CSCI 321 Computer Science III Fall 2018

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
* jjAs 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.

public class LinearProbingHashTable<Key, Value> {

private static final int INIT\_CAPACITY = 4;

private int n; // number of key-value pairs in the symbol table

private int m; // size of linear probing table

private Key[] keys; // the keys

private Value[] vals; // the values

// Initializes an empty symbol table.

public LinearProbingHashTable() {

this(INIT\_CAPACITY);

}

// Initializes an empty symbol table with the specified initial capacity.

public LinearProbingHashTable(int capacity) {

m = capacity;

n = 0;

keys = (Key[]) new Object[m];

vals = (Value[]) new Object[m];

}

// Returns the number of key-value pairs in this symbol table.

public int size() {

return n;

}

//Returns true if this symbol table is empty.

public boolean isEmpty() {

return size() == 0;

}

// Returns true if this symbol table contains the specified key.

public boolean contains(Key key) {

if (key == null) throw new IllegalArgumentException("argument to contains() is null");

return get(key) != null;

}

// hash function for keys - returns value between 0 and M-1

private int hash(Key key) {

return (key.hashCode() & 0x7fffffff) % m;

}

/\*\*

\* Inserts the specified key-value pair into the symbol table, overwriting the old

\* value with the new value if the symbol table already contains the specified key.

\* Deletes the specified key (and its associated value) from this symbol table

\* if the specified value is null.

\*/

public void put(Key key, Value val) {

if (key == null) throw new IllegalArgumentException("first argument to put() is null");

if (val == null) {

delete(key);

return;

}

int i;

for (i = hash(key); keys[i] != null; i = (i + 1) % m) {

if (keys[i].equals(key)) {

vals[i] = val;

return;

}

}

keys[i] = key;

vals[i] = val;

n++;

}

/\*\*

\* Returns the value associated with the specified key.

\*/

public Value get(Key key) {

if (key == null) throw new IllegalArgumentException("argument to get() is null");

for (int i = hash(key); keys[i] != null; i = (i + 1) % m)

if (keys[i].equals(key))

return vals[i];

return null;

}

/\*\*

\* Removes the specified key and its associated value from this symbol table

\* (if the key is in this symbol table).

\*/

public void delete(Key key) {

if (key == null) throw new IllegalArgumentException("argument to delete() is null");

if (!contains(key)) return;

// find position i of key

int i = hash(key);

while (!key.equals(keys[i])) {

i = (i + 1) % m;

}

// delete key and associated value

keys[i] = null;

vals[i] = null;

n--;

}

public static void main(String[] args) {

LinearProbingHashTable<String, String> st = new LinearProbingHashTable<String, String>();

st.put("www.uscupstate.edu", "USC Upstate");

st.put("www.google.com", "Google");

st.put("www.yahoo.com", "Yahoo");

// print keys

System.out.println("The value for www.uscupstate.edu is: " + st.get("www.uscupstate.edu"));

System.out.println("The value for www.google.com is: " + st.get("www.google.com"));

System.out.println("The value for www.yahoo.com is: " + st.get("www.yahoo.com"));

//get key value

System.out.println("The key for www.google.com: " + st.get("www.google.com"));

// delete some key-value pair entry and try to print the entries in the table again.

st.delete("www.yahoo.com");

System.out.println("The size of the table is: " + st.size());

System.out.println("The value for www.uscupstate.edu is: " + st.get("www.uscupstate.edu"));

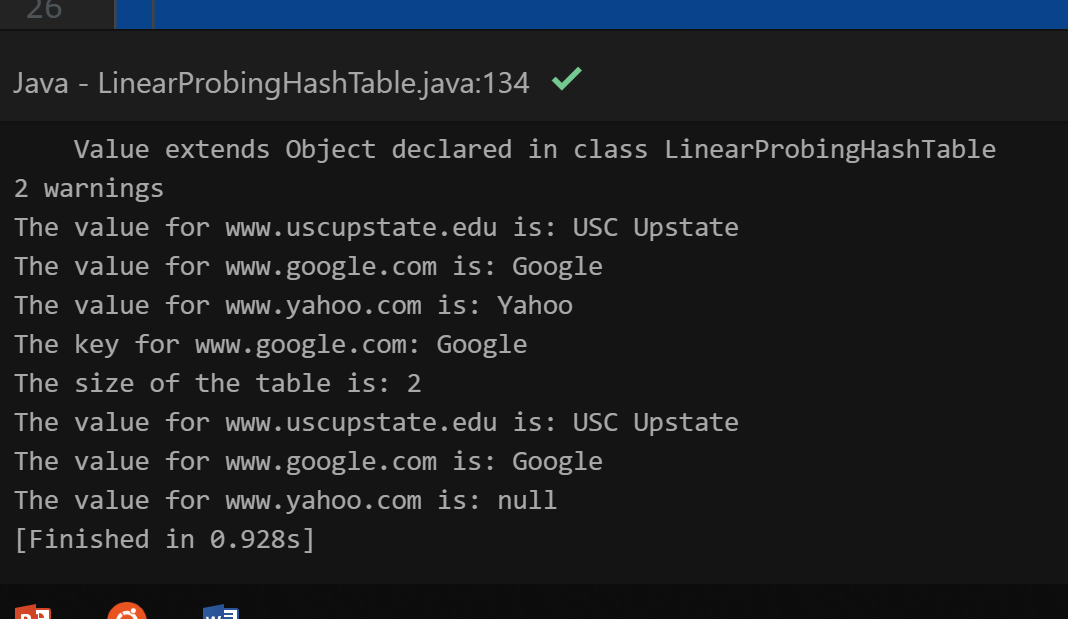
System.out.println("The value for www.google.com is: " + st.get("www.google.com"));

System.out.println("The value for www.google.com is: " + st.get("BB"));

System.out.println("The value for www.yahoo.com is: " + st.get("www.yahoo.com"));

}

}



**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.

public class doublehashing<Key, Value> {

private static final int INIT\_CAPACITY = 4;

private int n; // number of key-value pairs in the symbol table

private int m; // size of linear probing table

private Key[] keys; // the keys

private Value[] vals; // the values

// Initializes an empty symbol table.

public doublehashing() {

this(INIT\_CAPACITY);

}

// Initializes an empty symbol table with the specified initial capacity.

public doublehashing(int capacity) {

m = capacity;

n = 0;

keys = (Key[]) new Object[m];

vals = (Value[]) new Object[m];

}

// Returns the number of key-value pairs in this symbol table.

public int size() {

return n;

}

//Returns true if this symbol table is empty.

public boolean isEmpty() {

return size() == 0;

}

// Returns true if this symbol table contains the specified key.

public boolean contains(Key key) {

if (key == null) throw new IllegalArgumentException("argument to contains() is null");

return get(key) != null;

}

// hash function for keys - returns value between 0 and M-1

private int hash(Key key) {

return (key.hashCode() & 0x7fffffff) % m;

}

private int doublehash(Key key) {

return (key.hashCode() + 2 & 0x7fffffff) % m;

}

/\*\*

\* Inserts the specified key-value pair into the symbol table, overwriting the old

\* value with the new value if the symbol table already contains the specified key.

\* Deletes the specified key (and its associated value) from this symbol table

\* if the specified value is null.

\*/

public void put(Key key, Value val) {

if (key == null) throw new IllegalArgumentException("first argument to put() is null");

if (val == null) {

delete(key);

return;

}

int i;

for (i = hash(key); keys[i] != null; i = (i + doublehash(key)) % m) {

if (keys[i].equals(key)) {

vals[i] = val;

return;

}

}

keys[i] = key;

vals[i] = val;

n++;

}

/\*\*

\* Returns the value associated with the specified key.

\*/

public Value get(Key key) {

if (key == null) throw new IllegalArgumentException("argument to get() is null");

for (int i = hash(key); keys[i] != null ; i = i + doublehash(key))

if (keys[i].equals(key))

return vals[i];

return null;

}

/\*\*

\* Removes the specified key and its associated value from this symbol table

\* (if the key is in this symbol table).

\*/ public void delete(Key key) {

if (key == null) throw new IllegalArgumentException("argument to delete() is null");

if (!contains(key)) return;

// find position i of key

int i = hash(key);

while (!key.equals(keys[i])) {

i = (i + doublehash(key)) % m;

}

// delete key and associated value

keys[i] = null;

vals[i] = null;

n--;

}

public static void main(String[] args) {

doublehashing<String, String> st = new doublehashing<String, String>();

st.put("www.uscupstate.edu", "USC Upstate");

st.put("www.google.com", "Google");

st.put("www.yahoo.com", "Yahoo");

// print keys

System.out.println("The value for www.uscupstate.edu is: " + st.get("www.uscupstate.edu"));

System.out.println("The value for www.google.com is: " + st.get("www.google.com"));

System.out.println("The value for www.yahoo.com is: " + st.get("www.yahoo.com"));

//get key value

System.out.println("The key for www.google.com: " + st.get("www.google.com"));

// delete some key-value pair entry and try to print the entries in the table again.

st.delete("www.yahoo.com");

System.out.println("The size of the table is: " + st.size());

System.out.println("The value for www.uscupstate.edu is: " + st.get("www.uscupstate.edu"));

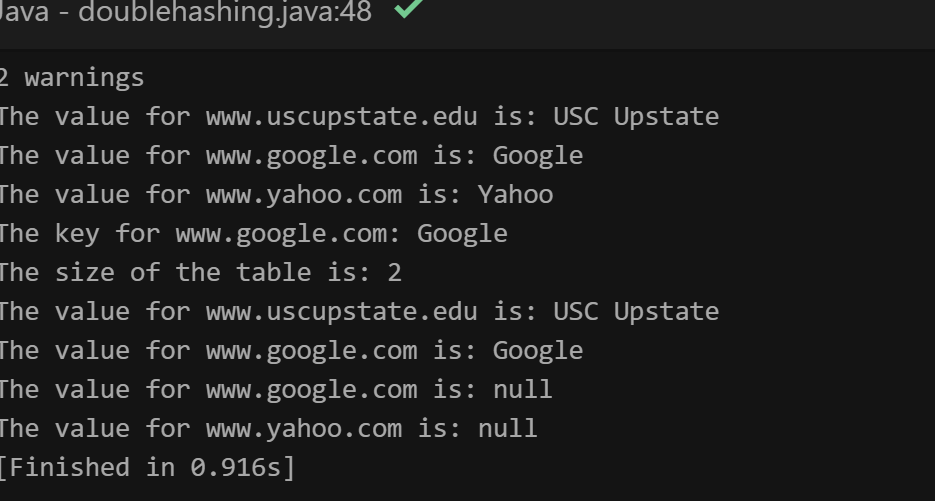
System.out.println("The value for www.google.com is: " + st.get("www.google.com"));

System.out.println("The value for www.google.com is: " + st.get("BB"));

System.out.println("The value for www.yahoo.com is: " + st.get("www.yahoo.com"));

}

}



**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