Chapitre 13 - Les arbres binaires de recherche

Objectifs :

- ▷ Connaître le principe d'un arbre binaire de recherche
- ▷ Rechercher une clé dans un arbre de recherche, insérer une clé.

1 Introduction

Les **arbres binaires de recherche** (notés souvent **ABR**) sont des structures de données fondamentales en informatique et en algorithmique. Ils sont largement utilisés dans de nombreuses applications informatiques pour leur efficacité dans la recherche, l'insertion et la suppression de données.

On les utilise par exemple pour :

- ▷ l'implémentation des bases de données indexées
- ▷ les systèmes de gestion de fichiers
- ▷ les algorithmes de recherche et de tri
- ▷ les structures de recherche dans les compilateurs et analyseurs syntaxiques

2 Principe de l'ABR

Un **arbre binaire de recherche (ABR)** est une structure de donnée composée de noeuds (on parlera aussi de **clés**).

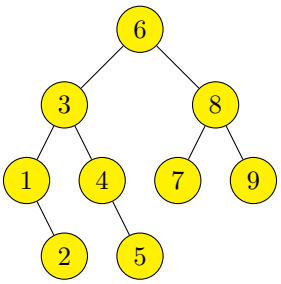
A retenir !

Chaque noeud a au plus 2 enfants ordonnés d'une manière particulière :

- les enfants **à gauche** d'un noeud ont des valeurs **inférieures ou égales** à celle du noeud.
- les enfants **à droite** d'un noeud ont des valeurs **strictement supérieures** à celle du noeud.

Et cela doit être vrai pour chaque noeud de l'arbre.

Par exemple :

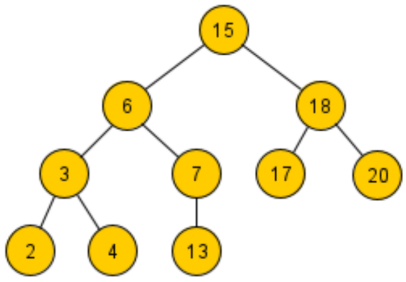


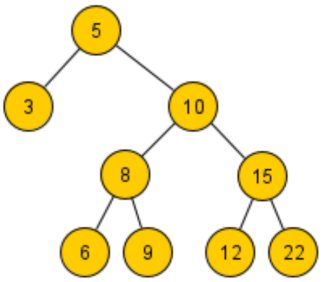
Les ABR sont utiles dans le cas où l'on effectue des **insertions** ou **suppressions** fréquentes d'éléments ou des **recherches** fréquentes.

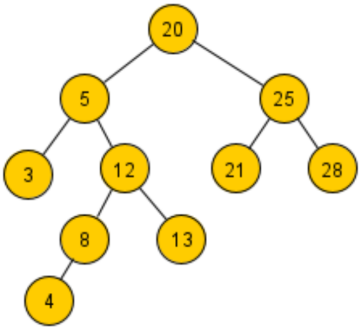
A retenir !

On parlera aussi d'un **arbre équilibré** s'il s'agit d'un arbre binaire tel que les deux sous-arbres partant de chaque nœud autre qu'une feuille ont **des profondeurs différant au maximum de 1**.

Exercice 1 : Préciser si les arbres ci-dessous sont des **ABR** ou pas et s'il s'agit d'**arbres équilibrés**.







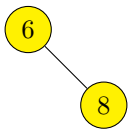
Réponse :

3 Comment construire un ABR ?

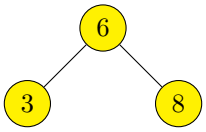
Par exemple, on souhaite représenter la liste de nombres `[6,8,3,1,4,9,2,7,5]`. On va suivre les étapes ci-dessous en respectant les **règles d'un ABR** édictées précédemment.

Dans un premier temps on crée la **racine** de l'arbre :

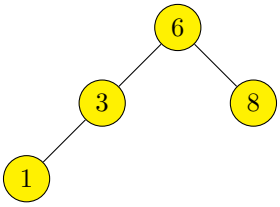
Puis, on insère les noeuds successifs en commençant par le **noeud 8** :



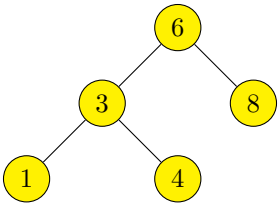
Puis, on insère le **noeud 3** :



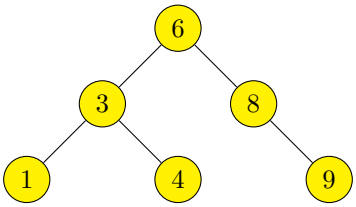
Puis, on insère le **noeud 1** :



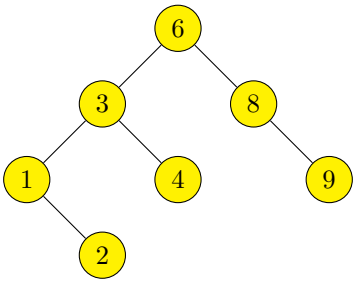
Puis, on insère le **noeud 4** :



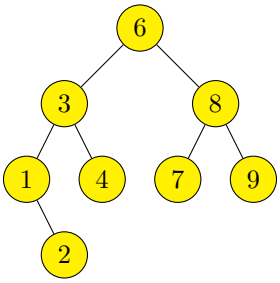
Puis, on insère le **noeud 9** :



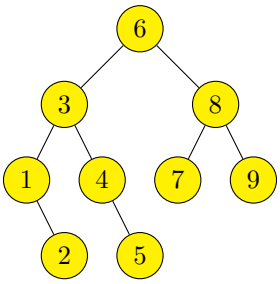
Puis, on insère le **noeud 2** :



Puis, on insère le **noeud 7** :



En insérant le dernier **noeud 5**, on obtient l'arbre binaire équilibré suivant :



4 Implémentation des arbres binaires de recherche

Pour implémenter un arbre binaire de recherche, nous allons utiliser une structure de classe. Nous travaillerons avec le fichier `abr.py` ci-joint, qui contient une classe `Noeud` et de quoi faire afficher graphiquement les arbres que nous construirons.

La classe `Noeud` possède 4 attributs :

- une valeur `value`
- un parent `parent`
- les deux enfants gauche et droite qui sont `left` et `right`

Voici la classe `abr.py` que nous utiliserons :

```
class Noeud:
    '''Noeud d'un arbre de recherche'''

    def __init__(self, value, left=None, right=None):
        ''' Le constructeur'''
        self.value = value
        self.left = left
        self.right = right
        self.parent = None

    def __str__(self):
        ''' Methode qui permet d'afficher la valeur de la racine
         avec la fonction print'''
        return str(self.value)


    def estFeuille(self):
        ''' Methode qui permet de savoir si le noeud est une feuille'''
        if not self.left and not self.right:
            return True
        else:
            return False
```

Exercice 2 : Compléter le code Python de la méthode `insert()` ci-dessous qui permet d'ajouter un nœud à l'arbre binaire de recherche.

```
    def insert(self, valeur):
        if ..................................... :
            if self.left is None:
                self.left = ....................................
                self.left.parent =
                    ....................................
            else:
                self.left.insert(valeur)
        elif ..................................... :
            if self.right is None:
                self.right = ....................................
                self.right.parent =
                    ....................................
            else:
                self.right.insert(valeur)
```

Exercice 3 : Rajouter la méthode `insert()` de l'exercice précédent à la classe `Noeud`, puis tester le programme suivant :

```
bst = Noeud(6)
bst.insert(8)
bst.insert(3)
bst.insert(1)
bst.insert(4)
bst.insert(9)
bst.insert(2)
bst.insert(7)
bst.insert(5)
graphicarbre(bst)
```

Exercice 4 : Écrire une fonction `insertion(arbre,val)` qui réalise le même travail mais sans que ce soit une méthode de classe.

1. Récupérer les méthodes des différents parcours (préfixe, infixe et suffixe).
2. Les faire fonctionner sur notre arbre.
3. Lequel d'entre eux affiche la liste triée ?

Exercice 5 : *Maximum*
Écrire un programme qui utilise une structure d'ABR pour trouver le maximum d'une liste d'entier.

Exercice 6 : *Minimum*
Écrire un programme qui utilise une structure d'ABR pour trouver le minimum d'une liste d'entier.

Exercice 7 : *Tri de liste*
Écrire un programme qui utilise une structure d'ABR pour trier par ordre croissant une liste d'entier.

Exercice 8 : *Recherche animal*
Écrire un programme qui utilise une structure d'ABR pour rechercher une valeur dans la liste suivante :

```
L = ['chat','chien','souris','araignée','crapaud','grenouille','lézard','zèbre']
```

Un tel algorithme est de complexité logarithmique.

Exercice 9 : *Chemin*
Écrire un programme qui utilise une structure d'ABR pour afficher le chemin depuis la racine jusqu'à un noeud donné.

Exercice 10 : *Sujet de BAC*
Faire l'exercice 1 du sujet de Bac 2021 suivant.

Exercice 11 : *Sujet de BAC (bis)*
Faire l'exercice 3 du sujet de Bac 2023 suivant.