Table de hachage

Ensimag 1A – Préparation au Projet C

1 Présentation

Une table de hachage est une structure de données permettant d'implémenter efficacement des dictionnaires. Sous des hypothèses raisonnables, les opérations principales (insertion, recherche, suppression) sont en moyenne à coût constant $O(1)$.

La structure est en fait un tableau T de m listes chaînées (figure 1 [1]). Lors de l'insertion d'un couple <cle,valeur>, une *fonction de hachage* est appliquée à la clé, qui retourne une valeur entière h(cle). L'insertion a alors lieu dans la liste chaînée d'index h(cle). Chaque liste T[k] contient donc tous les éléments dits en collision, c'est-à-dire dont la valeur de hachage h(cle) est k.

L'espace des indexes étant généralement plus petit que celui des valeurs de hachage, l'insertion a en fait lieu dans la liste en h(cle) modulo m. Les clés sont supposées uniques : une valeur n'est insérée que si elle n'est pas déjà présente dans la liste. Dans le cas général, aucun ordre n'est maintenu sur les listes de collision.

Les performances d'une table de hachage dépendent très fortement de la fonction de hachage. Dans le pire des cas, tous les éléments sont dans la même liste de collision et les coûts sont identiques à une liste chaînée simple. Une bonne fonction de hachage répartit au contraire les éléments de manière quasi-uniforme dans les listes de la table, tout en gardant un coût de calcul raisonnable.

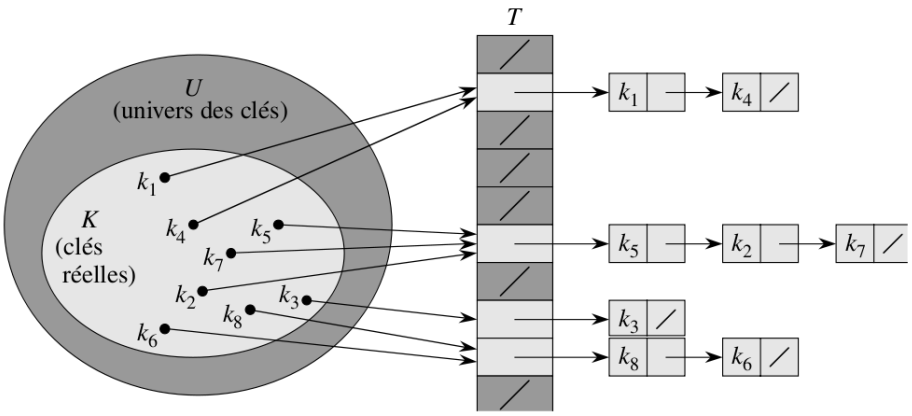

FIGURE 1 – Structure d'une table de hachage T. Chaque liste chaînée en T[k] contient tous les éléments dont la valeur de hachage h(cle) est k. Par exemple h(k1) = h(k4) et h(k5) = h(k2) = h(k7).

1. figure tirée de "Introduction to algorithms, second edition", Cormen et al., The MIT Press.

2 TP

Le travail demandé consiste à utiliser une table de hachage pour implémenter un annuaire associant des numéros de téléphone à des noms (les clés).

2.1 Données manipulées

Les données manipulées sont des noms et numéros de téléphone représentés comme des chaînes de caractères. En C, une chaîne est une suite de caractères se terminant par le caractère '\0'. La librairie standard du langage C fournit diverses fonctions de manipulation de chaînes de caractères, déclarées dans le fichier string.h.

2.2 Fonction de hachage

La fonction de hachage des noms est donnée par l'algorithme suivant [2] :

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 $hash$

2.3 Annuaire

Vous devez d'abord définir une structure d'annuaire **struct** annuaire comme un tableau de listes chaînées contenant des couples <nom, numero>. **Vous devez respecter scrupuleusement la spécification ci-dessous (pas de modification des prototypes)**. Les opérations demandées sur la structure d'annuaire sont :

— la création d'un annuaire initialement vide :

```
struct annuaire *creer();
```

— l'insertion d'un couple <nom, numero> :

```
char *inserer(struct annuaire *an, const char *nom, const char *numero);
```

Si l'annuaire ne contenait aucune entrée de clé nom, alors une nouvelle entrée associant nom à numero est ajoutée et la fonction retourne NULL. Important : les chaînes stockées dans la table sont des *copies* des chaînes nom et numero passées en paramètres [3]. Sinon, la valeur associée à nom est remplacée par le paramètre numero, et l'ancien numéro est retourné (attention, la chaîne retournée est à libérer par l'utilisateur !)

2. Fonction de hachage classique et efficace, introduite par Daniel J. Bernstein. Noter l'utilisation des valeurs 5381 (nombre premier) et 33 (= 32 + 1)

3. C'est un choix de spécification : copie profonde vs copie légère. Dans d'autres contextes on garde les adresses de chaînes existantes par ailleurs.

Note : nous introduisons ici le qualifieur **const** dans le type **const char** $*$, qui spécifie que le pointeur passé en paramètre de la fonction pointe sur des données utilisées en lecture seule, qui ne seront pas modifiées : la fonction insérer spécifiée n'écrira pas par effet de bord dans la mémoire de la chaîne de caractère qui lui est passée. Le respect de cette règle est vérifié par le compilateur lorsque vous écrivez le code de la fonction.

— la recherche d'un numéro :

```
const char *rechercher_numero(struct annuaire *an, const char *nom);
```

Retourne une copie du numéro associé à la clé nom si elle est présente dans l'annuaire, NULL sinon. De même que précédemment, **const char** $*$ pour le type de retour spécifie que l'on interdit à l'utilisateur de rechercher_numero d'écrire dans la chaîne de caractère renvoyée ; cela permet d'éviter un écrasement par l'utilisateur des chaînes de caractères stockées en interne dans la table de hachage.

— la suppression d'une entrée :

```
void supprimer(struct annuaire *an, const char *nom);
```

Supprime l'entrée de clé nom si elle est presente, sans effet sinon.

— la libération de toutes les entrées d'un annuaire puis de l'annuaire lui-même :

```
void liberer(struct annuaire *an);
```

En précondition pour toutes les procédures, le paramètre an est l'adresse valide d'un annuaire déjà existant, et les chaînes sont supposées non nulles.

Dans cette 1ère étape, on souhaite donc implémenter les structures nécessaires et les 4 fonctionnalités décrites. Pour les tests, on considérera l'exemple d'une table de hachage comportant 10 listes chainées.

2.4 Efficacité d'une table de hachage

Lorsque le nombre d'entrées grandit, le risque de collisions augmente et donc la longueur des listes et le coût des opérations. Inversement, si le nombre d'entrées diminue, la table va utiliser inutilement de la mémoire.

Dans cette 2ème étape, on souhaite donc disposer de fonctions (internes à votre librairie) permettant le redimensionnement de la table afin de minimiser les coûts 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).

2.5 Organisation du code.

La table de hachage devra être implémentée sous forme d'un module (*.o*) dont les fonctions sont exportées dans un fichier *.h*.

Ce module sera utilisé dans un programme qui implantera tous les tests nécessaires pour valider le bon fonctionnement de votre table de hachage.

Le code devra compiler sans erreur ni warning par un simple appel à *make*.