CSCI 321 Computer Science III Summer 2019

Assignment 4

Problem: ***Build a hash table for websites***

General Setup and Requirements:

* A website w consists of two fields: (url, name)
* Use w’s url to compute w’s hash key:
  + Hash code: url -> integer
  + Compression function: integer -> [0, m-1] where m is the size of the table
* In your hash table implementation, it should have at least three methods:
  + put(key k, value v): Put a new website by its key k to the hash table
  + get(key k): Return the website associated with k
  + delete(key k): Remove the website associated with k
* As a test suite, the following operations should be performed:
  + Put ([www.uscupstate.edu](http://www.uscupstate.edu), USC Upstate)
  + Put ([www.google.com](http://www.google.com), Google)
  + Put (www.yahoo.com, Yahoo)
  + Get [www.google.com](http://www.google.com)
  + Delete [www.yahoo.com](http://www.yahoo.com)
  + Get [www.uscupstate.edu](http://www.uscupstate.edu)
  + Get [www.google.com](http://www.google.com)
  + Get www.yahoo.com

**Part a**. Implement a hash table to store websites using **linear probing** to handle collision. Sample code “LinearProbingHashTable.java” is attached for your reference. It implements the hash table using linear probing. Attach your code and screenshots.

import java.util.Scanner;  
   
   
class LinearProbingHashTable  
{  
 private int currentSize, maxSize;   
 private String[] keys;   
 private String[] vals;   
   
 public LinearProbingHashTable(int capacity)   
 {  
 currentSize = 0;  
 maxSize = capacity;  
 keys = new String[maxSize];  
 vals = new String[maxSize];  
 }   
   
 public void makeEmpty()  
 {  
 currentSize = 0;  
 keys = new String[maxSize];  
 vals = new String[maxSize];  
 }  
   
 public int getSize()   
 {  
 return currentSize;  
 }  
   
 public boolean isFull()   
 {  
 return currentSize == maxSize;  
 }  
   
 public boolean isEmpty()   
 {  
 return getSize() == 0;  
 }  
   
 public boolean contains(String key)   
 {  
 return get(key) != null;  
 }  
   
 private int hash(String key)   
 {  
 return key.hashCode() % maxSize;  
 }   
   
 public void insert(String key, String val)   
 {   
 int tmp = hash(key);  
 int i = tmp;  
 do  
 {  
 if (keys[i] == null)  
 {  
 keys[i] = key;  
 vals[i] = val;  
 currentSize++;  
 return;  
 }  
 if (keys[i].equals(key))   
 {   
 vals[i] = val;   
 return;   
 }   
 i = (i + 1) % maxSize;   
 } while (i != tmp);   
 }  
   
 public String get(String key)   
 {  
 int i = hash(key);  
 while (keys[i] != null)  
 {  
 if (keys[i].equals(key))  
 return vals[i];  
 i = (i + 1) % maxSize;  
 }   
 return null;  
 }  
   
 public void remove(String key)   
 {  
 if (!contains(key))   
 return;  
   
 int i = hash(key);  
 while (!key.equals(keys[i]))   
 i = (i + 1) % maxSize;   
 keys[i] = vals[i] = null;  
   
 for (i = (i + 1) % maxSize; keys[i] != null; i = (i + 1) % maxSize)  
 {  
 String tmp1 = keys[i], tmp2 = vals[i];  
 keys[i] = vals[i] = null;  
 currentSize--;   
 insert(tmp1, tmp2);   
 }  
 currentSize--;   
 }   
   
 public void printHashTable()  
 {  
 System.out.println("\nHash Table: ");  
 for (int i = 0; i < maxSize; i++)  
 if (keys[i] != null)  
 System.out.println(keys[i] +" "+ vals[i]);  
 System.out.println();  
 }   
}  
   
public class LinearProbingHashTableTest  
{  
 public static void main(String[] args)  
 {  
 Scanner scan = new Scanner(System.in);  
 System.out.println("Hash Table Test\n\n");  
 System.out.println("Enter size");  
 LinearProbingHashTable lpht = new LinearProbingHashTable(scan.nextInt() );  
   
 char ch;  
 do   
 {  
 System.out.println("\nHash Table Operations\n");  
 System.out.println("1. insert ");  
 System.out.println("2. remove");  
 System.out.println("3. get");   
 System.out.println("4. clear");  
 System.out.println("5. size");  
   
 int choice = scan.nextInt();   
 switch (choice)  
 {  
 case 1 :   
 System.out.println("Enter key and value");  
 lpht.insert(scan.next(), scan.next() );   
 break;   
 case 2 :   
 System.out.println("Enter key");  
 lpht.remove( scan.next() );   
 break;   
 case 3 :   
 System.out.println("Enter key");  
 System.out.println("Value = "+ lpht.get( scan.next() ));   
 break;   
 case 4 :   
 lpht.makeEmpty();  
 System.out.println("Hash Table Cleared\n");  
 break;  
 case 5 :   
 System.out.println("Size = "+ lpht.getSize() );  
 break;   
 default :   
 System.out.println("Wrong Entry \n ");  
 break;   
 }  
 lpht.printHashTable();   
   
 System.out.println("\nDo you want to continue (Type y or n) \n");  
 ch = scan.next().charAt(0);   
 } while (ch == 'Y'|| ch == 'y');   
 }  
}

**Part b**. Implement a hash table to store websites using **double hashing** to handle collision. You can modify the Sample code in Part a for this part. Attach your code and screenshots.

**Note: You can refer to the class slides and the following to understand the idea of linear probing and double hashing.**

**Linear probing:** This technique is used when we have more index in the table then the values to be stored. Linear probing technique work on the concept of keep incrementing until you find the empty slot. The pseudo code looks like this.

index = h(k)

while( val(index) is occupied)

index = (index+1) mod m

**Double hashing technique:** In this technique, we use two hashing functions h1(k) and h2(k). If the slot at h1(k) is occupied then the second hashing function h2(k) used to increment the index. The pseudo-code looks like this.

index = h1(k)

while( val(index) is occupied)

index = (index + h2(k)) mod m