FERCHE Adelin-Flaviu
COULIBALY Mariame
Groupe 4

# MINI PROJET 2

Dans ce mini-projet, nous nous intéressons à la gestion de deux bibliothèques contenant des livres, l'une implémenter en une liste simplement chaînée et l'autre implémenter à l'aide d'une table de hachage. Le but est de comparer l'efficacité des deux structures de données.

## DESCRIPTION DES FICHIERS

- biblioLC.h :
    typedef struct Livre
    typedef struct Biblio

- biblioLC.c :
    Livre* creer_livre(int num, char* titre, char* auteur)
    void liberer_livre(Livre* l)
    Biblio* creer_biblio()
    void liberer_biblio(Biblio* b)
    void inserer_en_tete(Biblio* b, int num, char* titre, char* auteur)
    void affichage_livre(Livre* l)
    void affichage_biblio(Biblio* b)
    void recherche_numero(Biblio* b, int n)
    void recherche_titre(Biblio* b, char* t)
    Biblio* recherche_auteur(Biblio* b, char* a)
    Biblio* suppression_ouvrage(Biblio* b, int n, char* t, char* a)
    Biblio* fusion_biblio(Biblio* b1, Biblio* b2)
    Biblio* recherche_plusieurs_exemplaires(Biblio* b, int n)

- entreeSortieLC.h

- entreeSortieLC.c :
    Biblio* charger_n_entrees(char* nomfic, int n)
    void enregistrer_biblio(Biblio* b, char* nomfic)

- biblioH.h :
    typedef struct LivreH
    typedef struct BiblioH

- biblioH.c :
    int fonctionClef(char* auteur)
    LivreH* creer_livreH(int num, char* titre, char* auteur)
    void liberer_livreH(LivreH* l)
    BiblioH* creer_biblioH(int m)
    void liberer_biblioH(BiblioH* b)

int fonctionHachage(int cle, int m)
void inserer(BiblioH* b, int num, char* titre, char* auteur)
void affichage_livreH(LivreH* l)
void affichage_biblioH(BiblioH* b)
void recherche_numeroH(BiblioH* b, int n)
void recherche_titreH(BiblioH* b, char* t)
BiblioH* recherche_auteurH(BiblioH* b, char* a)
BiblioH* suppressionH(BiblioH* b, int n, char* t, char* a)
BiblioH* fusionH(BiblioH* b1, BiblioH* b2)
BiblioH* recherche_plusieurs_exemplairesH(BiblioH* b,int n)

- entreeSortieH.h

- entreeSortieH.c :
BiblioH* chargerH(char* nomfic, int n)
void enregistrerH(BiblioH* b, char* nomfic)

- main.c :
void menu1()
void menu2()
int main(int argc, char** argv)

- Makefile

- sortie_vitesse.txt

- tps_plusieurs_exemplaires.txt :
courbes_vitesse1.png
courbes_vitesse2.png

**EXERCICE 1 :**

**Q 1.1)**
On crée le fichier biblioLC.h qui est un header dans lequel on a répertorié tous les fonction contenues dans le ficher biblioLC.c associé.

**Q 1.2)**
Le fichier biblioLC.c contient un ensemble de fonctions : Une première fonction **creer_livre** permet de créer un livre, une autre ***liberer_livre*** réalise une desallocation et permet de libérer de l'espace dans la mémoire ayant déjà été allouée pour un livre lorsqu'il n'est plus disponible dans la bibliothèque par exemple. La fonction **creer_biblio** permet de créer une nouvelle bibliothèque vide et **libérer_biblio** permet à l'inverse de libérer la mémoire occupée par une bibliothèque. Enfin, la fonction **insérer_en_tete** permet d'ajouter un nouveau livre à une bibliothèque.

**Q 1.3)**
Ici, il était question de créer un fichier entree_SortieLC.h qui est l'header contenant l'ensemble des signatures des fonctions contenues dans entreeSortieLC.c . Ce dernier contenant 2 fonctions, la première permettant de lire les lignes contenues dans un fichier et de les stocker dans la bibliothèque et la suivante de stocker une bibliothèque au format adéquat (numéro titre auteur).

**Q 1.4)** La fonction **main(argc, argv)** doit faire en sorte que l'executable doit être lancé avec des argument, à savoir le fichier où est stocker la bibliothèque et le nombre de ligne qu'il faut lire. S'il manque au moins l'un de ces arguments, cela afficherai le message d'erreur «il manque des paramètres.

**Q 1.5)**
Cette question ne consistait qu'en la création d'un fichier Makefile permettant de compiler un à un les fichiers puis de créer l'exécutable. Cela permet une meilleure organisation et un meilleure contrôle de ses fichiers.

**Q 1.6)**
On a placer toute les fonctions demandées dans le fichier biblioLC.c et on les a également répertorier dans le header biblioLC.h

**Q 1.7)**
La fonction **void menu2( )** permet d'afficher l'ensemble des possibilités, représenté par les fonctions demandées pour la 1.3 et la 1.6, permettant à l'utilisateur de gérer la bibliothèque.

**Q 1.8)**
Une fois l'executable lancé, le menu apparaît à l'utilisateur demandant quel action il souhaite. Sa réponse est stocké dans une variable avec un fgets et selon son choix, il rentrera dans l'une des conditions du switch case. Grâce à une boucle do while, même si son action c'est réaliser, le menu réapparaîtra à l'utilisateur, pouvant faire un autre choix, tant que l'utilisateur ne tape pas «0», qui correspond à l'arrêt du programme.

## EXERCICE 2 :

Dans cet exercice il était toujours question de gestion de bibliothèque mais cette fois avec une table de hachage.

**Q 2.1)**
Comme pour la question. 1.1, on crée le fichier biblioH.h qui est une bibliothèque dans laquelle on a répertorié tous les fonction contenues dans le ficher biblioH.c associé permettant de gérer une bibliothèque avec une table de hachage.

**Q 2.2)**
On créé la fonction **fonctionClef** qui permet de d'obtenir un nombre qui est la somme des valeurs ASCII des lettres du nom de son auteur, le nombre ainsi obtenu n'est autre que la clé du livre.

**Q 2.3)**
Ici, il nous ait demander d'implémenter différentes fonctions permettant de créer un livre, **creer_livreH** , de libérer de l'espace mémoire alloué préalablement par un livre, **liberer_livreH,** de créer une bibliothèque et de libérer l'espace mémoire occupée par une bibliothèque ; respectivement **creer_biblioH** et **liberer_biblioH**.

**Q 2.4)**
On implémente f**onctionHachage** qui permet de transformer la clé obtenue grâce à **fonctionClef** en une valeur utilisable par la table de hachage.

**Q 2.5)**

Afin d'insérer un livre grâce à sa clé ainsi qu'a la fonction précédente il fallait se demander à quelle case de la table il correspondait pour l'intégrer en tête de liste chainée correspondant à cette case : on a donc pu créer la fonction **inserer**.

**Q 2.6)**
Cette question demande de réadapter les questions 1.3 et 1.6 à l'utilisation d'une table de hachage. Pour cela, on a crée deux nouveaux fichiers : entreeSortieH.h, un header, et entreeSortieH.c. Dans le fichier entreeSortie.c nous avons recréer une fonction charger et enregistrer qui font appel à des fonctions de biblioH.h. Puis, on a ajouté dans le fichier biblioH.c toutes les fonctions demandées dans la 1.6 en modifiant pour que cela fonctionne avec une table de hachage. Enfin, dans le fichier main.c, on a ajouté un deuxième menu, **menu1()**, qui demande à l'utilisateur de choisir son mode de gestion, soit avec liste chaînée ou soit avec table de hachage. Dans le Makefile, on a rajouté les instructions pour créer les fichiers objets biblioh.o et entreeSortieH.o qu'on a rajouté à l'instruction main pour créer l'exécutable.

**EXERCICE 3 :**

**Q 3.1)**
On observe que pour la liste chaînée la recherche par numéro et par titre sont plus rapides en moyenne que celle de la table de hachage. En effet, on a une complexité temporelle de O(nmax), où nmax est le nombre de livre, pour la liste chaînée dans les deux cas, alors que pour la table de hachage, on a une complexité temporelle de O(m*nmax), où nmax est le nombre de livre max de la bibliothèque et m la taille du tableau. En revanche, pour la recherche d'auteur, la table de hachage est plus rapide que la liste chaînée car l'indice du tableau correspond à la clé haché par rapport à la clé, obtenu par la somme des lettres de l'auteur d'un livre, et la taille du tableau. Ainsi, pour la recherche par un auteur, on peut se placer directement à la case du tableau est parcourir la liste chaînée contenue, donc une complexité temporelle de O(n), où n correspond au nombre de livre dans la case, alors que pour la liste chaînée, on a une complexité temporelle O(nmax), où nmax correspond au nombre de livre max de la bibliothèque.

**Q 3.2)**
En modifiant la taille de la table de hachage, l'évolution des temps de calcul augmente. En effet, même si la taille de la table de hachage augmente, plus parcourir la table de hachage sera long.

**Q 3.3)**
donnée : tps_plusieurs_exemplaires
graphiques obtenus : courbes_vitesse1.png courbes_vitesse2.png

**Q 3.4)**
Sur les graphiques, on observe que pour les deux cas plus le nombre de ligne n à lire est grand, plus le temps de calcul de la recherche de plusieurs exemplaires grandi. Néanmoins, les deux graphiques ne sont pas à la même échelle, environ $10^1$ s pour la liste chaînée, contre,10^-1 et 10^-2 s pour la table de hachage. Cette différence s'explique par le fait qu'il y a plusieurs exemplaires dans la bibliothèque. En effet, vu que la clé hachée correspond est obtenue grâce à l'auteur, cela veut dire que s'il y a plusieurs exemplaires, elles seront dans la même case de la table de hachage alors que dans la liste chaînée on ne connaît pas leur position, donc forcément faudra parcourir toute la liste. Or en s'intéressant au pire cas (plusieurs exemplaires absent) , on remarque que la table de hachage est plus longue que la liste. Pour la liste chaînée, on a une complexité temporelle de $O(nmax^2)$, où nmax est le nombre de livre dans la bibliothèque, dans le pire des cas, alors que pour la table de

hachage, on a une complexité temporelle de $O(m*n^2)$, où n est le nombre de livre qui varie par case et m la taille de la table de hachage.
Ainsi, la table de hachage sera meilleure dans  le cas où il y a plusieurs exemplaires dans une bibliothèque mais la liste chaînée sera meilleur dans le pire des cas, à savoir l'inexistence de plusieurs exemplaires.

En conclusion, la liste chaînée reste meilleure que la table de hachage, qui dans certain cas est meilleure, car dans la majorité des cas, la complexité temporelle de ses fonctions est plus petite que celle de la table de hachage.