# Chapitre 7 : Tables de hachage

Dans ce chapitre, nous présentons une nouvelle structure de données : les tables de hachage. Les tables de hachage ont pour vocation de stocker un ensemble de valeurs (pas d'ordre entre ces valeurs) tout en permettant de :

- déterminer rapidement si une valeur est dans la table,
- ajouter/supprimer rapidement des valeurs dans la table.

**Remarque :** Les dictionnaires en python sont des tables de hachage (version plus complexe que les tables de hachage présentées dans ce cours).

## 1 Principe

Quand on a un ensemble de valeurs, si l'on peut partitionner ces valeurs en différents "tas", cela accélère la recherche car il suffit de chercher si la valeur est dans le "tas" correspondant.

**Exemple :** si j'ai un ensemble de nombres, je peux partitionner cet ensemble en 10 tas numérotés de 0 à 9 tel que le tas numéro i contient l'ensemble des nombres terminant par le chiffre i. Pour déterminer si le nombre 48975 appartient à mon ensemble, il me suffit de déterminer s'il appartient au tas numéro 5.

Une table de hachage est un tableau dont chaque case contient un tas. Chaque tas est lui-même représenté par un tableau contenant l'ensemble des valeurs du tas[1].

## 2 Fonctions de hachage

Pour implémenter une table de hachage, il faut être capable de déterminer à quel tas appartient n'importe quelle valeur. Pour cela on utilise une *fonction de hachage* qui prend en paramètre une valeur et retourne le tas correspondant (l'indice du tableau contenant le tas). Pour être efficace, une fonction de hachage doit :

- s'exécuter rapidement (quel que soit la valeur),
- produire à chaque fois le même résultat lorsqu'elle est appelée avec le même paramètre.

**Remarque :** pour être certain que la valeur retournée correspond à un indice valide, on utilise l'opérateur modulo ($\%n$) où $n$ est le nombre de tas de la table.

Voici un exemple de fonction de hachage (les valeurs sont de types chaînes de caractères).

[1] Il existe d'autres possibilités pour implémenter une table de hachage que nous ne verrons pas.

```
def hachage(str):
    nombre =  0
    i = 0
    while i < len(str):
        nombre += ord(str[i])
        i += 1
    return nombre
```

Cette fonction retourne un entier pour chaque chaîne. Par exemple, `hachage("a")` retourne 97 et `hachage("bonjour")` retourne 767. La fonction de hachage permet de déterminer dans quelle case (tas) de la table chercher ; on cherche le mot dans la case d'indice `hachage(str)%len(table)`.

## 3 Notions de performances

L'efficacité de la recherche, de l'ajout et de la suppression dépend grandement du nombre de valeurs contenues dans un tas de la table. Si l'on note $c$ la taille du plus grand tas, alors la recherche, l'ajout et la suppression se font en $O(c)$.

Idéalement, on cherche à avoir une table de hachage très grande avec des valeurs bien réparties dans les différentes cases de manière à avoir une valeur petite pour $c$.

**Remarques :**

- La valeur $c$ dépend directement de la fonction de hachage. Le choix de la fonction de hachage a donc une grande influence sur l'efficacité de la table de hachage.

  Une fonction de hachage "parfaite" répartit équitablement les différentes valeurs dans les tas de la table (même nombre de valeurs dans chaque tas de la table).

- La valeur de $c$ varie au cours des ajouts/suppressions. Au départ, quand la table est vide, elle vaut 0. Dans le pire des cas, elle est égale au nombre de valeurs dans la table (si toutes les valeurs sont rangées dans le même tas).

- La dimension de la table est importante pour l'efficacité. Même si la fonction de hachage est "parfaite", on a $c \geq \frac{n}{k}$ où $n$ est le nombre de valeurs dans la table et $k$ le nombre de cases de la table.

  Il faut donc avoir une table avec suffisamment de cases par rapport au nombre de valeurs, quitte à recopier une table de hachage dans une autre plus grande.