Homework 6

COMP 272: Data Structures II

The goal of this assignment is to experiment with 3 different hash functions and determine the one best suited for storing a large English word list. The EnglishWordList.txt file has repetitious words that you first need to eliminate and then use a large enough table to store the words so that the search time is O(1) on an average. The number of unique non-empty words in the list is 250,154. We would need a table (slightly) larger than this. You will use a LinkedList to handle collisions by storing words that collide at any index. Your preferred hash table data structure is ArrayList<LinkedList<String>>. We are going to use two metrics (i) total number of collisions, and (ii) average size of the linked list, as indicative of the efficacy of the hash function. For each hash function below, you need to calculate these two metrics and pick the one you think is the most efficient. There is a third metric that you report out but not use it for picking the best one which is the time taken to hash all the words in the list. For timing use System.currentTimeMillis() before and after the critical hash and store call and take the difference. Don't include any I/O statement like Scanner method calls for timing purposes. For submission, you will produce the java code that uses the hash functions to store the words from the word list, the results of the metrics, your final best choice hash function, and the timing results. You will name the Java class MyHashSet for implementing hashing. You will also need to import three libraries

```
import java.util.*;
import java.security.*;
import java.io.*;
```

Here are the ideas for the three hash functions.

1. Use hashCode() for any string s, hashCode(s) = Math.abs(s.hashCode()) % tableSize, where tableSize = 262,127. Use an ArrayList<LinkedList> of capacity tableSize, and the given hash function H to store all the unique words in the EnglishWordList. Use the variable myHashSet for this data structure. Whenever a collision occurs, store the word in the LinkedList at the hash index where collision occurred. After reading all the words into myHashSet

2. Repeat the above with the following hash function H(s) = hash(key)%tableSize, where

```
public int hashCode(Object key) {
 int h;
  return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
 }
```

3. For the third hash function, the size of the table will be 262,144 = 2^{18}. For this we will use the one-way hash SHA-256 used in some computer security applications. We discussed this in class a little bit.

```
public int hashCode(String s) {
byte[] sb=s.getBytes();
     try {
```

```
     MessageDigest md = MessageDigest.getInstance("SHA-256");
     byte[] key=md.digest(sb);
    }
     catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
BitSet bs = BitSet.valueOf(key);
// using the BitSet bs, you will extract 18 bits based on the
//get() method of BitSet.  The 18 bits are extracted at the first
//18 prime numbers 2, 7,17,29,41,53,67,79,97,107,127,139,157,
173,191, 199,227,239.  Put the bits together in that //order to
form an integer and return it.  That will be the hash //value of
the key that you can use in the table of size 2^18.
}
```