Algorithmique
Hachage : Résolutions des collisions par calcul

Rappels

Pour toute clé x d'une collection d'éléments, la fonction de hachage h est une fonction uniforme telle que $h(x)$ (valeur de hachage primaire) donne l'indice, compris entre 0 et $m-1$, de x dans le tableau de hachage.

Si plusieurs clés ont la même valeur de hachage, on parle alors de *collision* (primaire).

Résolution des collisions par calcul

Parmi les méthodes de résolution des collisions, il existe les méthodes dites **directes** : par calcul. Les places des éléments en collision sont données grace à une fonction d'essais successifs qui pour chaque x retourne en m essais une permutation des m valeurs prises entre 0 et $m-1$:

$$\begin{array}{rccc} essai & : & E & \rightarrow & \{1,2,\ldots,m\}^m \\ & & x & \mapsto & (essai_1(x), essai_2(x), \ldots, essai_m(x)) \end{array}$$

avec

$\forall\ i \in \{1,\ldots,m\} \quad essai_i(x) \in \{0,\ldots,m-1\}$ et *si* $i \neq j$ *alors* $essai_i(x) \neq essai_j(x)$.

Opérations [1]

Recherche

Lorsque l'on recherche un élément x dans le tableau de hachage, on explore successivement les places correspondantes aux essais successifs jusqu'à ce que l'on trouve x ou une case "vide".

Insertion

On procède comme pour la recherche. Seulement, lorsque l'on tombe sur une case "vide", on ajoute l'élément dedans.

Suppression

La suppression génère un problème de réorganisation des données.

La solution est donc de ne pas supprimer réellement un élément, mais de marquer la case qu'il occupe comme *libre* (solution utilisée pour le hachage coalescent) : chaque case pourra alors être soit *vide* (état initial), *occupée* ou *libre* (a été occupée puis libérée).

Dès lors, il faut adapter les méthodes vues ci-dessus [2].

Pour la **recherche**, il suffit juste de ne pas tenir compte des cases *libres* (on continue la recherche).

Lors du parcours des cases données par les essais successifs dans l'**insertion**, on conserve l'éventuelle première place *libre* rencontrée :
- Si une telle place existe, alors on y insérera tout simplement l'élément (sauf s'il était déjà présent).
- Sinon, on procède à l'insertion "classique" dans la première case libre rencontrée.

Fonctions d'essais

Les différentes méthodes de hachage direct se caractérisent par le choix de la fonction d'essais successifs. Cette dernière associe une suite de m places dans le tableau. Il y a au plus $m!$ suites différentes dans le tableau, mais la plupart des méthodes en utilisent beaucoup moins.

1. Les algorithmes sont donnés en annexes
2. Dans les algorithmes en annexe, les différences sont en bleu.

Hachage linéaire

Dans le hachage linéaire, en cas de collision, on se contente de tester successivement les cases suivantes en remontant à la première case lorsque l'on est arrivé en bas du tableau (à $m-1$). Notons $a \oplus b$ cette opération (incrémentation modulo la taille m), la suite d'essais est donc tout simplement :

$essai_1(x) = h(x)$
$essai_2(x) = h(x) \oplus 1$
...
$essai_i(x) = h(x) \oplus i - 1$
...
$essai_m(x) = h(x) \oplus m - 1$

Remarques :
- Les éléments qui ont la même valeur de hachage primaire ont la même séquence d'essais.
- Plus les groupements sont importants (en taille), plus la probabilité de voir leur taille augmenter est importante : la probabilité qu'un groupement de k éléments sur un tableau de taille m voit sa taille augmenter au prochain ajout est de $(k+2)/m$.

Double hachage

Pour palier au problème de la formation de groupements d'éléments, il faut essayer de disperser un peu plus les éléments en cas de collision.

Nous pouvons alors imaginer la fonction d'essais suivante, où k est un entier fixé :

$$essai_i(x) = h(x) \oplus k(i-1)$$

Pour que la séquence de m essais référence les m valeurs, il faut que k et m soient premiers entre eux. En fait, cela ne réglera pas vraiment le problème d'accumulation d'éléments, mais ceux-ci se produiront de k en k. Pour éviter cela, il faudrait que l'incrément (k) dépende de l'élément x.

On introduit donc une deuxième fonction de hachage d (d'où le nom de **double hachage**). La séquence d'essais devient :

$$essai_i(x) = h(x) \oplus d(x)(i-1)$$

Là encore, il faut que chaque séquence d'essais nous garantisse les m valeurs. Ce qui répond au même critère que précédemment, à savoir que m et $d(x)$ doivent être premiers entre eux quel que soit x. Cela peut s'obtenir des deux façons suivantes :

- m est premier et $d \in [1, m-1]$
- $m = 2^p$ (il est pair) et $d(x)$ doit être impair pour tous les x.
  Ce que l'on peut obtenir en ayant $d(x) = 2d'(x) + 1$, avec $d'(x) \in [0, 2^{p-1} - 1]$.

Conclusion sur les méthodes de hachage

- Toutes les méthodes de hachage sont bien adaptées aux ensembles statiques. Elles sont à peu près équivalentes pour un faible taux de remplissage.
- Le hachage coalescent est la méthode la plus efficace si la zone de mémoire réservée au hachage doit être fixée à l'avance (zone de data statique).
- Le hachage linéaire est le moins rapide, mais cette méthode présente l'avantage d'être extrêmement simple à mettre en oeuvre.
- Le double hachage est plus performant que le hachage linéaire, mais il nécessite le calcul de deux fonctions de hachage (et donc leur écriture).
- A part le chaînage séparé, les méthodes de hachage supportent assez mal les suppressions. La solution qui consiste à utiliser un système de marquage augmente considérablement les temps de recherche.
- La plus performante des méthodes de hachage est incontestablement celle avec chaînage séparé.