Algorithmique TD6
L3 Informatique – ENS Cachan

14 août 2020

Exercice 1

On considère dans cet exercice des arbre binaires de recherche. En notant size(x) le nombre de nœuds d'un arbre enraciné en x, left(x) son fils gauche et right(x) son fils droit, on dit qu'un noeud est α-*équilibré* si :

$$\text{size}(\text{left}(x)) \leq \lceil \alpha(\text{size}(x) - 1) \rceil$$

et

$$\text{size}(\text{right}(x)) \leq \lceil \alpha(\text{size}(x) - 1) \rceil$$

Un arbre binaire est dit α-équilibré ssi tout nœud de l'arbre est α-équilibré.

1. Étant donné un noeud x d'un arbre, donner un algorithme pour rééquilibrer le sous-arbre enraciné en x afin d'obtenir un sous-arbre $\frac{1}{2}$-équilibré. L'algorithme devra avoir une complexité en temps et en espace linéaire en le nombre d'éléments du sous-arbre enraciné en x
2. Donner un algorithme de recherche dans un arbre binaire α-équilibré. Montrer que cet algorithme a une complexité en temps logarithmique en le nombre d'éléments de l'arbre.

On considère dans le reste du problème un $\alpha > \frac{1}{2}$. Lors de l'ajout et de la suppression d'éléments, on procède comme dans un arbre binaire de recherche habituel, sauf dans le cas ou l'on se retrouverait avec un sous-arbre qui n'est plus équilibré, auquel cas, on rééquilibre le sous-arbre maximal non équilibré avec la procédure de la question 1.

3. Implémenter l'algorithme d'insertion et de suppression d'un élément, et montrer qu'il conserve le α équilibrage d'un arbre.

On définit maintenant :

$$\Delta(x) = |\text{size}(\text{left}(x)) - \text{size}(\text{right}(x))|$$

et le potentiel d'un arbre par :

$$\Phi(t) = c \sum_{x \in T : \Delta(x) \geq 2} \Delta(x)$$

où c est une constante qu'il faudra déterminer.

4. Montrer qu'un arbre $\frac{1}{2}$-équilibré possède un potentiel nul.
5. Supposons que le rééquilibrage d'un arbre à m éléments un coût en temps exactement égal à m. Trouver une valeur de c telle que le coût amorti du rééquilibrage d'un arbre non α-équilibré soit en $O(1)$.
6. Montrer que la l'ajout et la suppression d'un élément dans un arbre α-équilibré a une complexité amortie en $O(\log(n))$ ou n est le nombre d'éléments de l'arbre.

Exercice 2

On s'intéresse à l'opération de jointure sur les arbres rouge-noir. Elle prend en entrée deux arbres rouge-noir T_1 et T_2, et un enregistrement x tels que pour tout nœuds x_1 de T_1 et x_2 de T_2, $x_1.clé \leq x.clé \leq x_2.clé$. Elle retourne un arbre rouge-noir T correspondant à l'union de T_1, x, et T_2.

1. On décide de conserver la hauteur noire d'un arbre rouge-noir T dans un champ $T.hb$. Montrer que ce champ peut être mis à jour par les opérations d'insertion et de suppression sans utiliser d'espace supplémentaire dans les nœuds de l'arbre ni augmenter la complexité asymptotique.

On souhaite implémenter l'opération Jointure(T_1, x, T_2). On note n le nombre total de nœuds dans T_1 et T_2.

2. On suppose que $T_1.hb \geq T_2.hb$. Décrire un algorithme en temps $O(\log(n))$ qui trouve un nœud noir y dans T_1 ayant une clé maximale parmi tous les nœuds noirs de hauteur noire $T_2.hb$.
3. Décrire un algorithme en temps $O(\log(n))$ qui réalise la jointure de T_1, x et T_2, en distinguant les cas $T_1.hb \geq T_2.hb$ et $T_1.hb < T_2.hb$.

Exercice 3

On se propose d'étudier les arbres splay (« Splay trees »), qui sont des arbres binaires de recherche qui "s'autoéquilibrent", en ramenant les éléments fréquemment accédés près de la racine de l'arbre. Afin de faire cela on introduit une opération, appelée « splay », qui prend un arbre A et un nœud x, et fais remonter x pour qu'il devienne la racine de l'arbre. L'opération applique récursivement des transformations locales qui ont pour but de faire remonter x petit à petit.

1. Distinguer trois cas différents (modulo symétries), et proposer une transformation locale qui fait remonter x pour chacun de ces cas, en utilisant des rotations simples ou doubles.
2. Expliquer comment implémenter les opérations suivantes en utilisant l'opération de « splay » :
   - Insérer un élément (à la fin, le nouvel élément doit être à la racine de l'arbre)
   - Supprimer un élément
   - Fusionner deux arbres splay
   - Séparer un arbre splay en deux : étant donné un nœud x de l'arbre, il faut rendre un arbre qui contient les éléments (strictement) plus petits que x, et un autre qui contient les éléments (strictement) plus grand que x.
3. On veut maintenant analyser la complexité amortie de opérations ci-dessus. On définit pour cela les notions suivantes :
   - La taille d'un nœud x : size(x) est le nombre de nœuds du sous arbre qui a r pour racine.
   - Le rang d'un nœud x : $\text{rank}(x) = \log_2(\text{size}(x))$

   Trouver une fonction de potentiel telle que l'opération de « splay » au nœud x ait un coût amorti inférieur à $3(\text{rank}(r) - \text{rank}(x)) + 1$, où r est la racine de l'arbre (on comptera uniquement le nombre de rotations effectuées).
4. En déduire la complexité amortie d'une suite d'opérations quelconques.