UNIVERSITÉ DE NANTES

Projet Algorithmique

Tableau d'ABR

Matthieu Gonzalez

Clément Leclercq

2018

Introduction

Dans le cadre du module conception et analyse d'algorithmes nous avons été amené à implémenter une structure de donnée basé sur les arbres binaires de recherches, appelé Tabr qui est un tableau contenant des intervalles. Pour chaque intervalle un arbre binaire de recherche est associé et doit contenir seulement les valeurs de cet intervalle.

Nous avons choisi le langage python pour l'implémentation du Tabr. D'une part car nous nous sentions plus à l'aise en terme de programmation dans ce langage.

D'autre part car nous avions déjà une librairie d'interface graphique grâce au projet d'optimisation pour la gestion.

Enfin car c'est un langage de programmation sur lequel nous souhaitions monter en compétence.

I- Structure de données

Nous avons décidé d'implémenter le Tabr sous la forme d'une list qui est l'équivalent des tableaux dynamique en python.
Cette list est composée de dictionnaires, qui est un tableau clé=>valeur, la clé étant l'intervalle sous la forme d'un tuple (début, fin), un tuple est comme une list mais non modifiable.
On peut accéder aux valeurs d'un tuple de même façon qu'une list en spécifiant l'indice tuple[i].
La valeur des dictionnaires est un arbre binaire de recherche, qui est une classe que nous avons traduit en python car nous avions effectué les tp en java. chaque arbre est composé d'instance de la classe Noeud que nous créé pour représenter un noeud d'un arbre binaire de recherche.

II- Description et complexité

1- Vérification des propriétés du Tabr

Dans un premier temps on crée un tableau qui va nous servir pour stocker les clés des dictionnaires pour vérifier la propriété intervalles disjoint.

On commence par une première boucle qui va nous permettre de parcourir tous les dictionnaires du Tabr.

Dans cette boucle on crée un tableau qui va contenir les entiers de l'arbre binaire de recherche de l'intervalle.

Dans cette boucle on ajoute les clés à la list créée, et on vérifie que clé[0]<= clé[1], si ce n'est pas le cas on passe un booléen à faux.

Puis on effectue un parcours préfixe pour récupérer tous les entiers de l'arbre courant que l'on stock dans le tableau.

Une fois cette liste récupérer on boucle dans cette liste pour voir si les entiers sont compris dans l'intervalle des clés du dictionnaire qui contient l'arbre si tel n'est pas le cas un booléen est passé à faux.

Enfin une fois ce parcours fini, on parcours le tableau des clés des dictionnaires que nous avons récupéré de 0 à n-1, n étant le nombre de clés. On vérifie que la valeur clé[1] à l'indice courant est inférieur à la valeur clé[0] de l'indice n+1. Si tel n'est pas le cas alors le Tabr est disjoint et n'est pas en ordre croissant.

On est contraint de parcourir toute le Tabr afin de vérifier toutes les propriétés.

Complexité :

Soit N le nombre de dictionnaires du Tabr

Soit M le nombre d'entiers contenu dans un arbre d'un dictionnaire

Complexité:

Parcours du Tabr de N dictionnaires soit $\Theta(N)$.

Pour chaque dictionnaire on effectue un parcours préfixe de l'arbre contenu qui est en $\Theta(M)$.

Puis on effectue un parcours de la liste des entiers récupérée par le parcours préfixe en $\Theta(M)$.

Enfin on effectue un parcours des clés des dictionnaires du Tabr qui est $\Theta(N)$.

Compléxité => 2N x 2E Soit $\Theta(N \times E)$.

2- Fichier vers Tabr

On commence par créer une instance de classe Tabr.

On récupérer le contenu du fichier puis on le découpe par ligne.

Pour chaque ligne on fractionne la ligne en deux à partir du caractère (";") le côté gauche correspond aux clés et le droit aux valeurs à insérer.

Puis on crée un dictionnaire avec comme clé les valeurs de gauche, on crée un arbre et pour chaque valeur récupéré des lignes on procède à une insertion dans l'arbre.

Complexité:

Soit N le nombre de lignes.

Soit M le nombre d'entiers à insérer.

Une création d'arbre est au mieux en Ω(M log M) la forme des données associées est un arbre complet

Au pire la création d'un arbre binaire de recherche est en O(M^2) la forme associé est un arbre filiforme. En moyenne la création est en O(M log M).

Complexité => O (N*M log M)

3- Tabr vers fichiers et vers écran

Pour chaque dictionnaire dans le Tabr, on effectue un parcours préfixe qui retourne la liste des entiers du dictionnaire. Puis on écrit cette liste dans le fichier.

Complexité :

Soit N le nombre de lignes

Soit M le nombre d'entiers à insérer

Parcours du Tabr en Θ(N), parcours préfixe en Θ(E). On fait N x E parcours préfixe et le temps d'écriture est constant car on écrit directement la liste sans la parcourir.

Complexité => Θ(NxE).

4- Insertion d'un entier

Soit N le nombre d'indice du Tabr
Soit M les éléments d'un arbre binaire de recherche du Tabr

On effectue un parcours du Tabr pour trouver l'intervalle qui contient le nombre à insérer. Si cet intervalle existe alors on fait une insertion.
Parcours du Tabr au mieux : O(1) lorsque la valeur à insérer est dans le premier intervalle du Tabr.
Parcours du Tabr au pire : O(N): La valeur à insérer est dans le dernier intervalle ou la valeur à insérer n'est dans aucun intervalle.

Insertion dans un arbre binaire de recherche est au mieux en O(1) lorsque l'insertion se fait dans un arbre binaire de recherche vide. Au pire l'insertion est en O(M), insertion dans un arbre filiforme. En moyenne l'insertion est en O(log M).

<u>Complexité:</u>
Au pire l'insertion d'un entier dans le Tabr => O(N x M)
Complexité au mieux O(1), lorsque la valeur à insérer est dans le premier intervalle et l'arbre binaire de recherche de cet intervalle est vide.

5- Suppression d'un entier

Pour supprimer un entier il faut parcourir le Tabr pour trouver l'intervalle, puis si l'intervalle contient l'entier à supprimer, on fait une recherche dans l'arbre binaire de recherche pour voir si la valeur est dans l'arbre. Si la valeur est dans l'arbre, on fait appel à la fonction suppression.

Complexité :

Soit N le nombre d'indice du Tabr.
Soit M les éléments d'un arbre binaire de recherche du Tabr.

Parcours du Tabr : au mieux O(1) la valeur à supprimer et dans le premier intervalle au pire O(N).
Recherche au mieux en O(1), au pire en O(M) et en moyenne O(log M)
Suppression au mieux en O(1) au pire en O(M) et en moyenne O(log M)
Suppression d'un entier: au mieux en O(1) lorsque la valeur à supprimer est dans le premier intervalle et est la racine de l'arbre binaire de recherche de cet intervalle.
Au pire la suppression se fait en O(NxM) lorsque la valeur à supprimer est dans le dernier intervalle et est une feuille de l'arbre binaire de recherche.
En moyenne la suppression se fait en O(NxLog M)

6 - Fusion de deux cases du Tabr

On construit la clé de la nouvelle case en prenant la première valeur de la clé de l'index passé en paramètre et la deuxième valeur de la clé index +1.
On parcourt les deux arbres pour récupérer tous les entiers contenu dans ces arbres que l'on stock dans un tableau.
On crée un nouvel arbre binaire de recherche avec les valeurs contenu dans le tableau. La racine étant la racine de l'arbre de l'index passé passé en paramètre.
On supprime les deux cases et on ajoute la nouvelle case, puis on tri le Tabr sur les clés.

Complexité:
Soit M la liste des entiers à insérer.
Soit N le nombre d'indice du Tabr.

Accès à l 'indexe souhaité et suppression des deux cases en O(1)
Parcours des arbres pour récupérer les entiers en Θ(M)
Une création d'arbre binaire de recherche est au mieux en $\mathbf{\Omega}$(M log M) la forme des données associées est un arbre complet.
Au pire la création d'un arbre binaire de recherche est en O(M^2) la forme associé est un arbre filiforme. En moyenne la création est en O(M log M).

Tri du Tabr par ordre croissant de la valeur des clés en O(N log N)

Complexité => O(M log M + N log N)

7-Tabr vers Abr

Soit M la liste des entiers à insérer.

Soit N le nombre d'indice du Tabr.

On parcourt tout le Tabr, on récupère les valeurs de tous les entiers de tous les arbres.

On supprime le Tabr.

On crée un arbre à partir du tableau des valeurs

Complexité :

Parcours du Tabr en Θ(N).

Récupération des valeurs d'un arbre binaire de recherche en O(M)

Suppression du Tabr en O(1)

Création de l'arbre binaire à partir de toutes les valeurs récupérées :

Une création d'arbre est au mieux en Ω(M log M) la forme des données associées est un arbre complet.

Au pire la création d'un arbre binaire de recherche est en O(M^2) la forme associé est un arbre filiforme. En moyenne la création est en O(M log M).

Complexité => O(N)+O(N x M) + O(1) + O(M log M)

On prend le terme dominant ici O(N x M)

Conclusion

Ce projet nous as permis d’apprendre à manier la récursivité en Python, et les arbres binaires de recherche. Nous avons pu manipuler également les dictionnaires et les tuples ainsi que le programmation orientée objet.