

CYCLE L (L2 2013)

STRUCTURES DE DONNEES TP N°4

But: Tables de hachage.

Sujet : construction et réglage d'une table de hachage ★★★★★

Ce sujet fait suite au TP 3 et le complète. L'énoncé est plus long, mais pas le temps de réalisation.

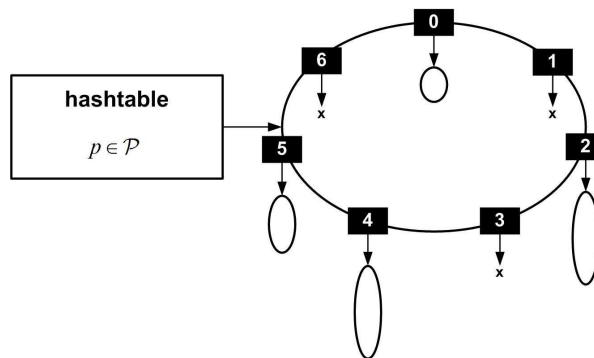
Rappel : Soit un alphabet $\Sigma_n = \{a_0 \dots a_{n-1}\}$, et soit le langage fini P_Σ constitué de l'ensemble des anagrammes du mot $w = a_0 \dots a_{n-1}$ alors P_Σ est isomorphe au groupe symétrique $\mathfrak{S}_n$.

Première partie : la structure (impératif)

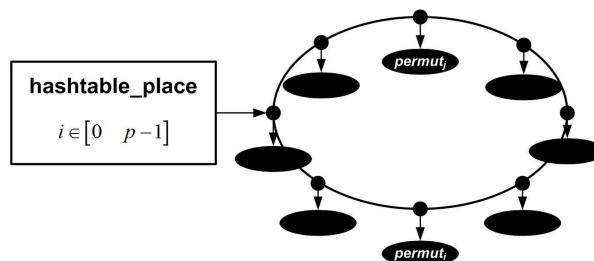
Commençons par le « squelette » de la table de hachage, que nous étoffons par étapes ci-dessous.

Il s'agit de construire une structure hashtable qui contient une liste chaînée circulaire de listes chaînées circulaires.

Chaque maillon de la liste chaînée circulaire de premier niveau représente une **place de la table de hachage**. Ce maillon comporte un index unique pris dans $[0 \ p-1]$ qui correspond à l'une des valeurs possibles retournées par la **fonction de hachage**. Deux maillons directement successifs contiennent deux valeurs d'index directement successives selon l'ordre croissant (sauf pour le cas du plus grand élément suivi directement du plus petit élément).



Chaque maillon d'une liste chaînée circulaire de second niveau contient une instance de la structure **permut** que vous avez définie au TP3 Part 2. Les maillons de cette liste de second niveau sont ordonnés selon l'ordre lexicographique sur les identifiants (anagrammes de P_Σ) des objets **permut**.



La structure principale comporte un paramètre p qui représente la taille de la liste circulaire de premier niveau et qui doit nécessairement être un nombre premier.

Seconde partie : les fonctionnalités de base (impératif)

Réaliser les fonctionnalités suivantes (les prototypes ci-dessous sont conseillés mais pas impératifs) :

Initialisation de la table avec un paramètre *p* : `hashtable *init_table(unsigned p);`

Instancie la structure avec sa liste circulaire de premier niveau dont les listes de second niveau sont initialisées à NULL.

Stockage d'un élément dans la table : `char *put(hashtable *ht, permut *s);`

Stocke s dans la table ht, en initialisant au besoin la liste circulaire de second niveau de destination, et dans tous les cas en insérant dans celle-ci un nouveau maillon à l'endroit indiqué par l'ordre lexicographique (voir plus haut). Retourne l'identifiant de s.

Extraction d'un élément de la table : `permut *get(hashtable *ht, char *id);`

Renvoie l'élément désigné par id, ou NULL si celui-ci n'est pas dans la table.

Retrait d'un élément de la table : `permut *rmv(hashtable *ht, char *id);`

Renvoie l'élément désigné par id après avoir supprimé son emplacement de la table, ou NULL si celui-ci n'est pas dans la table.

Application de la fonction de hachage :

`unsigned *apply_h(hashtable *ht, char *id);`

C'est votre fonction de base pour hacher : elle réalise l'interprétation numérique de id son hachage puis son modulo par le paramètre p de la table. Elle est appelée par les fonctions put, get, remove et rehash.

Troisième partie : jouer dynamiquement avec le hachage (option TD)

Réaliser plusieurs fonctions de hachage en vous inspirant de la littérature abondante à ce sujet.

Vos fonctions calculent une clé à partir de la version numérique *nid* de l'identifiant *id* des anagrammes, et elles doivent toutes partager la format commun suivant (seul leur nom doit les distinguer) : `unsigned <nom_fonction>(unsigned nid);`

Vous allez à présent ajouter un nouveau champ *h* de type `unsigned (*)(unsigned)` à votre structure `hashtable` qui est un pointeur sur la fonction de hachage à utiliser et initialement NULL.

Coup de pouce – `typedef unsigned (*hash_func)(unsigned);`

Pour configurer cette fonction, prévoyez deux fonctionnalités additionnelles pour manipuler votre table :

`unsigned (*)(unsigned) get_h(hashtable *ht);`

`void set_h(hashtable *ht, unsigned (*h)(unsigned));`

Modifiez la fonction `apply_h` vue plus haut de telle sorte qu'elle appelle préalablement la fonction représentée par le champ *h* de la table, avant d'effectuer un modulo *p* du résultat.

Si *h* est NULL, `apply_h` n'effectue qu'un simple modulo *p* du *nid*.

Si vous n'êtes pas à l'aise avec les pointeurs sur fonctions, vous pourrez trouver facilement quelques ressources sur Internet qui traitent de ce sujet.

Quatrième partie : analyse, réglages et ré-hachage (option ED)

Etudier les deux conditions d'uniformité et de densité et leur lien avec les décisions respectivement de changer de fonction de hachage et de ré-hacher la table.

Ajouter aux maillons de la chaîne circulaire de premier niveau (les places de la table), l'indicateur du nombre d'élément stockés (à maintenir à jour).

Ajouter le même indicateur (consolidé pour l'ensemble des places) à la structure racine qui représente la table, et compléter avec les deux indicateurs de la moyenne et de l'écart-type des nombres d'éléments stockés dans les différentes places.

Réaliser la fonction, prévoir les conditions de son déclenchement, l'intégrer dans votre existant :

```
void rehash_table(hashtable *ht, unsigned p);
```

Cette fonction reconstruit la table sur la base d'un nouveau paramètre p et y réorganise le stockage des éléments existants. PENSEZ PERFORMANCE !!

Cinquième partie : étude comparée de performances (option ABO)

Etudier les performances respectives en termes de moyenne et d'écart type, sur l'ensemble des échantillons, en phase de montée en charge (régime transitoire) puis après saturation de la création de contenus (régime permanent) de votre structure de stockage du TP 3 Part. 2 et de plusieurs configurations de votre table de hachage (config = fonction de hachage, condition de ré-hachage et paramètre p).

Les critères de performance sont d'abord le temps moyen d'accès aux objets stockés, puis tous ceux que vous pourriez estimer pertinents.

Pour réaliser cette partie, il est nécessaire de concevoir un protocole de tests de performances, notamment en « stressant » vos structures avec une avalanche de permutations.

Conseil – en réutilisant astucieusement le procédé de Joseph mis au point lors du TP 1, vous pouvez rapidement engendrer (pseudo)-aléatoirement de nombreuses permutations dans le cadre du protocole.

Produisez, si vous en êtes arrivé(e) là (et dans le but de cueillir vos lauriers) un tableau Excel qui met en évidence vos résultats sous forme de courbes comparées / t.

Livraison

Pour chaque binôme, remise des travaux (une seule par binôme) sous 144 h après la fin de la séance de TP (dans les 24 h suivantes, pénalité de trois points, au-delà, c'est zéro).

Envoi par email à franck.lepoivre@gmail.com, en précisant « TP SDD 4 – <NOM 1> – <NOM 2> » dans l'objet.

Forme et contenu du livrable : une archive (RAR), de nom « TP SDD 4 – <NOM 1> – <NOM 2> » contenant votre code source et un exécutable (Win32) qui produit la trace suivante :

```
Equipe
<Prénom1> <NOM1>
<Prénom2> <NOM2>
Groupe <Lettre Groupe>
TP 4 – Hashtable
<Vos traces pertinentes>
```