03-60-254: Data Structures and Algorithms
Lab Assignment 5: Word Count IV Using Hashing

1 Due date

In your lab section during the week November 26/28. To be submitted 15 minutes before the end of your lab section. Any submission within the last 15 minutes before the end of the lab session will not be accepted and a zero mark will be given. Late assignments will be given a zero mark. Submissions by email will not be accepted.

2 Regulations

This assignment must be done individually. Any similarity found between your code/answers and another student's, or a carbon-copy of an answer found in the web will be considered cheating. For the part where coding is required, you must implement your own program.

3 Objective

The aim of this assignment is to understand Hashing algorithm and its application.

4 The problem specification

The problem is the same as in A1, i.e., to count the frequency of words in a text file. Different from A2 and A4, this time we will not use a sorting method. Instead, similar to lab assignment 1, each time when we see a word we will find the word and its frequency, and update its frequency. Instead of the slow sequential searching as we did in A1, we will use hashing to access the word directly. The overall logic is described below:

```
//Hshmap254 to be implemented by you
HashMap254<String, Integer> map = new HashMap254<String, Integer>();
int len = tokens.length;
//Update the frequency
for (int i = 0; i < len; i++) {
        String token = tokens[i];
        Integer freq = map.get(token);
        if (freq == null)
                map.put(token, 1);
        else
                map.put(token, freq + 1);
}

//find the most popular word
int max = 0;
String maxWord="";
for (String k : map.keys())
        if (map.get(k) > max){
                max = map.get(k);
                maxWord=k;
        }
return new AbstractMap.SimpleEntry<String, Integer>(maxWord, max);
}
```

Your task is to implement HashMap254, so that word counting can work. Note that you are not allowed to use existing implementations, such as the Hashtable class in Java. There are several classes involved as listed below. Note that those class names in blue fonts are hyperlinks and can be clicked to download. To download, right click the link and use 'save as' to save to your local directory.

- WordCountHash: The counting program that uses HashMap254. For speed comparison, the code includes the HeapSort method. The input data are the same as before.
- HashMap254. This is the code you will work on. There are a few lines missing. One in the method *myHashCode*()

- LinkedListForHash254: We use separate chaining to handle the collision. The collisions are stored in this class. It is a linked list, each element stores a (word, freq) pair.

5 Your tasks

Implement the put(key, value) method (3 marks) The template for the *put*(*key*, *value*) method is listed as below. There are three lines missing. Please fill in the three lines so that the program can run and get the correct counting. Note that your counting result should be the same as using the Heap method.

```
public void put(Key key, Value val) {
        // double table size if load factor m/n >0.1
        if (n >= 10 * m)
                resize(2 * m);
        // Put your code here.
        // 1) get the hash value of the key i.
        // 2) then put (key, value) in the i-th linkedList implemented in
           LinkedListForHash254
        // 3) you need to handle the case whether the key is already there.
        int i = myhash(key);
        if (st[i].get(key)==null)
                n++;
        st[i].put(key, val);
}
```

Improve the hashing method (2 marks) In this implementation, you shall observe that the Hash method and Heap method have similar performances. It is against our knowledge that Hashing should be much faster than Heap sorting. A careful reading of the program reveals that the problem is in the *myhash*() method. It uses the polynomial method to calculate the hashcode of a string using the following formula:

$$hash = 31^{n-1}key[0] + 31^{n-2}key[1] + ... + key[n-1]$$

The approach and the formula have no problem, but the corresponding code below is too slow:

```
//calculate hashcode (h1) using the polynomial method:
for (int i = 0; i < k.length(); i++)
        hash=(int)Math.round((Math.pow(31,k.length()-i-1)))*k.charAt(i)+hash;
```

This code is slow because of the invocation of the power function Math.pow(). Please rewrite the code so that in each loop you only need to a multiplication once. Please refer to P.413 of the book for the detailed explanation. You need to demonstrate that your code reduces the running time at least by a half.

6 Bonus mark (1 mark)

If you run the hashing method on the hackedString200kwords.txt, you will find that it is extremely slow: my machine takes 4289ms using Hashing method, but only 115ms using HeapSort. The reason is that most of the words in the file have the same hashcode, hence the chain will be very long in the hashing method. Please rewrite the hashcode part so that it is more efficient. I see a reduction of the time from 4289ms to less than 200ms, i.e., 20 times more efficient. You will get this mark if your program is at least 10 times more efficient.