CS211 ALGORITHMS & DATA STRUCTURES II

LAB 7

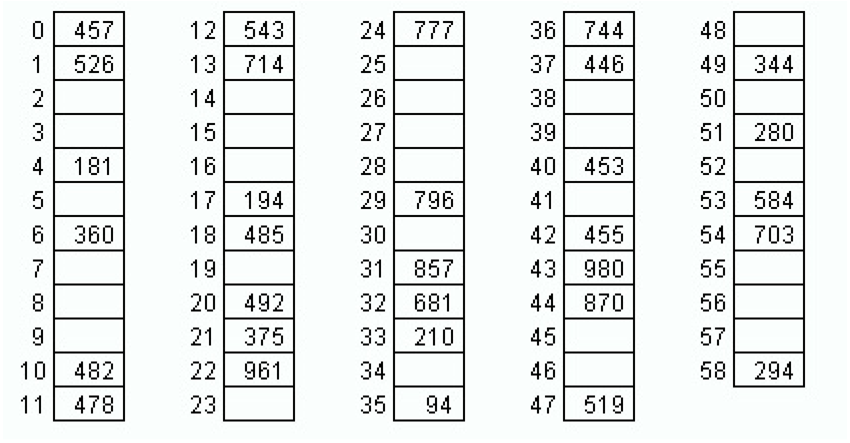
Dr. Phil Maguire

HASHING

Pen and paper exercise

Insert the values 444, 669 and 880 into the following hash table using the following algorithms. The double has function returns a value between 1 and 15.

i) Linear Probing
ii) Quadratic Probing
iii) Double Hashing



0	457	12	543	24	777	36	744	48	
1	526	13	714	25		37	446	49	344
2		14		26		38		50	
3		15		27		39		51	280
4	181	16		28		40	453	52	
5		17	194	29	796	41		53	584
6	360	18	485	30		42	455	54	703
7		19		31	857	43	980	55	
8		20	492	32	681	44	870	56	
9		21	375	33	210	45		57	
10	482	22	961	34		46		58	294
11	478	23		35	94	47	519		

Programming exercise

To practice this question ahead of the lab, you can use the code below and load in the file dictionary.txt

Problem Statement

The goal is to insert all the words into the hash table and then find them all again with the minimum number of collisions. You can use any hashing strategy you like. The main method that reads in the data is provided (see Java 8 code stub). There is also a HashTable class - you must use the check() and set() methods to check and set slots in the HashTable. Every time you make an unsuccessful check() call (i.e. a collision), a counter ticks up. You have to keep this as low as possible. You should just complete the fill() and find() methods, working with mytable, which is the HashTable object. When you run the code, it will output the number of collisions you have. If you see the word "error!" it means that your find() method is not finding the words properly in the HashTable.

Input Format

The input is already taken care of. All you need to do is complete the fill() method, which takes in an array of Strings to be filled into the HashTable object, and the find() method, which should return the slot in the HashTable where the word is to be found.

Output Format

There are two test cases. The first just checks that everything is working - there are few items inserted, so you should get zero collisions, and if you get zero collisions, then the first test case is passed. The second test case will push the load factor to 90%. You should minimize the collisions (anything under 1 million is decent).

NOTE: The second test case will not be passed (because there is no correct answer). Instead, you must call over a demonstrator and show them the number of collisions you achieved. Show them the number on the screen so they can write it down. Then submit your code as usual so there is a record of it.

This lab will be graded by how low you can get the number of collisions.

```
import java.util.*;
```

```
public class Stub{

    public static void main (String[] args){
        Scanner myscanner = new Scanner(System.in);
        int items = myscanner.nextInt();
        myscanner.nextLine();
        String[] contents = new String[items];
        for(int i=0;i<items;i++){
            contents[i]=myscanner.nextLine();
        }
        HashTable mytable = new HashTable();
        Solution mysolution = new Solution(mytable);
        mysolution.fill(contents);
        mysolution = new Solution(mytable);
        for(int i=0;i<items;i++){
            int rand = (int)(Math.random()*items);
            String temp = contents[i];
            contents[i]=contents[rand];
            contents[rand]=temp;
        }
        for(int i=0;i<items;i++){
            int slot = mysolution.find(contents[i]);
            if(!mytable.check(slot,contents[i])){
                System.out.println("error!");
            }
        }
        System.out.println(mytable.gettotal());
    }
}


class HashTable{

    public int size = 99991;
    private String[] hashTable;
    private int total=0;

    public HashTable(){
        hashTable = new String[size];
        for(int i=0;i<size;i++){
            hashTable[i]="";
        }
    }

    public boolean check(int slot, String check){
        if(hashTable[slot].equals(check)){
            return true;
        }else{
            total++;
            return false;
        }
    }

    public void set(int slot, String word){
        hashTable[slot]=word;
    }

    public int gettotal(){
        return total;
    }
}


class Solution{

    HashTable mytable = new HashTable();

    public Solution(HashTable input){

        mytable = input;
        //this is the constructor

    }

    public int find(String word){

        //fill this in so as to minimize collisions
        //this method should return the slot in the HashTable
where the word is

        return 0;
    }

    public void fill(String[] array){

        //fill this in using some hashing strategy
        //this should add all the words in the array into the
HashTable

    }
}
```