N. S. I. Term. - DS06 (jeudi 02 février 2023)

EXERCICE 1 (4 points)

Cet exercice porte sur les arbres binaires de recherche, la programmation orientée objet et la récursivité.

Dans cet exercice, 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 (nœuds sans sous-arbres). On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et la hauteur de l'arbre vide vaut 0.

1. On considère l'arbre binaire représenté ci-dessous:

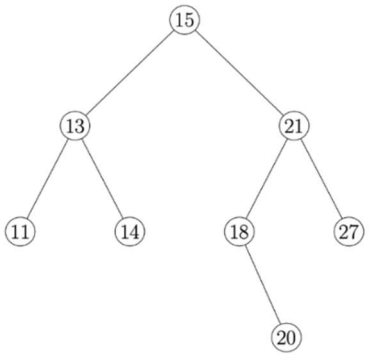


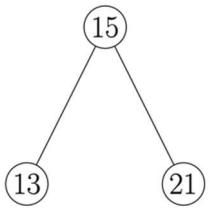
Figure 1

a. Donner la taille de cet arbre.
b. Donner la hauteur de cet arbre.
c. Représenter sur la copie le sous-arbre droit du nœud de valeur 15.
d. Justifier que l'arbre de la figure 1 est un arbre binaire de recherche.
e. On insère la valeur 17 dans l'arbre de la figure 1 de telle sorte que 17 soit une nouvelle feuille de l'arbre et que le nouvel arbre obtenu soit encore un arbre binaire de recherche. Représenter sur la copie ce nouvel arbre.

2. On considère la classe `Noeud` définie de la façon suivante en Python :

```
1  class Noeud:
2     def __init__(self, g, v, d):
3        self.gauche = g
4        self.valeur = v
5        self.droit = d
```

a. Parmi les trois instructions **(A)**, **(B)** et **(C)** suivantes, écrire sur la copie la lettre correspondant à celle qui construit et stocke dans la variable `abr` l'arbre représenté ci-contre.



(A) `abr=Noeud(Noeud(Noeud(None,13,None),15,None),21,None)`

(B) `abr=Noeud(None,13,Noeud(Noeud(None,15,None),21,None))`

(C) `abr=Noeud(Noeud(None,13,None),15,Noeud(None,21,None))`

b. Recopier et compléter la ligne 7 du code de la fonction `ins` ci-dessous qui prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et qui renvoie l'arbre obtenu suite à l'insertion de la valeur `v` dans l'arbre `abr`. Les lignes 8 et 9 permettent de ne pas insérer la valeur `v` si celle-ci est déjà présente dans `abr`.

```
1  def ins(v, abr):
2      if abr is None:
3          return Noeud(None, v, None)
4      if v > abr.valeur:
5          return Noeud(abr.gauche,abr.valeur,ins(v,abr.droit))
6      elif v < abr.valeur:
7          return ........................................
8      else:
9          return abr
```

3. La fonction `nb_sup` prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et renvoie le nombre de valeurs supérieures ou égales à la valeur `v` dans l'arbre `abr`.

Le code de cette fonction `nb_sup` est donné ci-dessous :

```
1  def nb_sup(v, abr):
2      if abr is None:
3          return 0
4      else:
5          if abr.valeur >= v:
6              return 1+nb_sup(v, abr.gauche)+nb_sup(v, abr.droit)
7          else:
8              return nb_sup(v, abr.gauche)+nb_sup(v, abr.droit)
```

a. On exécute l'instruction `nb_sup(16, abr)` dans laquelle `abr` est l'arbre initial de la figure 1. Déterminer le nombre d'appels à la fonction `nb_sup`.

b. L'arbre passé en paramètre étant un arbre binaire de recherche, on peut améliorer la fonction `nb_sup` précédente afin de réduire ce nombre d'appels. Écrire sur la copie le code modifié de cette fonction.

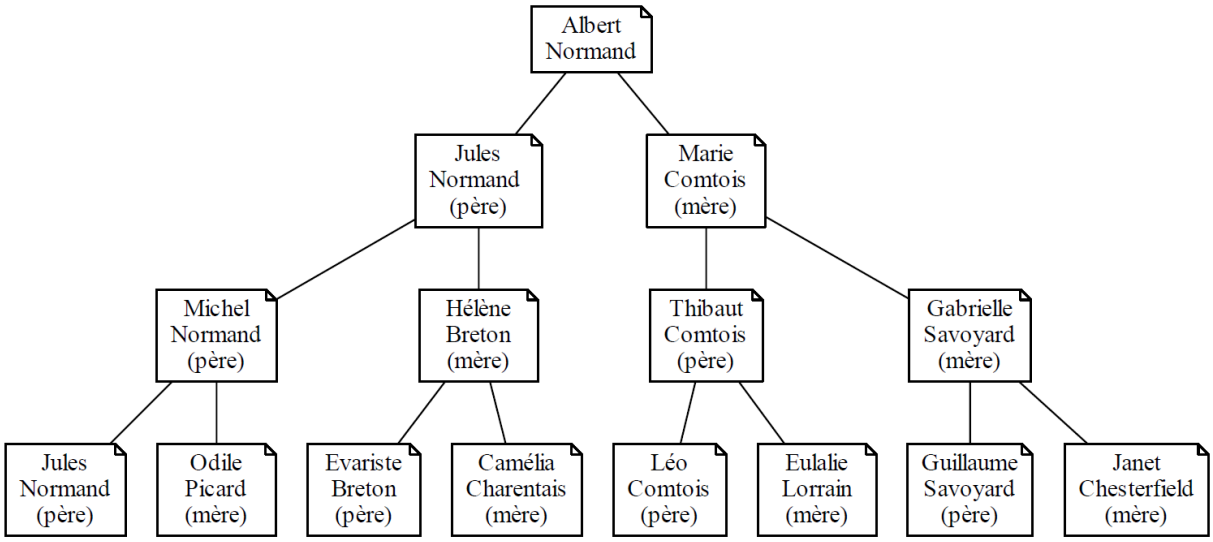
EXERCICE 4 (4 points)

Cet exercice porte sur l'algorithmique (arbres binaires en profondeurs préfixe et infixe).

On s'intéresse dans cet exercice à l'étude d'un arbre généalogique.
Voici un extrait de l'arbre généalogique fictif d'une personne nommée Albert Normand.

L'arbre généalogique est présenté avec les parents vers le bas et les enfants vers le haut.

Albert Normand est considéré comme la génération 0. On considère ses parents comme la génération 1, ses grands-parents comme la génération 2 et ainsi de suite pour les générations précédentes.



MODELISATION DE L'ARBRE

L'arbre généalogique d'un individu est modélisé par un arbre :

- chaque nœud de l'arbre représente un individu ;
- le premier nœud du sous-arbre gauche d'un individu est associé à son père ;
- le premier nœud du sous-arbre droit est associé à sa mère.

IMPLEMENTATION DE L'ARBRE

Pour implémenter l'arbre, on utilise des notions de programmation orientée objet.
Chaque nœud de l'arbre est représenté par un **objet** qui est l'instance d'une **classe** `Noeud` ayant trois attributs. Ainsi l'objet `N` de type `Noeud` aura les attributs suivants :

- `N.identite` de type tuple : `(prenom,nom)` de l'individu référencé par l'objet `N` ;
- `N.gauche` de type arbre binaire : le sous-arbre gauche de l'objet `N` ;
- `N.droit` de type arbre binaire : le sous-arbre droit de l'objet `N`.

Pour un individu, référencé par l'objet `N` de type `Noeud`, dont on ne connait pas les parents, on considèrera que `N.gauche` et `N.droit` ont la valeur `None`.

1.

a. Expliquer en quoi cet arbre généalogique est un arbre binaire.

b. Pourquoi un arbre généalogique n'est pas un arbre binaire de recherche (ABR) ?

2. On souhaite obtenir la liste de tous les ascendants (ancêtres) d'Albert Normand. Pour cela, on utilise un parcours en profondeur de l'arbre.

a. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur préfixe.

b. Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur infixe.

On donne ci-dessous le code incomplet de la fonction d'un parcours en profondeur de l'arbre, dans lequel il manque la ligne correspondant à l'instruction d'affichage du prénom et du nom de l'individu :

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
        noeud_actuel = racine_de_l_arbre
        parcours(noeud_actuel.gauche)
        parcours(noeud_actuel.droite)
```

c. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur préfixe.

d. Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur infixe.

3. On souhaite maintenant préciser la génération d'un individu dans l'implémentation de l'arbre généalogique. Lors de la création de l'instance, on donnera la valeur 0 par défaut.

a. Recopier et compléter la définition de la classe `Noeud` pour ajouter un attribut generation qui indique la génération d'un individu.

```
class Noeud() :
   def __init__(prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
        ……
```

b. Ecrire la fonction récursive `numerotation` qui parcourt l'arbre et modifie l'attribut generation de tous les ancêtres d'Albert Normand, de sorte que les parents d'Albert Normand soient la génération 1 etc…
Cette fonction prend en paramètres `racine_de_l_arbre` de type `Noeud` et `num_gen` de type entier.

```
def numerotation(racine_de_l_arbre, num_gen=0) :
    …
```

4. On donne la fonction suivante qui prend en paramètres l'objet `N` de type `Noeud` et la variable `affiche` de type booléen :

```
def mystere(N,affiche) :
    if N != None :
        if affiche :
            print( N.identite[0])
        mystere(N.gauche,False)
        mystere(N.droite,True)
```

Ecrire, dans l'ordre d'affichage, le résultat de l'exécution de `mystere(racine_de_l_arbre,False)` où `racine_de_l_arbre` est le nœud qui référence Albert Normand.