SORBONNE UNIVERSITÉ

Mini-Projet: 2i006

Gestion d'une bibliothèque

Réalisé par :

BOUSBA Abdellah

HADDADI Hacene

Enseignants :

Pierre Fouilhoux

Etienne Simon

Mathilde Carpentier

2ème année Licence Informatique
mono-disciplinaire
Groupe n°3
2018-2019

Description globale des structures et du code :

Structures utilisées :

s_livre : contient les informations d'un livre (numéro, titre et auteur) et un pointeur sur le livre suivant (NULL si non existant).

Biblio : représente la Bibliothèque qu'on veut gérer, contient une liste de livres (pointeur sur s_livre) et leur nombre.

cell_t : contient les informations d'un livre (numéro, titre et auteur) et un pointeur sur le livre suivant (NULL si non existant), de plus une clef qui nous permet de le gérer dans notre table de hachage.

tableHachage_t : représente la Bibliothèque qu'on veut gérer sous forme de table de hachage, contient le nombre de livres, la taille de la table et un tableau de cell_t ; chaque case contient une liste de livre pour gérer les collisions.

Fichiers utilisés :

biblio.h , biblioHash.h, entree_sortie.h : contiennent les signatures des fonctions et la déclarations des structures décrite précédemment des fichier .c correspondants.

entree_sortie.c : contient le code des fonctions pour la gestion d'entrée sortie des fichiers utilisés.

biblio.c : (resp. **biblioHash.c**) contient le code des fonctions pour la gestion de bibliothèque représentée par une liste chainée (resp. table de hachage)

main.c : (resp. **mainHash.c**) contient le menu d'interaction avec l'utilisateur pour le cas d'une liste chainée (resp. table de hachage)

mainTest.c : contient le test du code et la comparaison entre les deux structures.

Description des Algorithmes :

Cas de liste :

lecture_n_entree : si l'ouverture du fichier a échoué on affiche l'erreur, sinon on lit n ligne du fichier en utilisant entree_sortie.c, on initialise un nouveau livre à chaque fois et on l'ajoute en tête d'une liste temporaire qu'on retournera a la fin de la boucle après la fermeture du fichier.

intialise_biblio : on alloue une Biblio, initialise la liste a NULL, le nombre de livre a 0 et on la retourne.

intialise_livre : on alloue un livre, initialise ses attributs avec les arguments de la fonction et suiv a NULL et on le retourne (on utilise strdup pour les chaine de caractère).

liberer_livre : si le livre n'est pas NULL on libère les chaines de caractères d'abord ensuite le livre.

liberer_biblio : on parcours la liste, et on libère chaque livre.

inserer_livre : si le livre et biblio existent l->suiv pointe sur la liste des livre et on recule la liste vers l.

supprimer_livre : on traite le cas ou le livre est en tête de liste sinon on la parcours jusqu'à ce qu'on pointe sur son précédent, on change le chainage et on libère le livre.

recherche_num (resp.recherche_titre): on parcours la liste et retourne le livre si il correspond au numéro voulu (resp. titre en utilisant strcmp) à la sortie de la boucle on retourne NULL.

recherche_auteur : on parcourt la liste et insère le livre a une liste temps s'il est écrit par l'auteur voulu en utilisant strcmp, a la sortie de la boucle on retourne la liste temps.

recherche_double : on parcours toute la liste, pour chaque livre on recherche les livres qui ont le même auteur, on parcours cette dernière et on insère le livre dans une liste tmp dans le cas ou on passe par deux exemplaire qui on le même titre et que le livre n'existe pas déjà dans tmp.

Cas de table de hachage :

initTableHachage : on alloue la table, ensuite le tableau avec calloc pour avoir toutes les cases à NULL.

fonctionClef : retourne la somme des caractères d'une chaine de caractères

intialise_cell : le même algorithme que s_livre, on calcule la clé en plus

Hrecherche_auteur : la case du tableau avec l'indice haché contient la liste des livres qui ont le même auteur (avec quelques autres en plus dans le cas des collisions), alors on parcourt la liste et on ajoute un nouveau livre initialisé si il a le même auteur.

Hliberer_biblio : on parcours le tableau et on libère chaque liste, ensuite le tableau et la table.

PS : pour les fonctions restantes il suffit de calculer la clé haché pour récupérer une liste linéaire chaînée et appliquer dessus les mêmes algorithmes que dans la partie 1 sur les listes.

Comparaison des deux structures :

Soit m la taille de la table de hachage pour n=25000 :

m=1 :

recherche titre : temps d'exécution de **liste** : 0.929941, **table de hachage** : 0.914963

recherche auteur : temps d'exécution de **liste** : 0.993571, **table de hachage** : 1.005033

recherche numéro : temps d'exécution de **liste** : 0.959078, **table de hachage** : 0.942535

m=191 :

recherche titre : temps d'exécution de **liste** : 0.935981, **table de hachage** : 2.916333

recherche auteur : temps d'exécution de **liste** : 0.983449, **table de hachage** : 0.005033

recherche numéro : temps d'exécution de **liste** : 0.948023, **table de hachage** : 2.952566

m=1901 :

recherche titre : temps d'exécution de **liste** : 0.922269, **table de hachage** : 2.995621

recherche auteur : temps d'exécution de **liste** : 0.969958, **table de hachage** : 0.000733

recherche numéro : temps d'exécution de **liste** : 0.958821, **table de hachage** : 2.975421

Conclusion : la table de hachage est plus rapide pour la recherche avec auteur, par contre la liste est plus appropriée pour la recherche par titre ou numéro.

On remarque que le temps d'exécution dans le cas d'une table de hachage pour la recherche par auteur diminue à chaque fois qu'on augmente la taille de la table.

Etudions la vitesse de la fonction recherche_doublant :

La complexité est en $O(n^2)$ pour la structure liste et O(n) pour celle de la table de hachage dans le pire des cas. Et c'est exactement ce qu'on remarque dans les courbes suivantes, complexité quadratique pour la courbe de liste et une complexité linière pour tHash avec n le nombre de livres dans la bibliothèque.