

J-F. Kamp

R3.02 – TP2 – Table de Hachage

Septembre 2023

Deuxième TP d'exercices sur la manipulation de structures assez classiques (listes, arbres) enseignés en seconde année de BUT informatique. Les exercices sont à réaliser en Java, ils abordent également les notions de contrat et de généricité.

**Mise en pratique des
notions vues en cours**

TP2 : Table de Hachage

1. Objectifs

Construction de la classe *HashTable.java* (paquetage *datastruct*) sous Eclipse qui implante une table de hachage très simplifiée (en particulier, on considèrera que la table a une capacité de *TAILLE* tuples maximum, *TAILLE* étant une constante). On créera d'abord une interface *Table.java* et ensuite *HashTable.java* qui implémentera cette interface.

Pour ce TP, vous devez créer un nouveau projet (*projHashTable* par exemple) sous Eclipse. Pour le test, faire usage obligatoirement de *JUnit4*.

2. L'interface Table

L'interface *Table.java* du paquetage *datastruct* contient la signature des 3 méthodes suivantes :

- *public Object select (String key) ;*

Cette méthode renvoie la donnée du *Tuple* correspondant à la clé de recherche *key* passée en paramètre. Renvoie *null* si aucun tuple ne correspond à la clé.

- *public boolean insert (String key, Object data) ;*

Si aucun *Tuple* de la table ne correspond à la clé passée en paramètre, cette méthode insert au bon endroit dans la table un nouveau *Tuple* dont la clé (*key*) et la donnée (*data*) sont passées en paramètres. Renvoie faux si l'insertion n'est pas possible (soit la clé existe déjà, soit le tableau est plein).

Note importante : en cas de collision (i.e. lorsque l'indice « ind » calculé avec *computeIndex(...)*, voir ci-dessous, correspond à une case déjà occupée), on avance de façon circulaire case par case sur le tableau en commençant par la case « ind + 1 » et en s'arrêtant dès qu'une case a un contenu « null ». L'indice de la première case rencontrée avec un contenu « null » est l'emplacement du nouveau *Tuple* inséré dans le tableau.

- *public boolean delete (String key) ;*

Supprime de la table le *Tuple* correspondant à la clé passée en paramètre. Renvoie faux si la suppression n'est pas possible (i.e. aucun *Tuple* ne correspond à cette clé dans la table).

3. La classe HashTable

Cette classe du paquetage *datastruct* implémente l'interface *Table* décrite ci-dessus.

Elle possède 2 attributs :

- *Tuple[] table* : le tableau qui contiendra tous les tuples
- *int nbTuples* : le nombre de tuples que contient le tableau (forcément $\leq TAILLE$)

Et une constante *int TAILLE = 10* qui définit la taille du tableau (une fois pour toute).

En plus des 3 méthodes de l'interface, on développera le constructeur et les méthodes suivantes :

- *public HashTable()* : le constructeur qui doit initialiser les attributs
- *public int getNbTuples ()* : simple accesseur
- *private int computeIndex (String key)* : calcule, sur la base des codes ASCII des caractères qui forment la clé, l'indice du tableau où le *Tuple* sera mémorisé ($0 \leq indice < TAILLE$). Méthode à utiliser dans le code des méthodes *insert* et *select*.

- *private int circularSearch (String key, int indice)* : renvoie l'indice de tableau où se trouve le tuple recherché (ou -1 s'il ne s'y trouve pas). La recherche sur le tableau est circulaire et commence à (indice + 1). Cette méthode est appelée par *select* ou *delete* lorsque le tuple recherché se trouve non pas à l'indice calculé par *computeIndex* mais à un autre emplacement trouvé par parcours circulaire du tableau (voir « Note importante » ci-dessus).
- *public String toString()* : renvoie l'entièreté de la table sous forme d'une chaîne de caractères

On trouvera ensuite 1 classe interne privée *Tuple* décrite ci-dessous.

4. La classe interne *Tuple*

```
private class Tuple {  
    // Attributs  
    String key ;           // la clé d'identification du Tuple  
    Object data ;          // la donnée du Tuple  
  
    // Constructeur d'un Tuple, initialisation des attributs  
    Tuple ( String key, Object data ) { ... }  
  
    // Méthode qui compare la clé du Tuple avec une autre clé (otherKey).  
    // Méthode utilisée pour rechercher le Tuple dans la table (le tableau).  
    // Renvoie vrai si les deux clés sont identiques (faux sinon).  
    boolean sameKey ( String otherKey ) { ... }  
  
    // Renvoie le Tuple sous forme d'une chaîne de caractères qui contiendra  
    // sa clé et sa donnée  
    public String toString() { ... }  
}
```