1. Цель работы

Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

2. Вариант задания

№ вар.	Форма ключа	Количество сегментов	Метод хеширования (разрешение коллизий)
10	БццццБ	2000	Линейное опробование

3. Описание хеш-функции

Используя заданный формат ключа и количество сегментов в таблице, составляем хеш-функцию. Ключ представляет собой строку, содержащую буквы и цифры в определенных позициях.

Для проверки формата ключа используем функцию boolean itsKey(String key) она проверяет строку на паттерн формата ключа, если не соответствует, то возвращает false.

Хеш-функция, принимает входную строку в качестве ключа и выполняет ряд операций для генерации числового значения (хеша). Вот как происходит процесс хеширования:

- Инициализация: Создается объект StringBuilder для сборки строки.
- Извлечение символов: Получается ASCII-код первого и шестого символа входной строки, а также подстрока, состоящая из символов с позиции 1 по 5 (не включая 5).
- Сборка строки: Первый символ добавляется в объект StringBuilder, за которым следует добавление подстроки.
- Преобразование в число: Полученная строка преобразуется в целое число, к которому прибавляется значение ASCII-кода шестого символа.

- Вычисление хеша: Возвращается остаток от деления полученной суммы на (length - 1), где length - это длина строки.

Эта хеш-функция предназначена для быстрого преобразования входной строки в числовое значение, которое может быть использовано, для индексации объектов в хеш-таблице.

```
private int hashCode(String key) {
        StringBuilder stringBuilder = new StringBuilder();
        int first = key.charAt(0);

        int last = key.charAt(5);
        String sub = key.substring(1, 5);

        stringBuilder.append(first);
        stringBuilder.append(sub);
        int hash = Integer.parseInt(stringBuilder.toString())+last;
        return hash % (length - 1);
    }
```

4. Результаты анализа хеш-функции

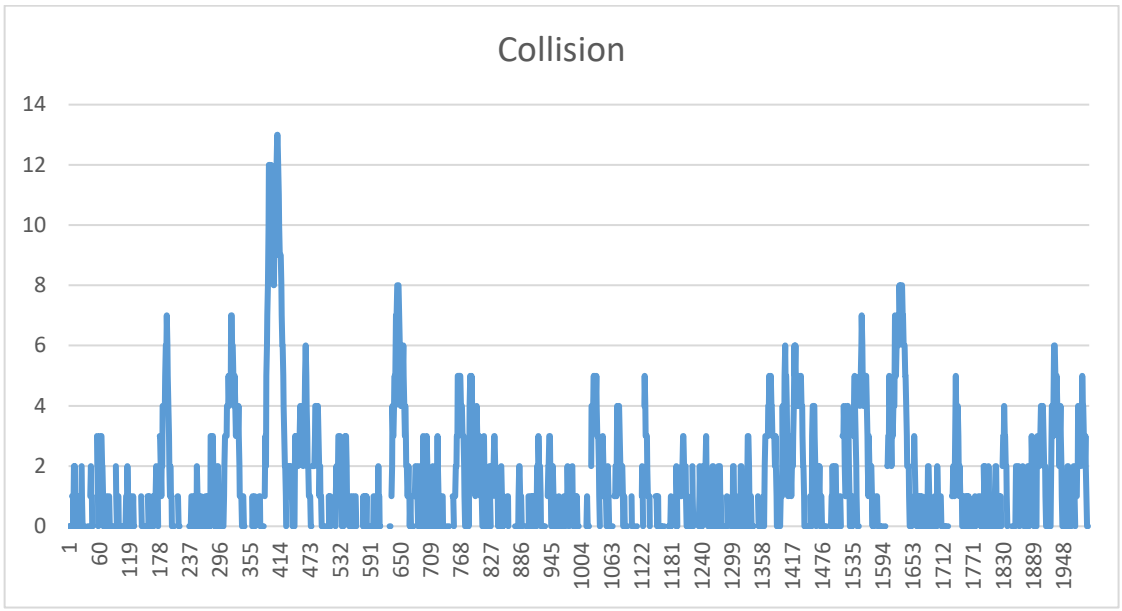

Рис. 1 Результат экспериментального анализа хеш-функции.

Сгенерировано 1500 ключей на 2000 сегментов.

Данный эксперимент показал, что хеш функция неплохо справилась, но надо оговориться, что для этого эксперимента был изменен исходный код программы.

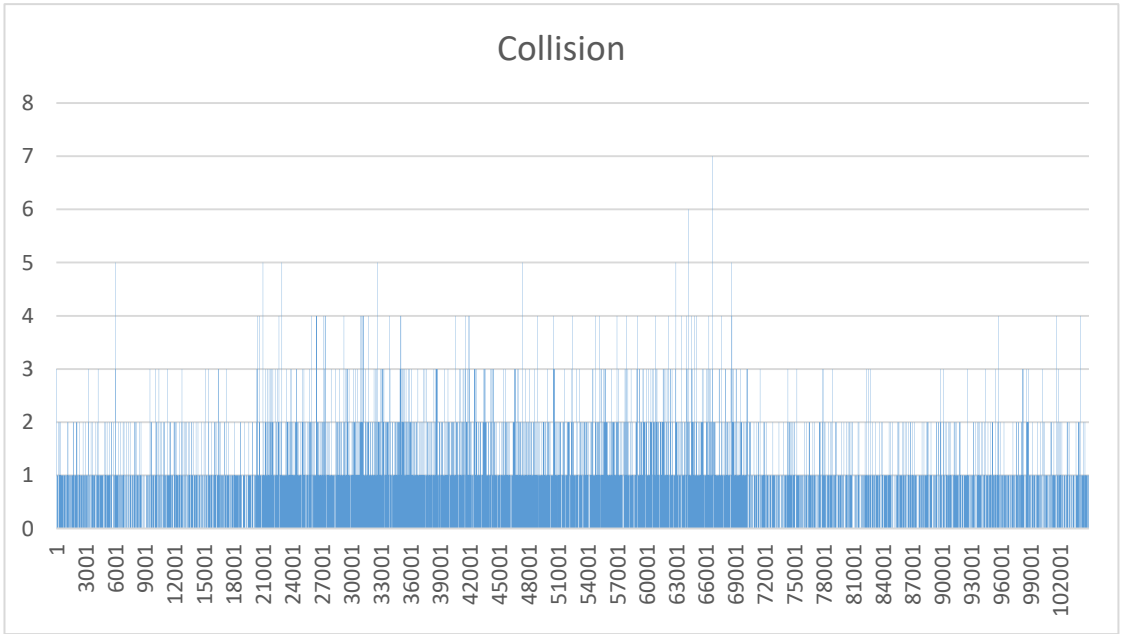

Рис.2 Результат экспериментального анализа хеш-функции

Сгенерировано 40000 ключей.

Как видно из Рис. 2, коллизии равномерно распределены, что достигается за счет увеличения размера таблицы. Когда таблица заполняется на 80%, ее размер увеличивается в 3 раза. Первоначальная таблица содержит 16 бакетов.

5. Листинг программы, реализующей хеш-таблицу и заданный перечень функции

```
package myHash;


import myHash.buket.Buket;


import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class HashTable {



    String filePath = "output.csv";
    private int length = 16;
    private Buket table[] = new Buket[length];
    private int capacity = 0;
```

```
private BufferedWriter writer;



public HashTable() throws IOException {
    try {

        writer = new BufferedWriter(new FileWriter(filePath));


        writer.write("Collision,Segments");
        writer.newLine();

    } catch (IOException e) {
        System.out.println("Ошибка при записи в файл: " + e.getMessage());
    }
}


public void put(String key) throws IOException {
    boolean flagColl = false;
    if (!itsKey(key)) {
        System.out.println("Ключ не соответствует шаблону \"БццццБ\"");
    } else {
        int index = hashCode(key);
```

```
if (table[index] == null) {
    table[index] = new Buket(key);
    capacity++;
} else if (containsKey(key)) {


} else {
    int temp = index;
    if (table[temp] != null) {
        table[temp].addCollision();
    }
    while (table[index] != null) {


        if (flagColl) {
            if (table[temp] != null) {
                table[temp].addCollision();
            }
        }
        index++;
        temp = index;
        if (index == length - 1) {
            index = 0;
        }
        flagColl = true;
```

```
            }
            table[temp] = new Buket(key);
            capacity++;
        }
        if (capacity > length * 0.8) {
            reHashTable();
        }


    }
}

private void reHashTable() throws IOException {
    this.length = length*3;
    this.capacity = 0;
    Buket temp[] = table;
    table = new Buket[this.length];
    for (Buket b : temp) {
        if (b != null) {
            put(b.getKey());
        }
    }
```

```
}

public void printToFile() throws IOException {

    int index = 0;
    while (index != length - 1) {
        if (table[index] != null) {
            writer.write(table[index].getCollision() + "," + index);
            writer.newLine();
        } else {
            writer.write("" + "," + index);
            writer.newLine();
        }
        index++;
    }
    writer.close();
}

private int hashCode(String key) {
    StringBuilder stringBuilder = new StringBuilder();
    int first = key.charAt(0);

    int last = key.charAt(5);
```

```
    String sub = key.substring(1, 5);

    stringBuilder.append(first);
    stringBuilder.append(sub);


    int hash = Integer.parseInt(stringBuilder.toString())+last;

    return hash % (length - 1);
}

private boolean itsKey(String key) {

    String pattern = "[A-Z]\\d{4}[A-Z]";

    Pattern regex = Pattern.compile(pattern);
    Matcher matcher = regex.matcher(key);

    return matcher.matches();
}

public void printTable() {
    for (int i = 0; i < length; i++) {
```

```
        if (table[i] == null) {
            System.out.println(i + " " + "NULL");
        } else {
            System.out.println(i + " " + table[i].toString());
        }
    }
}

public String get(String key) {
    if (containsKey(key)) {

        int index = hashCode(key);
        if (table[index].getKey() != key) {

            while (table[index].getKey() != key) {

                index++;
                if (index == length - 1) {
                    index = 0;
                }
            }
            return table[index].getData();
```

```
        } else return table[index].getData();


    } else {
        return null;
    }
}

public boolean containsKey(String key) {

    int startIndex = hashCode(key);
    int counter = 0;
    if (table[startIndex] == null) {
        return false;
    } else {
        while (key != table[startIndex].getKey()) {
            startIndex++;
            counter++;
            if (startIndex == length - 1) {
                startIndex = 0;
            }
            if (counter == length * 1.5) {
                return false;
            }
```

```
            if (table[startIndex] == null) {
                return false;
            }
        }
        return true;
    }
}

public void remove(String key) {

    int deleteIndex = hashCode(key);

    if (table[deleteIndex] == null) {
        return;
    }
    while (key != table[deleteIndex].getKey()) {
        deleteIndex++;

        if (deleteIndex == length - 1) {
            deleteIndex = 0;
        }
    }
```

```
        int temp = deleteIndex;

        while (hashCode(table[temp].getKey()) < deleteIndex) {
            temp++;
            if (temp == length - 1) {
                temp = 0;
            }
        }
        table[deleteIndex] = table[temp];
        table[temp] = null;
    }

    public String getSegment(int segment) {
        if (segment > length) {
            return null;
        }
        if (table[segment] == null) {
            return null;
        } else return table[segment].getData();
    }

}
```

put(String key): Метод для вставки элемента в хеш-таблицу. При вставке происходит обработка коллизий, проверка соответствия ключа заданному шаблону и перехеширование, если таблица заполняется более чем на 80% своей емкости.

reHashTable(): Приватный метод, который увеличивает размер таблицы в 3 раза и перехеширует элементы. Этот метод вызывается при переполнении таблицы.

printToFile(): Метод для вывода содержимого таблицы в файл. Он записывает количество коллизий и номер сегмента в каждой строке файла.

hashCode(String key): Приватный метод для генерации хеш-кода на основе входной строки key.

itsKey(String key): Приватный метод для проверки соответствия входной строки ключевому шаблону "[A-Z]\d{4}[A-Z]".

printTable(): Метод для вывода содержимого таблицы на консоль.

get(String key): Метод для получения значения по ключу из таблицы.

containsKey(String key): Метод для проверки наличия ключа в таблице.

remove(String key): Метод для удаления элемента из таблицы по заданному ключу.

getSegment(int segment): Метод для получения значения из определенного сегмента таблицы.

Эти методы обеспечивают функциональность хеш-таблицы, включая добавление, удаление, получение значений и управление коллизиями.

```
package myHash.buket;


public class Buket {
```

```
private String key;
private String data;


private int collision = 0;

@Override
public String toString() {
    return "Buket{" +
            "key='" + key + '\'' +
            ", data='" + data + '\'' +
            ", collision=" + collision +
            '}';
}

public Buket(String key) {
    this.key = key;
    this.data = key;
}

public String getData() {
    return data;
```

```
    }

    public String getKey() {
        return key;
    }

    public int getCollision() {
        return collision;
    }

    public void addCollision() {
        this.collision++;
    }
}
```

Buket(String key) : Конструктор класса, принимает ключ и устанавливает данные равными ключу.

toString() : Метод, возвращающий строковое представление объекта, включая ключ, данные и количество коллизий.

getData() : Метод для получения данных, связанных с объектом.

getKey() : Метод для получения ключа, связанного с объектом.

getCollision() : Метод для получения количества коллизий, связанных с объектом.

addCollision() : Метод для увеличения счетчика коллизий на единицу.

6. Выводы по работе.

Целью данной работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц. Мой код представляет собой реализацию простой хеш-таблицы с обширным функционалом, включающим основные операции работы с данными. Работа с этим кодом позволяет изучить основные принципы хеширования данных и получить практические навыки реализации хеш-таблиц, что соответствует заявленной цели.