IFT339 - Exercices sur les arbres

Dans les questions qui suivent, on suppose qu'un arbre T est représenté par un objet Noeud*, qui est la racine de l'arbre. Si T est binaire, l'objet noeud a deux enfants gauche, droit. Si T n'est pas nécessairement binaire, les enfants d'un noeud sont dans un vector<Noeud*>. On suppose aussi que chaque noeud connaît son parent.

Exercice 1.
Supposons que la classe Noeud possède une variable membre hauteur, représentant la hauteur du noeud.

a) En supposant qu'un arbre binaire T est donné, donnez le code d'un fonction qui affecte la variable hauteur de chaque noeud correctement. Quelle est la complexité de votre algorithme?

b) Si T n'est pas nécessairement binaire, que devez-vous modifier pour implémenter la même fonction? Quelle est la complexité de cet algorithme modifié?

Exercice 2.

a) Donnez le code d'une fonction qui reçoit un arbre T et qui détermine si T est un ABR ou non.

b) Donnez le code d'une fonction qui détermine si T est un arbre AVL ou non.

Exercice 3.
Donnez une version itérative, donc non-récursive, de la fonction contient d'un ABR.

Exercice 4.
Montrez qu'avec un arbre AVL, on peut trier une liste de n éléments en temps $O(n \log n)$.

Exercice 5.
Vous avez une liste d'entiers dans un vecteur vec, et vous voulez les éléments uniques de vec. Vous devez donc produire une liste dans laquelle chaque nombre présent dans vec se trouve une seule fois dans la liste (donc sans doublons). Par exemple, si vec = [1,5,3,2,1,3,5,4] alors on voudrait en sortie [1,5,3,2,4] (l'ordre des éléments de sortie n'est toutefois pas important).

Avec ce qu'on a vu en classe, ceci peut se faire en temps $O(n \log n)$, avec n le nombre d'éléments de vec. Montrez comment.

Exercice 6.
Une métrique parfois utilisée pour mesurer le débalancement d'un arbre possiblement non-binaire est le *poids en profondeur*.

Soit T un arbre et v un de ses noeuds. La profondeur de v, dénotée $prof(v)$, est le nombre d'arêtes à parcourir pour passer de v à la racine (donc la racine a une profondeur de 0, ses enfants on une profondeur de 1, ses petits-enfants de 2, etc.). On définit le poids de profondeur de T, dénoté $PP(T)$, comme la somme des profondeurs de tous ses noeuds :

$$PP(T) = \sum_{v \in T} prof(v)$$

Où $v \in T$ signifie « v est un nœud de T ».

a) Donnez le code d'un algorithme qui calcule $PP(T)$.

b) Si T est un arbre avec n noeuds, quel est le minimum de $PP(T)$? Quel est le maximum? Donnez des exemples d'arbres qui atteignent le minimum et le maximum.

Exercice 7.
Dans un *parcours en profondeur* d'un arbre T, on commence par visiter la racine, puis on répète la règle suivante jusqu'à ce que tous les nœuds aient été visités : *« parmi tous les nœuds non-visités dont le parent est visité, on visite le nœud le plus profond »*

Montrez comment implémenter un parcours en profondeur.

Exercice 8.
Dans un *parcours en largeur*, on fait le contraire de l'exercice précédent. C'est-à-dire, on commence par visiter la racine, puis on répète le règle suivante : *« parmi tous les nœuds non-visités qui ont un parent visité, on visite le nœud le moins profond ».*

En fait, ce parcours commence par la racine, puis visite tous ses enfants, puis lorsque c'est fait, visite tous ses petits-enfants, puis ses petits-petits-enfants, etc.

Montrez comment implémenter le parcours en profondeur et le parcours en largeur d'un arbre.

Exercice 9.
Soit T un ABR. Implémentez une fonction rebalancer, qui créé un nouvel ABR dont la hauteur est $O(\log n)$. J'y arrive en temps $O(n \log n)$.

Suggestion. Triez les éléments de T, puis recréez un nouvel ABR en y insérant les valeurs dans un ordre optimal. Pour voir comment faire ça, pratiquez-vous à trouver un ordre d'insertion de 1-2-3-4-5-6-7 qui minimise la hauteur.

Exercice 10.
Soit un nœud x d'un arbre T dans lequel chaque nœud a une valeur différente. On dénote par $c(x)$ l'ensemble des valeurs qui apparaissent dans les nœuds du sous-arbre enraciné en x. Supposons que l'on vous donne un ensemble de valeurs C (vous pouvez supposer que C est un vector, un array, une list, à vous de choisir...) et que l'on vous demande *« est-ce qu'il existe un nœud x dans T tel que $c(x) = C$ »*? En d'autres termes, est-ce qu'il y a un sous-arbre qui contient exactement les valeurs de C?

a) Implémentez une fonction qui retourne un tel nœud x s'il existe, ou faux sinon. Si n est la taille de T et m la taille de C, quelle est la complexité?

Ce n'est pas trop difficile d'y arriver en temps $O(n^2 \log m)$. Pouvez-vous atteindre $O(n \log m)$?

b) Implémentez la même fonction, mais dans le cas où T est un ABR. Mon implémentation nécessite un temps $O(h \log m + m \log m)$, où h est la hauteur de T.