

Devoir surveillé n°6 – NSI
30/01/2023**Exercice 1 (7 points)**

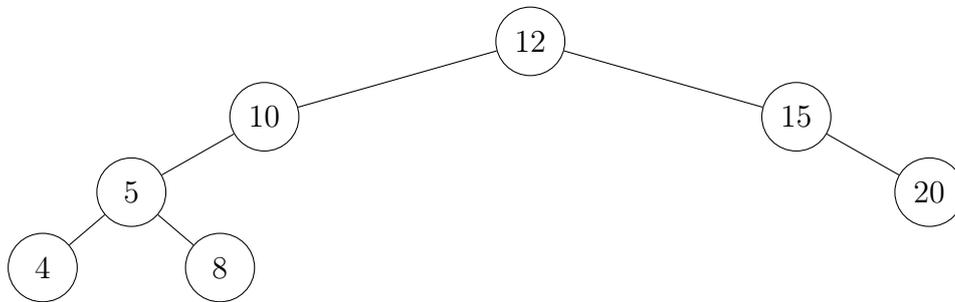
On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit. Un nœud sans sous-arbre est appelé feuille. La taille d'un arbre est le nombre de nœuds qu'il contient ; sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles. Ainsi la hauteur d'un arbre réduit à un nœud, c'est-à-dire la racine, est 1.

Dans un arbre binaire de recherche, chaque nœud contient une clé, ici un nombre entier, qui est :

- strictement supérieure à toutes les clés des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les clés des nœuds du sous-arbre droit.

Ainsi les clés de cet arbre sont toutes distinctes.

On considère l'arbre binaire de recherche ci-dessous :



1. Quelle est la taille de l'arbre ci-dessus ? Et quelle est sa hauteur ?
2. Un arbre binaire de recherche est dit « bien construit » s'il n'existe pas d'arbre de hauteur inférieure qui pourrait contenir tous ses nœuds. L'arbre ci-dessus n'est pas « bien construit ». Proposer un arbre binaire de recherche contenant les mêmes clés et dont la hauteur est plus petite que celle de l'arbre initial.
3. les classes `Noeud` et `Arbre` ci-dessous permettent de mettre en œuvre en Python la structure d'arbre binaire de recherche. La méthode `insere` permet d'insérer récursivement une nouvelle clé.

```
class Noeud :
    def __init__(self, cle):
        self.cle = cle
        self.gauche = None
        self.droit = None
    def insere(self, cle):
        if cle < self.cle :
            if self.gauche == None :
                self.gauche = Noeud(cle)
            else :
                self.gauche.insere(cle)
        elif cle > self.cle :
            if self.droit == None :
                self.droit = Noeud(cle)
            else :
                self.droit.insere(cle)
```

```

class Arbre :
    def __init__(self, cle):
        self.racine = Noeud(cle)
    def insere(self, cle):
        self.racine.insere(cle)

```

Donner la représentation de l'arbre codé par les instructions ci-dessous :

```

a = Arbre(10)
a.insere(20)
a.insere(15)
a.insere(12)
a.insere(8)
a.insere(4)
a.insere(5)

```

4. Pour calculer la hauteur d'un arbre non vide, on a écrit la méthode ci-dessous dans la classe Noeud.

```

def hauteur(self):
    if self.gauche == None and self.droit == None:
        return 1
    if self.gauche == None:
        return 1+self.droit.hauteur()
    elif self.droit == None:
        return 1+self.gauche.hauteur()
    else:
        hg = self.gauche.hauteur()
        hd = self.droit.hauteur()
        if hg > hd:
            return hg+1
        else:
            return hd+1

```

Écrire la méthode `hauteur` de la classe `Arbre` qui renvoie la hauteur de l'arbre en utilisant celle de la classe `Noeud`.

5. Écrire les méthodes `taille` des classes `Noeud` et `Arbre` permettant de calculer la taille d'un arbre.
6. **(Bonus ; à ne traiter qu'une fois tout le devoir effectué)**
 On souhaite écrire une méthode `bien_construit` de la classe `Arbre` qui renvoie la valeur `True` si l'arbre est « bien construit » et `False` sinon.
 On rappelle que la taille maximale d'un arbre binaire de recherche de hauteur h est $2^h - 1$.
- (a) Quelle est la taille minimale, notée t_{min} , d'un arbre binaire de recherche « bien construit » de hauteur h ?
- (b) Écrire la méthode `bien_construit` demandée.

Exercice 2 (3 points)

On reprend la notion d'arbre binaire de recherche ainsi que la classe `Noeud` de l'exercice précédent, notamment avec les attributs `cle`, `gauche` et `droit`.

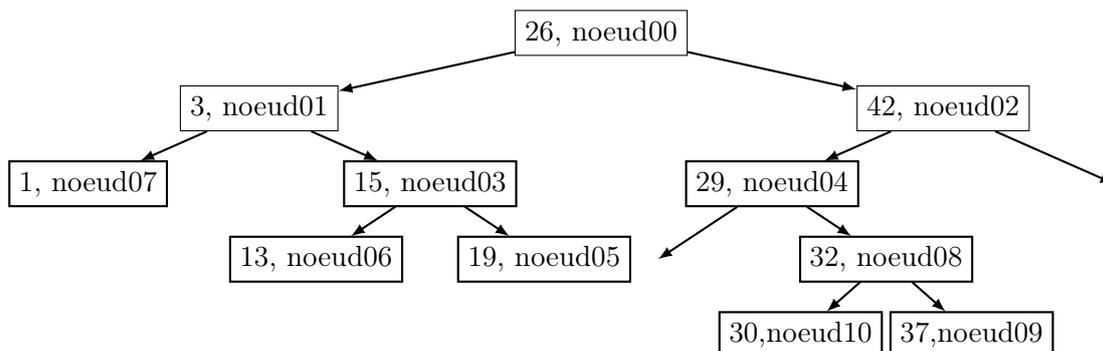
On donne ci-dessous un exemple d'arbre binaire de recherche dans lequel on a stocké dans cet ordre les clés :

[26, 3, 42, 15, 29, 19, 13, 1, 32, 37, 30]

L'étiquette d'un nœud indique la clé du nœud suivie du nom du nœud.

Les nœuds ont été nommés dans l'ordre de leur insertion dans l'arbre ci-dessous, à partir de 00.

Ainsi, « 29, noeud04 » signifie que le nœud nommé `noeud04` possède la clé 29.



1. Préciser toutes les valeurs entières des clés que l'on peut stocker dans le nœud fils gauche du noeud04 (vide pour l'instant), en respectant les règles sur les arbres binaires de recherche.
2. Voici une fonction récursive permettant de parcourir et d'afficher les clés de l'arbre :

```
def Parcours(A): # A est un arbre binaire de recherche
    if not A is None:
        print(A.cle)
        Parcours(A.gauche)
        Parcours(A.droit)
```

- (a) Écrire la liste de toutes les clés dans l'ordre où elles seront affichées.
 - (b) Préciser le type de parcours d'arbres binaires de recherche réalisé parmi les propositions suivantes : suffixe, préfixe ou infixe.
3. En vous inspirant de la fonction précédente, écrire une fonction `Parcours2` permettant de parcourir et d'afficher les clés de l'arbre A dans l'ordre croissant.