Exercice : Arbre binaire de recherche (ABR)

Dans cet exercice, les arbres binaires de recherche ne peuvent pas comporter plusieurs fois la même clé. De plus, un arbre binaire de recherche limité à un nœud a une hauteur de 1.
On considère l'arbre binaire de recherche représenté ci-dessous (figure 1), où val représente un nombre entier :

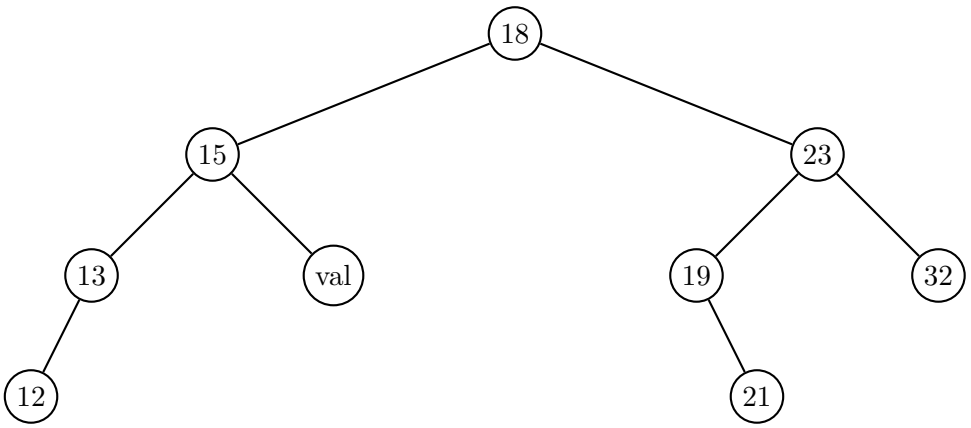

Figure 1

1) **a)** Donner le nombre de feuilles de cet arbre et préciser leur valeur.

b) Donner le sous-arbre gauche du nœud 23.

c) Donner la hauteur et la taille de l'arbre.

d) Donner les valeurs entières possibles de val pour cet arbre binaire de recherche.

On suppose, pour la suite de cet exercice, que val est égal à 16.

2) On rappelle qu'un parcours infixe depuis un nœud consiste, dans l'ordre, à faire un parcours infixe sur le sous-arbre gauche, afficher le nœud puis faire un parcours infixe sur le sous-arbre droit.

Dans le cas d'un parcours suffixe, on fait un parcours suffixe sur le sous-arbre gauche puis un parcours suffixe sur le sous-arbre droit, avant d'afficher le nœud .

a) Donner les valeurs d'affichage des nœuds dans le cas d'un parcours infixe de l'arbre de la figure 1.

b) Donner les valeurs d'affichage des nœuds dans le cas d'un parcours suffixe de l'arbre de la figure 1.

3) On considère la classe Noeud définie de la façon suivante en Python :

```
class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere :
            if v == n.v:
                est_insere = True          } Bloc 1
            elif v < n.v:
                if n.ag != None:           }
                    n = n.ag               }
                else:                      } Bloc 2
                    n.ag = Noeud(v)        }
                    est_insere = True      }
            else:
                if n.ad != None:           }
                    n = n.ad               }
                else:                      } Bloc 3
                    n.ad = Noeud(v)        }
                    est_insere = True      }

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)
```

a) Représenter l'arbre construit suite à l'exécution de l'instruction suivante :

```
racine = Noeud(18)
racine.insere_tout([12, 13, 15, 16, 19, 21, 32, 23])
```

b) Écrire les deux instructions permettant de construire l'arbre de la figure 1. On rappelle que le nombre val est égal à 16.

c) On considère l'arbre tel qu'il est présenté sur la figure 1. Déterminer l'ordre d'exécution des blocs (repérés de 1 à 3) suite à l'application de la méthode insere(19) au nœud racine de cet arbre.

4) Écrire une méthode recherche(self, v) qui prend en argument un entier v et renvoie la valeur True si cet entier est une valeur de l'arbre, False sinon.