# DD1339 Introduktion till datalogi 2013/2014

# Uppgift nummer: 4

# Namn: Marcus Larsson

# Grupp nummer: 5

# Övningsledare: Marcus Dicander

# Betyg: ..... Datum: .............. Rättad av: .......................................

# Exercise Implement StringDictionary

## StringHash

/\*\*

\* This is like a simple HashSet. Built in educational purpose. The Set can only contain Strings and only have 3 public methods.

\* All methods are expected to compute on O(1) for n elements in list. As long as the size of the StringHash created is big enough.

\* Recommended size is between half to the total number of elements you intend to store in the Set.

\* @author Marcus

\*/

public class StringHash implements StringDictionary {

private ListElement[] table;

/\*\*

\* Add the specified String to the Set. String will be added if Set currently does not contain the matching String.

\*

\* @param s The String object to add to the Set.

\* @return True if element did not exist before and it was added. False if matching String already existed.

\* @throws Throws IlleagalArgumentException if parameter is not a String.

\*/

@Override

public boolean add(String s) {

if(!(s instanceof String)){

throw new IllegalArgumentException("Invalid type, only String is valid");

}

if(contains(s)){

return false;

}

int index = Math.abs(s.hashCode()%table.length);

ListElement current = table[index];

table[index] = new ListElement();

table[index].value=s;

table[index].next=current;

return true;

}

/\*\*

\* Remove the specified String from the Set. If it exists, method removes element and return true. If not. method returns false.

\*

\* @param s The matching String to delete.

\* @return True if the matching String existed. Else false.

\*/

@Override

public boolean remove(String s) {

if(!(s instanceof String)){

return false;

}

int index = Math.abs(s.hashCode()%table.length);

ListElement current = table[index];

if(current==null){

return false;

}

ListElement next = current.next;

if(current.value.equals(s)){

table[index]=next;

return true;

}

while(current.next!=null){

if(current.next.value.equals(s)){

current.next=next.next;

return true;

}

current=current.next;

next=next.next;

}

return false;

}

/\*\*

\* Searching the set for the specified String.

\*

\* @param s The String to search for

\* @return True if the StringHash contains the given String.

\*/

@Override

public boolean contains(String s) {

if(!(s instanceof String)){

return false;

}

int index = Math.abs(s.hashCode()%table.length);

ListElement current = table[index];

while(current!=null){

if(current.value.equals(s)){

return true;

}

current=current.next;

}

return false;

}

/\*\*

\* Private class that represent the elements in the array. (simple linked list).

\*/

private static class ListElement {

String value;

ListElement next;

}

/\*\*

\* Constructor of StringHash.

\* Specified size must be greater than 0.

\*

\* @param size Specify the size that the Set should have. Good number is between half of the number up to the number of elements you expect to have in the set.

\* @throws IllegalArgumentException if parameter size<=0

\*/

public StringHash(int size){

if(size<=0){

throw new IllegalArgumentException("size="+size);

}

table = new ListElement[size];

}

}

## StringHashTest

import org.junit.Test;

import static org.junit.Assert.\*;

/\*\*

\* TestClass for StringHash

\* @author Marcus

\*/

public class StringHashTest {

public StringHashTest() {

}

/\*\*

\* Test of add method, of class StringHash.

\*/

@Test

public void testAdd() {

StringHash testSet = new StringHash(1000);

assertTrue(testSet.add(""));

assertFalse(testSet.add(""));

for(int i=0; i<1000;i++){

assertTrue(testSet.add("test"+i));

}

//Test to add 2 elements that will end up in same index of array, since these strings generate same hash.

assertTrue(testSet.add("FB"));

assertTrue(testSet.add("Ea"));

//Same thing again but try to add everything in StringHash with size=1.

StringHash testSet2 = new StringHash(1);

assertTrue(testSet2.add(""));

assertFalse(testSet2.add(""));

for(int i=0; i<1000;i++){

assertTrue(testSet2.add("test"+i));

}

//Test to add 2 elements that will end up in same index of array, since these strings generate same hash.

assertTrue(testSet2.add("FB"));

assertTrue(testSet2.add("Ea"));

}

/\*\*

\*

\*/

@Test (expected=IllegalArgumentException.class)

public void testAddNull(){

StringHash testSet = new StringHash(1000);

testSet.add(null);

}

/\*\*

\* Test of remove method, of class StringHash.

\*/

@Test

public void testRemove() {

StringHash testSet = new StringHash(1000);

assertFalse(testSet.contains(""));

testSet.add("");

assertTrue(testSet.remove(""));

assertFalse(testSet.contains(""));

assertFalse(testSet.remove(null));

//Test to add 2 elements that will end up in same index of array, since these strings generate same hash.

testSet.add("FB");

testSet.add("Ea");

assertTrue(testSet.remove("Ea"));

assertFalse(testSet.contains("Ea"));

assertTrue(testSet.contains("FB"));

for(int i=0; i<1000;i++){

testSet.add("test"+i);

}

for(int j=0; j<1000;j++){

assertTrue(testSet.contains("test"+j));

assertTrue(testSet.remove("test"+j));

for(int k=j+1; k<1000;k++){

assertTrue(testSet.contains("test"+k));

}

assertFalse(testSet.contains("test"+j));

}

}

/\*\*

\* Test of contains method, of class StringHash.

\*/

@Test

public void testContains() {

StringHash testSet = new StringHash(1000);

testSet.add("");

assertTrue(testSet.contains(""));

//Test to add 2 elements that will end up in same index of array, since these strings generate same hash.

testSet.add("FB");

assertTrue(testSet.contains("FB"));

assertFalse(testSet.contains("Ea"));

testSet.add("Ea");

assertTrue(testSet.contains("Ea"));

assertTrue(testSet.contains("FB"));

for(int i=0; i<1000;i++){

assertFalse(testSet.contains("test"+i));

testSet.add("test"+i);

assertTrue(testSet.contains("test"+i));

}

}

/\*\*

\* Test to find null in Set.

\* Cannot contain null but should not crash if I check for it.

\*/

@Test

public void testContainsNull(){

StringHash testSet = new StringHash(1000);

assertFalse(testSet.contains(null));

}

/\*\*

\* Will the to create a StringHash object with size=0

\*/

@Test (expected=IllegalArgumentException.class)

public void testCreateEmptyHashString() {

StringHash testSet = new StringHash(0);

}

/\*\*

\* Will test to create a StringHash object with negative size.

\*/

@Test (expected=IllegalArgumentException.class)

public void testCreateNegativeSizeHashString() {

StringHash testSet = new StringHash(-666);

}

}

## StringDictionary

/\*\*

\* An interface describing a dictionary of strings.

\* The dictionary cannot contain duplicate strings.

\*

\* @author Stefan Nilsson

\* @version 2012-12-14

\*/

public interface StringDictionary {

/\*\*

\* Adds the given string to this table.

\* Returns <code>true</code> if the dictionary

\* did not already contain the given string.

\*

\* Complexity: O(1) expected time.

\*/

public boolean add(String s);

/\*\*

\* Removes the given string from this dictionary

\* if it is present. Returns <code>true</code> if

\* the dictionay contained the specified element.

\*

\* Complexity: O(1) expected time.

\*/

public boolean remove(String s);

/\*\*

\* Returns <code>true</code> if the string is

\* in this dictionary.

\*

\* Complexity: O(1) expected time.

\*/

public boolean contains(String s);

}

# Exercise Medelfallstid

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sökning | Insättning | Borttagning |
| Osorterad vektor | O(n) (O(1) om man vet index) | O(1) | O(n) (O(1) om man vet index och inte ska flytta resten.) |
| Sorterad vektor | O(log n) (O(1) om man vet index) | O(n) | O(log n) |
| Osorterad enkellänkad lista | O(n) | O(1) | O(n) (måste hitta elementet) |
| Sorterad enkellänkad lista | O(n) | O(n) | O(n) |
| hashtaball | O(1) | O(1) | O(1) |

Motivering:

I hashtabellen är storleken lika med antalet element. Då får vi platsen till objektet direkt, och kan lägga till och ta bort hur som helst direkt. Därför är alla O(1).