Arbres 3/3 : Il est temps d'allumer un ordinateur

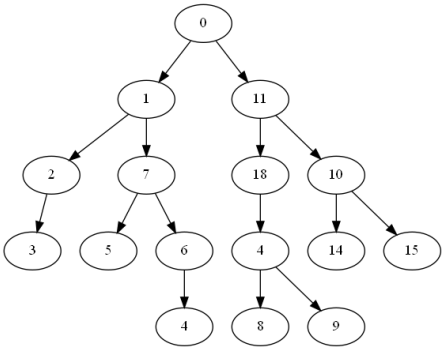
- Pour pouvoir concevoir et afficher des arbres, vous aurez besoin des modules:
  - `pillow` : permet de manipuler des fichiers images
  - `graphviz` : permet de concevoir des arbres, nécéssite d'abord l'installation sur votre machine du logiciel 'graphviz'(https://graphviz.org/download/ (https://graphviz.org/download/))
  - `Arbre` : c'est un fichier `Arbre.py` qui se trouve dans le même répertoire que cette feuille.Il contient des classes et des fonctions qui simplifient la conception d'arbres.
- Rappel pour installer un module sous python :
  - `pip install nom_du_module`
  - Si cela ne fonctionne pas : `python -m pip  install nom_du_module`

In [ ]:
```
#Modules nécéssaires
from PIL import Image
from graphviz import Digraph
from Arbre import *
```

In [ ]:
```
#Exemple d'utilisation

a1 = Arbre(1)
b = Arbre(2)
c = Arbre(3)
d = Arbre(4)
a1.ajoute(b, c)
b.ajoute(d, Arbre(5))

filename='a1.png'
a1.save(filename=filename)

pil_img=Image.open(filename)
display(pil_img)
```

Exercice 1 :

Ecrire le code qui permet de générer l'arbre ci-contre.



Exercice 2 :

On veut écrire une fonction récursive `genere_arbre(hauteur, max_enfants, n_max)` qui génère des arbres de façon aléatoire.Elle prend en paramètres:

- `hauteur` : hauteur de l'arbre généré, de type entier positif.
- `max_enfants` : nombre maximal d'enfants de chaque sous-arbre de l'arbre, de type entier positif.
- `n_max` : valeur maximale des étiquettes de l'arbre , de type entier positif.

Ainsi, l'appel de `genere_arbre(3, 3, 16)` doit générer un arbre de hauteur 3, dont chaque sous arbre comporte un nombre aléatoire d'enfants compris entre 0 et 3 et dont chaque étiquette est un nombre aléatoire compris entre 0 et 16.

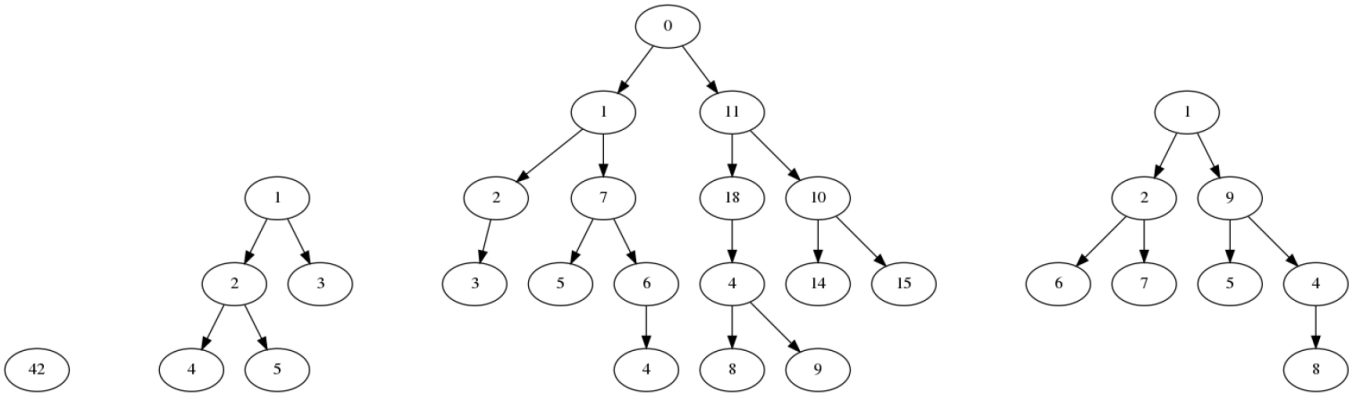
Voici le descriptif de cette fonction:

- On crée un arbre sans enfants avec une étiquette aléatoire.
- Le cas de base est celui ou `hauteur` vaut 1. Dans ce cas, on renvoie cet arbre.
- Le cas récursif genere des sous-arbres de hauteur `hauteur-1` pour chacun des enfants de l'arbre.

In [ ]:
```
from random import *

def genere_arbre(hauteur, max_enfants, n_max):
    tree=Arbre(randint(1,n_max))
    #code à compléter









    return tree


#Exemple :
H=3 #hauteur de l'arbre
E=3 #nombre max d'enfants
N=16 #valeur max étiquette
TREE=genere_arbre(H,E,N)

filename='arbre{}_{}_{}.png'.format(H,E,N)
TREE.save(filename=filename)

pil_img=Image.open(filename)
display(pil_img)
print(TREE)
```

Exercice 3 :

On veut écrire la fonction récursive `taille(arbre)` qui prend en paramètre un arbre et qui calcule sa taille, c'est à dire son nombre d'étiquettes. Dans cet exercice, on considère les arbres `a0` , `a1` , et `a5` respectivement affichés ci-dessous:



1. Indiquer ce que doit renvoyer:

- `taille(a0)` :
- `taille(a1)` :
- `taille(a5)` :

2. Ecrire le code de cette fonction.On pourra utiliserla fonction `est_une_feuille(arbre)` .

In [ ]:
```
def est_une_feuille(arbre):
    """
    Prend un arbre en parametre
    renvoie True si  cet arbre est une feuille (s'il n'a pas d'enfants)
    et False sinon
    """
    return arbre.enfants == []

def taille(arbre):









    pass


#les arbres a1 et a5 sont déjà créées plus haut
a0=Arbre(42)

print (taille(a0), taille(a1),taille(a5))
```

Exercice 4:

On veut écrire la fonction récursive `contient(etiquette,arbre)` qui prend en paramètres une étiquette et un arbre et qui renvoie `True` si l'étiquette est présente et `False` sinon Dans cet exercice, on considère les arbres `a0` , `a1` , et `a5` précédents.

1. Indiquer ce que doit renvoyer:

- `contient(42,a0)` :
- `contient(42,a1)` :
- `contient(18,a5)` :

2. Ecrire le code de cette fonction.On pourra utiliser la fonction `est_une_feuille(arbre)` .

In [ ]:

```
def contient(etiquette,arbre):









    pass

print(contient(42,a0), contient(42,a1),contient(18,a5))
```