TD n°3

Arbre Binaire de Recherche (suite et fin ?)

Exercice 1 [Suppression dans un ABR]

Nous avons observé lors du TD précédent, qu'il est possible de supprimer un nœud d'un arbre binaire de recherche. Le but de cet exercice est de concevoir et de comparer deux algorithmes de suppression.

Soit x le noeud que l'on cherche à supprimer. Il n'est pas compliqué de supprimer x s'il ne possède ni fils gauche, ni fils droit—et si x ne possède qu'un seul fils, il suffit de le remplacer par son fils. Les deux algorithmes étudiés dans cet exercice diffèrent dans la manière dont il supprime un nœud possédant deux fils.

Le premier algorithme cherche à déplacer entièrement le sous-arbre gauche de x dans le sous-arbre droit de x (ou inversement).

- À quel(s) nœud(s) du sous-arbre droit de x est-il possible de rattacher le sous-arbre gauche de x ?
- Dans le pire des cas quel est le rapport entre la hauteur de l'arbre avant et après suppresssion ?
- Écrivez cet algorithme et calculez sa complexité en moyenne et dans le pire des cas.

Le second algorithme cherche à ne pas déséquilibrer inutilement l'arbre après suppression. Si l'on nomme T l'arbre original et T' celui obtebu après suppresion, il est possible d'obtenir la propriété suivante :

$$hauteur(T) - 1 \leqslant hauteur(T') \leqslant hauteur(T)$$

- Écrivez un algorithme de suppression utilisant la fonction `Node Successeur(x : Node)` (ou la fonction `Node Predecesseur(x : Node)`).
- Montrez que votre algorithme vérifie la propriété ci-dessus.
- Calculez sa complexité en moyenne et dans le pire des cas.

Exercice 2 [Petite pause]

Écrivez un algorithme, qui donne le nombre de nœud d'un arbre T. Quelle est sa complexité, dans le pire des cas et en moyenne ?

Exercice 3 [Insertion à la racine]

Dans un arbre binaire de recherche, avec la méthode d'insertion classique, toutes les nouvelles valeurs sont placées aux feuilles de l'arbre. Si l'on souhaite accéder à un nœud insérer récemment dans l'arbre, il faudra parcourir toute la hauteur de l'arbre. Dans certaines applications, on souhaite accéder plus fréquemment aux derniers éléments insérés. Il s'agit du principe LRU[1]. Dans ce cas particulier, il peut-être intéressant d'insérer à la racine. En procédant de la sorte, les valeurs auxquelles on souhaite accéder le plus souvent ont plus de change d'être à une hauteur faible.

En considérant l'exemple de la figure 1 :

- tracez le chemin qui va de la racine au nœud où classiquement il faudrait insérer le nœud 33.
- si on enlève les arêtes de ce chemin, quelles sont les caractéristiques des sous arbres restants ?
- que peux-t'on dire de l'ordre dans lequel l'on rencontre sur ce chemin les nœuds dont l'étiquette est plus petite que 33 ? Et de l'ordre de ceux dont l'étiquette est plus grande que 33 ?

[1] pour Least Recently Used !

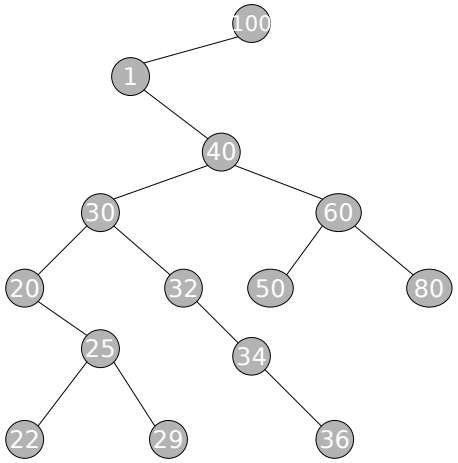

FIG. 1 – un arbre binaire de recherche

À partir des réponses aux questions précédentes, proposez un algorithme d'insertion à la racine dans un arbre binaire de recherche.

Indication : Il est possible d'utiliser deux listes de sous arbres, une pour les "plus grands" et une autre pour les "plus petits".

Évaluez la complexité de cet algorithme.

Exercice 4 [Petite pause bis]

Quelle pourrait être la compléxité d'une fonction qui teste si un arbre binaire étiquetté par des entiers est un arbre binaire de recherche ?

Exercice 5 [Accès au k-ième élément]

Étant donné un arbre binaire de recherche on souhaite connaître la valeur du $k^{\text{ième}}$-élément.

1. Combien de temps cela prend-il de trouver le $k^{\text{ième}}$-élément dans un tableau de n entiers triés ?
2. Proposez un méthode simple qui utilise un parcours et une structure auxilliaire pour trouver le k-ième de l'arbre. Quelle est la complexité de cette méthode ?
3. Proposez un méthode qui utilise n'utilise pas de structure auxilliaire pour trouver le k-ième de l'arbre. Quelle est la complexité de cette méthode ? (*Indication : Vous pourrez-vous aider de la fonction définit à l'exercice 2*)
4. En acceptant de modifier légèrement la structure des arbres binaires, pouvez-vous proposer une méthode avec une meilleur complexité ? Vous vérifirez que votre modification ne modifie pas la compléxité des fonctions d'insertion et de suppression d'un nœud dans un ABR.