Vincent CHESNEAU
Tanguy HELBERT
M1 MIAGE ALT

UNIVERSITÉ DE NANTES

Unité de Formation et de Recherche

Filière Licence MIAGE

X7IP050 : Compléments en algorithmique

2016-2017

Projet : Tableau d'Arbre Binaire de Recherche

1ère année de master MIAGE

Réalisé par : Vincent CHESNEAU
Tanguy HELBERT

Encadré par : Guillaume FERTIN

12/01/2017

1. Introduction

Au cours du module de compléments en algorithmique nous avons eu à réaliser un projet. Celui-ci consistait à la réalisation d'un programme pouvant gérer plusieurs actions sur un tableau d'Arbre Binaire de Recherche (TABR).

Pour la phase de développement, nous avons analysé et donc conclu que certaines fonctionnalités utiliseraient les mêmes modules. Nous avons donc créé des méthodes qui seront réutilisés dans plusieurs parties de notre programme.

Afin de présenter ce rapport nous allons suivre différents axes. Dans un premier temps, nous parlerons des fonctions de création, de sauvegarde et d'affichage du TABR. Ensuite, nous enchaînerons avec les procédures permettant de manipuler le tableau. Dans un troisième temps, nous expliquerons les différents jeux d'essais que nous avons effectués et nous terminerons par une conclusion.

Pour la suite du rapport, nous allons utiliser les variables suivantes : n définit le nombre de cases du TABR, m le nombre de nœuds dans un ABR contenu dans une case, m' présente également le nombre de nœuds, pour un ABR extérieur au tableau.

2. Génération, sauvegarde et affichage des TABR

2.1 Fichier vers TABR

Cette méthode demande à l'utilisateur de choisir un fichier texte à importer. Une fois cela fait, nous parcourons le fichier, constitué de la façon suivante : intervalle ; ABR, et nous ajoutons dans un premier temps les différentes cases contenant les intervalles de valeurs. Dans un second temps nous parcourons une nouvelle fois la liste, en récupérant cette fois la chaîne décrivant les données de l'ABR.

Il y a donc deux parcours du fichier, cependant, lors de l'ajout des nœuds, nous utilisons la fonction insere de la classe ArbreBinaireRecherche, qui est de complexité n (parcours de l'ABR, de m nœuds, à la recherche de la place du futur nœud). Nous avons donc une complexité O(n * log(m)).

N.B. : La fonction a été divisé en 2 parcours car, initialement, nous vérifions directement et corrigeons les erreurs potentiels du fichier (début et fin interchangés, nœud de la liste à placer dans une autre case, etc…).

2.2 TABR vers fichier et Affichage à l'écran

En ce qui concerne cette fonction, elle réalise l'objectif inverse de la première : nous voulons sauvegarder le TABR en cours dans un fichier nommé par l'utilisateur. Ainsi nous parcouru juste les lignes du tableau et utilisons une fonction retournant la ligne à écrire dans le fichier. Cette dernière utilise la méthode récupérant le parcours suffixe de l'ABR qui le concatène à l'intervalle de la case.

Quant à l'affichage du TABR à l'utilisateur, les lignes, au lieu d'être écrites dans un fichier sont directement affichés à l'écran.

Avec un seul parcours de tableau nous avons une première complexité à n. De même que précédemment, nous parcourons l'ensemble de l'ABR, constitué de m nœuds, ce qui revient à une complexité O(n * m) pour les deux méthodes.

2.3 TABR aléatoire

genereRandomTABR représente la fonction permettant de générer, de manière aléatoire, un TABR à partir de 2 paramètres. Ces paramètres sont demandés à l'utilisateur en deux temps : le premier représentant le nombre de cases à créer, et le deuxième définissant la valeur maximale du tableau. Les intervalles sont définis aléatoirement en s'assurant qu'ils puissent contenir au moins trois valeurs. Suite à cela, nous générons l'arbre avec des valeurs comprises dans les intervalles créées.

Nous parcourons les n cases du TABR, pour ensuite utiliser la fonction de génération aléatoire d'ABR. A l'intérieur de celle-ci, nous parcourons les m éléments et insérons les nouveaux à l'aide de la fonction inserer, interne à la classe de l'arbre. Nous pouvons donc estimer la complexité au pire à O(n * m * log(m)).

2.4 Vérification

La méthode de vérification permet de confirmer si le TABR en cours est correct. Ainsi, nous allons donc vérifier, un à un, les critères exigés. Un rapport est renvoyé au client qui détaille chaque contrôle de contraintes, que ce soit un message de confirmation ou d'alerte.

Dans un souci d'afficher simplement les résultats des contrôles à l'utilisateur, nous avons choisi de parcours autant de fois le TABR qu'il y a de critères. Cependant, seule la vérification de la disjonction des éléments est coûteuse puisque nous allons chercher chacun des nœuds. Cette boucle majorera donc la complexité de cette fonction qui se retrouve être en O(n * m) encore une fois.

3. Manipulation des TABR

3.1 Insertion et suppression d'un entier

La fonction d'insertion et de suppression dans le TABR se réalise en deux temps :

1- Tout d'abord, nous vérifions que l'élément entré en paramètre peut exister dans le TABR. Pour cela, nous venons vérifier son appartenance aux intervalles des cases du tableau. Puis nous travaillons sur l'arbre concerné
2- Ensuite, nous vérifions que l'élément n'existe pas dans le tableau, s'il existe, un message de retour est envoyé à l'utilisateur et la fonction s'arrête
3- Sinon :
    a. Soit nous insérons simplement l'élément dans l'ABR correspondant grâce à la fonction insere
    b. Ou bien nous le supprimons à l'aide de la méthode supprime de la classe de l'ABR

Ce qui revient à avoir une complexité de O(n * log(m)), pour les deux fonctionnalités, puisque nous effectuons des recherches sur un arbre trié.

3.2 Fusion de deux cases du TABR

Pour fusionner deux cases du TABR en cours en une seule, il est demandé à l'utilisateur de rentrer la case qu'il souhaite fusionner avec la suivante. Nous vérifions alors, que l'indice i en paramètre est valide (positif et strictement inférieur à la taille du tableau) et nous procédons à la fusion. Pour cela, nous récupérons les deux cases considérés et affectons au nœud, contenant la plus grande valeur de l'ABR i, la racine, en tant que sous-arbre droit, de l'ABR i+1. Les intervalles des deux cases sont alors fusionnés et il y a décalage des cellules vers la gauche.

Encore une fois, nous retrouvons une complexité en O(n * log(m)) puisque nous devons parcourir le TABR pour effectuer le décalage et que nous devons retrouver le plus grand élément du premier arbre.

3.3 Equilibre en profondeur d'un ABR

De la même manière que pour la fusion, avant de procéder au test de l'équilibrage d'un ABR, nous vérifions d'abord si l'indice rentré est bien valide.

Cela vérifié, nous pouvons donc tester si l'arbre est équilibré en profondeur. Pour y répondre, nous allons juste trouver le premier nœud possédant un déséquilibre. Nous commençons d'abord par tester le déséquilibrage de la racine, et, si les profondeurs de ses fils se compensent, nous testons récursivement le déséquilibrages des sous-arbres. Dès qu'un nœud déséquilibré est trouvé, la fonction s'arrête et renvoie le détail de celui-ci avec les profondeurs de ses fils. Si tous les nœuds sont équilibrés, alors l'utilisateur est informé de ce fait.

Dans cette méthode, nous allons donc parcourir, au pire des cas, l'ensemble des nœuds binaires de l'arbre, tout en allant chercher leur profondeur. Ce qui revient à parcourir deux arbres en plus (les fils gauche et droit). Nous allons donc obtenir O(m*m*m) soit $O(m^3)$.

3.4 ABR vers TABR

La construction d'un TABR provenant d'un ABR est réalisée en plusieurs étapes. Tout d'abord, nous demandons à l'utilisateur de rentrer les valeurs de son ABR, qui est construit au fur et à mesure, sans doublons. Puis, il sera amené à continuer la procédure en définissant tous les maximums des intervalles du TABR qu'il souhaite.

Une fois cela terminé, un tri de cette liste de maximum est effectué et les valeurs incorrectes (négatives ou hors des valeurs min et max de l'ABR) sont retirées. Ensuite, il ne reste plus qu'à créer les cases avec leur intervalle et leur arbre. Pour ce dernier, il faudra lire le parcours suffixe, en partant de la fin, de l'arbre afin d'ajouter tous les éléments dans le TABR avec insertTABR.

La complexité calculée résulte en O(m' * (n * log(m))) avec m' représentant tous les éléments de l'arbre source.

3.5 TABR vers ABR

Cette dernière fonctionnalité permet de fusionner toutes les cases du tableau afin de créer un unique arbre binaire de recherche. Dans un souci de propreté, nous avons estimé que la case se situant au milieu du tableau serait une pseudo médiane des valeurs trouvés dans les ABR de celui-ci. Ainsi, nous commençons par intégrer cet ABR à notre nouvel arbre et nous ajoutons les premières cases, en tant que sous-arbre gauche, et les dernières, en tant que sous-arbre droit.

Une nouvelle fois nous parcours les n cases du TABR et nous suivons le chemin suffixe de l'arbre contenu, tout cela afin d'utiliser la fonction d'insertion pour chacune des valeurs. Nous obtenons alors la complexité O(n * m * log(m')) avec m' représentant le nombre de nœuds dans le nouvel arbre.

4. Jeux d'essais

Afin de valider nos fonctionnalités, nous les avons rigoureusement testés avec des jeux d'essais permettant de simuler plusieurs cas :

```
Creation du TABR par fichier :
TABR construit avec le fichier txt/test.txt

Affichage :
1:6;1:3:2
9:22;11:14:9
50:75;55:75:62:60
78:80;80

Insertion dans TABR (hors interv) :
/// L'element 7 n'est pas compris dans les intervalles du TABR - echec de
l'insertion

Insertion dans TABR (existe) :
/// L'element 3 existe deja dans le tableau

Insertion dans TABR (vrai) :
/// L'element 79 a ete insere dans le tableau :
1:6;1:3:2
9:22;11:14:9
50:75;55:75:62:60
78:80;79:80

Suppresion dans TABR (hors interv) :
/// L'element 7 n'est pas compris dans les intervalles du TABR - echec de la
suppression
Suppression dans TABR (existe pas) :
/// L'element 66 n'existe pas dans le tableau
Suppression dans TABR (vrai) :
/// L'element 14 a ete supprime du tableau :
1:6;1:3:2
9:22;11:9
50:75;55:75:62:60
78:80;79:80

Fusion de deux cases (hors interv)
/// L'indice 6 n'appartient aux bornes du TABR
```

```
Fusion de deux cases (max)
/// Vous ne pouvez pas fusionner la derniere case (pas de case suivante)
Fusion de deux cases (vrai)
/// Les cases 1 et 2 ont bien été fusionnées !
1:6;1:3:2
9:75;55:75:62:60:11:9
78:80;79:80

Equilibre correct
/// L'arbre, stocke a l'indice 2, est equilibre
Equilibre incorrect
/// L'arbre, stocke a l'indice 1, n'est pas equilibre pour le noeud : 9 ;
profondeur du sag : -1 ; profondeur du sad : 4
 deseq(9) = (-1) - (4) = -5

ABR en paramètre :
     13
    /  \
  10   16
 /       \
5         24
         /
       20

Intervalles : [1 ; 17 ; 9 ; 15 ; -1 ; 40]

Creation d'un TABR via un ABR
5:9;5
10:15;10:13
16:17;16
18:24;20:24

Creation d'un ABR regroupant tout le TABR
10:13:5:20:24:16

Creation d'un TABR aleatoire (10,80)
1:5;2:3:1
8:11;9:8:10
15:19;16:15:17
22:29;27:25:22
33:37;34:33:35
37:44;41:40:43:42:39
46:53;46:48:47:51:52:49
59:63;59:62:60
64:68;65:67:66
75:80;76:78:75
```

5. Conclusion

Pour finir nous avons pu développer toutes les fonctionnalités demandées. Notre projet possède une interface logicielle simple d'utilisation pour un utilisateur lambda. Il sera en mesure de profiter aisément des méthodes implémentées. Ce projet a été très instructif car nous avons pu mettre en pratique nos connaissances sur les arbres dans un contexte informatique.

Du fait des différences entre les méthodes et de leur nombre, nous avons pu travailler chacun de notre côté en synchronisant nos travaux à l'aide du gestionnaire de version Git. Cela nous a permis de raffermir notre capacité à développer en équipe.