

Compte-rendu LU2IN006

Structures de données

Mini-Projet :

Gestion d'une bibliothèque

PENG Kairui 21219046

WENG Julie 21209248

I. Introduction

Ce rapport présente une analyse comparative de deux structures de données différentes, à savoir les listes chaînées et les tables de hachage, dans le contexte de la gestion d'une bibliothèque (biblio) et de ses livres (livre). Le projet se divise en trois parties principales : l'implémentation à l'aide de listes chaînées, l'implémentation à l'aide de tables de hachage, et enfin la comparaison des performances entre les deux approches. Enfin, nous fournissons et analysons les résultats des tests de performance afin de tirer des conclusions sur la pertinence de chaque structure de données pour la gestion des bibliothèques.

II. Description des codes

1. Les Listes Chaînées

Structure

Pour les listes chaînées, il y a deux structures : Livre et Biblio.

```
4  typedef struct livre{  
5      int num;           //numéro du livre  
6      char *titre;       //titre du livre (pointeur de chaîne de caractères)  
7      char *auteur;      //auteur du livre (pointeur de chaîne de caractères)  
8      struct livre * suiv; //pointeur vers le livre suivant  
9  } Livre;  
10  
11  
12 typedef struct{      //Tête fictive  
13     Livre* L;         //Premier élément  
14 } Biblio;
```

L : pointeur vers le premier élément, c'est-à-dire un pointeur de type Livre. Cette structure peut être considérée comme une tête de la bibliothèque, ne stockant pas réellement de données de livre mais servant de point de départ pour accéder à la liste des livres.

Opérations sur Livre et Biblio

Nous avons réalisé : la création, l'affichage, la suppression et libération de Livre et de Biblio. Nous devons faire attention à utiliser la fonction strdup() pour copier les pointeurs de chaîne de caractères, ce qui copie également la mémoire ; vérifier si l'allocation de mémoire avec malloc() réussit ; libérer la mémoire des pointeurs de chaîne de caractères ; lors de la libération de la

mémoire de Biblio, il est nécessaire de parcourir toute la liste chaînée et de

```
17 /* créer un livre */
18 Livre* creer_livre(int num,char* titre,char* auteur);
19
20 /* réaliser une désallocation */
21 void liberer_livre(Livre* l);
22
23 /* afficher une Livre */
24 void afficher_livre(Livre* l);
25
26 /* créer une bibliothèque vide */
27 Biblio* creer_biblio();
28
29 /* libérer la mémoire occupée par une bibliothèque */
30 void liberer_biblio(Biblio* b);
31
32 /* l'affichage d'une bibliothèque */
33 void afficher_biblio(Biblio* b);
34
35 void supprimer_livre(Biblio* b, int num, char* titre, char* auteur) {
36     if (b==NULL) {
37         return ;
38     }
39     Livre *tmp = b->L;
40     Livre *precedent = NULL;
41     while (tmp) {
42         if ((tmp->num==num) && (strcmp(tmp->titre,titre)==0) && (strcmp(tmp->auteur,auteur)==0)) {
43             if (precedent==NULL) { // si supprimer la tête
44                 b->L=tmp->suiv;
45             } else {
46                 precedent->suiv = tmp->suiv;
47             }
48             liberer_livre(tmp);
49             return ;
50         } else {
51             precedent = tmp;
52             tmp = tmp->suiv;
53         }
54     }
55 }
```

libérer la mémoire de chaque livre individuellement.

Recherche dans Biblio par son numéro, son titre , tous les livres de même auteur et les livres identiques. Dans ce type d'opérations, nous vérifions d'abord si Biblio est vide pour éviter les erreurs de référence, puis nous initialisons un pointeur pointant vers le premier élément de la liste chaînée et parcourons toute la liste jusqu'à la fin (tmp == NULL) ou jusqu'à ce que nous trouvions un élément répondant aux critères et effectuons les opérations correspondantes.

```
91 Livre* rechercher_par_titre(Biblio* b, char* titre) {
92     if (b == NULL) {
93         return NULL;
94     }
95     Livre *tmp = b->L;
96     while (tmp) {
97         if (strcmp(tmp->titre,titre)==0) { //si les titres sont le même
98             return tmp;
99         }
100        tmp = tmp->suiv;
101    }
102    return NULL;
103 }
```

```

126 Biblio* rechercher_livres_meme_auteur(Biblio* b, char* auteur) {
127     if (b==NULL) {
128         return NULL;
129     }
130     Livre *tmp = b->L;
131     Biblio *newBiblio = creer_biblio();
132     while (tmp) {
133         if (strcmp(tmp->auteur,auteur)==0) { //si d'un même auteur
134             inserer_en_tete(newBiblio,tmp->num,tmp->titre,tmp->auteur);
135         }
136         tmp = tmp->suiv;
137     }
138     return newBiblio;
139 }
```

Pour la fonction *Biblio* rechercher_livres_identiques(Biblio* b)*, il commence par vérifier si la Biblio donnée est vide, puis parcourt toute la liste chaînée. Pour chaque livre, il parcourt la liste chaînée suivante pour rechercher d'autres livres ayant le même titre et le même auteur. S'il trouve des livres correspondants, il les insère dans une nouvelle Biblio. Enfin, il retourne la nouvelle Biblio contenant les livres identiques.

```

184 Biblio* rechercher_livres_identiques(Biblio* b) {
185     if (b==NULL) {
186         return NULL;
187     }
188     Livre *tmp1 = b->L;
189     Livre *tmp2;
190     Biblio *b_iden = creer_biblio(); //bibliothèque des livres identiques
191     while (tmp1) {
192         tmp2 = tmp1->suiv;
193         while (tmp2) {
194             if ((strcmp(tmp2->titre,tmp1->titre)==0) && (strcmp(tmp2->auteur,tmp1->auteur)==0)) {
195                 inserer_en_tete(b_iden,tmp2->num,tmp2->titre,tmp2->auteur);
196                 break;
197             }
198             tmp2 = tmp2->suiv;
199         }
200         tmp1 = tmp1->suiv;
201     }
202     return b_iden;
203 }
```

La manipulation de Biblio : *insérer* et *fusionner*. Ces deux opérations sont relativement simples, mais il est important de prendre en compte différentes situations où les pointeurs ou les structures peuvent être nuls, et de fournir des solutions appropriées (affichage des messages d'erreur, création de nouveaux pointeurs, etc.).

```

41 /* ajoute un nouveau livre au début de la bibliothèque */
42 void inserer_en_tete(Biblio* b,int num,char* titre,char* auteur);
165 Biblio* fusionner_biblio(Biblio* b1, Biblio* b2) {
166     if (b2 == NULL) {
167         return b1;
168     }
169     if (b1 == NULL) {
170         b1 = b2;
171         free(b2);
172         return b1;
173     }
174     Livre *tmp = b1->L;
175     while (tmp->suiv) {
176         tmp = tmp->suiv;
177     }
178     tmp->suiv = b2->L;
179     free(b2);
180     return b1;
181 }
```

```

11 void menu() {
12     printf("\nCe sont les actions possibles sur la bibliothèque:\n");
13     printf("0-sortie du programme\n");
14     printf("1-Affichage\n");
15     printf("2-Inserer ouvrage\n");
16     printf("3-la recherche d'un ouvrage par son numéro\n");
17     printf("4-la recherche d'un ouvrage par son titre\n");
18     printf("5-la recherche de tous les livres d'un même auteur\n");
19     printf("6-la suppression d'un ouvrage\n");
20     printf("7-la fusion de deux bibliothèques : bibliotheque et bNew\n");
21     printf("8-la recherche de tous les ouvrages identiques\n");
22 }

```

La fonction charger_n_entrees est utilisée pour lire un nombre spécifié d'entrées à partir d'un fichier et les charger dans une structure 'Biblio'. Elle ouvre le fichier spécifié, lit les données ligne par ligne, puis analyse chaque ligne pour extraire le numéro, le titre et l'auteur d'un livre, qu'elle insère ensuite dans une structure 'Biblio'. Enfin, elle renvoie la structure 'Biblio' créée.

- La fonction enregistrer_biblio est utilisée pour écrire les informations des livres de la structure 'Biblio` dans un fichier. Elle ouvre le fichier spécifié, puis parcourt chaque livre de la structure 'Biblio', écrivant son numéro, son titre et son auteur dans le fichier.

Ces deux fonctions comprennent des vérifications pour l'ouverture et la fermeture des fichiers, ainsi que pour l'allocation de mémoire.

```

5 #include "biblioLC.h"
6
7
8 /* permettant de lire n lignes du fichier et de les stocker dans une bibliothèque */
9 Biblio* charger_n_entrees(char* nomfic, int n);
10
11 /* permet de stocker une bibliothèque dans un fichier au bon format : numéro titre auteur. */
12 void enregistrer_biblio(Biblio *b, char* nomfic);

```

Application dans mainLC.c

```

25 int main(int argc, char** argv){
26     if (argc != 3) { //la taille de la commande './main <NomFichier> <NumLigne>' est 3
27         printf("Erreur de la commande.\n");
28         return 0;
29     }
30     char* nomFichier = argv[1];
31     int nombreLignes = atoi(argv[2]); //str ==> int
32     Biblio *bibliotheque = charger_n_entrees(nomFichier, nombreLignes);
33
34     if (bibliotheque == NULL) {
35         printf("Erreur lors du chargement de la bibliothèque depuis le fichier %s\n", nomFichier);
36         return 0;
37     }
38     afficher_biblio(bibliotheque);

```

Ce programme gère une bibliothèque de livres en utilisant une liste chaînée. Voici un aperçu de son fonctionnement :

1. Le programme commence par charger un certain nombre d'entrées à partir d'un fichier spécifié en ligne de commande. Si le chargement échoue, il affiche un message d'erreur.

2. Ensuite, il affiche un menu d'options pour l'utilisateur, telles que l'affichage de la bibliothèque, l'insertion d'un nouveau livre, la recherche d'un livre par son numéro ou son titre, la recherche de tous les livres d'un même auteur, la suppression d'un livre, la fusion de deux bibliothèques, et la recherche de tous les ouvrages identiques.

3. L'utilisateur peut sélectionner une action en saisissant le numéro correspondant dans le menu. Le programme effectue ensuite l'action choisie et affiche les résultats.

4. L'utilisateur peut répéter ces actions autant de fois qu'il le souhaite jusqu'à ce qu'il choisisse de quitter le programme en saisissant 0.

```
48     do {
49         menu();
50         scanf(" %d", &rep);
51         switch (rep){
52             case 1:
53                 printf("Affichage :\n");
54                 afficher_biblioH(bibliotheque);
55                 break;
56
57             case 2:
58                 printf("Veuillez écrire le numéro, le titre et l'auteur de l'ouvrage.\n");
59                 /* On suppose que le titre et l'auteur ne contiennent pas d'espace */
60                 if (scanf(" %d %s %s", &num, titre, auteur)==3){
61                     inserer(bibliotheque, num, titre, auteur);
62                     printf("Ajout fait \n");
63                 }else{
64                     printf("Erreur format\n");
65                 }
66                 break;
67
68             case 3:
69                 printf("Veuillez écrire le numéro \n");
70                 if (scanf(" %d", &num)==1){
71                     livre = rechercher_par_numero_H(bibliotheque, num);
72                     if (livre==NULL) {
73                         printf("Désolé, on n'a pas trouvé l'ouvrage\n");
74                     } else {
75                         printf("l'ouvrage trouvé :\n");
76                         afficher_livreH(livre);
77                     }
78                     printf("recherche fini \n");
79                 }else{
80                     printf("Erreur format\n");
81                 }
82                 break;
83         }
84     }
85 }
```

5. Une fois que l'utilisateur quitte le programme, toutes les ressources allouées sont libérées pour éviter les fuites de mémoire.

Le programme est conçu pour être interactif et convivial, permettant à l'utilisateur de gérer facilement la bibliothèque de livres de différentes manières.

Par exemple, on retrouve et supprime le livre identique dans le tableau.

```
kairuipeng@KairuideMacBook-Pro code % ./mainLC GdeBiblio.txt or 1000
num: 999, titre: ZUNSL, auteur: hgxyjyazitnxg
num: 998, titre: QIRNNNCHGU, auteur: cmklulo
num: 997, titre: HHGZBCOGOT, auteur: knuyzq
num: 996, titre: CPQO, auteur: funsrptv
num: 995, titre: QNVCYEAMLMRUVND, auteur: fivufok
num: 994, titre: DGJTHNQ, auteur: lpvtpqodujin
num: 993, titre: VDQUGA, auteur: mlgymh
num: 992, titre: EIPLBZMPZXQZEWWVW, auteur: qgatvl
num: 991, titre: LZAFGUSZS, auteur: cdnljarmzdi
num: 990, titre: GNGA, auteur: jwubywfqiao
```

```
ce sont les actions possibles sur la bibliothèque:  
0-sortie du programme  
1-Affichage  
2-Inserer ouvrage  
3-la recherche d'un ouvrage par son numéro  
4-la recherche d'un ouvrage par son titre  
5-la recherche de tous les livres d'un même auteur  
6-la suppression d'un ouvrage  
7-la fusion de deux bibliothèques : bibliotheque et bNew  
8-la recherche de tous les ouvrages identiques  
8  
num: 3, titre: KEZXDU, auteur: xdrwv  
rechercher fini  
0-sortie du programme  
1-Affichage  
2-Inserer ouvrage  
3-la recherche d'un ouvrage par son numéro  
4-la recherche d'un ouvrage par son titre  
5-la recherche de tous les livres d'un même auteur  
6-la suppression d'un ouvrage  
7-la fusion de deux bibliothèques : bibliotheque et bNew  
8-la recherche de tous les ouvrages identiques  
6  
Veuillez ecrire le numero, le titre et l'auteur de l'ouvrage.  
3 KEZXDU xdrwv  
Supprimer fini
```

Nous avons à nouveau cherché livre identique et avons constaté que le résultat était vide.

```
ce sont les actions possibles sur la bibliothèque:  
0-sortie du programme  
1-Affichage  
2-Inserer ouvrage  
3-la recherche d'un ouvrage par son numéro  
4-la recherche d'un ouvrage par son titre  
5-la recherche de tous les livres d'un même auteur  
6-la suppression d'un ouvrage  
7-la fusion de deux bibliothèques : bibliotheque et bNew  
8-la recherche de tous les ouvrages identiques  
8  
rechercher fini
```

Quitter le programme.

```
ce sont les actions possibles sur la bibliothèque:  
0-sortie du programme  
1-Affichage  
2-Inserer ouvrage  
3-la recherche d'un ouvrage par son numéro  
4-la recherche d'un ouvrage par son titre  
5-la recherche de tous les livres d'un même auteur  
6-la suppression d'un ouvrage  
7-la fusion de deux bibliothèques : bibliotheque et bNew  
8-la recherche de tous les ouvrages identiques  
0  
Merci, et au revoir  
kairuipeng@KairuideMacBook-Pro code % |
```

2. Les Tables de Hachage

Structure

```
4  typedef struct livreH{
5      int cle; //la clé utilisée pour l'indexation dans la table de hachage
6      int num; // le numéro du livre
7      char *titre; // le titre du livre (chaîne de caractères)
8      char *auteur; // l'auteur du livre (chaîne de caractères)
9      struct livreH *suivant; //
10 } LivreH;
11
12 typedef struct table {
13     int nE; /*nombre d'éléments contenus dans la table de hachage */
14     int m; /*taille de la table de hachage */
15     LivreH** T; /*table de hachage avec resolution des collisions par chainage */
16 } BiblioH;
```

la clé associée à un livre est la somme des valeurs ASCII de chaque lettre du nom de son auteur; suivant : un pointeur vers le livre suivant ayant la même valeur de hachage, pour résoudre les collisions; T : un pointeur vers un tableau de pointeurs LivreH, utilisé pour stocker les éléments de la table de hachage et résoudre les collisions à l'aide de listes chaînées.

Opérations sur Livre et Biblio

Dans la série de codes de la table de hachage, nous avons principalement ajouté des définitions et des opérations pour la structure de la table de hachage, tout en conservant les fonctions de base liées à la liste chaînée.

```
19 /* obtenir la clé associée à la livre */
20 int fonctionClef(char* auteur);
34 /* la définition de la fonction de hachage, renvoie l'index dans la table de hachage */
35 int fonctionHachage(int cle, int m);
```

Voici quelques codes spécifiques. Nous utilisons la fonction de hachage

$$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor, \text{ où } k \text{ est la clé associée à un livre et } A = \frac{\sqrt{5} - 1}{2}.$$

```
9  int fonctionClef(char* auteur) {
10    int cle = 0;
11    char *tmp = auteur;
12    while(*tmp != '\0') {
13        cle = cle + (int)(*tmp);
14        tmp++;
15    }
16    return cle;
17 }
18 int fonctionHachage(int cle, int m) {
19    double A = (sqrt(5.0) - 1) / 2;
20    return (int)(m * (cle*A - (int)(cle*A)));
21 }
```

Pour les opérations sur la bibliothèque gérée par une table de hachage, essentiellement, elle vérifie d'abord si la table de hachage est vide, puis elle parcourt chaque position d'index de la table de hachage à l'aide d'une boucle. Pendant le parcours, elle utilise un compteur pour s'assurer de ne pas dépasser

le nombre réel d'éléments dans la table de hachage. Pour chaque position d'index, elle vérifie si la liste chaînée est vide. Si ce n'est pas le cas, elle parcourt la liste et effectue les opérations associées, puis incrémenté le compteur. Enfin, la fonction retourne.

```

151 LivreH* rechercher_par_titre_H(BiblioH* bh, char* titre) {
152     if (bh == NULL) {
153         return NULL;
154     }
155     int cpt = 0;
156     for (int i=0;i<bh->m;i++) {
157         if (cpt >= bh->nE) {
158             break;
159         }
160         if (bh->T[i] != NULL) {
161             LivreH *tmp = bh->T[i];
162             while(tmp != NULL) {
163                 if (strcmp(tmp->titre,titre)==0) {
164                     return tmp;
165                 }
166                 tmp = tmp->suivant;
167                 cpt++;
168             }
169         }
170     }
171     return NULL;
172 }
```

Dans les fonctions où l'auteur est utilisé comme critère, il commence par calculer la fonction de hachage à partir du nom de l'auteur pour déterminer l'index dans la table de hachage. Ensuite, elle parcourt la liste chaînée à cet index, vérifiant si chaque livre a le même auteur que celui donné. Pour chaque livre ayant le même auteur, elle l'insère dans une nouvelle table de hachage. Enfin, elle retourne cette nouvelle table de hachage. En réalité, tous les livres ayant le même auteur se trouvent nécessairement dans la même liste chaînée à un index donné de la table de hachage car les clés sont égalisées.

```

175 BiblioH* rechercher_livres_meme_auteur_H(BiblioH* bh, char* auteur) {
176     if ((bh==NULL) || (bh->T==NULL)) {
177         return NULL;
178     }
179     int index = fonctionHachage(fonctionClef(auteur),bh->m); //l'index dans la table hachage
180     BiblioH *bh_new = creer_biblioH(bh->m);
181     LivreH *tmp = bh->T[index];
182     while (tmp != NULL) {
183         if (tmp->clef == fonctionClef(auteur)) { //si l'autuer est la même
184             inserer(bh_new,tmp->num,tmp->titre,tmp->auteur);
185         }
186         tmp = tmp ->suivant;
187     }
188     return bh_new;
189 }
190 }
```

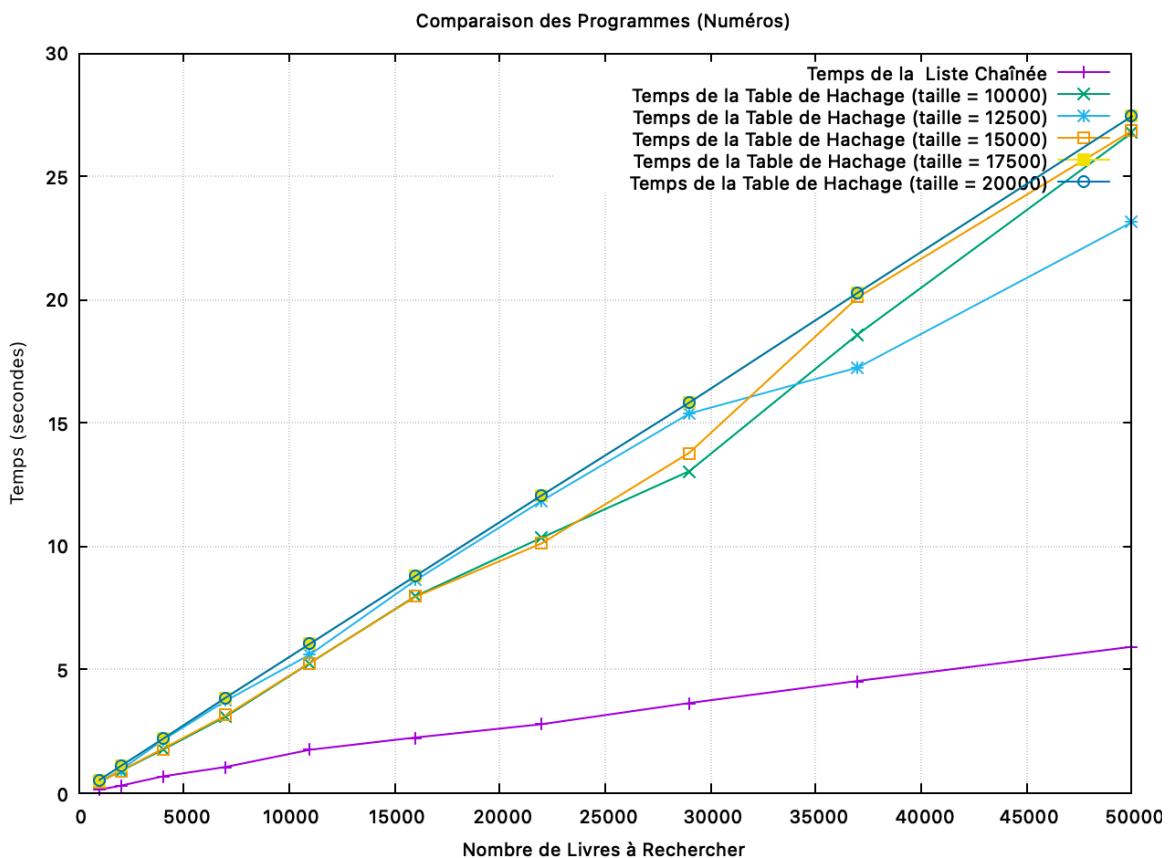
Application dans mainH.c

Ces fonctions sont essentiellement similaires à celles des listes chaînées, donc je vais les omettre ici.

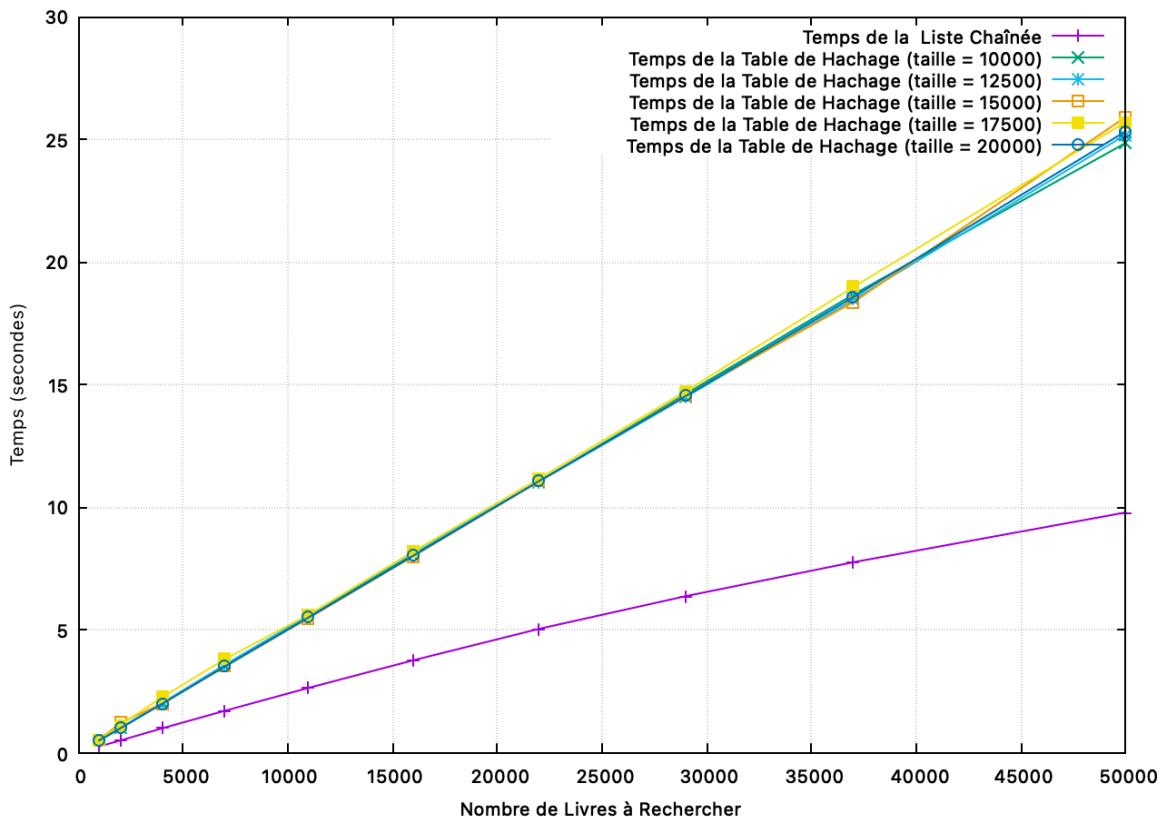
III. Comparaison des deux structures

Dans la fonction 'mainCP.c', nous commençons par stocker respectivement les 99999 livres de la base de données dans une liste chaînée et une table de hachage, puis nous modifions le nombre de livres à lire. Selon nos connaissances, les performances de la table de hachage sont optimales lorsque le facteur de charge se situe entre 0,5 et 1. Par conséquent, nous avons choisi de fixer la taille de la table de hachage entre 10 000 et 20 000 (avec 9 999 éléments stockés à l'intérieur). Enfin, nous comparons le temps pris par les deux méthodes.

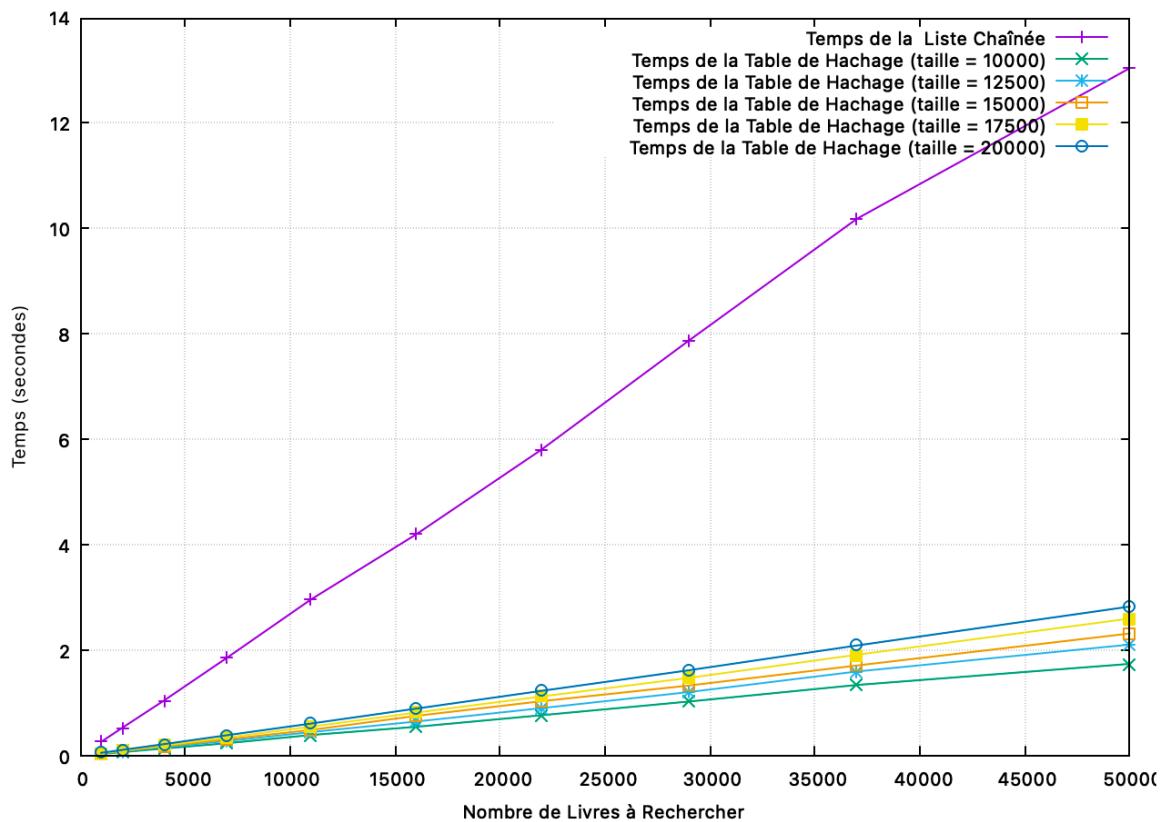
Voici des comparaisons des recherches effectuées avec des listes chaînées et des tables de hachage en utilisant respectivement le numéro, le titre et l'auteur.



Comparaison du Programme (Titre)



Comparaison du Programme (Auteur)



Les Réponse aux Questions

Q 3.1

Selon les données que nous avons recueillies, nous pouvons constater que lors de la recherche par numéro et par titre, la liste chaînée est la plus appropriée, tandis que lors de la recherche par auteur, la supériorité de la table de hachage est incontestable.

Q3.2

Selon nos découvertes, plus la taille de la table de hachage est grande, plus la vitesse de calcul est lente. Cela est dû au fait qu'avec l'augmentation de la taille de la table de hachage, davantage d'opérations de parcours sont nécessaires lors de la recherche, ce qui ralentit la vitesse de calcul.

Analyse des résultats

Pour la recherche par numéro ou titre, la complexité dans le pire des cas est $O(n)$ pour les listes chaînées et $O(m+n)$ pour les tables de hachage. Dans une liste chaînée, il peut être nécessaire de parcourir toute la liste pour trouver un élément correspondant, donc la complexité dépend du nombre d'éléments n dans la liste. En revanche, dans une table de hachage, dans le pire des cas, il peut être nécessaire de parcourir toute la table de hachage pour rechercher un élément, ce qui donne une complexité de $O(m+n)$, où m est la taille de la table de hachage.

Les listes chaînées conservent l'ordre d'insertion des éléments, donc lors d'un parcours dans l'ordre d'insertion, aucune opération supplémentaire n'est requise. En revanche, les éléments dans une table de hachage ne sont pas nécessairement stockés dans l'ordre d'insertion, ce qui peut nécessiter plus de calculs pour déterminer l'emplacement de stockage des éléments, ce qui entraîne des coûts supplémentaires en termes de temps.

Quant à la recherche par auteur, la complexité dans le pire des cas est également $O(n)$ pour les listes chaînées, car il faut parcourir toute la liste pour trouver un auteur correspondant. En revanche, dans le meilleur des cas, une table de hachage peut atteindre une complexité constante $O(1)$, car la fonction de hachage peut mapper directement l'auteur à une position d'index.

Conclusion

En résumé, les listes chaînées peuvent être plus adaptées dans certains cas, notamment pour préserver l'ordre des éléments. Cependant, les tables de hachage peuvent offrir des performances de recherche plus efficaces dans

d'autres cas, en particulier pour la recherche par auteur. Le choix de la structure de données dépend donc des besoins spécifiques et du contexte d'utilisation.