ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

RAPPORT DE LABORATOIRE 2

PRÉSENTÉ À

SABEUR LAFI

DANS LE CADRE DU COURS :

ALGORITHMES

ELE440-01

PAR

JULIEN LEMAY (LEMJ16059303)

ALEXANDRE LESSARD (LESA30099400)

MONTRÉAL, LE 30 OCTOBRE 2015

Sommaire

1 Introduction

Dans le cadre de ce laboratoire, différents algorithmes de recherches ont été étudiés afin de nous familiariser avec ce type d'algorithme.

Ainsi, l'équipe a réalisé une application C++ afin de pouvoir expérimenter avec les différents algorithmes de recherches et comparer leur différent niveau d'efficacité grâce aux calculs asymptomatique et aux baromètres.

Le rapport suivant est séparé en trois grandes sections. Tout d'abord, il y a la présentation des différents algorithmes de recherches. Ensuite, l'analyse théorique ainsi que les calculs asymptomatiques seront présentées. Finalement, une justification sera faite pour les différents choix heuristiques seront fait par l'équipe pour améliorer les performances du programme.

2 Les algorithmes de recherche

Dans la section suivante, nous expliquerons chacun des algorithmes de recherches utilisés dans le cadre de ce laboratoire. Nous inclurons leur pseudo-code, une petite explication de l'algorithme ainsi que des principales difficultés rencontrées lors de leur implémentation.

2.1 Recherche séquentielle

2.1.1 Pseudo-code

Pour toutes les cases du tableau

    Si c'est la valeur qu'on recherche

        Retourner cette valeur.

    Sinon

        Retourner -1.

2.1.2 Explication

La recherche séquentielle est la plus simple. Elle consiste à tout simplement itérer à travers le tableau de valeurs jusqu'à ce qu'elle trouve la valeur recherchée.

2.1.3 Difficultés

Aucune difficulté n'a été rencontrée lors de l'implémentation de cette fonction.

2.2 Recherche binaire

2.2.1 Pseudo-code

Tant que la valeur du milieu de la plage n'est pas la valeur recherchée

    Si la valeur du milieu de la plage est plus petite que la valeur de la clé

        Le début de la plage devient le milieu + 1

    Sinon

        La fin de la plage devient le milieu – 1

    Le milieu devient la moyenne entre le début et la fin de la plage.

2.2.2 Explication

Cet algorithme coupe constamment le problème en deux ce qui permet d'éliminer beaucoup de données rapidement. Cependant, pour utiliser cet algorithme, il est nécessaire de trier les données avant de commencer la recherche. Son fonctionnement consiste à isoler la valeur entre une limite basse (le début) et une limite haute (la fin). À chaque itération, la fonction compare la clé avec la valeur du milieu entre le début et la fin puis retranche de la plage les nombre plus haut que le milieu si la clé est inférieure à celui-ci et vice versa pour le contraire.

2.2.3 Difficultés

On a eu des difficultés avec nos IDE puisqu'un d'entre nous utilisait les normes de C++ 2011 tandis que l'autre utilisait ceux de 1998, donc lorsqu'on partageait le code ça ne fonctionnait que sur un seul ordinateur à la fois.

2.3 Recherche par d'arbre binaire

2.3.1 Pseudo-code

Tant que le résultat recherché n'est pas atteint

    Si la clé est plus petite que la valeur du nœud

        Descendre au nœud de gauche

    Sinon

        Descendre au nœud de droite

2.3.2 Explication

Le problème avec l'arbre c'est qu'il doit être souvent re-balancé pour avoir la même profondeur sur toutes les branches de l'arbre. Le fonctionnement de cet algorithme est simplement de descendre dans l'hiérarchie de nombre utilisant deux simple règles qui ne changent jamais : si la clé est plus petite que la valeur du nœud, on descend au prochain nœud à gauche et si la clé est plus grande que la valeur du nœud, on descend vers le prochain nœud à droite jusqu'à ce qu'on retrouve la valeur demandée.

2.3.3 Difficultés

Puisqu'un des membres de l'équipe était un peu moins familier avec la programmation orientée objet, le code est inspiré d'un tutoriel en JAVA pour comprendre comment coder l'algorithme. Nous avons eu beaucoup de problèmes avec les pointeurs NULL puisqu'en JAVA, les pointeurs n'existent pas.

2.4 Recherche par table de hachage

2.4.1 Pseudo-code

Garder le restant de la division entre la valeur et un nombre de cases.

> Tant que la case n'est pas vide
>
> > Pointer vers la prochaine maille de la chaine.
>
> Placer l'information à la suite des autres informations dans la case portant le restant de la division.

2.4.2 Explication

Cette fonction de hachage divise simplement la clé par le nombre de case qu'il y a d'alloué et garde le restant de l'opération pour lui attribuer cette case. Si de l'information se trouve déjà dans la case, l'information devient une liste chainée. L'algorithme doit alors itérer à travers tous les mailles de la chaine jusqu'à ce qu'il en trouve une de libre pour y insérer la nouvelle information.

Par la suite lorsqu'on recherche de l'information à partir d'une clé il suffit de passer la clé dans la fonction de hachage et elle vous retournera la case dans laquelle l'information est stockée, cependant il faudra peut être fouillé un peu avant de trouver.

2.4.3 Difficultés

Ayant un peu de difficulté à comprendre le fonctionnement d'une fonction pour table de hachage, nous nous sommes inspirés d'exemples sur internet. Beaucoup de temps aura été perdu ici mais c'était intéressant tout de même. Nous n'avons pas pris le temps d'optimiser la fonction de hachage.

3 L'analyse théorique

Dans la section suivante, l'analyse théorique est faite pour chacun des algorithmes de recherches. Cette section inclut les calculs asymptotiques, les baromètres retenus ainsi que la comparaison entre l'analyse théorique et les résultats obtenus avec notre programme et ce, pour chacun des algorithmes étudiés dans ce laboratoire.

3.1 Recherche séquentielle

3.1.1 Barèmes et formule

O(N)

L'ordre asymptotique de cette recherche est de N puisque le pire cas serait lorsque l'élément recherché se trouve à la toute fin du tableau de données.

3.1.2 Analyse

L'analyse de cet algorithme est assez, simple. Grandement inefficace lorsque le nombre de données est trop grand, mais peut être plus rapide que d'autres lorsque le nombre d'éléments n'est pas très gros puisque les autres fonctions nécessitent du temps pour préparer les données avant de lancer la recherche.

3.2 Recherche binaire

3.2.1 Barèmes et formule

O(Log(N))

Sans compter l'algorithme qui trie les données, la formule asymptotique est Log(N). Ce résultat est déjà beaucoup mieux que la recherche séquentielle mais peut-on trouver mieux. La formule étant plutôt rapide, devrait utiliser l'ordre asymptotique de l'algorithme de tri.

3.2.2 Analyse

Temps de préparation = 310175

Temps de recherche = 22990

Temps total= 333165

Temps amortie= 389

Par l'analyse des données on remarque que le temps de préparation est très important comparé au temps de recherche. On peut donc conclure que le temps de recherche équivaut au temps de trie des données.

3.3 Recherche par d'arbre binaire

3.3.1 Barèmes et formule

O(Log(N))

Comme la recherche binaire l'arbre binaire possède une formule asymptotique de Log(N). La seule chose qui change est le temps de préparation. Puisque les données n'ont pas besoin d'être triés, on gagne du temps, mais ici l'arbre doit être ajusté souvent. La formule asymptotique étant très petite, le temps de préparation peut avoir un grand impact sur cet algorithme.

3.3.2 Analyse

Temps de préparation = 352097

Temps de recherche = 31507

Temps total = 383604

Temps amortie = 448

On remarque que le temps de préparation de l'arbre est ici aussi beaucoup plus grand que le temps de recherche. Il faut donc attribuer ici la formule asymptotique de cette recherche au temps de la construction de l'arbre.

3.4 Recherche par table de hachage

3.4.1 Barèmes et formule

O(N)

La formule asymptotique de cet algorithme est N, cependant cette valeur n'est pas significative puisqu'il suffit d'alloué plus d'espace pour entreposer les données. De cette manière, si le facteur de chargement est de 1, la formule asymptotique est de 1 et si on alloue une seule case pour toutes les données, cet algorithme est aussi efficace que l'algorithme séquentielle puisqu'elle fait exactement la même chose c'est-à-dire, traverser le tableau au complet jusqu'à ce qu'elle tombe sur ce qu'elle cherche.

3.4.2 Analyse

Temps de préparation = 40968

Temps de recherche = 3312

Temps total = 44280

Temps amortie = 51

Ici on peut constater que le temps de recherche est pratiquement négligeable et que le temps de préparation est énormément plus rapide que les autres algorithmes. Nous avons ici une formule gagnante.

4 Choix heuristique

Dans cette section, il sera questions des différents choix heuristiques qui ont été choisis par l'équipe pour améliorer les performances du programme. Cette section inclut la justification des choix heuristiques pour la recherche binaire et la recherche « optimisée ».

4.1 Recherche binaire

Le choix de l'algorithme de tri était clair grâce au laboratoire 1. Lors de ce dernier laboratoire nous sommes arrivés à la conclusion que 'algorithme de tri fusion était le meilleur puisqu'il était capable efficacement de trier n'importe quel données peu importe le nombre, le rang ou le degré de désordre. C'est pour cette raison que nous avons utilisé cet algorithme pour trier les données de la recherche binaire.

4.2 Recherche optimisée

La table de hachage n'a pas été incluse dans cette fonction puisqu'elle est excellente pour trier toutes sortes de données. Son seul défaut est que c'est un algorithme qui prend beaucoup de mémoire.

La recherche séquentielle est beaucoup trop inefficace, mais peut être utile lorsque la charge n'est pas trop grosse.

La recherche par arbre binaire est pratique puisqu'il n'y a pas besoin de trier les données mais on doit tout de même balancer l'arbre. Le nombre de données importe peu et le rang aussi. Par contre, le désordre des données permet à l'arbre se re-balancer moins souvent si celui-ci est à un taux de 50%. C'est pour cela que nous avons choisi l'arbre dans ces situations. Pour le reste du temps, ayant un bon algorithme de tri trouvé à partir du premier laboratoire de ce cours, la recherche binaire est simple mais efficace aussi. Puisque c'est le seul qui reste, pour tout le reste, l'algorithme de tri et l'algorithme de recherche ne semble pas différencier les degrés de désordre, rang ou nombre de données. Sa performance reste stable peut importe ce qu'on lui donne comme charge.

5 Conclusion

En bref, dans ce laboratoire, nous avons utilisé le premier laboratoire sur les algorithmes de trie pour construire des algorithmes de recherche. Quatre algorithmes ont été vus : recherche séquentielle, recherche binaire, recherche par arbre binaire et recherche par table de hachage. La recherche séquentielle étant la plus simple et la plus évidente regarde tous les éléments de la table jusqu'à ce qu'elle trouve ce qu'elle cherche. La recherche binaire coupe le problème en deux à chacune des itérations ce qui permet de sauver beaucoup de temps. Le seul inconvénient est que les données doivent être en ordre avant de commencer la recherche. La recherche par arbre binaire coupe le problème en deux aussi, mais ne nécessite pas que les données soient triées. Cependant l'arbre doit être en premier lieu construit puis balancé. Bien que ces opérations soient plus rapides, ils prennent quand même un temps considérable. Finalement, la table de hachage permet de catégoriser tous les informations dans des cases selon une règle que l'auteur choisit. Comme la recherche binaire, les données doivent être triées dans la table de hachage mais beaucoup plus rapidement. Le seul défaut qui peut être pas si grave la majorité du temps est que cet algorithme demande beaucoup d'espace. La table de hachage est un bon choix d'algorithme dans tous les cas sauf peut-être pour des nombre ridiculement petit de données à chercher. Dans ce cas on utilise la recherche séquentielle. Sinon, pour des degrés de désordre de 50%, nous proposons d'utiliser l'arbre binaire pour minimiser le balancement de l'Arbre. Puis, si rien ne rentre dans ces catégories, utilisez la recherche binaire.

6 Références

Kezakoo (2013, 7 sept) Recherche binaire et recherche linéaire :
https://www.youtube.com/watch?v=MkbxOZAJM_w

Derek Banas (2013, 28 mars), Java Binary Search Tree :
https://www.youtube.com/watch?v=M6lYob8STMI

Paul Programming (2013, 28 mai) How to create a hash table project in C++ :
https://www.youtube.com/watch?v=m6n_rozU8dA