

# Tables de hachage

On suppose donnés deux ensembles  $K$  (ensemble des clefs) et  $E$ , ainsi qu'un sous-ensemble  $m$  de  $K \times E$  tel qu'à chaque  $k \in K$  corresponde au plus un couple  $(k, e) \in m$ . Par exemple, les clefs peuvent être des numéros de téléphone et les éléments de  $E$  des noms,  $m$  faisant correspondre un nom d'abonné à chaque numéro.

Une manière simple d'implémenter  $m$  est de construire la liste des couples  $(k, e) \in m$ . Pour retrouver l'élément de  $E$  correspondant à une clef donnée, il suffit alors de faire une recherche sur la liste.

Les *tables de hachage* constituent une implémentation plus efficace de ce problème, alliant les avantages d'un tableau pour l'accès instantané à une donnée, et d'une liste pour la souplesse de l'ajout de nouveaux éléments.

## 1 Principe

L'idée consiste à définir une fonction de hachage  $h_M : K \rightarrow \{0, \dots, M-1\}$  ( $M$  sera appelé la *longueur* de la table de hachage) qui à chaque clef fait correspondre un entier compris entre 0 et  $M-1$ . On utilise ensuite un tableau de  $M$  listes pour stocker les éléments : la liste  $i$  contient les couples  $(k, e)$  tels que  $h_M(k) = i$ .

Pour que le hachage soit efficace, il est préférable que les images des clefs soient équiréparties dans  $\{0, \dots, M-1\}$ .

Dans ce problème, on suppose que les clefs sont des chaînes de caractères : soit

$$s = a_0 \dots a_n$$

une chaîne, et soit  $c_i$  le code ASCII du caractère  $a_i$  (donné par `int_of_char` en Caml). À une telle chaîne on fait correspondre l'entier

$$k = \sum_{i=0}^n 256^i c_i.$$

$s$  peut ainsi être vue comme la représentation en base 256 de l'entier  $k$ .

**Question 1.** *Écrire une fonction `conv : string → int` implémentant ce codage, en temps linéaire en la longueur de la chaîne.*

Une fonction de hachage simple sur les entiers peut être définie par :

$$f_M(k) = k \bmod M.$$

**Question 2.** *Écrire une fonction `hash : int → string → int` qui implémente cette fonction de hachage sur les chaînes de caractères (pour un entier  $M$  passé en argument).*

## 2 Tables de hachage de taille fixe

On définit le type des tables de hachage de clefs de type 'a vers des valeurs de type 'b de la façon suivante :

```
type ('a,'b) tableh = { hash : 'a → int ;  
                        donnees : ('a * 'b) list vect };;
```

Soit  $m$  une table de hachage,  $m.hash$  est la fonction de hachage  $h_M$  et  $m.donnees$  est un tableau de longueur  $M$ . La  $i$ -ième case de ce tableau contient la liste des couples  $(k, e)$  tels que  $h_M(k) = i$ .

**Question 3.** *Écrire une fonction `creer : ('a → int) → int → ('a * 'b) tableh` telle que `creer h M` renvoie une nouvelle table de hachage vide de longueur  $M$ .*

**Question 4.** *Écrire une fonction `ajoute : ('a * 'b) tableh → 'a → 'b → unit()` telle que `ajoute m k e` ajoute l'entrée  $(k, e)$  à la table  $m$ . On n'oubliera pas le cas où la clef  $k$  est déjà présente dans la table.*

**Question 5.** *Écrire une fonction `trouve : ('a * 'b) tableh → 'a → 'b` telle que `trouve m k` renvoie l'élément  $e$  tel que  $(k, e) \in m$  s'il existe, et lève une exception sinon.*

## 3 Tables de hachage de taille dynamique

La partie coûteuse de la recherche d'une clef dans une table de hachage est la recherche dans la liste. On veut pouvoir garder des listes courtes (en moyenne), et donc augmenter la longueur du tableau lorsqu'il y a trop d'éléments. Pour cela, on redéfinit le type de la table de hachage :

```
type ('a,'b) dyn_tableh = { hash : int → 'a → int ;  
                           mutable taille : int ;  
                           mutable donnees : ('a * 'b) list vect };;
```

Le nouveau champ `taille` permet de stocker le nombre d'entrées de la table. La fonction de hachage prend à présent la longueur  $M$  de la table en argument.

**Question 6.** *Réécrire la fonction de création `dyn_creer`.*

On utilisera le principe de redimensionnement suivant : à l'ajout d'un élément, si le nombre total d'éléments `taille` est supérieur au double de la longueur courante  $M$  de la table, alors on réarrange la table sur une nouvelle longueur de  $2M + 1$  avant de procéder à l'ajout.

**Question 7.** *Écrire une fonction `dyn_rearrange : ('a * 'b) dyn_tableh → unit()` qui réarrange une table comme décrit ci-dessus (i.e., en faisant passer sa longueur de  $M$  à  $2M + 1$ ).*

**Question 8.** *En déduire les nouvelles fonctions `dyn_ajoute` et `dyn_trouve`.*

**Question 9.** *Évaluer la complexité des algorithmes d'insertion et de recherche dans une table de hachage, en fonction du nombre total d'éléments de la table et de sa longueur. On supposera que la fonction de hachage s'évalue en  $O(1)$  opérations élémentaires. Quelle est la complexité de l'algorithme de recherche si l'on suppose adopté le principe de redimensionnement ci-dessus ?*