Yoan LE NEVEZ & Anh Tu NGUYEN

Rapport Mini-Projet 2 : Gestion d'une bibliothèque

Ce mini-projet consiste à créer 2 structures différentes pour créer un livre contenant des attributs, et une bibliothèque contenant plusieurs livres. On va utiliser une liste simplement chaînée, et une table de hachage. On va créer les mêmes fonctions pour les différentes structures et par la suite, on va comparer les temps mis entre les deux structures pour voir lequel est le plus performant en terme de temps.

On a donc 4 structures créer : pour la liste, on a une structure livre avec 3 attributs et un pointeur vers un autre livre et une structure bibliothèque qui a seulement un pointeur livre (cette structure est donc une tête fictive), et pour la table de hachage, on a une autre structure livre qui a en plus une clef (clef pour la table de hachage) avec les 3 attributs et le pointeur suivant, et la bibliothèque ici aura 2 attributs : le nombre d'éléments, donc de livres contenus dans la tableau de hachage, la taille de la table de hachage, et un tableau de livres qui sera la table de hachage, les livres seront positionnés dedans en fonction de l'attribut clef de la struct livre.
Pour les différencier, on a appellé les structures LivreH et BiblioH pour la table de hachage.

Les fonctions principalement créer dans les fichiers biblioLC.c et biblioH.c sont des fonctions pour créer une des structures, l'afficher, insérer un élément, supprimer un élément, une recherche fait par un des attributs de la struct livre, eventuellement une fonction fusion, et une fonction libérer pour la mémoire en fin de programme. Or, la table de hachage nécéssitera quelques fonctions en plus tels que une fonction pour créer la clef du livre et une fonction de hachage qui va transformer la clé pour éviter toutes collisions. Les fichiers biblioLC.h et biblioH.h vont venir contenir la définition des fonctions, des structures, et les #include nécessaires.
Il y a de même les fichiers entreeSortieLC.c et entreeSortieH.c qui permettent de lire n lignes d'un fichier où le nom est passé en argument et de le stocker dans une bibliothèque grâce à la fonction charger_n_entrees(), et au contraire une fonction qui permet de stocker une bibliothèque dans un fichier qui est enrgistrer_biblio().
On a enfin les fichiers main.c, mainH.c qui permettent d'utiliser toutes nos fonctions créer auparavant.
Pour se faire, on a créer un menu interactif, où chaque touche entrée par l'utilisateur entre 1-10 (et 0 pour mettre fin au programme) permet de faire une action sur un fichier passé en paramètre avec le nombre de lignes à lire. Par exemple, la touche 4 va permettre de rechercher un livre dans la bibliothèque contenue dans le fichier par son titre.
On a donc un fichier main.c pour la struct de liste chaînée, mainH.c pour la table de hachage, et le fichier main_exo3.c pour comparer les 2 structures en termes de temps pour les mêmes fonctions.

Les différents jeux de tests permettent de voir que les fonctions telles que la création d'une bibliothèque, de livres, l'insertion d'un livre, les fonctions de recherche et de suppression font bien leurs rôles, on

fait aussi gaffe au cas où un livre ou une bibliothèque est vide et il n'y a pas d'erreur tel que Segmentation fault ou autre.
A l'aide de la fonction clock (), on peut voir que les temps mis sont très minimes et donc que les performances de nos programmes sont bonnes et nos programmes font bien leurs rôles tout en évitant les fuites de mémoires (pour tous les fichiers main et jeu de test).

Réponses aux questions :

3.1)

Pour une taille de n=100, il y a très peu de différences de temps entre la liste et la table de hachage. Les 2 donnent le même temps quand un livre n'est pas trouvée et de même quand il est trouvé.
Or, pour la recherche par auteur, la table de hachage est beaucoup plus rapide grâce à la clef de hachage qui permet de directement aller à la position de l'ouvrage recherché. (test avec le fichier main_exo3.c)
L'utilisation de la liste ou de la table de hachage ne change donc pas beaucoup le temps de recherche, hormis pour la recherche par auteur où il y a un grande différence.

3.2)

On remarque que plus la taille augmente (pour la liste et la table de hachage), et plus l'écart de temps entre les deux augmentent.
La table de hachage commence à prendre de plus en plus de temps comparé à la liste. Or, pour la recherche par auteur, la table de hachage reste toujours plus rapide que la liste peu importe sa taille.

Avec n=1000 :
Liste : Numero=0.000012, Titre=0.000013, Auteur=0.000014
Table de Hachage : Numero=0.000017 , Titre=0.000052 , Auteur=0.000008

Avec n=10000 :
Liste : Numero=0.000105, Titre=0.000086, Auteur=0.000111
Table de Hachage : Numero=0.000652 , Titre=0.000352 , Auteur=0.000034

3.4)

Etant donné que pour une table de hachage, si le nombre de cases du tableau est au moins proportionnel au nombnre d'éléments à stocker, et que dans notre cas, ce l'est par n = 100, m += 1000, on a donc une complexité de O(1) pour l'ajout d'éléments, la suppression, et la recherche d'éléments.
Or, pour une liste simplement chaînée, la recherche est en O(n) donc plus la taille n augmente, et plus on voit la différence entre les 2 courbes où celle de la table reste assez stable tandis que la courbe de la liste augmente grandement.