TD – Tables de hachage

Nous avons étudié précédemment deux types de conteneur : (i) les tableaux et (ii) les éléments chainés. Les premiers sont dits à *accès direct* : on peut aisément et rapidement retrouver un élément à partir de sa position. Les seconds sont dits à *accès séquentiel* : accéder un élément de position donnée nécessite d'avoir accédé tous les éléments de position inférieure. Dans les deux cas cependant, la recherche d'un élément dans le conteneur est de complexité O(n) : il faut parcourir les éléments les uns après les autres, en accès direct ou séquentiel, jusqu'à trouver, ou pas, celui recherché.

Nous allons développer et étudier dans ce TD un nouveau type de conteneur : les tables de hachage (hash tables). Elles ont la propriété importante d'avoir une complexité de recherche constante, i.e. O(1), tout en maintenant une complexité d'ajout et de retrait constante également.

Table des matières

1 Principe

Le principe d'une table de hachage est simple : (i) sa structure de base est un tableau, contenant les éléments stockés, (ii) l'indice d'un élément dans ce tableau est calculé par une fonction, dite de hachage, à partir de l'élément lui-même. C'est à cet indice qu'on insère l'élément, qu'on va le chercher pour déterminer s'il est présent ou qu'on va le chercher pour le retirer.

La principale qualité d'une fonction de hachage est de retourner des indices très dispersés, de façon à ce que les éléments occupent – idéalement – des cellules distinctes du tableau. Il est cependant possible que la fonction retourne le même indice pour deux éléments distincts : c'est ce qu'on appelle une *collision*. Pour prendre ne compte cette possibilité, une cellule du tableau ne contient pas un seul élément, mais une *liste* d'éléments ayant la même valeur de hachage. Cette liste, dite de collision, est implémentée sous forme d'éléments chainés, avec un simple pointeur de tête : l'ajout et le retrait d'un élément se fait par la tête.

Les tables de hachage sont utilisées pour implémenter efficacement des structures de données de haut niveau telles que les ensembles (au sens mathématique) ou les dictionnaires (associations clé - valeur). Nous nous focalisons dans ce TD sur les dictionnaires : la fonction de hachage sera appliquée à la partie clé d'un couple (clé, valeur).

2 Version simple

Les fonctions essentielles d'un dictionnaire sont :

- ajouter(cle, valeur), qui ajoute le nouveau couple (cle, valeur) si la clé n'existe pas encore, ou remplace la valeur courante de la clé par la nouvelle valeur si la clé existe déjà,
- valeur(cle), qui retourne la valeur courante associée à la clé si elle existe, ou NULL si elle n'existe pas,
- retirer(cle), qui retire le couple (cle, valeur) correspondant à la clé,
- taille(), qui donne le nombre de couples (clé, valeur) existants.

Pour simplifier, nous développons dans un premier temps un dictionnaire dont les clés et les valeurs sont des chaines de caractères.

2.1 Interface du service

Sur le modèle de l'interface liste, écrivez le fichier le fichier `dictionnaire.h` matérialisant l'interface d'un dictionnaire telle que décrite ci-dessus.

2.2 Programme de test

Développez dans le fichier `testDictionnaire.c` un programme de test fonctionnel du service, qui exerce toutes ses fonctions essentielles.

2.3 Implémentation du service

Développez dans le fichier `dictionnaire.c` une implémentation simple du service :

Le tableau est de taille fixe, défini par une constante, par exemple TAILLE_TABLEAU.

La fonction de hachage est également simple : elle retourne par exemple le premier caractère de la clé, considéré comme un entier, ou la somme des caractères de la clé pour limiter les risques de collision. Noter que dans tous les cas, ce n'est pas la valeur h retournée par la fonction que l'on utilise directement pour indicer le tableau, mais h % TAILLE_TABLEAU.

2.4 Test du service

Testez que votre implémentation fonctionne correctement.

Testez ensuite qu'elle continue à fonctionner correctement même avec une fonction de hachage très dégradée, qui retourne par exemple la même valeur quel que soit l'élément. Vous aurez ainsi testé que vous avez correctement développé les opérations d'ajout, de retrait et de recherche dans les listes de collision.

3 Redimensionnement dynamique du tableau

Les performances d'accès – ajout, recherche et retrait – de la table vont se dégrader si le taux de collisions devient trop élevé, notamment parce que le nombre d'éléments est trop grand par rapport à la taille du tableau.

Développez une seconde version de votre implémentation qui agrandisse automatiquement le tableau quand le taux de collision dépasse un certain seuil et repositionne les éléments dans le nouveau tableau.

Développez le mécanisme inverse, qui rapetisse automatiquement le tableau, à partir d'un critère et d'un seuil que vous définirez.