ALGORITHMES SUR LES ARBRES BINAIRES

NOTATIONS

À chaque nœud d'un arbre binaire, on associe :

- une **clé** (« valeur » associée au nœud, on peut aussi utiliser le terme « valeur » ou le terme « étiquette »),
- un « **sous-arbre gauche** »
- un « **sous-arbre droit** »

Un sous-arbre, droite ou gauche, est un arbre, même s'il contient un seul nœud ou pas de nœud de tout.

Un sous-arbre vide est noté **NIL** (du latin nihil qui signifie « rien »).

Soit un arbre **`T`** :

- **`T.racine`** correspond au nœud racine de l'arbre **`T`**.

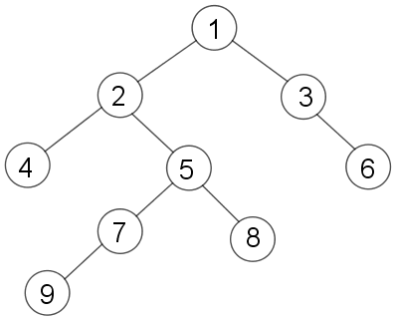
Soit un nœud **`x`** :

- **`x.gauche`** correspond au sous-arbre gauche du nœud **`x`**
- **`x.droit`** correspond au sous-arbre droit du nœud **`x`**
- **`x.cle`** correspond à la clé du nœud **`x`**

Si le nœud **`x`** est une feuille, alors **`x.gauche`** et **`x.droite`** sont des arbres vides (NIL)

CALCULER LA HAUTEUR D'UN ARBRE

Le nombre de niveaux total est appelé **hauteur de l'arbre**.



Hauteur de l'arbre = 4

```
class Noeud:
    def __init__(self,key):
        self.gauche = None
        self.droit = None
        self.val = key

def Hauteur(noeud):
    if noeud is None:
        return 0
    else:
        lheight = Hauteur(noeud.gauche)
        rheight = Hauteur(noeud.droit)
        return 1 + max(lheight, rheight)
```

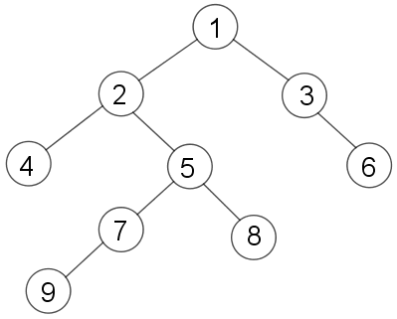
```
# test de la fonction Hauteur

racine = Noeud(1)
racine.gauche = Noeud(2)
racine.droit = Noeud(3)
racine.gauche.gauche = Noeud(4)
racine.gauche.droit = Noeud(5)
racine.droit.droit = Noeud(6)
racine.gauche.droit.gauche = Noeud(7)
racine.gauche.droit.droit = Noeud(8)
racine.gauche.droit.gauche.gauche = Noeud(9)

print(Hauteur(racine))
```

CALCULER LA TAILLE D'UN ARBRE

La **taille d'un arbre** est le nombre de nœuds présents dans l'arbre.



Taille de l'arbre = 9

```
class Noeud:
    def __init__(self,key):
        self.gauche = None
        self.droit = None
        self.val = key

def Taille(noeud):
    if noeud is None:
        return 0
    else:
        ltaille = Taille(noeud.gauche)
        rtaille = Taille(noeud.droit)
        return 1 + lheight + rheight
```

```
# test de la fonction Hauteur

racine = Noeud(1)
racine.gauche = Noeud(2)
racine.droit = Noeud(3)
racine.gauche.gauche = Noeud(4)
racine.gauche.droit = Noeud(5)
racine.droit.droit = Noeud(6)
racine.gauche.droit.gauche = Noeud(7)
racine.gauche.droit.droit = Noeud(8)
racine.gauche.droit.gauche.gauche = Noeud(9)

print(Taille(racine))
```

PARCOURS EN LARGEUR D'ABORD

METHODE 1 (AVEC UNE FILE)

```
def ParcoursLargeur (noeud):
    if noeud is None:
        return
    file = []
    file.append(noeud)

    while(len(file) > 0):
        # afficher et retirer
        # le premier élément
        print(file[0].val)
        node = file.pop(0)

        # ajouter l'enfant gauche
        # de l'élément retiré
        if node.gauche is not None:
            file.append(node.gauche)

        # ajouter l'enfant droit
        # de l'élément retiré
        if node.droit is not None:
            file.append(node.droit)
```

METHODE 2

```
def AfficheNiveau(noeud, niveau):
    if noeud is None:
        return
    if niveau == 1:
        print(noeud.val)
    elif niveau > 1:
        AfficheNiveau(noeud.gauche, niveau-1)
        AfficheNiveau(noeud.droit, niveau-1)

def ParcoursLargeur(noeud):
    h = Hauteur(noeud)
    for i in range(1, h+1):
        AfficheNiveau(noeud, i)
```

PARCOURS INFIXE

```
def infixe(noeud):
    if noeud==None:
        return
    if noeud.gauche!=None:
        infixe(noeud.gauche)
    print(noeud.val)
    if noeud.droit!=None:
        infixe(noeud.droit)
```

PARCOURS PREFIXE

```
def prefixe(noeud):
    if noeud==None:
        return
    print(noeud.val)
    if noeud.gauche!=None:
        prefixe(noeud.gauche)
    if noeud.droit!=None:
        prefixe(noeud.droit)
```

PARCOURS SUFFIXE

```
def postfixe(noeud):
    if noeud==None:
        return
    if noeud.gauche!=None:
        postfixe(noeud.gauche)
    if noeud.droit!=None:
        postfixe(noeud.droit)
    print(noeud.val)
```

RECHERCHE D'UNE CLE DANS UN ARBRE BINAIRE DE RECHERCHE

Si la clé k est présente dans l'arbre binaire de recherche, alors l'algorithme renvoie vrai, sinon il renvoie faux.

```
def ArbreRecherche(noeud, k):
    if noeud is None:
        print("FAUX")
        return
    if k == noeud.val:
        print("VRAI")
        return
    elif k < noeud.val:
        ArbreRecherche(noeud.gauche, k)
    else:
        ArbreRecherche(noeud.droit, k)
```

INSERTION D'UNE CLE DANS UN ARBRE BINAIRE DE RECHERCHE

```
def ArbreInserer(noeud, k):
    if noeud is None:
        noeud = Noeud(k)
    else:
        if noeud.val < k:
            if noeud.droit is None:
                noeud.droit = Noeud(k)
            else:
                ArbreInserer(noeud.droit, k)
        else:
            if noeud.gauche is None:
                noeud.gauche = Noeud(k)
            else:
                ArbreInserer(noeud.gauche, k)
```

APPLICATION :
CONSTRUCTION D'UN ARBRE BINAIRE DE RECHERCHE

Construction d'un ABR par ajouts successifs des valeurs 14, 10, 35, 6, 30, 33, 11, 16, 8 et 18.

```
""" création de la classe noeud """
class Noeud:
    def __init__(self,key):
        self.gauche = None
        self.droit = None
        self.val = key

def ArbreInserer(noeud, k):
    if noeud is None:
        noeud = Noeud(k)
    else:
        if noeud.val < k:
            if noeud.droit is None:
                noeud.droit = Noeud(k)
            else:
                ArbreInserer(noeud.droit, k)
        else:
            if noeud.gauche is None:
                noeud.gauche = Noeud(k)
            else:
                ArbreInserer(noeud.gauche, k)

""" Création de l'arbre binaire de recherche """
racine = Noeud(14)
ArbreInserer(racine,10)
ArbreInserer(racine,35)
ArbreInserer(racine,6)
ArbreInserer(racine,30)
ArbreInserer(racine,33)
ArbreInserer(racine,11)
ArbreInserer(racine,16)
ArbreInserer(racine,8)
ArbreInserer(racine,18)
```