https://www.laurentbloch.net/BlogLB/Combiner-vecteurs-et-listes

Combiner vecteurs et listes associatives : les hash tables

- Zinformatiques - Cours de bioinformatique au CNAM -

Date de mise en ligne : vendredi 17 décembre 2004

Sommaire

Pour illustrer l'usage des vecteurs par leur application à la construction de tables associatives *(hash-tables)*, nous prendrons comme exemple la réalisation d'un programme d'annuaire électronique simplifié.

Notre programme d'annuaire devra répondre aux messages suivants :

– pour introduire des données dans l'annuaire :

```
 1:=> (cet-annuaire 'peupler)
Entrez nom et numéro : "Edgar" 87
Entrez nom et numéro : "Toto" 12
Entrez nom et numéro : "Lili" 28
Entrez nom et numéro : "Eloïse" 45
Entrez nom et numéro : "" 0
#f
```

Attention : les chaînes de caractères doivent être tapées entre guillemets.

– pour interroger l'annuaire :

```
 1:=> (cet-annuaire 'interrogation "Toto")
12
1:=> (cet-annuaire 'interrogation "Albert")
Pas d'abonné au numéro demandé
1:=> (cet-annuaire 'interrogation "Eloïse")
45
```

Pour améliorer le temps de réponse aux interrogations, ce programme gardera en mémoire l'ensemble de l'annuaire. La première idée qui vient à l'esprit sera de construire une liste d'associations : chaque personne identifiée dans l'annuaire sera représentée par un doublet dont le `car` sera une chaîne de caractères donnant son nom et le `cdr` son numéro de téléphone.

Ce programme naïf donnerait peut-être satisfaction si l'annuaire est petit, mais si les effectifs sont importants l'usage direct des listes d'associations aura vite deux inconvénients :

* à chaque ajout d'un nom dans l'annuaire, une nouvelle liste est créée par l'appel de `cons`, ce qui sera vite insupportable ;

* la consultation par `assoc` parcourt séquentiellement l'annuaire à chaque fois, ce qui va finir par prendre trop de temps, plus précisément le temps moyen d'accès à un élément de l'annuaire est une fonction linéaire de la taille de la liste.

Comment surmonter ces inconvénients ? L'adressage associatif

Soit pour représenter notre annuaire un vecteur de taille *n*. Pour insérer à sa place chaque doublet représentatif d'un couple nom-numéro, nous allons essayer de construire une fonction *f* qui à un nom fasse correspondre un entier compris entre 0 et *n*-1. Le nom est utilisé comme clé d'accès à la table. Pour une valeur c de la clé, l'entrée de l'annuaire sera rangée dans l'élément du vecteur de rang *i = f(c)* (on dit aussi alvéole).

Cette fonction permettra également de retrouver le doublet à chaque interrogation, et ce en un laps de temps constant, égal au temps de calcul de la fonction plus le temps d'accès à un élément de vecteur.

La fonction *f* s'appelle la fonction d'association, ou d'adressage dispersé, ou de dispersion, ou de hachage, ou de *hash-coding*.

L'idéal serait que la fonction *f* soit injective, c'est-à-dire que chaque valeur de *i* corresponde à une seule valeur possible de *c*. C'est malheureusement irréalisable. Même si on imagine pouvoir inventer une fonction telle que deux clés différentes ne puissent pas donner une même valeur de *i*, un exercice difficile et même insoluble dans le cas général, il faudrait avoir un vecteur dont la taille serait au moins égale au nombre de clés possibles, qui est immense.

Plusieurs valeurs de *c* peuvent donc donner par *f* la même valeur de *i*, c'est ce que l'on appelle une collision, ou une synonymie. En cas de collision, un doublet va vouloir se ranger dans un élément de vecteur déjà occupé. Afin de faire face à cette situation inévitable, chaque élément de vecteur ne sera pas un doublet, mais une liste associative.

Nous allons choisir une fonction *f* telle que chaque indice i corresponde à un ensemble de clés *ci1, ci2, ... cij ..., cip* d'effectifs sensiblement voisins ; s'il y a *N* individus dans notre annuaire, la longueur de chaque liste sera de l'ordre de *N/n*. Le choix judicieux de *n* et de *f* devrait assurer une efficacité raisonnable à l'algorithme.

[https://www.laurentbloch.net/BlogLB/IMG/gif/hash.gif]
Adressage associatif

Une façon simple d'associer un nombre à un nom, c'est d'aditionner les codes ASCII des caractères du nom :

```
 (apply + (map char->integer (string->list "Aéa")))
395
```

Ce nombre n'est pas compris entre 0 et *n*-1, mais le reste de sa division par *n* l'est :

```
 (remainder (apply + (map char->integer (string->list "Aéa")))  23)
4
```

Voici donc la fonction d'association que nous utiliserons :

```
(define (hash nom taille-table)
   (remainder
     (apply + (map char->integer (string->list nom)))
     taille-table))
```

et le programme de création de l'annuaire :

```
(define (make-annuaire)
 (let* ((n 23) ; nombre d'éléments du vecteur
        (un-annuaire (make-vector n '())))
   (lambda message
     (case (car message)
       ((peupler)
         (let boucle ()
            (display "Entrez nom et numéro : ")
            (let* ((le-nom (read))
                   (numero (read)))
               (if (not (string=? le-nom ""))
                   (begin
                      (vec!:ajouter le-nom
                                    numero
                                    un-annuaire)
                      (boucle))))))

       ((interrogation)
         (let* ((le-nom (cadr message))
                (la-case (hash le-nom n))
                (reponse
                   (assoc le-nom (vector-ref
                            un-annuaire la-case))))
           (if reponse
              (cdr reponse)
              "Pas d'abonné au numéro demandé")))
       ((donner)
          un-annuaire)
       ((afficher)
        (vector:for-each print un-annuaire))))))
```

et quelques procédures auxiliaires :

```
 (define (vector:for-each proc V)
  (let ((longueur (vector-length V)))
    (let boucle ((index 0))
       (if (not (= index longueur))
          (begin
             (proc (vector-ref V index))
             (boucle (+ index 1)))))))

(define (assoc!:ajouter nom numero liste)
  (cons (cons nom numero) liste))

(define (vec!:ajouter nom numero vecteur)
  (let* ((n (vector-length vecteur))
         (une-case (hash nom n))
         (elem (vector-ref vecteur une-case)))
    (vector-set! vecteur une-case
                 (assoc!:ajouter nom numero elem))))
```