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
Ответы на вопросы 3
Постановка задачи 4
Подход к решению 5
Алгоритмы операций на псевдокоде 5
Код приложения 6
Скриншот результатов выполнения операций с хеш-таблицей 12
Скриншоты содержания файла и хеш-таблицы 15
Вывод 16

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.
2) Что такое коллизия?
   Коллизия хеш-функции — это равенство значений хеш-функции на двух различных блоках данных.
3) 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).
4) 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
5) 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
6) 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.
7) Что такое «первичный кластер» в таблице с открытым адресом?
   Первичным кластером является первичный ключ хеш-таблицы с открытым адресом
8) 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

```java
class ValueEntry implements Serializable  {
    String telefonNumber;
    String adress;
    int fileNumber;

    ValueEntry(String telefonNumber, String adress) {
        this.telefonNumber = telefonNumber;
        this.adress = adress;
    }

    public ValueEntry clone() {
        return new ValueEntry(telefonNumber, adress);
    }

    @Override
    public String toString() {
        return "PHONE: " + telefonNumber + " | ADRESS: " + adress + " | FileNumber: " + fileNumber;
    }
}
```

Рисунок 1 – Класс создания переменных

```
class HashTable implements Serializable  {
    private int hashSize, size, totalSize;
    private ValueEntry[] table;

    public HashTable(int ts) {
        size = 0;
        hashSize = ts;
        table = new ValueEntry[hashSize];

        for (int i = 0; i < hashSize; i++)
            table[i] = null;
        totalSize = getPrime();
    }
```

Рисунок 2 – Конструктор хеш-таблицы

```
public int getPrime() {
    for (int i = hashSize - 1; i >= 1; i--) {
        int cnt = 0;
        for (int j = 2; j * j <= i; j++)
            if (i % j == 0)
                cnt++;
        if (cnt == 0)
            return i;
    }
    return 3;
}
```

Рисунок 3 – Метод возращения простого числа

```
public void saveFile() {
    ObjectOutputStream outputStream;
    try {
        outputStream = new ObjectOutputStream(new FileOutputStream( name: "SiAOD_Test.txt"));
        outputStream.writeObject(this);
        outputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Рисунок 4 – Метод сохранения файла

```java
public ValueEntry getTelefoNumber(String telefonNumber) {
    int hash1 = hasingOne(telefonNumber);
    int hash2 = hashingTwo(telefonNumber);

    while (table[hash1] != null
            && !table[hash1].telefonNumber.equals(telefonNumber)) {
        hash1 += hash2;
        hash1 %= hashSize;
    }
    return table[hash1];
}
```

Рисунок 5 – Метод получения номера телефона (ключа)

```java
    public void insert(String telefonNumber, String adress) {
        if (size >= hashSize) {
            rehash();
        }
        int hashing1 = hasingOne(telefonNumber);
        int hashing2 = hashingTwo(telefonNumber);
        while (table[hashing1] != null) {
            int oldHash = hashing1;
            hashing1 += hashing2;
            hashing1 %= hashSize;

            //Только на случай тестирования коллизии!!!
//            System.out.println("При попытке добавления нового ключа " + telefonNumber +
//                    " произошла коллизия и хеш '" + oldHash +
//                    "' оказался занят. Сгенерировали новый: '" + hashing1 + "'");
        }

        int nextFileNumber = getNextFileNumber();
        table[hashing1] = new ValueEntry(telefonNumber, adress);
        table[hashing1].fileNumber = nextFileNumber;
        size++;
        saveFile();
    }
```

Рисунок 6 – Метод вставки элемента в хеш-таблицу

```java
public void removeFunc(String telefonNumber) {
    int hash1 = hasingOne(telefonNumber);
    int hash2 = hashingTwo(telefonNumber);
    while (table[hash1] != null
            && !table[hash1].telefonNumber.equals(telefonNumber)) {
        hash1 += hash2;
        hash1 %= hashSize;
    }
    table[hash1] = null;
    size--;
    saveFile();
}
```

Рисунок 7 – Метод удаления элемента из хеш-таблицы

```java
private int hasingOne(String y) {
    int myhashVal1 = y.hashCode();
    myhashVal1 %= hashSize;
    if (myhashVal1 < 0)
        myhashVal1 += hashSize;
    return myhashVal1;
}

private int hashingTwo(String y) {
    int myhashVal2 = y.hashCode();
    myhashVal2 %= hashSize;
    if (myhashVal2 < 0)
        myhashVal2 += hashSize;
    return totalSize - myhashVal2 % totalSize;
}
```

Рисунок 8 – Хеш-функции

```java
public void printHashTable() {
    System.out.println("\nHash Table");

    for (int i = 0; i < hashSize; i++)
        if (table[i] != null)
            System.out.println("PHONE: " + table[i].telefonNumber + " | ADRESS: " + table[i].adress);
}
```

Рисунок 9 – Вывод всей хеш-таблицы

```java
private void rehash() {
    ValueEntry[] oldTable = table.clone();

    hashSize *= 2;
    table = new ValueEntry[hashSize];

    for (int i = 0; i < oldTable.length; i++) {
        int hashing1 = hasingOne(oldTable[i].telefonNumber);
        int hashing2 = hashingTwo(oldTable[i].telefonNumber);
        while (table[hashing1] != null) {
            hashing1 += hashing2;
            hashing1 %= hashSize;
        }
        table[hashing1] = oldTable[i].clone();
    }
}
```

Рисунок 10 – Метод рехеширования

```
public int getNextFileNumber() {
    return Arrays.stream(table).filter(Objects::nonNull).mapToInt(v -> v.fileNumber).max().orElse( other: 0) + 1;
}
```

Рисунок 11 – Метод получения следующего номера файла

```
public ValueEntry getValueByFileNumber(int fileNumber) {
    return Arrays.stream(table).filter(Objects::nonNull).filter(v -> v.fileNumber ==
            fileNumber).findFirst().orElse( other: null);
}
```

Рисунок 12 – Метод получения значения по номеру в файле

```
public void removeValueByFileNumber(int fileNumber) {
    for (int i = 0; i < table.length; i++) {
        if(table[i] != null) {
            if(table[i].fileNumber == fileNumber) {
                table[i] = null;
            }
        }
    }
    saveFile();
}
```

Рисунок 13 – Метод удаления значения по номеру в файле

```
//Создание таблицы
HashTable table = new HashTable( ts: 5);
table.insert( telefonNumber: "89266715863",  adress: "Moscow");
table.insert( telefonNumber: "89934444449",  adress: "St. Peterburg");
table.insert( telefonNumber: "89231236412",  adress: "Kazan");
table.insert( telefonNumber: "88421674819",  adress: "Novosibirsk");
table.insert( telefonNumber: "80248124167",  adress: "Vladivostok");
table.printHashTable();
```

Рисунок 14 – Создание таблицы

```
//Добавление
System.out.println("Enter <NUMBER> <ADRESS>");
String addNum = in.next();
String addAdress = in.next();
table.insert(addNum, addAdress);
System.out.println("Complete!");
table.printHashTable();
```

Рисунок 15 – Добавление нового элемента в таблицу

```
//Поиск
System.out.printf("\nEnter the number you want to find: ");
String searchNum = in.next();
System.out.printf(table.getTelefoNumber(searchNum).toString());
```

Рисунок 16 – Поиск элемента в таблице

```
//Удаление
System.out.println("\nEnter the number you want remove: ");
String removeNum = in.next();
table.removeFunc(removeNum);
System.out.println("Complete!");
table.printHashTable();
```

Рисунок 17 – Удаление элемента из таблицы

```
//Время чтения
System.out.println("\nTesting Reading time");
long time = System.currentTimeMillis();
table.getTelefoNumber( telefonNumber: "89266715863");
System.out.println("Time is: " + (System.currentTimeMillis() - time) + "ms");
```

Рисунок 18 – Время поиска элемента в таблице

```
//Считываем с файла
long time2 = System.currentTimeMillis();
ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream( name: "SiAOD_Test.txt"));
HashTable tableNew = (HashTable) inputStream.readObject();
System.out.println("\nFrom the File: ");
tableNew.printHashTable();
System.out.println("Time is: " + (System.currentTimeMillis() - time2) + "ms");
```

Рисунок 19 – Считывание с файла (с временем чтения)

```
//Тест рехеширования
table = new HashTable( ts: 1);
System.out.println("\nTesting Reheshing");
//Создал таблицу на 1 элемент и добавил туда 3 элемента
table.insert( telefonNumber: "89023482402",  adress: "Domodedovo");
table.insert( telefonNumber: "89230301020",  adress: "Sheremet'evo");
table.insert( telefonNumber: "89991301230",  adress: "Vnukovo");
//Все элементы были добавлены
table.printHashTable();
```

Рисунок 20 – Тест рехеширования

```
//Тест коллизии
System.out.println("\nТестирование коллизий: попробуем добавить в таблицу 2 одинаковых ключа");
HashTable table2 = new HashTable( ts: 2);
table2.insert( telefonNumber: "89999999999",  adress: "Sheremet'evo");
table2.insert( telefonNumber: "89999999999",  adress: "Vnukovo");
```

Рисунок 21 – Тест коллизии

```
//Достаем из файла по файловому номеру '2'
System.out.println("\nТестирование доступа по файловому номеру: " +
        "добавим в таблицу 5 записей и достанем значение по номеру '3'");
table2 = new HashTable( ts: 5);
table2.insert( telefonNumber: "89999999991",  adress: "Sheremet'evo");
table2.insert( telefonNumber: "89999999992",  adress: "Vnukovo");
table2.insert( telefonNumber: "89999999993",  adress: "Domodevo");
table2.insert( telefonNumber: "89999999994",  adress: "Vnukovo");
table2.insert( telefonNumber: "89999999995",  adress: "Vnukovo");

ObjectInputStream inputStream2 = new ObjectInputStream(new FileInputStream( name: "SiAOD_Test.txt"));
HashTable tableNew2 = (HashTable) inputStream2.readObject();
ValueEntry valueEntry = tableNew2.getValueByFileNumber(3);
System.out.println("Значение: " + valueEntry);
```

Рисунок 22 – Тест доставания элемента по номеру в файле

```
//Удаление и получение по номеру
System.out.println("\nПопробуем удалить значение из файла с номером '3' и снова получить");
inputStream2 = new ObjectInputStream(new FileInputStream( name: "SiAOD_Test.txt"));
tableNew2 = (HashTable) inputStream2.readObject();
tableNew2.removeValueByFileNumber(3);
System.out.println("Удалили, пробуем получить");

inputStream2 = new ObjectInputStream(new FileInputStream( name: "SiAOD_Test.txt"));
tableNew2 = (HashTable) inputStream2.readObject();
valueEntry = tableNew2.getValueByFileNumber(3);
System.out.println("Значение: " + valueEntry);
```

Рисунок 23 – Тест удаления и получения по номеру в файле

Скриншот результатов выполнения операций с хеш-таблицей

```
Hash Table
PHONE: 88421674819 | ADRESS: Novosibirsk
PHONE: 89266715863 | ADRESS: Moscow
PHONE: 80248124167 | ADRESS: Vladivostok
PHONE: 89934444449 | ADRESS: St. Peterburg
PHONE: 89231236412 | ADRESS: Kazan
```

Рисунок 24 – Результат заполнения таблицы

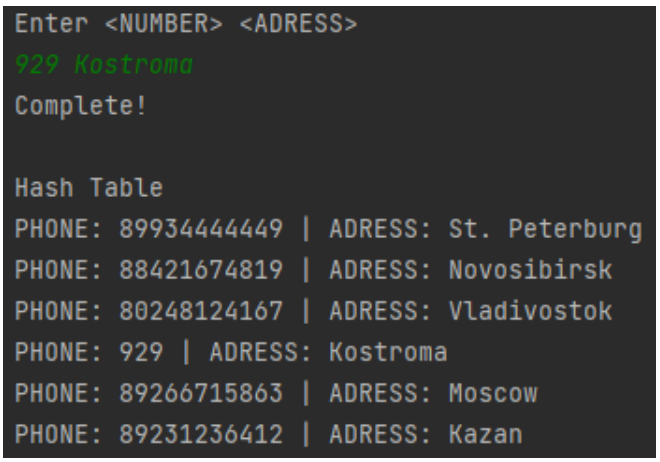

Рисунок 25 – Результат вставления нового номера

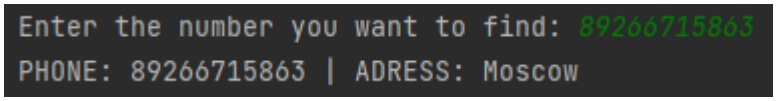

Рисунок 26 – Результат поиска по номеру

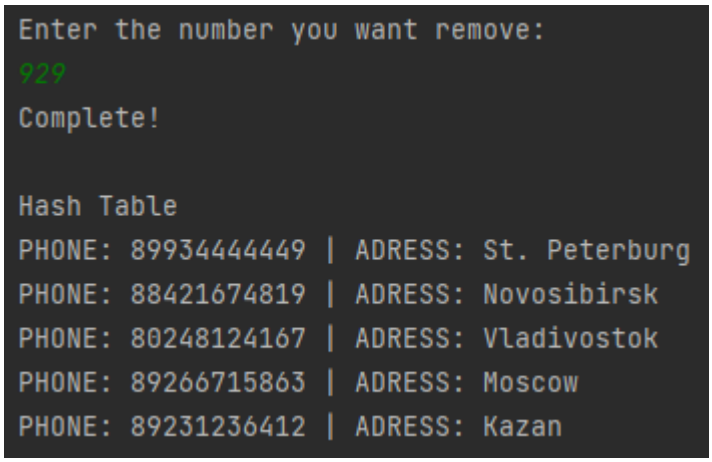

Рисунок 27 – Результат удаления по номеру

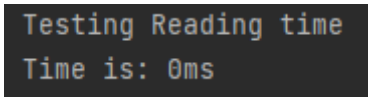

Рисунок 28 – Результат считывания времени поиска элемента

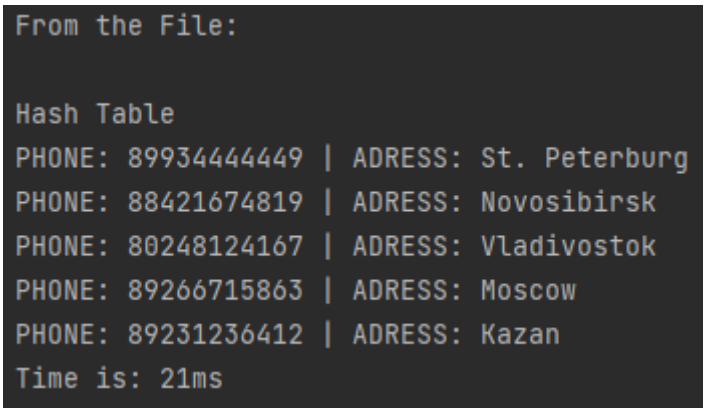

Рисунок 29 – Результат считывания с файла и вывода, с временем считывания

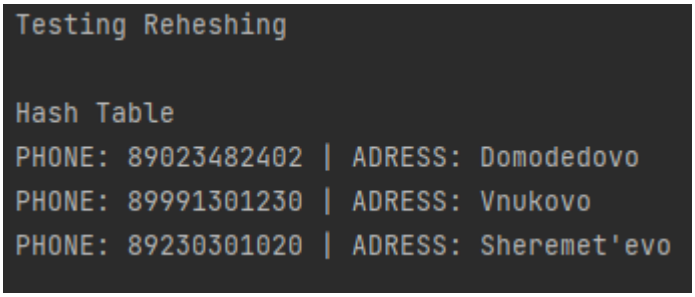

Рисунок 30 – Результат рехеширования (добавили 3 элемента в таблицу с 1 ячейкой изначально)

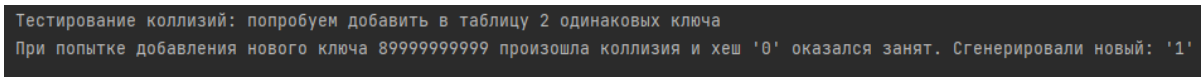

Рисунок 31 – Результат тестирования коллизии

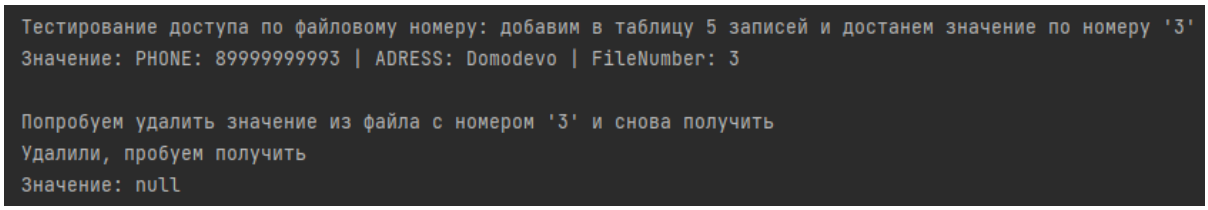

Рисунок 32 – Результат тестирования удаления элемента по номеру в файле

Скриншоты содержания файла и хеш-таблицы

```
From the File:

Hash Table
PHONE: 88421674819 | ADRESS: Novosibirsk
PHONE: 89264038712 | ADRESS: Podolsk
PHONE: 89266715863 | ADRESS: Moscow
PHONE: 80248124167 | ADRESS: Vladivostok
PHONE: 89934444449 | ADRESS: St. Peterburg
```

Рисунок 15 – Полученная хеш-таблица

SiAOD_Test.txt

```
aced 0005 7372 0011 7275 2e74 6173 6b2e
4861 7368 5461 626c 654f b7ee 1a94 edc0
7502 0004 4900 0868 6173 6853 697a 6549
0004 7369 7a65 4900 0974 6f74 616c 5369
7a65 5b00 0574 6162 6c65 7400 155b 4c72
752f 7461 736b 2f56 616c 7565 456e 7472
793b 7870 0000 0005 0000 0005 0000 0003
7572 0015 5b4c 7275 2e74 6173 6b2e 5661
6c75 6545 6e74 7279 3b99 7a81 2b22 dda9
1102 0000 7870 0000 0005 7372 0012 7275
2e74 6173 6b2e 5661 6c75 6545 6e74 7279
4b54 79cb 8692 0a56 0200 0349 000a 6669
6c65 4e75 6d62 6572 4c00 0661 6472 6573
7374 0012 4c6a 6176 612f 6c61 6e67 2f53
7472 696e 673b 4c00 0d74 656c 6566 6f6e
4e75 6d62 6572 7100 7e00 0678 7000 0000
0574 000b 566c 6164 6976 6f73 746f 6b74
000b 3830 3234 3831 3234 3136 3773 7100
7e00 0500 0000 0474 000b 4e6f 766f 7369
6269 7273 6b74 000b 3838 3432 3136 3734
3831 3973 7100 7e00 0500 0000 0174 0006
4d6f 7363 6f77 7400 0b38 3932 3636 3731
3538 3633 7371 007e 0005 0000 0002 7400
0d53 742e 2050 6574 6572 6275 7267 7400
0b38 3939 3334 3434 3434 3439 7371 007e
0005 0000 0003 7400 054b 617a 616e 7400
0b38 3932 3331 3233 3634 3132
```

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.