Algo 16 et 17 – **Implémentation d'AB avec un modèle objet et algorithmes récursifs**

Nous allons utiliser une implémentation simple mais dans laquelle on ne distingue pas réellement Arbre Binaire NON VIDE et Noeud : un Arbre NON VIDE est un alias de son Noeud-racine.

```
AB = AB VIDE | AB NON VIDE
AB VIDE = (None, None, None)
AB NON VIDE = (Element, AB, AB)
Noeud =       (Element, AB, AB)
```

I et II – Module : implémentation d'un Arbre Binaire simple sous forme objet

```
"""Module implémentant un Arbre Binaire (AB) sous forme d'objets
"""

# Déclaration de la classe AB

class AB:
    """Implémentation d'un Arbre Binaire sous forme objet simple"""

    def __init__(self, valeur, gauche, droite):

        self.v = valeur
        self.g = gauche
        self.d = droite

    def relier_a_gauche(self:'AB NON VIDE', sous_arbre:'AB'):
        """Méthode qui définit sous_arbre comme le sous-arbre gauche de l'arbre actif"""
        self.g = sous_arbre

    def relier_a_droite(self:'AB NON VIDE', sous_arbre:'AB'):
        """Méthode qui définit sous_arbre comme le sous-arbre droite de l'arbre actif"""
        self.d = sous_arbre

    def definir_donnees_racine(self:'AB', element:'Element NON VIDE'):
        """Méthode qui place l'élément NON VIDE en tant que racine de l'arbre"""
        self.v = element

    def vider(self:'AB'):
        """Méthode qui transforme l'arbre en AB VIDE"""

        self.v = None
        self.g = None
        self.d = None

# Fonctions d'interface (accessibles de l'extérieur)

def contenu(noeud:'Noeud') -> 'Element':
    """Renvoie le contenu associée au noeud fourni"""
    return noeud.v

def nv_ABV() -> 'AB VIDE':
    """Renvoie un nouvel Arbre Binaire Vide"""
    return AB(None, None, None)

def nv_AB(valeur:'Element', g:'AB', d:'AB') -> 'AB NON VIDE':
    """Renvoie un nouvel AB dont la racine porte valeur et qui mène aux sous-arbres g et d"""
    return AB(valeur, g, d)

def est_ABV(arbre:'AB') -> bool:
    """Prédicat qui renvoie True si l'arbre est vide"""
    return arbre.v is None and arbre.g is None and arbre.d is None

def racine(arbre:'AB NON VIDE') -> 'Noeud':
    """Renvoie le noeud-racine de l'Arbre Binaire NON VIDE"""
    return arbre    # dans cette implémentation, un AB NON VIDE est un alias de sa racine

def gauche(arbre:'AB NON VIDE') -> 'AB':
    """Renvoie le sous-arbre gauche de l'Arbre Binaire NON VIDE"""
    return arbre.g

def droite(arbre:'AB NON VIDE') -> 'AB':
    """Renvoie le sous-arbre droite de l'Arbre Binaire NON VIDE"""
    return arbre.d
```

III – Réalisation d'autres fonctions à l'aide des primitives

```
# Importation

from arbre_binaire_ifa import contenu
from arbre_binaire_ifa import nv_ABV
from arbre_binaire_ifa import nv_AB
from arbre_binaire_ifa import est_ABV
from arbre_binaire_ifa import racine
from arbre_binaire_ifa import gauche
from arbre_binaire_ifa import droite

# Déclarations des fonctions

def parcours_prefixe(arbre:'AB') -> None:
    """Exploration (et affichage) en profondeur en préfixe RGD"""
    if not est_ABV(arbre):
        print(contenu(racine(arbre)))        # Traitement de la racine en 1er
        parcours_prefixe(gauche(arbre))
        parcours_prefixe(droite(arbre))

def parcours_infixe(arbre:'AB') -> None:
    """Exploration (et affichage) en profondeur en infixe GRD"""
    if not est_ABV(arbre):
        parcours_suffixe(gauche(arbre))
        print(contenu(racine(arbre)))        # Traitement de la racine en 2e
        parcours_suffixe(droite(arbre))

def parcours_suffixe(arbre:'AB') -> None:
    """Exploration (et affichage) en profondeur en postfixe GDR"""
    if not est_ABV(arbre):
        parcours_suffixe(gauche(arbre))
        parcours_suffixe(droite(arbre))
        print(contenu(racine(arbre)))         # Traitement de la racine en 3e

def hauteur(arbre:'AB', profondeur_vide=0) -> int:
    """Renvoie la hauteur de l'Arbre Binaire"""
    if est_ABV(arbre):
        return profondeur_vide
    else:
        return 1 + max(hauteur(gauche(arbre)), hauteur(droite(arbre)))

def nb_feuilles(arbre:'AB') -> int:
    """Renvoie le nombre de feuilles de l'Arbre Binaire"""
    if est_ABV(arbre):
        return 0
    elif est_ ABV(gauche(arbre)) and est_ ABV(droite(arbre)):
        return 1
    else:
        return nb_feuilles(gauche(arbre)) + nb_feuilles(droite(arbre))

def taille(arbre:'AB') -> int:
    """Renvoie la taille de l'Arbre Binaire"""
    if est_ABV(arbre):
        return 0
    elif est_ ABV(gauche(arbre)) and est_ ABV(droite(arbre)):
        return 1
    else:
        return 1 + taille(gauche(arbre)) + taille(droite(arbre))

def est_present(arbre:'AB', c:'Element') -> bool:
    """Prédicat : True si c est la clé d'un des noeuds"""
    if est_ABV(arbre):
        return False
    elif contenu(racine(arbre)) == c:
        return True
    else:
        return est_present(gauche(arbre), c) or est_present(droite(arbre), c)

# Instructions du programme principal

if __name__ == "__main__":
    ad = nv_AB("D", nv_ABV(), nv_ABV())
    af = nv_AB("F", ad,       nv_ABV())
    ae = nv_AB("E", nv_ABV(), af)
    ah = nv_AB("H", nv_ABV(), nv_ABV())
    ag = nv_AB("G", nv_ABV(), ah)
    ab = nv_AB("B", nv_ABV(), nv_ABV())
    ac = nv_AB("C", ag,       ab)
    aa = nv_AB("A", ac,       ae)
```