Chap 08. Arbres Binaires de Recherche

Livre p 157 – Chap 9 Arbres binaires de recherche

1. Les Arbres Binaires de Recherche

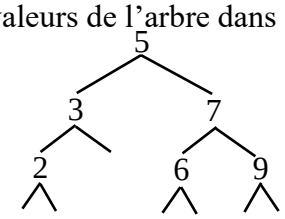
a. Définition

Un **ABR : Arbre Binaire de Recherche** en français (BST : Binary Search Tree en anglais**),** est un arbre binaire dont toutes les valeurs sont ordonnées en respectant la règle suivante pour tous les nœuds non vides :

- Toutes les valeurs du sous-arbre gauche sont inférieures ou égales à la valeur du nœud considéré.
- Toutes les valeurs du sous-arbre droit sont supérieures ou égales à la valeur du nœud considéré.

Remarque : Dans certains cas, on peut interdire la possibilité d'avoir des valeurs égales dans l'ABR.
Remarque : La parcours infixe d'un ABR renvoie la liste des valeurs de l'arbre dans l'ordre croissant.
Exemple :

- Parcours **infixe** : 2 – 3 – 5 – 6 – 7 – 9
  plus précisément : (((, 2,) 3,), 5, ((, 6,), 7, (, 9,)))



b. Recherche

Comme son nom l'indique, la principale utilisation d'un ABR est d'effectuer des recherches !
Une recherche dans un ABR ressemble à une recherche dichotomique :

- Si la racine est vide : La recherche aboutit à un échec.
- Si la valeur est égale à celle située à la racine : La recherche aboutit à un succès.
- Si la valeur est inférieure à celle située à la racine : On continue, de manière récursive, par une recherche dans le sous-arbre gauche.
- Si la valeur est supérieure à celle située à la racine : On continue, de manière récursive, par une recherche dans le sous-arbre droit.

Remarque ; Cette méthode de recherche est optimale lorsque l'arbre est complet, ou au moins équilibré. Le pire cas est celui où l'arbre est dégénéré !

c. Insertion

Pour insérer une nouvelle valeur dans un ABR, le même type d'algorithme :

- Si la racine est vide : On insère la valeur ici en créant un nouveau nœud contenant cette valeur.
- Si la valeur est inférieure à celle située à la racine : On continue, de manière récursive, à essayer d'insérer la valeur dans le sous-arbre gauche.
- Si la valeur est supérieure ou égale à celle située à la racine : On continue, de manière récursive, à essayer d'insérer la valeur dans le sous-arbre gauche.

Remarque : Le choix des inégalités larges et strictes est arbitraire. Le mieux est d'essayer d'équilibrer l'arbre au maximum…

d. Suppression

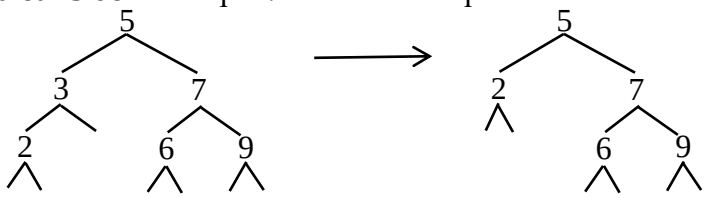
Pour supprimer une valeur dans un ABR, on commence par parcourir l'arbre afin de trouver où se situe le nœud à supprimer puis :

- Si le nœud est une feuille : il suffit de supprimer ce nœud !
- Si le nœud n'a qu'un fils : il suffit de remplacer ce nœud par son fils (en gardant intacte toute la descendance s'il y en a une)
- Si le nœud possède deux fils : il suffit de remplacer la valeur de ce nœud par la valeur de la feuille du fils gauche ayant la plus grande valeur ou du fils droit ayant la plus petite valeur.

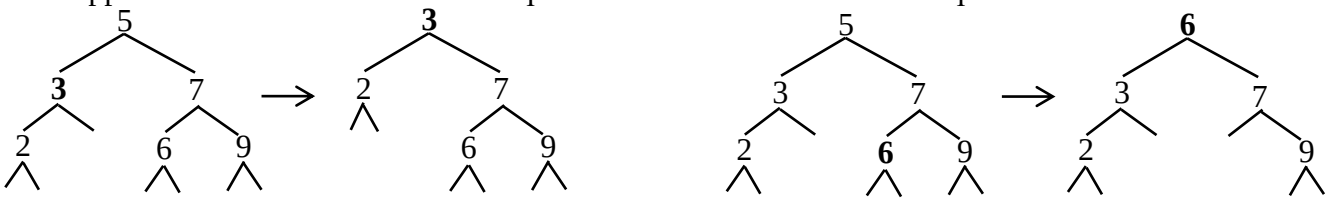
Remarque : Dans le cas de deux fils, le choix est arbitraire. Le mieux est d'essayer d'équilibrer l'arbre au maximum ou d'alterner le plus grand descendant à gauche avec le plus petit descendant à droite…

Exemples :

- Suppression de la valeur 3 de l'exemple : Ce nœud n'a qu'un fils…



- Suppression de la valeur 5 de l'exemple : Ce nœud a deux fils… deux possibilités !



Maximum du fils gauche ou Minimum du fils droit

2. Implémentation en Python

En python, on peut reprendre les objets **Noeud** et **Arbre** du chapitre précédent ou recréer un objet **ABR** :

- Recherche d'une valeur dans un ABR nommé tree.
  Exemple : valeur in tree
- Ajout d'une valeur dans un ABR nommé tree.
  Exemple : tree.ajouter(valeur)

```
class Noeud:
    """un noeud d'un arbre binaire"""
    def __init__(self, g, v, d):
        self.gauche = g
        self.valeur = v
        self.droit  = d

def appartient(x, a):
    """détermine si x apparaît dans l'ABR a"""
    if a is None:
        return False
    if x < a.valeur:
        return appartient(x, a.gauche)
    elif x > a.valeur:
        return appartient(x, a.droit)
    else:
        return True

def ajoute(x, a):
    """ajoute x à l'arbre a, renvoie un nouvel arbre"""
    if a is None:
        return Noeud(None, x, None)
    if x < a.valeur:
        return Noeud(ajoute(x, a.gauche), a.valeur, a.droit)
    else:
        return Noeud(a.gauche, a.valeur, ajoute(x, a.droit))
```

```
class ABR:
    """un arbre binaire de recherche"""
    def __init__(self):
        self.racine = None

    def __contains__(self, x) :
        return appartient(x, self.racine)

    def ajouter(self, x):
        self.racine = ajoute(x, self.racine)
```