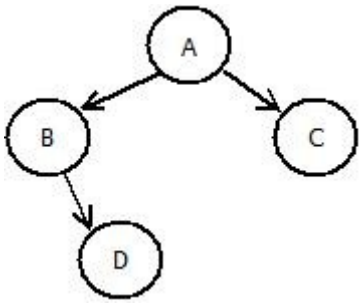
Michaël Guedj

TP d'Algorithmique avancée

Arbres binaires

Soit l'arbre binaire schématisé ainsi :



Que nous implémentons en Python comme suit :

```
class Noeud(object):
    # proprietes
    def __init__(self):
        self.val = None
        self.fg = None
        self.fd = None

    def __str__(self):
        return self.val


# -- creation des sommets
a = Noeud() # a : un objet, instance de la classe Noeud
a.val = 'A'

b = Noeud()
b.val = 'B'

c = Noeud()
c.val = 'C'

d = Noeud()
d.val = 'D'


# -- creation des arcs
a.fg = b
a.fd = c

b.fd = d
```

Nous donnons, en outre, l'implémentation du prédicat `est_feuille` (resp. `est_vide`) ; prenant en argument un sommet ; et testant si ce sommet est une feuille (resp. si ce sommet est vide).

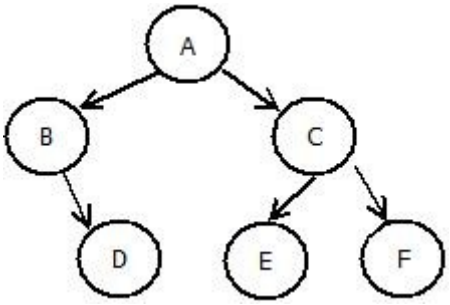
```
def est_feuille(s):
    if est_vide(s):
        return False
    return s.fg == None and s.fd == None

def est_vide(s):
    return s == None
```

Si *s* est un nœud, on note *Arbre(s)* l'arbre ayant pour racine le nœud *s*.

Exercice 1

Implémentez l'arbre binaire schématisé ci-dessous.



Exercice 2

Implémentez la fonction `taille`, prenant en argument un nœud *s* ; et retournant la **taille** de *Arbre(s)* ; i.e. le nombre de ses nœuds.

Exercice 3

Implémentez la fonction `trouver`, prenant en argument un nœud *s*, ainsi qu'une valeur *x* ; et retournant *Vrai* si *x* est présent dans *Arbre(s)* ; et *Faux* sinon.

Exercice 4

Implémentez la fonction `profondeur`, prenant en argument un nœud *s* ; et retournant la **profondeur** de *Arbre(s)*.

Exercice 5

Implémentez le **parcours préfixe** d'un arbre binaire.

Exercice 6

Implémentez le **parcours infixe** d'un arbre binaire.

Exercice 7 (***)

Réfléchir à l'implémentation du prédicat `est_parfait` ; retournant *Vrai* si le nœud donné en argument est la racine d'un **arbre binaire parfait** ; et *Faux* sinon.