



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Algorithmique et structures de données

S14 Tables de hachage



Auteurs :
Marc Roten

Professeur :
Rudolph SCHEURER

17 mai 2018

Table des matières

1	Hachage ouvert et Hachage fermé	2
1.1	Hachage fermé	2
1.2	Hachage ouvert	2
2	PolynomThing	3
3	SetOfString	4
3.1	TargetIndex	4
3.1.1	PseudoCode	4
3.1.2	Code utilisé	4
3.2	Autres Méthodes	5
3.2.1	add	5
3.2.2	doubleTable	5
3.2.3	Union	5
3.2.4	intersection	6
3.3	SetOfStringItr	6
4	Ex4	7
4.1	Tables de Hachage, sécurité des systèmes informatiques	7
4.2	Tables de hachage, intégrité des données échangées dans un réseau	7

Table des figures

1	Donnée Exercice 1	2
2	Avec le Hachage fermé	2
3	Avec le Hachage Ouvert	2
4	Résultat à la console PolynomThing	3
5	Résultat à la console	7

1 Hachage ouvert et Hachage fermé

- Repérer les incohérences dans l'évolution de cette table de hachage d'objets (on fait 3 ajouts puis 1 retrait). (a) Hachage ouvert, (b) Hachage fermé.

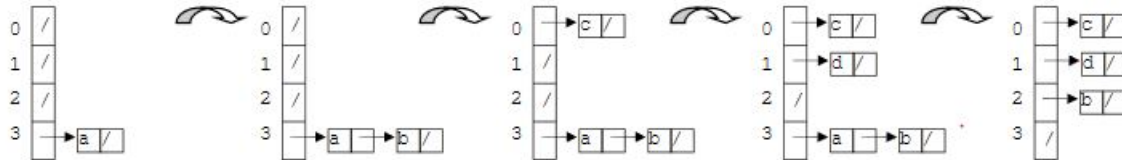


FIGURE 1 – Donnée Exercice 1

1.1 Hachage fermé

index	total	busy	elt
0			
1			
2			
3	1	yes	a

index	total	busy	elt
0		yes	b
1			
2			
3	2	yes	a

index	total	busy	elt
0	1	yes	b
1		yes	c
2			
3	2	yes	a

index	total	busy	elt
0	1	yes	b
1	1	yes	c
2		yes	d
3	2	yes	a

index	total	busy	elt
0	1	yes	b
1	1	yes	c
2		yes	d
3	1	yes	

FIGURE 2 – Avec le Hachage fermé

Avec cette méthode de Hachage, il n'y a que le a qui change, il n'y a donc pas de raison pour le d d'être à l'index 3. Le tableau reste comme indiqué en Figure 2.

1.2 Hachage ouvert

index	total	busy	elt
0			
1			
2			
3	1	yes	a

index	total	busy	elt
0			
1			
2			
3	2	yes	a,b

index	total	busy	elt
0	1	yes	c
1			
2			
3	2	yes	a,b

index	total	busy	elt
0	1	yes	c
1	1	yes	d
2			
3	2	yes	a,b

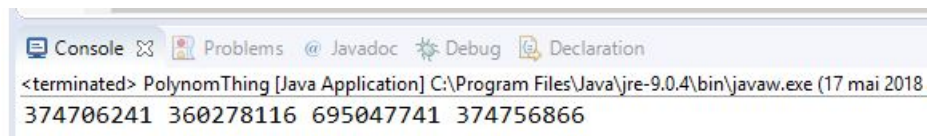
index	total	busy	elt
0	1	yes	b
1	1	yes	c
2	0		
3	1	yes	d

FIGURE 3 – Avec le Hachage Ouvert

Le Hachage ouvert peut contenir plusieurs éléments au même index. C'est la principale différence avec le hachage fermé qui lui décale les éléments pointant vers le même index, jusqu'à ce qu'un index soit libre.

2 PolynomThing

```
@Override public int hashCode() {  
    int hash = 42;  
    hash = 15* hash +(isReducible() ? 0:1);  
    hash = 15* hash +name.hashCode();  
    if(coef !=null){  
        for(double e : coef){  
            long l = Double.doubleToLongBits(e);  
            hash = 15*hash +(int) (l^(1>>>32));  
            //slide 17 theorie s14  
        }  
    }  
    return hash;  
}
```



The screenshot shows a Java IDE console window with the following text:

```
<terminated> PolynomThing [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (17 mai 2018  
374706241 360278116 695047741 374756866
```

FIGURE 4 – Résultat à la console PolynomThing

3 SetOfString

3.1 TargetIndex

Exprimer en pseudo-code, puis implémenter la methode `targetIndex()`. Cette méthode est utilisée pour implémenter les autres méthodes (`add/remove/contains`).

3.1.1 PseudoCode

je parcours mon tableau `elt`. Dès que j'ai un élément dans mon tableau egal à l'élément donné en paramètre, je retourne l'index en question

3.1.2 Code utilisé

```
int targetIndex(String e) {
    for(int i=0;i<elt.length;i++) {
        if(e==elt[i]) {
            return i;
        }
    }
    return 0;
}
```

3.2 Autres Méthodes

3.2.1 add

```
public void add(String e) {
    if (crtSize*2 >= capacity()) doubleTable();
    int index = targetIndex(e);
    busy.set(index);
    elt[index] = e;
    total[hashString(e)]++;
    crtSize++;
}
```

3.2.2 doubleTable

```
private void doubleTable() {
    int len = capacity()*2;
    int[] myTempInt = new int [len];
    String[] myTempString = new String[len];
    BitSet myTempBs = new BitSet(len);
    myTempBs.or(busy);
    for (int i=0; i<capacity(); i++){
        myTempString[i] = elt[i];
        myTempInt[i] = total[i];
    }
    busy = myTempBs;
    elt = myTempString;
    total = myTempInt;
}
```

3.2.3 Union

```
public void union (SetOfStrings s) {
    for (int i = 0; i < s.capacity(); i++) {
        if (s.elt[i] != elt[targetIndex(s.elt[i])]) {
            add(s.elt[i]);
        }
    }
}
```

3.2.4 intersection

```
public void intersection(SetOfStrings s) {  
    //chaque elt qui n'est pas egal a l'elt a targetindex est retire  
    for(int i = 0; i < s.capacity(); i++) {  
        if (elt[i] != s.elt[targetIndex(elt[i])]) remove(elt[i]);  
    }  
}
```

3.3 SetOfStringItr

```
public class SetOfStringsItr {  
    private int count;  
    private int index;  
    private SetOfStrings set;  
    // -----  
    public SetOfStringsItr (SetOfStrings theSet) {  
        this.set = theSet;  
        index = -1;  
        count = 0;  
    }  
    // -----  
    public boolean hasMoreElements() {  
        return count < set.size();  
    }  
    // -----  
    public String nextElement() {  
        index++;  
        while(!set.busy.get(index)){  
            index++;  
        }  
        count++;  
        return set.elt[index];  
    }  
    // -----  
}
```



```
<terminated> SetOfStringsTest [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (17 mai 2018 :  
Using seed 730  
Add/remove seems OK  
Iterator seems OK  
Union seems OK  
Intersection seems OK  
  
Test passed successfully !
```

FIGURE 5 – Résultat à la console

4 Ex4

4.1 Tables de Hachage, sécurité des systèmes informatiques

Les Tables de hachages sont utiles pour stocker dans des systèmes de base de données pour pouvoir stocker des mots de passe par exemple, sans avoir à les stocker en clair. C'est un gros point positif pour la sécurité et l'intégrité des mots de passe des utilisateurs.

4.2 Tables de hachage, intégrité des données échangées dans un réseau

Pour la sécurité des systèmes informatique, on peut utiliser un certain HashCode sur des données, faire transiter les données avec le Hashcode, et réappliquer le hashcode sur le mot de passe pour vérifier que les données n'aient pas été corrompues par un tiers, éviter l'installation de logiciels malveillants