TD8

Programmation en C (LC4)

Semaine du 25 mars 2013

1 Tables de hachage

En programmation, on est souvent amené à utiliser des listes d'associations. Une liste d'association associe des valeurs à des indices. Par exemple, si on implémente un annuaire téléphonique, on veut pouvoir retrouver rapidement le numéro de téléphone (la valeur) correspondant à un nom donné (l'indice). Si les indices sont des entiers, on peut utiliser des tableaux pour implémenter nos liste d'association. Le gros avantages est alors que l'accès à une valeur se fait en temps constant. Par contre, non seulement on est limité à avoir des indices entier, mais en plus, si les indices sont pris dans un ensemble très grand, on est obliger d'allouer un énorme tableau (songer à un annuaire inversé). Les tables de hachages sont une structure de donnée associant des valeurs à des indices sans les inconvénients des tableaux, mais permettant tout de même un accès en temps constant en moyenne.

Dans la suite de cet exercice, on utilisera l'ensemble des chaînes de caractères comme indice de case. Pour cela, on va d'abord calculer à partir de la chaîne qui nous sert d'indice (appelée « clé »), un « haché » qui sera le véritable indice de la case dans un tableau (qui, lui, est de taille raisonnable). Évidemment, la fonction qui calcule le haché ne peut être injective (plusieurs clés différentes peuvent produire le même haché), et on utilisera non pas un tableau donnant directement les valeurs associées aux clés mais un tableau de listes de couples (clé, valeur) où chaque liste du tableau correspond à un haché différent. Si la fonction qui calcule le haché est bien construite et que le tableau est suffisamment grand, les listes associées aux mêmes hachés seront courtes et les opérations d'insertion, de recherche et de suppression d'un élément seront efficaces.

On travaille avec les types suivants :

```
typedef struct liste_s {
    char *cle;
    int valeur;
    struct liste_s *suivant;
} *liste_t;

typedef struct table_de_hachage_s {
    int taille;
    liste_t *tableau;
} *table_de_hachage_t;
```

La figure 1 montre une table de hachage représentant un répertoire téléphonique (à un nom correspond un numéro de téléphone).

Exercice 1 Écrire une fonction `table_de_hachage_t cree_table_de_hachage(int taille)` qui crée une table de hachage vide avec un tableau de la taille indiquée (mais qui ne contient que des listes vides puisqu'il n'y a pas encore d'élément dans la table de hachage).

Exercice 2 Écrire une fonction `void detruit_table_de_hachage(table_de_hachage_t table)` qui libère la mémoire occupée par une table de hachage donnée en argument.

Exercice 3 Écrire une fonction `int hachage(table_de_hachage_t table, char *cle)` qui calcule le haché d'une clé, c'est-à-dire l'indice du tableau de listes de la table de hachage que l'on doit utiliser pour placer les valeurs associées à la clé donnée en argument : pour cet exercice, le haché sera simplement la somme des valeurs associées aux caractères de la clé modulo la taille du tableau.

Exercice 4 Écrire une fonction `void insere(table_de_hachage_t table, char *cle, int valeur)` qui insère dans la table de hachage la valeur associée à la clé donnée en argument.

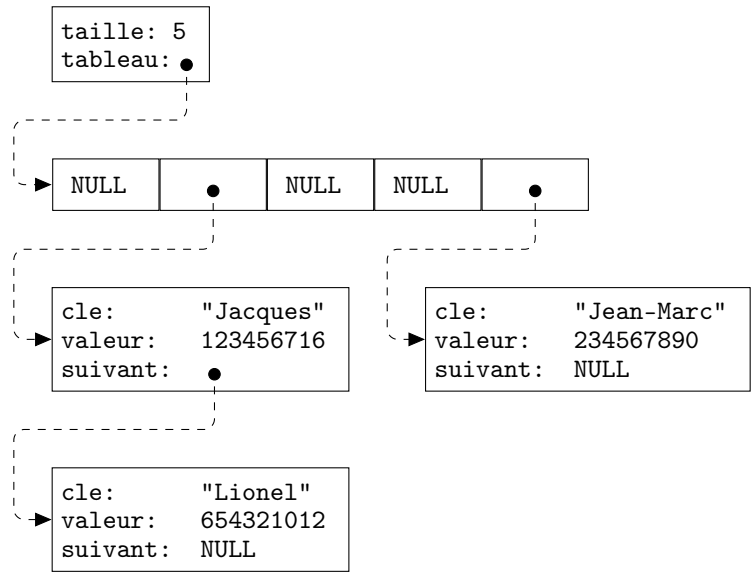

FIGURE 1 – Une table de hachage (`"Jacques"` et `"Lionel"` on le même haché)

Exercice 5 Écrire une fonction `int recherche(table_de_hachage_t table, char *cle, int *valeur)` qui cherche si une entrée de la table a pour clé celle indiquée en argument, et le cas échéant renvoie la valeur correspondante via le pointeur donné en argument. Cette fonction renvoie un entier qui indique si la recherche a abouti.

Exercice 6 Écrire une fonction `void supprime(table_de_hachage_t table, char *cle)` qui efface l'éventuelle entrée de la table ayant pour clé celle donnée en argument (et on suppose qu'il y a au plus une seul telle entrée dans la table).

Exercice 7 Écrire une fonction `void affiche_numero(int numero)` qui affiche un numéro de téléphone donné sous la forme d'un entier : ainsi si `numero = 564302120`, la fonction devra afficher `05.64.30.21.20` (Pourquoi doit on omettre le `0` du préfixe téléphonique ?). Écrire dans la fonction `main()` une séquence d'opérations qui crée une table de hachage représentant un répertoire téléphonique, insère plusieurs entrées dans la table (noms et numéros), effectue une recherche puis supprime l'élément trouvé, fait à nouveau la même recherche et enfin détruit la table.

Nous allons maintenant écrire quelques fonction pour nous permettre d'utiliser ces tables de hachage. Dans un premier temps, le programme lira sur l'entrée standard des couples noms-numeros et les stockera dans une table de hachage, puis dans un second temps lira des noms, et affichera si ce nom est dans l'annuaire (i.e. la table de hachage) son numero de telephone.

Exercice 8 À l'aide de la fonction `scanf`, écrire une fonction `liste_t lis_numero()` qui lit sur l'entrée standard un non puis un numero (représenté par un entier en base 10), et renvoie un `liste_t` ne contenant que ce numero.

Exercice 9 Modifier `lis_numero` pour qu'elle puisse lire des numeros de la forme `XX.XX.XX.XX.XX`.

Exercice 10 Écrire une fonction `void affiche_table(table_de_hachage_t table)` qui affiche tous les couples clé-valeur présent dans la table de hachage passée en argument.

Exercice 11 À l'aide des fonctions précédentes, écrire une fonction `main` qui lit sur l'entrée standard un entier n, puis lit n lignes contenant chacune un nom et un numero, et les stocke dans une table de hachage de taille $3n$ (afin d'éviter les collisions) puis affiche la table de hachage.

Exercice 12 Modifier la fonction `main` pour qu'après avoir lu les numéros, elle lise des noms et affiche si ce nom a une entrée dans la table de hachage, le numero associé à ce nom, sinon un message d'erreur.