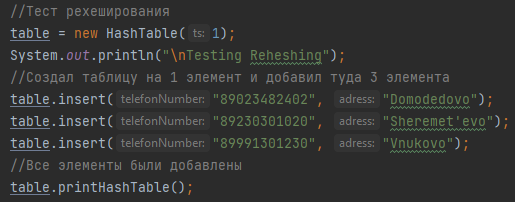


Рисунок 20 – Тест рехеширования

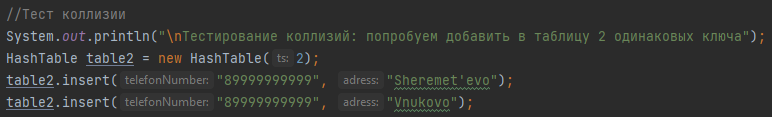


Рисунок 21 – Тест коллизии

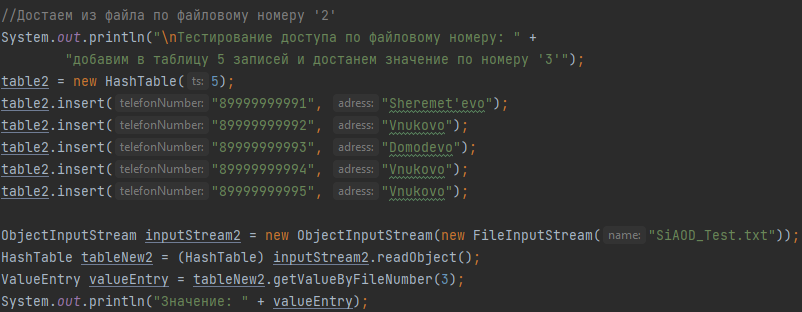


Рисунок 22 – Тест доставания элемента по номеру в файле

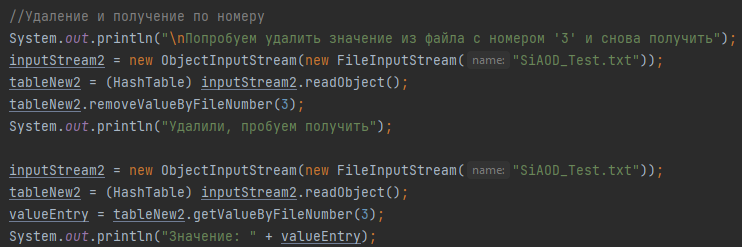


Рисунок 23 – Тест удаления и получения по номеру в файле

# **Скриншот результатов выполнения операций с хеш-таблицей**

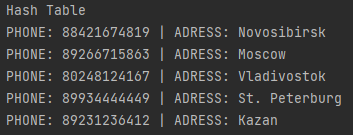


Рисунок 24 – Результат заполнения таблицы

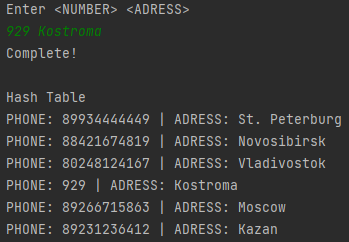


Рисунок 25 – Результат вставления нового номера



Рисунок 26 – Результат поиска по номеру

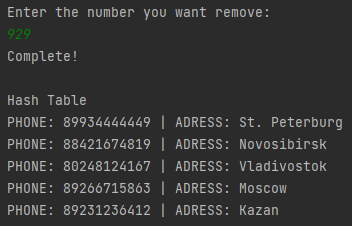


Рисунок 27 – Результат удаления по номеру



Рисунок 28 – Результат считывания времени поиска элемента

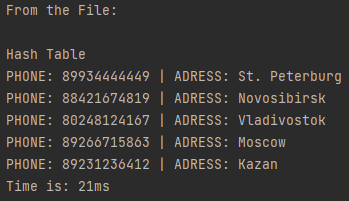


Рисунок 29 – Результат считывания с файла и вывода, с временем считывания

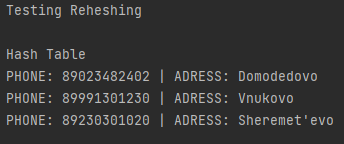


Рисунок 30 – Результат рехеширования (добавили 3 элемента в таблицу с 1 ячейкой изначально)



Рисунок 31 – Результат тестирования коллизии

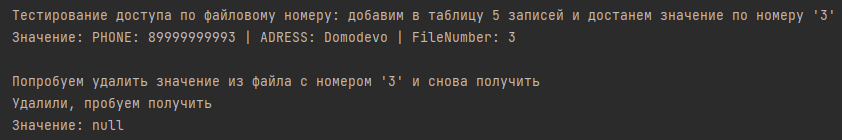


Рисунок 32 – Результат тестирования удаления элемента по номеру в файле

# **Скриншоты содержания файла и хеш-таблицы**

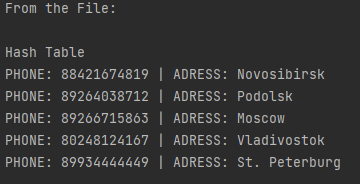


Рисунок 15 – Полученная хеш-таблица

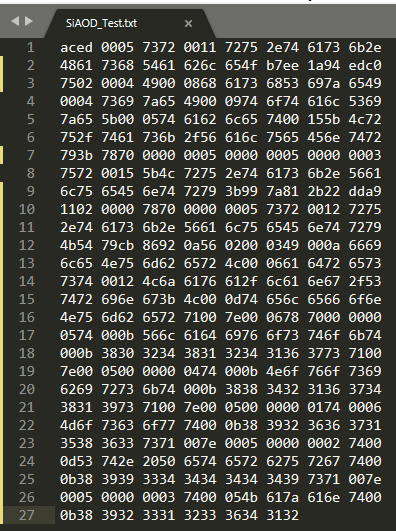


Рисунок 16 – Полученная хеш-таблица в файле

Считаю важным объяснить, что такая запись файла получилась из-за особенности языка, я не стал редактировать это в программе, ведь она и так все отлично читает с него.

# **Вывод**

В результате выполнения работы я приобрел навыки по разработке хеш-таблиц и научился их применять в языке программирования Java.