

# TD n°1

## Tables de hachage

### Motivation

Considérons le problème suivant. Une collection de données est organisée en enregistrements. L'un des champs d'un enregistrement s'appelle la clef. On ne connaît pas *a priori* le type de la clef. Par contre, la clef est unique (il existe au plus un enregistrement de clef donnée dans la collection.) On souhaite pouvoir gérer la collection de la façon suivante :

- Rechercher dans la collection un enregistrement de clef donnée (la recherche peut être infructueuse)
- Insérer un nouvel enregistrement dans la collection
- Supprimer de la collection un enregistrement (de clef donnée)

Si la clef est un petit nombre (typiquement si le nombre d'enregistrements est  $\leq n$  et la clef un entier entre 0 et  $n$ ) un tableau fait très bien l'affaire. Mais *quid* si la clef est un champ de 1ko ? De 10Mo ?

### Fonction de hachage

Une fonction de hachage est quelque chose qui sert à rendre la clef plus petite. Si  $E$  est l'ensemble des clefs possibles, pour un  $n$  donné, avec  $n \ll |E|$ , une fonction de hachage est tout simplement une fonction  $h : E \rightarrow [0, n[$ .

Vous en connaissez quelques unes sans doute : MD5 avec  $n = 2^{128}$ , SHA1 avec  $n = 2^{160}$ , ... Une «bonne» fonction de hachage peut prendre en entrée n'importe quelle donnée binaire, et rend un *hash* qui est un nombre (plus ou moins) petit.

### Table de hachage

Étant donnés

- Un type de donné Enregistrement comportant le champ clef de type  $E$
- une borne  $n$  «raisonnable» sur le nombre max d'enregistrements d'une collection
- une fonction de hachage  $h : E \rightarrow [0, n[$

une table de hachage est tout simplement un tableau  $T$  de  $n$  Enregistrements tel que l'enregistrement  $x$  est stocké en

$$T[h(x.clef)]$$

$T[x] = \text{nil}$  si aucun enregistrement n'a la clef  $x$  (la case est vide). On a alors :

- Rechercher la clef  $z$  : return  $T[h(z)]$  (nb : nil signale un échec de la recherche)
- Insérer un enregistrement  $x$  :  $T[h(x.clef)] = x$
- Supprimer un enregistrement de clef  $z$  :  $T[h(z)] = \text{nil}$

### Collisions et hachage parfait

Une *collision* entre deux enregistrements  $x$  et  $y$  arrive quand  $h(x.clef) = h(y.clef) = z$ . Il faut donc les ranger dans  $T[z]$  tous les deux, ce qui n'est pas possible ! Avec l'algorithme d'insertion ci-dessus, si on insère  $x$  puis  $y$ ,  $y$  écrase  $x$  ce qui est un bug... Deux solutions :

- Le *hachage parfait* qui est une fonction de hachage qui ne produit pas de collisions. À part le cas trivial  $h(x) = x$ , écrire des fonctions de hachage parfait est algorithmiquement dur et nécessite de connaître la collection à l'avance (le problème ne peut pas être résolu dynamiquement)
- Reste à prévoir la gestion des collisions. Que faire ? C'est le but de la fin de ce TD que d'y répondre

## Exercice 1. Fonction de hachage

Étant donné un entier strictement positif  $n$ , on considère la fonction de hachage  $h$  qui associe à tout entier un nombre compris entre 0 et  $n - 1$ , définie par :

$$h(x) = x \bmod n$$

1. Soit  $k$  un entier non nul. Quelle est la probabilité pour que deux entiers  $x, y \in [0 \dots, k \times n[$  vérifient  $h(x) = h(y)$  ? (On supposera les nombres  $x$  et  $y$  tirés uniformément dans cet intervalle.)
2. Soit  $p$  un entier positif ou nul et soit  $n = 2^p$ . Quelle est la condition sur l'écriture en binaire de deux entiers  $x$  et  $y$  pour que l'on ait  $h(x) = h(y)$  ?
3. On suppose à présent que  $n = 2^p - 1$ , avec  $p \geq 2$ .
  - (a) Montrer que, pour tout entier positif ou nul  $k$ ,  $2^k \bmod n \neq 0$ .
  - (b) Montrer que, pour tout entier positif  $x$  et pour tout entier positif ou nul  $k$ ,  $(x + 2^k) \bmod n \neq x \bmod n$ .
  - (c) En déduire que si l'entier  $y$  se déduit de  $x$  par changement d'un seul bit de sa représentation en binaire, alors  $h(x) \neq h(y)$ .
  - (d) Montrer aussi que si l'entier  $y$  se déduit de  $x$  par échange d'un 0 et d'un 1 adjacents dans sa représentation en binaire, alors  $h(x) \neq h(y)$ .

## Exercice 2. Gestion des collisions

On considère quatre méthodes pour résoudre les collisions :

1. Chaînage. Toutes les valeurs de même *hash* sont stockées dans une liste chaînée, en mémoire externe (il faut allouer un espace supplémentaire)
2. Chaînage en ABR. Idem que précédemment, mais on organise les données en arbre binaire de recherche (la clef est un nombre)
3. Débordement simple (ou sondage linéaire) : on se donne une constante  $c$ , souvent  $c = 1$ . Pour insérer une valeur  $x$ , avec  $z = h(x.clef)$ , si  $T[z]$  est occupé, on regarde  $T[z + c]$ ,  $T[z + 2c]$ , etc.
4. Débordement quadratique (ou sondage quadratique) : on se donne deux constantes  $c$  et  $d$ . Pour insérer une valeur  $x$ , avec  $z = h(x.clef)$ , si  $T[z]$  est occupé, on regarde  $T[z + c + d]$ ,  $T[z + 2c + 4d]$ , ...  $T[z + ic + i^2d]$  au  $i$ ème essai.
5. Double hachage : on a deux fonctions  $h_1$  et  $h_2$ . Au  $i$ ème essai on essaie la case  $T[h(x.clef, i)]$  avec  $h(k, i) = h_1(k) + ih_2(k) \bmod n$ .
6. Débordement avec pointeur : Dans chaque Enregistrement on a un champ privé *next*. Pour  $z = h(x.clef)$ , si  $T[z]$  est occupé, on cherche la première case vide plus loin,  $T[w]$ . On stocke  $x$  dans  $T[w]$ , et  $T[z].next = w$ . On construit ainsi des listes chaînées *internes* qui se terminent par un *next* à nil.

En vous divisant en six groupes :

- lire la partie du Cormen correspondante
- Discuter de l'implémentation
- Écrire le code de recherche, d'insertion, et de suppression d'une valeur
- faire un exemple pour une table de hachage avec  $n = 11$ .

la fonction  $h$  est le modulo 11 (pour le double hachage :  $h_1(k) = k \bmod n$  et  $h_2(k) = 1 + k \bmod (n - 1)$ )

Les valeurs insérées ont comme clef : 10, 22, 31, 4, 15, 28, 83, 88, 59, 37, puis suppression de la clef 22, suppression de la clef 4 et recherche de la clef 59
- Discuter de la complexité, en temps et espace, de faire  $m$  insertion dans une table de taille  $n$ . Attention dans certains cas, c'est dur !
- Discuter avantages et inconvénients de la méthode

Chaque groupe présentera ses résultats aux autres