10.01.2021

ALGORITHMS & DATA STRUCTURES

DOCUMENT TRACKİNG PROGRAM

Aydın Güler

1. PURPOSE

The subject of the project is to implement the functions of a hash-based document tracking program. This java program will enable to speed up the content search of documents. In this program, the document will be indexed and inserted to a database as all search engines do to all the documents on the internet. In this way, searching a word will be very fast.

In this work, NetBeans IDE 8.2 is used as working environment.

2. THE ALGORITHM

HashTable

Thanks to hashing, a specific object can be uniquely identified from a group of similar objects. Here, there are some examples of hashing used in daily life:

- In universities, a specific number is allocated to each student that can be used to retrieve data about them.
- Similarly, in libraries, a unique number is allocated to each book that can be used to identify the book's details.

These unique numbers are obtained with a hash function. The function returns a unique number to each different object. Then, these numbers are used to determine an index value to store the values in an array. The general idea in hashing is to place each entry in the array uniformly. When a searching is performed, the element will be accessed in O(1) time.

Open Addressing

In Open Addressing method, all elements are stored in an array itself. When a new element needs to be inserted to the array, the hash index of this element's hash value is computed first. Then, the place corresponding to this index number in the array is checked. If the place is not occupied then, the element is inserted there else it continues until an unoccupied place is found.

Index	
0	unintelligible 1
1	
2	except 3
3	
4	
5	is 13
6	

.

.

n-4	
n-3	was 3
n-2	
n-1	
n	house-front 1

Figure 1

Figure 1 shows the general structure of open addressing method that used in this program.

There are 3 types of open addressing method. These are:

- Linear Probing
- Quadratic Probing
- Double Hashing

In this program, double hashing is used to distribute clusters evenly on database.

Double Hashing for Distributing Clusters

The only difference between the 3 types of open addressing methods mentioned above is the interval between successive probes. Double hashing computes these intervals with a second hash function.

```
Index2 = hashFunction2(element) & tableSize;
Index = hashFunction1(element) % tableSize;
Index = (index + 1*index2) % tableSize;
Index = (index + 2*index2) % tableSize;
Index = (index + 3*index2) % tableSize;
Index = (index + 4*index2) % tableSize;
              .
              .
              .
Index = (index + n*index2) % tableSize;
```

The probing process will continue until an unoccupied place is found.

3. SOLUTION

NetBeans IDE 8.2 was used to compile this project. The codes are written in Java. This report will not include the whole code that is used because of it is too long. But it will explain the variables, methods and how they are work.

The main purpose of this program is to create an open address hash structure will be used to trace a text file. The program traces each word and keep the number of occurrences of each word.

The main file of the project is `DocumentTrackingProgram.java` includes all operation codes, another file is `HashObject.java` has a class to implement a Hash Table object. The other file is named `HashTable.java` has a class called `HashTable` that has whole the processes and methods which are defined in `HashTableInterface.java`.

HashTableInterface.java

`HashTableInterface.java` is an interface that specifies what will be used in this project as a method.

```
public interface HashTableInterface {
    Integer GetHash (String mystring);
    void ReadFileandGenerateHash (String filename, int size);
    void DisplayResult (String Outputfile);
    void DisplayResult ();
    void DisplayResultOrdered (String Outputfile);
    int showFrequency (String myword);
    String showMaxRepeatedWord ();
    boolean checkWord (String myword);
    float TestEfficiency();
}
```

HashObject.java

`HashObject.java` has a class called `HashObject` that consists of various methods to implement a HashObject. These methods are:

```
private int getPrime()
  {
    for (int i = tableSize - 1; i >= 1; i--)
    {
      int fact = 0;
      for (int j = 2; j <= (int) Math.sqrt(i); j++)
        if (i % j == 0)
          fact++;
      if (fact == 0)
        return i;
    }
    return 3;}
```

In the method `getPrime()` shown above, returns a prime number smaller than the table size. This prime number will be used in hashFunction2.

```
public String key(int i){
        String key;
        key = table[i].split(" ")[0];
        return key;
}

public int frequency(int i){
        int freq;
        freq = Integer.parseInt(table[i].split(" ")[1]);
        return freq;
}
```

The, `public String key(int i)` and `public int frequency(int i)` methods shown above, split the string and returns the key and frequency parts of it. Because the key and the frequency are in same string variable.

```
public int hashFunction1(String hashThisKey )
{
        int hashVal = hashThisKey.hashCode();
        hashVal %= tableSize;
        if (hashVal < 0)
            hashVal += tableSize;
        return hashVal;
}
public int hashFunction2(String hashThisKey )
{
        int hashVal = hashThisKey.hashCode();
        if (hashVal < 0)
            hashVal += tableSize;
        return primeSize - hashVal % primeSize;
}
```

In the `hashFunction1(String hashThisKey)` method, an unique numeric value of the key is created. This numeric value is computed as: s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]. Then, this hash value is divided into table size to get a reminder. If the reminder is negative, it will be positive by adding the table size. Then, hash value is returned. In `hashFunction2(String hashThisKey)` there is a difference, Hash value is subtracted from prime number which is smaller than the table size and divided into prime number.

```
public void insert(int index, String key)
{
        int dummyFrequency;
        if (table[index] != null && key(index).equals(key)){
            String stringFrequency;
            dummyFrequency =+ frequency(index);
            dummyFrequency++;
            stringFrequency = String.valueOf(dummyFrequency);
            table[index] = key(index) +" "+ stringFrequency;
        }
        else{
            String newKeyValue;
            newKeyValue = key +" "+1;
            table[index] = newKeyValue;
        }
}
```

Key and frequency pair will be inserted to table with `insert(int index, String key)` method shown above. If the key is already in the table then, it just increments the frequency of the key in the table[index]. If the key is being inserted first time then, it gives initial frequency as 1.

Methods and Explanations of HashTable.java

```
public Integer GetHash(String mystring)
```

A hash value of mystring is computed and this value is divided into table size to get the index value. Then this value is returned. hash1 represents the first hash function hash2 represents the second hash function. If table hash1 is not null and the key that assigned to that location is not equal to mystring then, it will be entered to while loop and hash2 is added to hash1 until an unoccupied place is found.

Here, in the while loop, number of collusions are incremented to update the collision number.

```
public Integer GetHash(String mystring) {
    int hash1 = ht.hashFunction1( mystring );
    int hash2 = ht.hashFunction2( mystring );
    while (ht.table[hash1] != null && !ht.key(hash1).equals(mystring))
    {
        hash1 += hash2;
        hash1 %= ht.tableSize;
        collisions++;
    }
    return hash1;
}
```

```
public void ReadFileandGenerateHash(String filename, int size)
```

Here file is read, and open hash structure is created. Also an object of HW4_HashObject class is created here. File is read line by line and each line is splitted by the space regex to get each word separately. Punctuation marks are removed here. Then, a hash value of each word is taken by using GetHash method and the result is assigned to a string variable named index. Then, myWord is inserted to the place corresponding to this index number in the array.

```
ht = new HashObject(size);
                    .
                    .
                    .
for (int k = 0; k<lineArray.length; k++){
    String myWord = lineArray[k].replaceAll("[,.!?:;]","");
    int index;
    index = GetHash(myWord);
    ht.insert(index, myWord);
}
                    ...
```

```
public void DisplayResult(String Outputfile)
```

In this method, all the keys and their frequencies are written to a file. BufferedWriter and FileWriter are used to write the results to the file.

```
for (int i = 0; i < ht.tableSize; i++){
    if (ht.table[i] != null)
        output.write(ht.key(i) +" "+ht.frequency(i)+"\n");
}
```

```
public void DisplayResult()
```

All the keys and their frequencies are displayed on the screen.

```
for (int i = 0; i < ht.tableSize; i++)
    if (ht.table[i] != null)
        System.out.println(ht.key(i) +" "+ht.frequency(i));
```

```
public void DisplayResultOrdered(String Outputfile)
```

All the keys and their frequencies are ordered and written to a file. Key and frequencies are copied to another array. Then, this array is sorted in a descending order and written to a file.

```
for (int i = 0; i < orderedArray.length; i++){
    for (int j = i+1; j < orderedArray.length; j++){
        if (orderedArray[i] != null && orderedArray[j] != null &&
                Integer.parseInt(orderedArray[i].split(" ")[1]) <
                Integer.parseInt(orderedArray[j].split(" ")[1])){
            dummy = orderedArray[i];
            orderedArray[i] = orderedArray[j];
            orderedArray[j] = dummy;
        }
    }

}
```

public int showFrequency(String myword)

It gets the hash value of myword then, checks this value whether it is in the table or not. Then, the frequency of this word is displayed on the screen. If it is not in the table then, -1 is returned.

```
int index = GetHash( myword ); // Gets the hash value.
    int freq;
    if (ht.table[index] != null && ht.key(index).equals(myword)){
          System.out.println("The frequency of "'+myword+"' is:"
                                +ht.frequency(index));
          freq = ht.frequency(index);
    }
     else{
           System.out.println("The frequency of "' +myword+"' is: " +0);
           freq = -1;
    }
    return(freq);
```

public String showMaxRepeatedWord()

Finds the max repeated word and returns its frequency. The main idea is to check each words' frequencies. Thus, most repeated word is found.

```
String maxRepeatedWord = null;
    int frequencyofMaxRepeatedWord = 0;
    for (int i = 0; i < ht.tableSize; i++){
                if (ht.table[i] != null && ht.frequency(i) >=
                   frequencyofMaxRepeatedWord){
                    maxRepeatedWord = ht.key(i);
                    frequencyofMaxRepeatedWord = ht.frequency(i);
                }
    }
System.out.println("The most repeated word is "'
+maxRepeatedWord+"' and the frequency is: "
+frequencyofMaxRepeatedWord);

return(maxRepeatedWord);
```

```
public boolean checkWord(String myword)
```

It gets the hash value of myword then checks this value whether it is in the table or not. If myword is found in the table, it returns true and prints a message else it will return false and print not found message.

```
    int index = GetHash(myword);
    if (ht.table[index] != null && ht.key(index).equals(myword)){
        System.out.println("" +myword+"" is found and number of
occurrences is: "
                                +ht.frequency(index));
        return true;
    }else{
        System.out.println("" +myword+"" is not found in the text");
        return false;
    }
```

```
public float TestEfficiency()
```

Collisions are printed here.

```
System.out.println("There are "+collisions+" collusion occurred.");
        return(collisions); ndex
```

DocumentTrackingProgram.java

Main program:

```
HashTable myHashTable = new HashTable();
myHashTable.ReadFileandGenerateHash("d:\\FileToHash.txt", 1000);
myHashTable.DisplayResult();
myHashTable.DisplayResult("d:\\NotOrdered.txt");
myHashTable.DisplayResultOrdered("d:\\DescendingOrder.txt");
myHashTable.showFrequency("telescreen");
myHashTable.checkWord("helicopter");
myHashTable.showMaxRepeatedWord();
myHashTable.TestEfficiency()
```

and the output will be like below.

thought 1
patrol 1
series 1
...
...
...
street 1
Any 1
distaste 1

Text file is created.
Ordered in descending order and the text file is created.
The frequency of 'telescreen' is: 3
'helicopter' is found and number of occurrences is: 1
The most repeated word is 'the' and the frequency is: 37
There are 44 collusion occurred.

When the table size is 5000, there will be 5 collusions.
When the table size is 10000, there will be 3 collusions.

The files FileToHash.txt, NotOrdered.txt and DescendingOrder.txt are exists in the DocumentTrackingProgram folder.