Algorithmique
Hachage : Principe et méthodes de bases

1 Principe du hachage

On utilise les méthodes de hachage lorsque l'on doit classer les éléments d'une collection appartenant à un univers très grand. La taille de ce dernier nous empêche de le représenter en réservant une place mémoire par clé possible. On utilise alors une *fonction de hachage* h qui associe à chaque clé un entier compris entre 0 et $m-1$ (ou 1 et m), où m est choisi en fonction de la taille prévue de la collection.

Pour toute clé x de la collection, $h(x)$ (valeur de hachage primaire) donne l'indice de x dans le tableau de hachage. Cela nous permet de le rechercher, l'ajouter ou le supprimer. Le choix de la fonction de hachage est fondamental, celle-ci doit être :

uniforme : c'est à dire que tous les éléments sont répartis le plus uniformément possible dans le tableau,

facile et rapide à calculer : le but étant de ne pas affaiblir la méthode par un calcul long et fastidieux,

déterministe : renvoyer toujours le même résultat.

Si la fonction h est injective, il n'y a aucun problème : chaque élément aura sa place unique dans le tableau. Si ce n'est pas le cas, plusieurs élément peuvent avoir la même valeur de hachage, on parle alors de *collision*. Les méthodes de résolutions des collisions (directes et indirectes) seront vues dans des cours spécifiques.

2 Fonctions de hachage

Important : il n'existe pas de fonction de hachage universelle. Celle-ci doit à chaque fois être adaptée aux données qu'elle hache et à l'application qui gère ces données.

Sont présentés ici quelques principes de constructions de fonctions de hachage modulables entre eux. Les clés sont des mots représentés en mémoire par une suite de bits interprétable comme un entier.

Il faut construire une fonction h des suites de bits dans un intervalle de m entiers : $h \ : \ \{0,1\}^* \to [0, m-1]$

2.1 Extraction

Cette méthode consiste à extraire un certains nombre de bits : p bits. L'intervalle se ramène donc à $[0, 2^p-1]$. Dans le cas où $m = 2p$, le mot obtenu formé des p bits donne directement une valeur d'indice dans le tableau.

Même si cette méthode est facile à mettre en oeuvre, le problème est que l'utilisation partielle de la clé ne donne pas de bons résultats car elle n'utilise pas tous les bits de la représentation. Celle-ci n'est bien adaptée que dans le cas où l'on connaît les données à l'avance ou lorsque certains bits de la codification sont non significatifs.

2.2 Compression

Dans cette méthode, les bits de la représentation sont découpés en sous-mots de longueurs identiques qui sont combinés à l'aide d'opérateurs sur bits. L'utilisation du `ET` ou du `OU` donne des résultats systématiquement plus petits (`ET`) ou plus grand (`OU`). Pour cette raison, il est préférable d'utiliser `OU exclusif`.

Le problème de cette méthode est de "hacher" de la même manière toutes les permutations d'une clé.

Une solution consiste à introduire des décalages (des rotations de bits qui dépendent de la position du caractère). Si cette technique peut supprimer les collisions entre anagrammes, elle peut aussi en créer de nouvelles.

2.3 Division

Ici, la valeur de hachage est tout simplement le reste de la division entière de la clé par m. Cette fonction, facile à calculer peut provoquer des collisions fréquentes si m a de petits diviseurs. Prendre m premier permet de limiter (sans éliminer) les collisions.

2.4 Multiplication

Étant donné un réel θ, tel que $0 < \theta < 1$, $h(e) = \lfloor ((e * \theta) \ \mathbf{mod}\ 1) * m \rfloor$ ($\mathbf{mod}\ 1 \equiv$ partie décimale).

Le choix de θ est important : une valeur trop proche de 0 ou de 1 provoquera des accumulations aux extrémités du tableau.