Mini-Projet 2 : Gestion d'une Bibliothèque

CARAUX Théophane 28602627
Drouard François-Xavier 3800028

Objectif : Apprendre à comparer des structures de données, pour implémenter une bibliothèque nous allons utiliser une liste chainée de struct et une table de hachage de struct. Pour créer nos bibliothèques nous devrons lire dans un fichier.

EXERCICE 1 :

Dans cet exercice, nous allons utiliser la liste chaînée de struct pour implémenter notre bibliothèque.

1.1) Pour cela nous utilisons une struct Livre (qui contient son numéro, son titre, son auteur et un pointeur vers le prochain livre) et une struct Biblio (qui contient la liste chainée des livres). Nous plaçons les définitions de ces structures dans Biblio.h.

1..2) On crée des fonctions qui permettent d'utiliser ces structures, (les fonctions sont commentées dans leurs fichiers).

1.3) On crée deux autres fonctions qui permettent d'utiliser les fichiers dans entreeSortieLC.c / .h.

1.4) On teste nos fonctions dans un main, les fonctions marchent comme souhaité.

1.5) Make est un logiciel qui construit automatiquement des fichiers, on l'utilise pour compiler les programmes que l'on souhaite pour notre exécutable.

1.6) De même que pour la question 1.3, les fonctions sont toutes commentées dans leurs fichiers.

1.7) La fonction menu() sert à présenter les interactions liées à la bibliothèque. Nous avons ajouté un peu de couleur à l'affichage du menu pour le différenciers des actions lancées.

1.8) Grâce au switch case et à notre boucle infini, l'utilisateur peut interagir avec les différentes propositions qui lui sont faites avec la fonction menu tant qu'il ne décide pas de sortir.
Le switch case permet de cibler les requêtes de l'utilisateur.
Nous utilisons des fgets car ils sont plus sûrs que des scanf et permettent d'obtenir la saisie clavier de l'utilisateur et de bien comprendre sa demande.
Le code est commenté et bien expliqué dans le main.c.

EXERCICE 2 :

Pour cet exercice nous allons maintenant utiliser la table de hachage, car les listes ne sont pas très efficaces pour effectuer de nombreuses recherches, comme c'est ici le cas pour les bibliothèques.
Nous allons donc utiliser 2 structures ici :

- LivreH, qui possède tous les attributs de Livre précédemment créé mais avec une clé en plus.
- BiblioH, qui contient son nombre d'éléments, une table de hachage LivreH et sa taille.

2.1) On crée donc nos nouveaux fichiers.

2.2) Cette fonction rend la somme des valeurs ASCII d'une chaîne de caractère.

2.3) Les fonctions sont commentées dans leurs fichiers.

2.4) On utilise une fonction de hachage pour éviter les collisions afin d'améliorer les performances de recherche.

2.5) Code commenté dans son fichier.

2.6) Mêmes fonctions que dans le premier exercice, mais utiliser plus intelligemment grâce la table de hachage.
Les codes sont commentés.

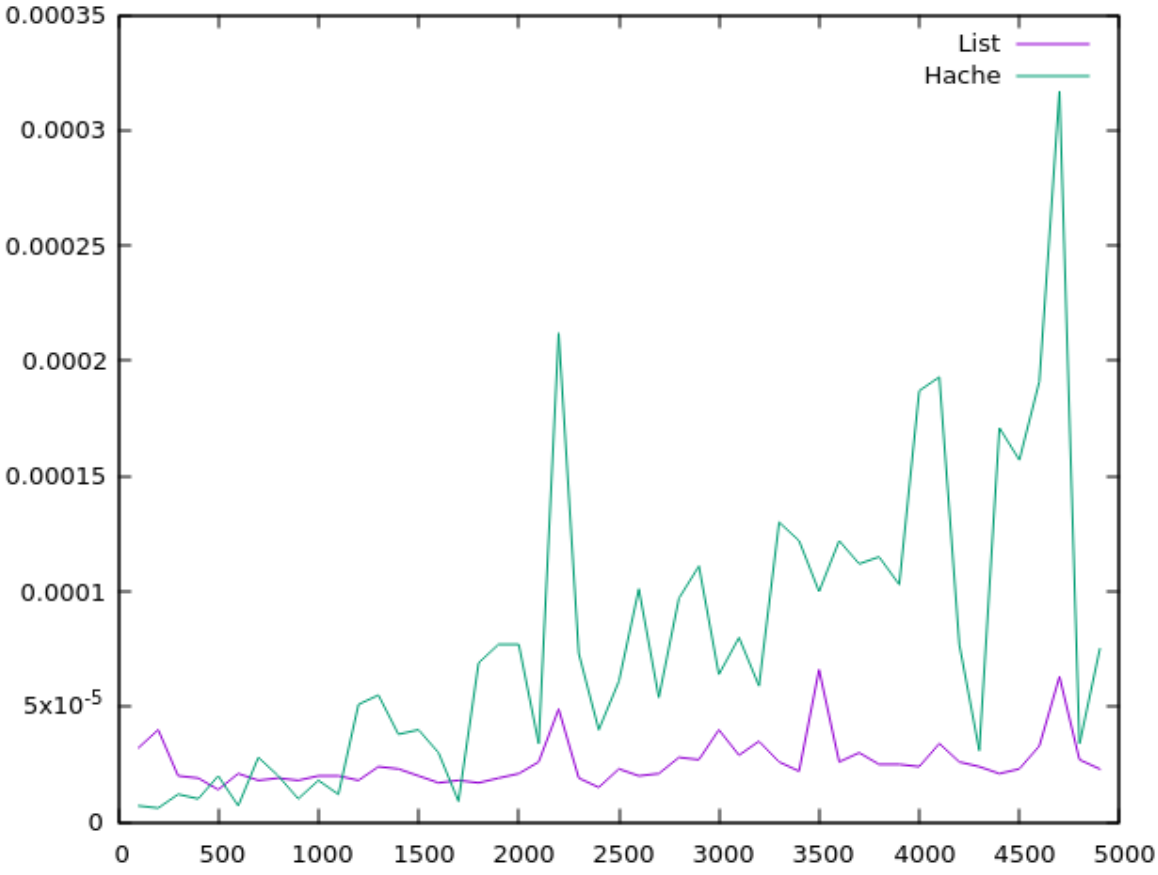
EXERCICE 3 :

Nous allons enfin comparer nos structures en termes de recherche, nous allons chercher des livres dans nos bibliothèques et comparer les temps pour voir laquelle des structures est la plus utile en fonction de la recherche.

Pour les prochaines questions, tout se passe dans /exo3/main.c.
Pour chaque questions on calcule le temps cpu pour les recherches sur nos deux structures, on écrit les résultats dans des fichiers qui portent le nom de leur test. Ces fichiers résultats sont dans exo3/resultats/ .
On en a fait les graphes grâce à gnuplot pour tous ces fichiers, on ne montre ici que les plus intéressants et les plus utiles.

3.1) On calcule les temps cpu pour les fonctions de recherches grâce à clock(). On écrit dans des fichiers les résultats obtenus et on observe les résultats.

recherche_num :



On voit que pour chercher un livre par son numéro il est plus efficace d'utiliser la liste plutôt que la table de hachage.

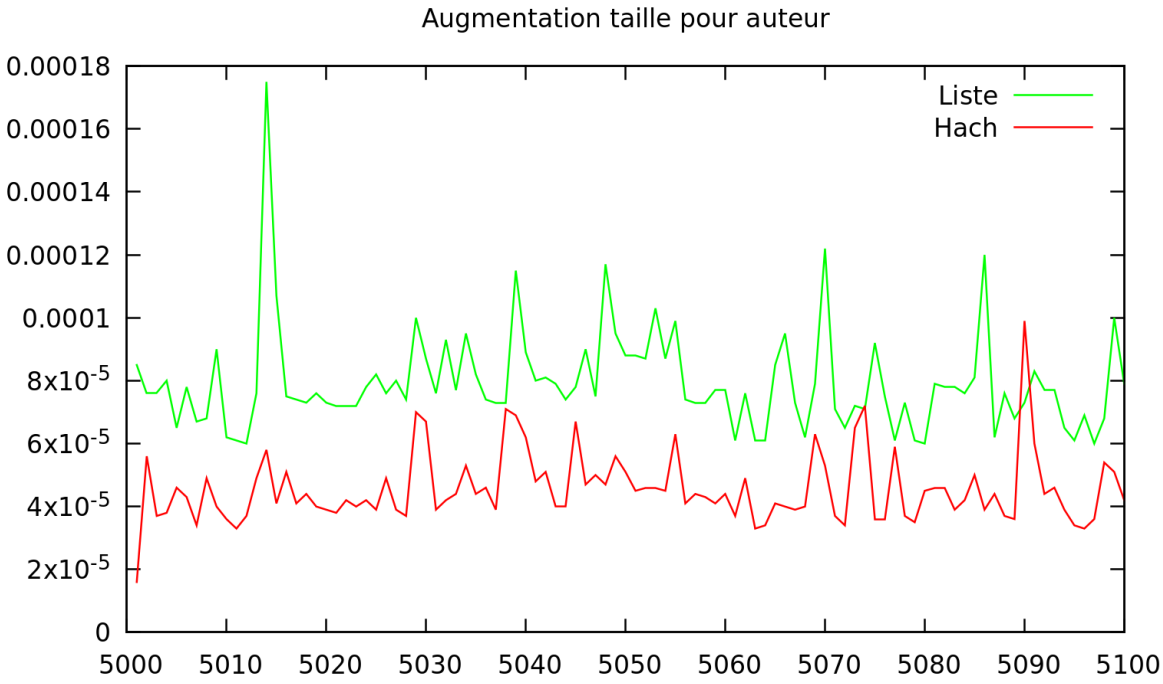
recherche_titre :
Pour chercher un livre par son titre, les fonctions se valent. On obtient presque les mêmes résultats de temps pour les deux structures. C'est normal car on les parcourt de la même manière pour trouver le titre.

recherche_auteur :
La table de hachage est beaucoup plus rapide que la liste car elle trouve directement dans quelle case T[cle] se trouve le livre et parcourt beaucoup moins de listes. On le constate clairement dans le fichier comparaison_auteur.txt, les résultats de la table de hachage sont beaucoup plus rapides que ceux de la liste.

3.2) On a donc diminué et augmenté la taille de notre table de hachage pour les mêmes calculs et on va donc re comparer les résultats.

Pour recherche_num et recherche_titre, les temps ne changent pas vraiment. Car en effet la table de hachage ne prend en compte que l'auteur mais parcourt les titres et les num comme une liste chaînée. Changer de taille ne change donc pas leur temps.

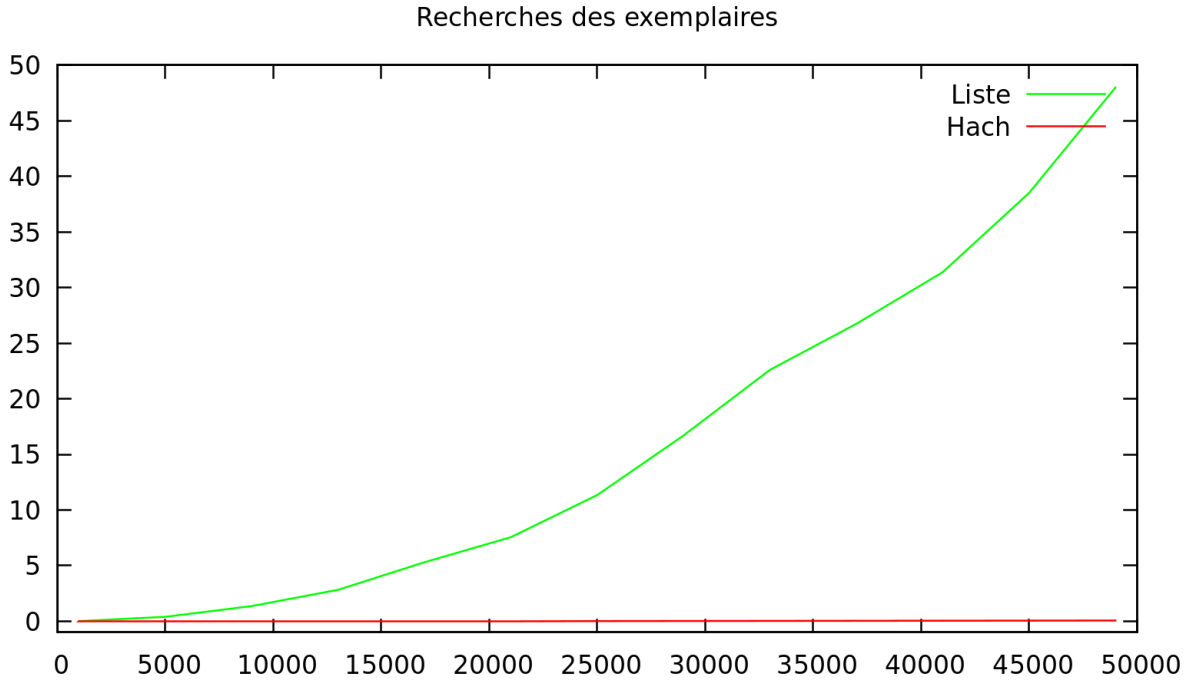
Pour la recherche_auteur, plus on augmente la taille et plus la recherche est rapide grâce à la clé.
Voici le résultat :



On voit donc bien que les résultats de la table de hachage sont meilleurs que ceux de la liste.

3.3) Le main est commenté. Je fais une boucle de 1000 jusqu'à 50 000 avec un pas de 4000 pour que ça ne soit pas trop long quand même. Et à chaque fois je charge les n lignes du fichier, j'enregistre les temps et je libère la mémoire.

On obtient donc :



3.4) La structure de hachage est donc beaucoup plus efficace sur de grandes recherches.
Pire cas liste : si il n'y a pas d'exemplaires on a une complexité de $O(n^2)$
Pire cas table de hachage : complexité de $O(m)$ avec m constant