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

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;

    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;
    }
}
```

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

```
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)

```
public void insert(String telefonNumber, String adress) {
    if (size >= hashSize) {
        rehash();
    }

    int hashing1 = hasingOne(telefonNumber);
    int hashing2 = hashingTwo(telefonNumber);
    while (table[hashing1] != null) {
        hashing1 += hashing2;
        hashing1 %= hashSize;
    }

    table[hashing1] = new ValueEntry(telefonNumber, adress);
    size++;
    saveFile();
}
```

Рисунок 6 – Метод вставки элемента в хеш-таблицу

```
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

```
//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 11 – 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 12 – 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 13 – 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 14 – 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 15 – 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 16 – 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 17 – Тест рехеширования

Тест с коллизией сделать невозможно, потому что у нас в задании двойное хеширование. Благодаря такому способу хеширования мы избегаем коллизии.

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 18 – Результат заполнения таблицы

```
Enter <NUMBER> <ADRESS>
929 Kostroma
Complete!

Hash Table
PHONE: 89934444449 | ADRESS: St. Peterburg
PHONE: 88421674819 | ADRESS: Novosibirsk
PHONE: 80248124167 | ADRESS: Vladivostok
PHONE: 929 | ADRESS: Kostroma
PHONE: 89266715863 | ADRESS: Moscow
PHONE: 89231236412 | ADRESS: Kazan
```

Рисунок 19 – Результат вставления нового номера

```
Enter the number you want to find: 89266715863
PHONE: 89266715863 | ADRESS: Moscow
```

Рисунок 20 – Результат поиска по номеру

```
Enter the number you want remove:
929
Complete!

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

Рисунок 21 – Результат удаления по номеру

```
Testing Reading time
Time is: 0ms
```

Рисунок 22 – Результат считывания времени поиска элемента

```
From the File:

Hash Table
PHONE: 89934444449 | ADRESS: St. Peterburg
PHONE: 88421674819 | ADRESS: Novosibirsk
PHONE: 80248124167 | ADRESS: Vladivostok
PHONE: 89266715863 | ADRESS: Moscow
PHONE: 89231236412 | ADRESS: Kazan
Time is: 21ms
```

Рисунок 23 – Результат считывания с файла и вывода, с временем считывания

```
Testing Reheshing

Hash Table
PHONE: 89023482402 | ADRESS: Domodedovo
PHONE: 89991301230 | ADRESS: Vnukovo
PHONE: 89230301020 | ADRESS: Sheremet'evo
```

Рисунок 24 – Тест рехеширвоания (добавили 3 элемента в таблицу с 1 ячейкой изначально)

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

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.