Table de hachage

Ensimag 1A - Préparation au Projet C

Année scolaire 2011 – 2012

1 Présentation

Une table de hachage est une structure de stockage dont la manipulation (insertion, recherche, suppression) est en moyenne peu coûteuse : en O(1).

Pour ce faire, on applique une fonction de hachage aux données à stocker puis on travaille avec cet index plutot qu'avec la clé. L'espace des index étant borné, on doit gèrer les collisions, c'est-à-dire le cas de deux clés distinctes donnant le même index. On va résoudre ce problème grâce à la structuration de la table de hachage, en stockant toutes les entrées ayant le même index dans une liste chaînée. La figure 1 [1] illustre une table de hachage.

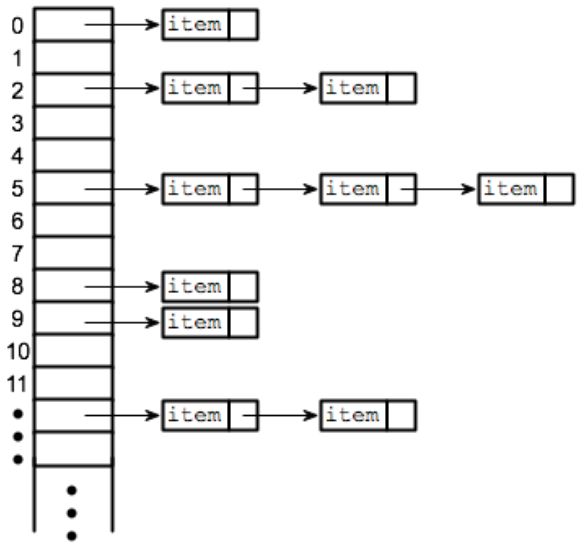

FIGURE 1 – Illustration du fonctionnement d'une table de hachage

1.1 Organisation du code.

Vous implanterez la table de hachage sous forme d'un module (*.o*) dont vous exporterez les fonctions dans un fichier *.h*.

Vous utiliserez votre module dans un programme qui implantera tous les tests nécessaire pour valider le bon fonctionnement de votre table de hachage.

Votre code compilera sans erreur (ni warning) par un simple appel à *make*.

1. Figure tirée du livre en ligne "Introduction to Programming Using Java, Sixth Edition" par David J. Eck

1.2 Objectifs

Votre table de hachage servira à stocker un annuaire dont les clés seront composées du nom et qui copiera nom et numéro dans votre table.

2 Fonction de hachage

Implantez la fonction de hachage donnée par l'algorithme 1.

Algorithm 1 Fonction de hachage

Require: a string str
Ensure: an unsigned int corresponding to the hash value of the key.
  $hash \leftarrow 5381$
  $c \leftarrow$ first character of str
  while $c \neq$ '\0' **do**
    $hash \leftarrow hash$ * 33 + c
    $c \leftarrow$ next character of str
  end while
  return $hash$

Rappel : en C, une chaîne de caractères (string) est une suite de caractères se terminant par le caractère '\0'. La librairie standard du langage C vous fournit diverses fonctions via le fichier *string.h*
Pour de meilleurs performances, on pourra aussi noter que *33 = 32 + 1*.

3 Structuration de la table

Concrètement, pour trouver un contact : on calcule son hash en appelant la fonction de hachage sur la clé (dans notre cas, le nom), on réduit le hash modulo la taille du tableau pour trouver l'index et finalement on accède à la liste chaînée de l'index. La liste contient donc tous les contacts dont l'index est identique.

Voici la représentation C d'un contact :

```
typedef struct _contact contact;

struct _contact {
        unsigned char *nom;    /* le nom du contact, sous forme de chaine de caracteres */
        unsigned char *numero; /* le numero du contact, sous forme de chaine de caracteres */
};
```

Vous définirez la notion de contact au sein même de votre librairie.

Implantez les structures nécessaire à votre annuaire ainsi qu'une fonction d'initialisation (par défaut on souhaite 10 cases dans le tableau).

4 Utilisation de l'annuaire

L'utilisateur de votre librairie doit pouvoir insérer, supprimer et rechercher un contact dans votre annuaire.
L'insertion nécessite que l'utilisateur vous fournisse un contact, que vous devrez dupliquer dans votre annuaire. La suppression n'a besoin que du nom du contact pour l'effacer. La recherche remplira le contact passé en paramètre (dont le nom sera renseigné).

Implantez ces trois fonctionnalitées.
L'utilisateur devra être prévenu du bon déroulement (ou non) de sa commande.

5 Efficacité d'une table de hachage

Lorsque le nombre de contact grandit, on augmente le risque de collision et donc la longueur des listes. Inversement, si le nombre de contact diminue, la table va utiliser inutilement de la mémoire.
On souhaite donc disposer de fonctions (interne à votre librairie) permettant le redimensionnement de la table afin de minimiser le temps de recherche au sein d'une liste ou de réduire l'empreinte mémoire d'une table majoritairement vide.
Ces fonctions seront appelées lorsque le tableau sera rempli à plus de 75% ou à moins de 15% (sans toutefois passer en dessous de 10 cases).

Implantez ces deux fonctions et modifiez les fonctions d'insertions et de suppressions.

6 Pour aller plus loin

Utilisez désormais votre code via une librairie plutôt que sous forme d'un module.