EXERCICE 4

Cet exercice est un exercice d'algorithmique et de programmation en langage Python utilisant les structures de données du type arbre binaire.

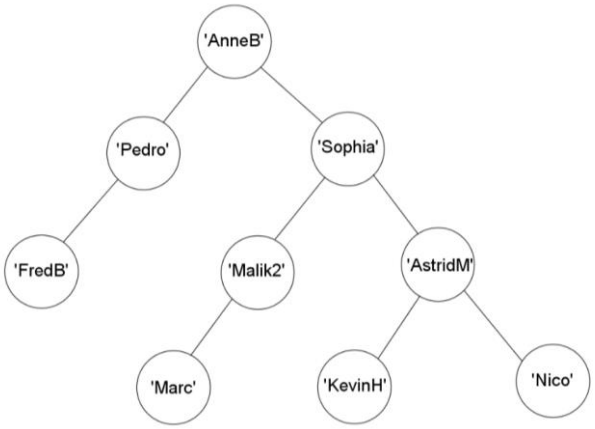
Un club de passionnés d'informatique fonctionne de la façon suivante : pour être membre du club, à l'exception du fondateur, il faut être parrainé. De plus, chaque membre peut parrainer au maximum deux personnes.

Dans ce club, on distingue trois profils de membres :

- membre or : le membre a parrainé deux personnes.
- membre argent : le membre a parrainé une seule personne.
- membre bronze : le membre n'a parrainé personne.

On peut modéliser ce fonctionnement de parrainage à l'aide d'un arbre binaire dont les étiquettes sont les pseudonymes des membres du club.

On donne ci-dessous l'arbre P représentant les membres du club issus des parrainages de AnneB, fondatrice du club. Par exemple, Sophia a parrainé Malik2 et AstridM.



On munit la structure de données arbre binaire des opérations suivantes :

`estvide(Arbre)` renvoie `True` si `Arbre` est vide, `False` sinon

`racine(Arbre)` renvoie l'étiquette du nœud racine de `Arbre`

`gauche(Arbre)` renvoie le sous-arbre gauche de `Arbre`

`droit(Arbre)` renvoie le sous-arbre droit de `Arbre`

1) On définit la hauteur d'un arbre non vide comme la longueur (en nombre d'arêtes) du plus long chemin allant d'une feuille à la racine. Un arbre vide a une hauteur égale à - 1. Un arbre qui n'a qu'un seul nœud a une hauteur de 0.

   a) **Indiquer** la hauteur de l'arbre P.

   b) **Recopier** et **compléter** le programme implémentant la fonction récursive `hauteur` qui prend en argument un arbre binaire `A` et qui renvoie la hauteur de cet arbre. On pourra utiliser la fonction `max` renvoyant la valeur maximale entre deux valeurs.

```
1. def hauteur(A) :
2.      if …… :
3.             return -1
4.      else :
5.             g=hauteur(gauche(A))
6.             d= ……
7.             return 1+ …….
```

   c) **Indiquer** le type de la valeur renvoyée par la fonction `hauteur`.

2) La fonction `membres` ci-dessous prend en argument un arbre binaire `A` et une liste `L`.

```
1. def membres(A,L) :
2.      if not(estvide(A)) :
3.             L.append(racine(A))
4.             membres(gauche(A),L)
5.             membres(droit(A),L)
```

   a) **Écrire** le contenu de `L_membres` après l'exécution des instructions ci-dessous où `P` désigne l'arbre P :

```
L_membres=[ ]
membres(P,L_membres)
```

   b) **Indiquer** le nom du type de parcours d'arbre binaire réalisé par la fonction `membres`.

3) Dans cette question, on s'intéresse aux profils des membres (or, argent ou bronze).

   a) **Indiquer** les étiquettes des feuilles de l'arbre P.

   b) À partir des propositions suivantes, **indiquer** le profil des membres dont les pseudonymes sont les étiquettes (définies en début de sujet) des feuilles.
   Réponse A : membre or.
   Réponse B : membre argent.
   Réponse C : membre bronze.
   Réponse D : on ne peut pas savoir.

c) **Écrire** la fonction `profil` prenant en argument un arbre binaire non vide A et qui renvoie le profil du membre dont le pseudonyme est l'étiquette de la racine de A, sous la forme d'une chaine de caractères : '`membre_or`', '`membre_argent`' ou '`membre_bronze`'. Par exemple, l'appel à `profil(P)` doit renvoyer '`membre_or`' qui correspond au profil du membre 'AnneB', racine de P.

4) Afin d'obtenir un tableau dont chaque élément est un tuple contenant le pseudonyme d'un membre et son profil, on propose la fonction `membres_profiles` définie ci-dessous :

```
1. def membres_profiles(A,L) :
2.       if not(estvide(A)) :
3.              L.append((racine(A),profil(A)))
4.              membres_profiles(gauche(A),L)
5.              membres_profiles(droit(A),L)
```

On exécute les instructions ci-dessous :

```
L2=[ ]
membres_profiles(arbre2,L2)
```

À l'issue de ces instructions on a obtenu le tableau suivant :

```
L2=[('LeaC','membre_or'),('Ali','membre_bronze'),('Tom45','membre_argent'),('Vero','membre_bronze')]
```

Dessiner un arbre possible pouvant correspondre à `arbre2`.

5) Chaque année, les membres reversent une cotisation en fonction de leur profil.

- membre or : cotisation de 20 €
- membre argent : cotisation de 30 €
- membre bronze : cotisation de 40 €

Écrire une fonction `cotisation` en langage Python ou en pseudo-code qui prend en argument un arbre binaire modélisant les parrainages comme explicité au début de l'exercice et qui renvoie le montant total des cotisations reçues par le club. On pourra utiliser la fonction `membres_profiles` de la question précédente.