Tables de hachage (*hash tables*)

1 Introduction

1.1 Le problème

Il est fréquent de devoir stocker dans une structure de données un ensemble d'élements pour pouvoir contrôler rapidement leur existence, comme les mots d'un dictionnaire, tout en devant mettre à jour cet ensemble fréquement.

C'est le cas par exemple des pages web dans le cache d'un proxy ou des noms de machines dans le cache d'un serveur DNS.

Comme structure de données pour stocker un ensemble d'élement, on connait la liste par exemple, elle est mise à jour très rapidement (il suffit d'ajouter un élément en tête) mais la recherche de l'existence d'un élément est longue puisqu'elle demande en moyenne le parcours de la moitié de la liste. On connait aussi le tableau qui lui sera logiquement trié ce qui donne une recherche en temps $O(\ln(n))$ où n est la taille du tableau mais l'ajout ou la suppression d'un élément sera longue.

1.2 Une solution

Voyons maintenant une structure qui permet à la fois une mise à jour rapide et une recherche rapide en ne gardant que les avantage des deux systèmes. L'idée est que la liste permet une mise à jour rapide mais elle ne doit pas être trop grande pour que la recherche soit rapide aussi : on découpe donc l'ensemble en plusieurs petites listes. La partition se fait selon ce que l'on appelle une *fonction de hachage*. Cette fonction associe à un élément de l'ensemble une valeur entre 0 et $M-1$ qui doit être toujours identique pour un objet identique. On utilise alors un tableau de taille M pour stocker les M listes d'éléments ainsi formées. Ces listes seront de taille moyenne N/M si N est la taille de l'ensemble et si la fonction de hachage est suffisament aléatoire. La recherche d'un élément ne prendra donc qu'un temps en $O(N/M)$ et la mise en place d'un nouvel élément prendra en temps constant. On voit que pour que cela fonctionne correctement N et M doivent être du même ordre de grandeur et la fonction de hachage doit être rapide à calculer.

Remarque 1 *On appelle* collision *le fait que deux éléments différents aient la même valeur par la fonction de hachage.*

Le plus difficile est donc de définir une bonne fonction de hachage, c'est-à-dire une fonction rapide à calculer qui se comporte comme une fonction aléatoire pour limiter au maximun les collisions et donc avoir des listes les plus courtes possibles.

2 Réalisation

On va supposer ici que les enregistrements sont des chaînes de caractères, mais le principe se généralise quel que soit le type des clés. Une fonction de hachage possible est la suivante : soit une chaîne de caractères

$$s = a_0 a_1 \dots a_n$$

et soit c_i le code ASCII du caractère a_i (il est donné par la fonction `int_of_char`). On choisit alors comme clé :

$$k = \sum_{i=0}^{i=n} 128^i . c_i$$

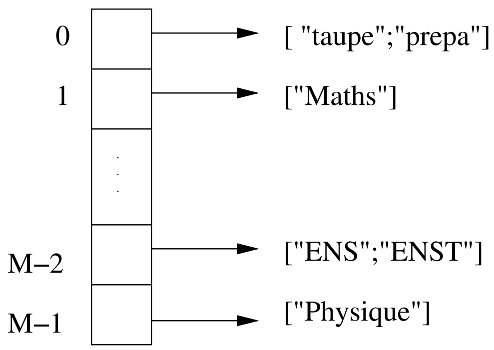
puis comme fonction de hachage :

$$h(k) = k \bmod M$$

Question 1 *Pourquoi est-il souhaitable de prendre M premier ?*

Question 2 *Ecrire une fonction* `hash` *qui implémente cette fonction de hachage sur les chaînes de caractères (pour une valeur de M passée en argument).*

Comme nous l'avons expliqué, une table de hachage de taille M est un tableau de listes de taille M. A chaque indice i est stockée la liste des enregistrements pour lesquels la fonction de hachage vaut i.



Le type des tables de hachage est tout simplement le suivant :

```
type 'a tableh = 'a list vect ;;
```

Question 3 *Ecrire une fonction* `nouvelle : int -> 'a tableh` *donnant une nouvelle table de hachage pour une taille donnée en argument.*

Question 4 *Ecrire une fonction* `ajoute : string tableh -> string -> unit` *ajoutant une chaîne dans une table de hachage.*

Question 5 *Ecrire une fonction* `trouve : string tableh -> string -> bool` *déterminant si une chaîne est présente ou non dans une table de hachage.*

3 Amélioration : tables de hachage de taille dynamique

Si l'on ne connait pas du tout à l'avance la taille de l'ensemble à stocker, on ne peut pas choisir de bonne valeur M, il faut alors faire varier cette valeur dynamiquement. L'idée est d'augmenter la taille du tableau lorsque les listes deviennent trop longues : on garde ainsi des listes courtes dans lesquelles la recherche reste rapide.

Pour cela, on redéfinit le type des tables de hachage de la manière suivante :

```
type 'a tableh = { mutable max     : int ;
                   mutable donnees : 'a list vect } ;;
```

où `max` est la longueur maximale des listes, et `donnees` le tableau. Ces deux champs sont modifiables (`mutable`) car leurs valeurs doivent pouvoir être modifiées.

Question 6 *Ré-écrire les fonctions* `nouvelle`*,* `ajoute` *et* `trouve` *pour les tables de hachage de taille dynamique. On fera les choix suivants : initialement* `max` *vaut 3 puis, lorsque la taille d'une liste dépasse la valeur de* `max`*, la table de hachage est re-dimensionnée. Pour cela la valeur de M est remplacée par $2M+1$ et la valeur de* `max` *par $2\times$*`max`*. On prendra soin de conserver l'ordre des éléments dans les listes au moment du re-dimensionnement.*