**Algorithmique / Travaux pratiques**

S14 – Tables de hachage



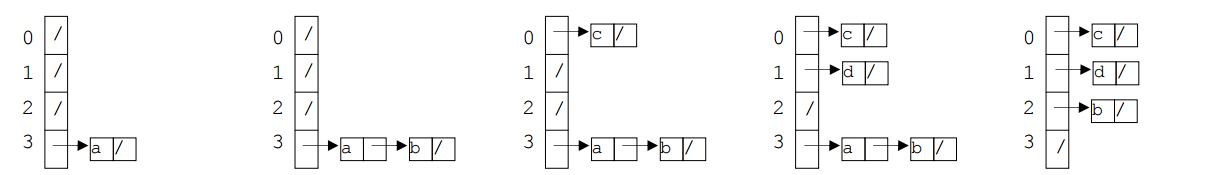


26.05.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

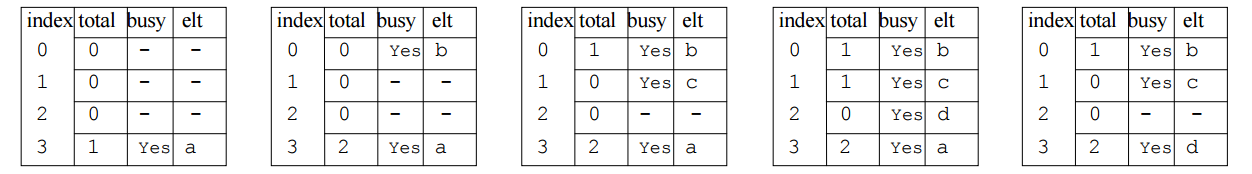
# Dans la première évolution (hachage ouvert), c’est la dernière table qui est incohérente.



|  |  |  |  |
| --- | --- | --- | --- |
| 0 |  |  | c |
| 1 |  |  | d |
| 2 |  |  |  |
| 3 |  |  | b |

Elle devrait avoir cette forme-ci :

Dans la deuxième évolution (hachage fermé), c’est aussi la dernière table qui est incohérente.



Elle devrait avoir cette forme-ci :

|  |  |  |  |
| --- | --- | --- | --- |
| Index  0  1  2  3 | total | busy | elt |
| 1 | Yes | b |
| 1 | Yes | c |
| 0 | Yes | d |
| 1 | - | - |

# Voici la méthode hashCode() de la classe PolynomThing

@Override

**public** **int** hashCode() {

**int** result = 19;

// each char

**for** (**int** i = 0; i < name.length(); i++) {

**int** c = (**int**) name.charAt(i);

result = result \* 31 + c;

}

// boolean

**int** c = (isReducible ? 1 : 0);

result = result \* 31 + c;

// tab

**for** (**int** i = 0; i < degree(); i++) {

c = (**int**) Double.doubleToRawLongBits(coef[i]);

result = result \* 31 + c;

}

**return** Math.abs(result);

//return ((int) Math.abs(result) % degree());

}

Résultat du hash pour les valeurs suivantes :

PolynomThing a = new PolynomThing(new double[] { 2, 3, 4 }, true, "foo");

PolynomThing b = new PolynomThing(new double[] { 2, 3, 5 }, true, "bar");

PolynomThing c = new PolynomThing(new double[] { 2, 3 }, true, "demo");

PolynomThing d = new PolynomThing(new double[] { 2, 3, 4 }, false,"foo");



# Voici les classes SetOfString et SetOfStringsItr

SetOfString :

package s14;

import java.util.BitSet;

public class SetOfStrings {

BitSet busy;

String [] elt;

int [] total;

int crtSize;

static final int DEFAULT\_CAPACITY = 5;

// ------------------------------------------------------------

public SetOfStrings () {

this(DEFAULT\_CAPACITY);

}

public SetOfStrings (int initialCapacity) {

initialCapacity = Math.max(2, initialCapacity);

elt = new String[initialCapacity];

total = new int[initialCapacity];

busy = new BitSet(initialCapacity);

crtSize = 0;

}

int capacity() {

return elt.length;

}

// Here is the hash function :

int hashString(String s) {

int h = s.hashCode() % capacity();

if (h<0) h=-h;

return h;

}

// PRE: table is not full

// returns the index where element e is stored,

// or, if absent, the index where e should be stored

int targetIndex(String e) {

int h = hashString(e);

int t = total[h];

int i = t, j = h;

if(j == elt.length)

j=0;

do{

if(busy.get(j) && elt[j].equals(e))

return j;

else{

if(busy.get(j) && hashString(elt[j]) == h)

i--;

}

j++;

if(j == elt.length)

j=0;

}while (i > 0 && j != h);

j = h;

if(j == elt.length)

j=0;

do{

if(!busy.get(j) ){

return j;

}

else

j++;

if(j == elt.length)

j=0;

}while (j != h);

return 0;

}

public void add(String e) {

if (crtSize\*2 >= capacity())

doubleTable();

if(!contains(e)){

elt[targetIndex(e)] = e;

busy.set(targetIndex(e));

total[hashString(e)]++;

crtSize++;

}

}

private void doubleTable() {

SetOfStrings tempSet = new SetOfStrings(this.elt.length\*2);

for (int i = 0; i < elt.length; i++) {

if(busy.get(i)){

tempSet.add(elt[i]);

}

}

busy = tempSet.busy;

total = tempSet.total;

elt = tempSet.elt;

crtSize = tempSet.crtSize;

}

public void remove(String e) {

int i = targetIndex(e);

if (!busy.get(i)) return; // elt is absent

int h = hashString(e);

total[h]--;

elt[i]=null;

busy.clear(i);

crtSize--;

}

public boolean contains(String e) {

return busy.get(targetIndex(e));

}

public void union (SetOfStrings s) {

for (int i = 0; i < s.elt.length; i++) {

if(s.busy.get(i))

this.add(s.elt[i]);

}

}

public void intersection(SetOfStrings s) {

SetOfStringsItr itr = new SetOfStringsItr(s);

SetOfStrings temp = new SetOfStrings();

while(itr.hasMoreElements()){

String elt = itr.nextElement();

if(this.contains(elt))

temp.add(elt);

}

this.total = temp.total;

this.busy = temp.busy;

this.elt = temp.elt;

this.crtSize = temp.crtSize;

}

public int size() {

return crtSize;

}

public boolean isEmpty() {

return size() == 0;

}

public String [] arrayFromSet() {

String t [] = new String [size()];

int i=0;

SetOfStringsItr itr = new SetOfStringsItr(this);

while (itr.hasMoreElements()) {

t[i++] = itr.nextElement();

}

return t;

}

public String toString() {

SetOfStringsItr itr = new SetOfStringsItr(this);

if (isEmpty()) return "{}";

String r = "{" + itr.nextElement();

while (itr.hasMoreElements()) {

r += ", " + itr.nextElement();

}

return r + "}";

}

// ------------------------------------------------------------

// tiny example

// ------------------------------------------------------------

public static void main(String [] args) {

String a="abc";

String b="defhijk";

String c="hahaha";

SetOfStrings s=new SetOfStrings();

s.add(a); s.add(b); s.remove(a);

if (s.contains(a) || s.contains(c) || !s.contains(b))

System.out.println("bad news...");

else

System.out.println("ok");

}

}

SetOfStringsItr :

package s14;

public class SetOfStringsItr {

SetOfStrings theSet;

int eltShowed = 0;

int currentIndex = 0;

// ------------------------------------------------------------

public SetOfStringsItr (SetOfStrings theSet) {

this.theSet = theSet;

}

// ------------------------------------------------------------

public boolean hasMoreElements() {

return eltShowed < theSet.busy.cardinality();

}

// ------------------------------------------------------------

public String nextElement() {

int next = theSet.busy.nextSetBit(currentIndex);

currentIndex = next+1;

eltShowed++;

return theSet.elt[next];

}

// ------------------------------------------------------------

}

# One-way function

**A**) Sécurité des systèmes informatiques

Afin de stocker un mot de passe dans une base de données.

B) Intégrité des données échangées sur un réseau

Pour contrôler l’intégrité d’un fichier échanger sur internet. Imaginons un document important. Par exemple, lorsqu’on télécharge ce fichier, une chaîne de caractère est fournie avec, ceci est alors le hash du document. Dès lors que le document est télécharger, on va hacher le contenu du document et le comparer à celui reçu avec le téléchargement. Si le hash est le même, le document n’a alors subit aucune modification pendant le transfert.