Mini-Projet

TP3 et TP4

Aura Leirós García
LU2IN006 - Structures de données
Sorbonne Université

Contents

1 Résolution de l'exercice 3

À continuation, la réponse aux différentes questions posées lors de l'exercice 3.

1.1 Exercice 3.1

On observe clairement que dans les deux cas, livre présent et livre absent, les listes chaînées sont beaucoup plus efficaces dans la recherche par numéro et par titre, tandis qu'en revanche, lorsqu'on effectue une recherche par auteur, on constate que les tables de hachage sont plus rapides.

On peut expliquer ce phénomène du fait que les clés (et par extension la fonction de hachage) sont calculées avec le nom de l'auteur. Ainsi, on peut déterminer dans quelle case exacte se trouvent les livres de cet auteur, et il suffit alors de parcourir une petite liste chaînée. En revanche, pour les autres deux fonctions, où l'on ne peut pas calculer la clé, il est nécessaire de parcourir toute la table.

1.2 Exercice 3.2

Comme indiqué précédemment, on observe clairement l'impact sur notre fonction de recherche par auteur.

On vérifie bien que quand notre facteur de charge est plus grand on a une légère amélioration des temps de recherche, expliqué par la réduction des collisions, entre autres.

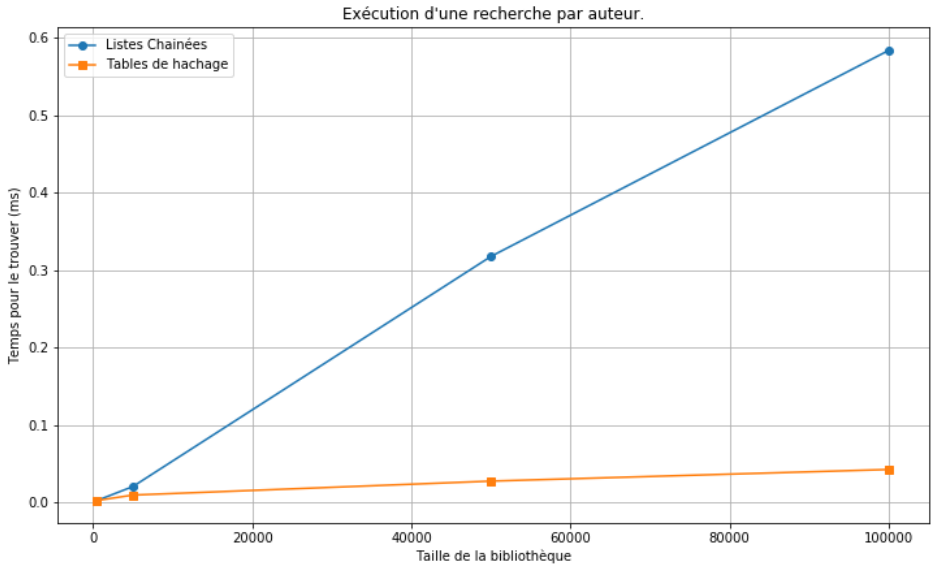
1.3 Exercice 3.3 et 3.4

D'abord, recherche par titre() où on trouve que la fonction avec listes chaînées se rapproche d'une complexité $O(n^2)$ tandis que celui avec une table hachage est une complexité $O(n))$ beaucoup plus efficient.

1.3.1 Recherche par titre

À niveau théorique, la version avec listes chaînées à une complexité pire-cas théorique de $O(n)$, égale à la implementation avec les tables de hachage.

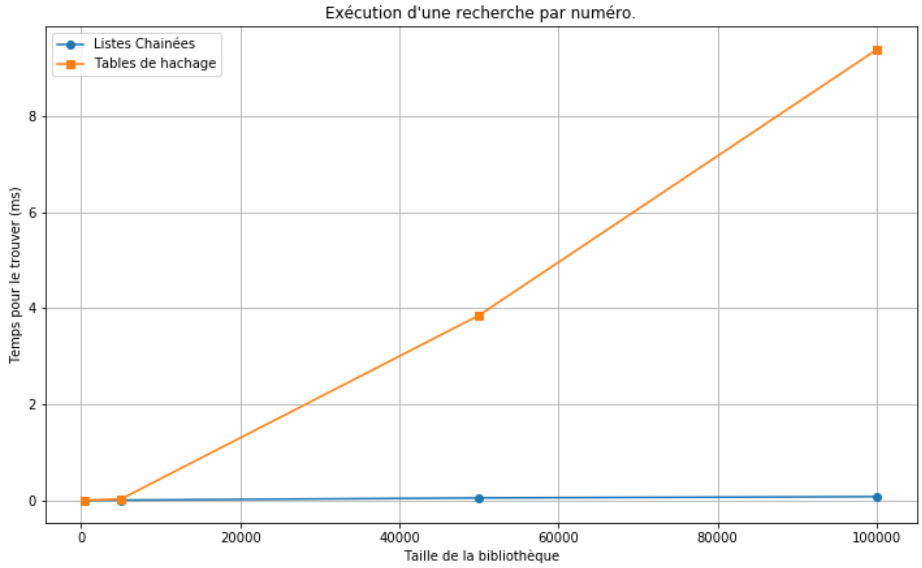
Par contre, à niveau des tests on voit que la complexité des listes chaînées est beaucoup plus (possiblement du à opérations cachés) et celui des tables de hachage est plutôt entre $O(1)$ et $O(n)$.



1.3.2 Recherche par numéro d'enregistrement

À niveau théorique, la complexité attendue pour les listes chaînées est de $O(n)$ qui se correspond avec les données obtenues.

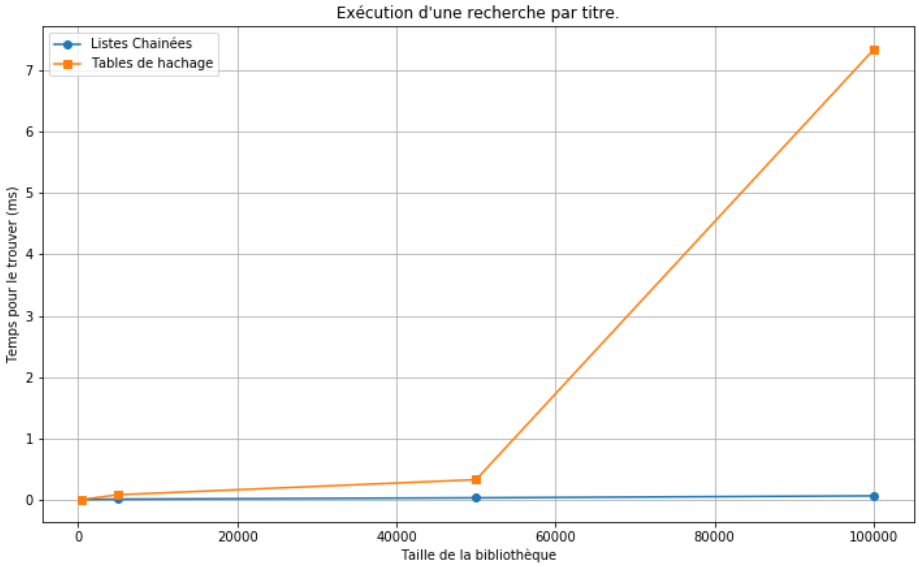
Par contre pour ce table de hachage on a la particularité de qu'on a pas moyen de calculer la clé, ce qui nous fait perdre la totalité des avantages d'une table de hachage classique, et nous donne une complexité pire-cas théorique de $O(n)$ dans le cas de qu'on doit à parcourir tous les livres. Par contre ne se correspond pas aux données obtenues.



1.3.3 Recherche par titre

Comme pour la recherche par numéro d'enregistrement, la complexité attendue pour les opérations dans les listes chaînées est de $O(n)$, et il en va de même pour les tables de hachage.

Cependant, tout comme dans le cas du numéro, bien que les listes chaînées affichent la complexité attendue, il semble y avoir un problème avec notre table de hachage, potentiellement lié à un problème avec la méthode de hachage utilisé dans la conception.


Exécution d'une recherche par titre.
Listes Chainées
Tables de hachage
Temps pour le trouver (ms)
0
1
2
3
4
5
6
7
0
20000
40000
60000
80000
100000
Taille de la bibliothèque

2 Rapport du code

Dans ce projet on a visé le développement d'un logiciel de gestion de librairies avec deux structures de données assez courantes, listes chaînées et tables de hachage avec des fonctions pour l'usage courant d'un utilisateur.

Le code fonctionne, à été désigne d'accord aux bonnes pratiques de programmation et compile sans aucun type d'erreur avec le flag -Wall activé, il contient des jeux de testes qui testent des edge-cases sans aucun type de fuite de mémoire, le code à été testé avec Valgrind et l'option –leak-test=full activé.

2.1 Structures utilisés

Tout d'abord le code est composé des fichiers suivants :

```
main. c - Point d'entrée du programme.
main_aux.c - Code rélatif aux fonctions auxiliaires du main et vitesse.
main_aux.h - Header des fonctions de main_aux.c
biblioLC.c - Code rélatif aux listes chaînées.
biblioLC.h - Header des fonctions de biblioLC.c
entreeSortieLC.c - Traitement des fichiers avec listes chaînées.
entreeSortieLC.h - Header des fonctions de entreeSortieLC.c
biblioH.c - Code rélatif aux tables de hachage.
biblioH.h - Header des fonctions de biblioH.c
entreeSortieH.c - Traitement des fichiers avec tables de hachage.
entreeSortieH.h - Header des fonctions de entreeSortieH.c
test.c - Jeux de tests.
```

```
test.h - Header des fonctions de test.c

makefile - Fichier pour la gestion de compilations
```

Comme indique préalablement, la priorité lors du développement à été toujours construire un programme modulaire, avec une structure solide, résistance aux problèmes liées à la mémoire et paramètres incorrectes et une gestion gracieuse des erreurs avec le printage au stderr.

Particulièrement, remarquer l'utilisation de fonctions dites "sécurises" tel que strncpy et strndup, l'utilisation quasi-exclusivement de mémoire dynamique et une gestion du buffer que nous permet d'initialiser uniquement la mémoire dont on a besoin et sécuriser le programme de problèmes liées au overflow.

2.1.1 Listes Chaînées

Pour les fonctions qu'ont involucré listes chaînées on a utilisé des méthodes assez classiques et standard, spécialement au niveau de recherche avec différemment donnés où on a utilisé pointeurs pour parcourir le tableau avec une condition pour vérifier à chaque fois, donc, la plupart des fonctions sont solides et avec une complexité pire-cas $O(n)$.

2.2 Tables de Hachage

L'autre structure de données utilisé, avec une solution aux collisions par chaînage mais avec un problème de conception déjà remarque, le hachage se réalise avec le nom de l'auteur ce que ne nous permet pas de bénéficier du hachage pour itérer directement dans la case correspondant dans le tableau, ce qui fait que soit uniquement intéressant de l'utiliser pour les recherches avec auteur.

3 Jeux de testes

Les jeux de testes qui sont dans les fichiers test.c et test.h servent à vérifier le bon fonctionnement du programme, pas uniquement quand tous les paramètres sont valides mais aussi quand ils ne le sont pas.

Veuillez noter que plusieurs tests ne sont pas actifs (ils sont commentés) dans le code pour éviter des prints au stderr dans les exécutions et ne pas avoir à rédiger tous les erreurs du programme vers un fichier null lors de la phase de testing.

4 Résultats

On arrive à la même conclusion de base que dans la résolution de l'exercice 3, la méthode pour établir une clé et réaliser le hachage n'est pas l'idéale car on ne peut pas utiliser l'avantage des tables de hachage pour accéder rapidement aux données. Cela se traduit les résultats et fais qu'on a une grosse perte de rendement par rapport aux listes chaînées.

On observe que dans le cas de la recherche par auteur nous obtenons une complexité réelle sur nous tests de $O(1)$ qui nous permet d'apprécier l'efficacité de ce type de structure.

En conclusion, avec les conditions actuelles, la liste chaînée est bien plus efficace que la table de hachage dans ce cas concret d'application.

5 Conclusion

Comme conclusion à ce projet, je tiens à souligner une fois de plus la solidité du programme et l'attention portée aux détails que j'ai essayé d'avoir. Bien que le programme soit loin d'être prêt pour un environnement de production, il a constitué une très bonne base pour renforcer mes connaissances et bonnes pratiques appris en cours.