Exercice 2 : Métropole 2 – 2022 pages 2/14 , 3/14

Exercice 1 : Métropole – septembre 2022 pages 2/13 , 3/13

Exercice 2 : Asie 2 – 2022 pages 4/14 , 5/14 , 6/14

Exercice 4 : Mayotte 2 – 2022 pages 8/13 , 9/13

Exercice 3 : Amérique du Sud 1 – 2022 pages 6/14 , à 9/14

Exercice 5 : Amérique du Sud 2 – 2022 pages 11/12 , à 12/12

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:

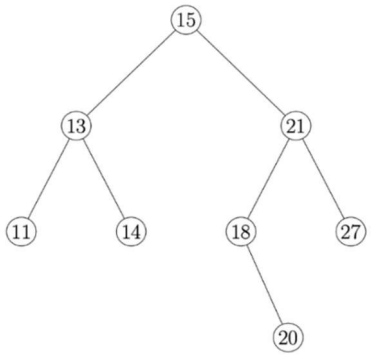


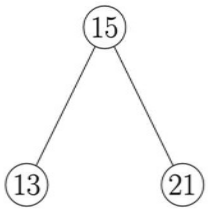
Figure 1

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 1 (4 points)

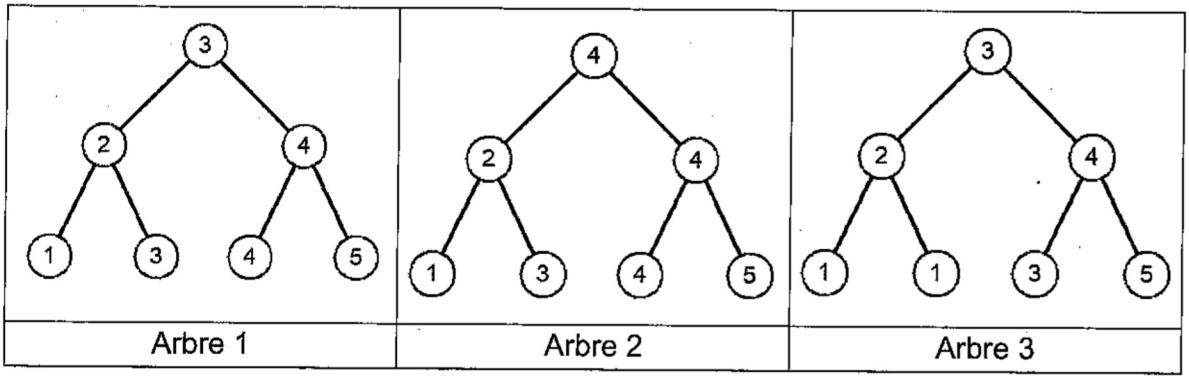
Cet exercice porte sur le thème "Algorithmique", les arbres binaires de recherche et leurs parcours.

Rappel : Un arbre binaire de recherche (ABR) est un arbre binaire étiqueté avec des clés tel que :

- Les clés du sous arbre gauche sont inférieures ou égales à celle de la racine ;
- Les clés du sous arbre droit sont strictement supérieures à celle de la racine ;
- Les deux sous arbres sont eux-mêmes des arbres binaires de recherche.

Partie A : Préambule

1) Recopier sur votre copie le ou les numéro(s) correspondant aux arbres binaires de recherche parmi les arbres suivants :



Partie B : Analyse

On considère la structure de données abstraites ABR (Arbre Binaire de Recherche) que l'on munit des opérations suivantes :

```
Structure de données : ABR
Utilise : Booleen, Element
Opérations :
```

- `creer_arbre : ∅ →ABR`

  `creer_arbre()` renvoie un arbre vide.

- `est_vide : ABR→Booleen`

  `est_vide(a)` renvoie True si l'arbre a est vide et False sinon.

- `racine : ABR→ Element`

  `racine(a)` renvoie la clé de la racine de l'arbre non vide a.

- `sous_arbre_gauche : ABR→ABR`

  `sous_arbre_gauche(a)` renvoie le sous-arbre gauche de l'arbre non vide a.

- `sous_arbre_droit : ABR→ABR`

  `sous_arbre_droit(a)` renvoie le sous-arbre droit de l'arbre non vide a.

- `inserer : ABR, Element→Rien`

  `inserer(a, e)` insère la clé e dans l'arbre a.

2.a) Dans un ABR, où se trouve le plus petit élément ? Justifier.

Pour rechercher une clé dans un ABR, il faut comparer la clé donnée avec la clé située à la racine. Si cette clé est à la racine, la fonction renvoie vrai sinon il faut procéder récursivement sur les sous arbres à gauche ou à droite.

2.b) En utilisant les fonctions ci-dessus, écrire une fonction récursive **`RechercheValeur`** prenant en arguments la clé recherchée et l'arbre ABR considéré. Cette fonction retourne un booléen (vrai ou faux) indiquant si la clé est présente dans l'arbre ou non.

3. On considère l'ABR ci-contre :

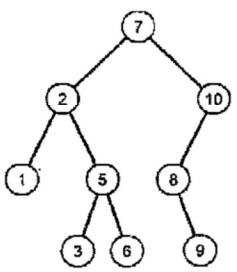
3.a) Dire à quel type de parcours correspond le résultat suivant où les clés sont triées dans l'ordre croissant :

1 – 2 – 3 – 5 – 6 – 7 – 8 – 9 - 10

3.b) Donner le parcours préfixe de cet arbre.

3.c) Donner le parcours suffixe de cet arbre.

3.d) Donner le parcours en largeur de cet arbre.



EXERCICE 2 (4 points)

Thèmes abordés : arbres binaires de recherche.

Un **arbre binaire de recherche** est un arbre binaire pour lequel chaque nœud possède une étiquette dont la valeur est supérieure ou égale à toutes les étiquettes des nœuds de son fils gauche et strictement inférieure à celles des nœuds de son fils droit. On rappelle que :

- sa taille est son nombre de nœuds ;
- sa hauteur est le nombre de niveaux qu'il contient.

Un éditeur réédite des ouvrages. Il doit gérer un nombre important d'auteurs de la littérature. Pour stocker le nom des auteurs, il utilise un programme informatique qui les enregistre dans un arbre binaire de recherche.

L'arbre vide sera noté `Null` pour les algorithmes de cet exercice.

Si `A` est un nœud non vide, `valeur(A)` renvoie le nom de l'auteur ; `fils_gauche(A)` renvoie le fils gauche du nœud `A` et `fils_droit(A)` renvoie le fils droit du nœud `A`.

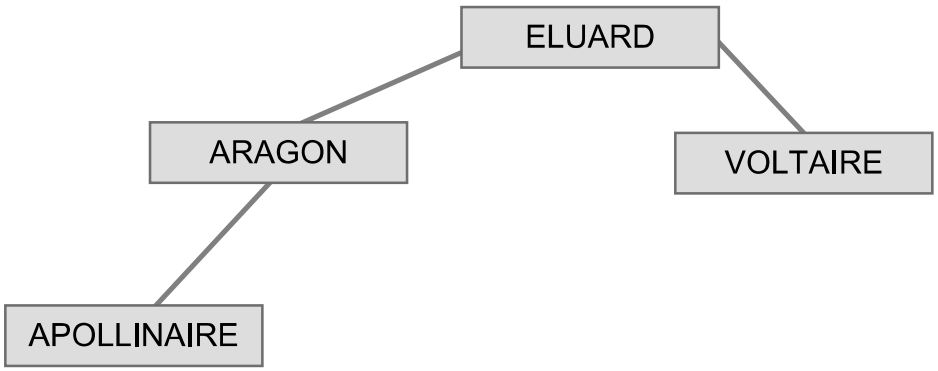
L'ordre alphabétique est utilisé pour classer le nom des auteurs.

Par exemple, on a `APOLLINAIRE < BAUDELAIRE`

Ainsi, pour tout nœud `A`, si `fils_gauche(A)` et `fils_droit(A)` ne sont pas `Null`, on a :

`valeur(fils_gauche(A)) ≤ valeur(A) < valeur(fils_droit(A))`.

Par exemple, l'arbre binaire suivant `A1` est un arbre binaire de recherche :



1.

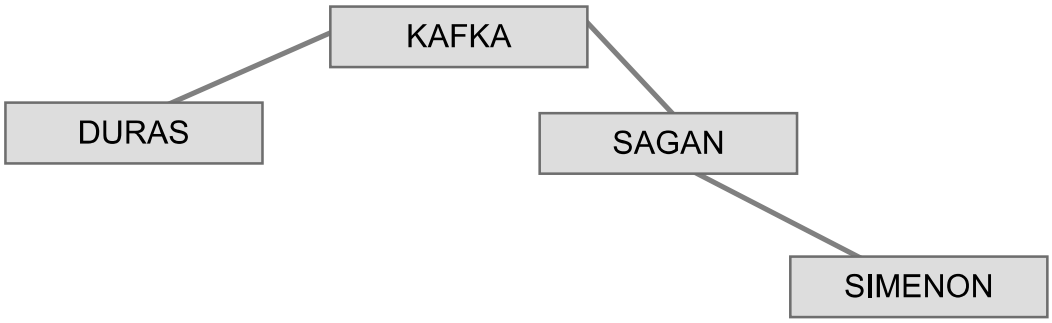
a. Recopier et compléter l'arbre binaire de recherche précédent en insérant successivement dans cet ordre les noms suivants :

DUMAS ; HUGO ; ZWEIG ; ZOLA

b. Quelle est la taille de l'arbre obtenu ? Quelle est la hauteur de cet arbre ?

c. Plus généralement, si l'arbre est de hauteur h, quel est le nombre maximal d'auteurs enregistrés dans cet arbre en fonction de h ?

On définit ici l'équilibre d'un arbre binaire : il s'agit d'un nombre entier positif ou négatif. Il vaut 0 si l'arbre est vide. Sinon il vaut la différence des hauteurs des sous-arbres gauche et droit de l'arbre.
Par exemple, si on considère l'arbre suivant que l'on nommera `A2` :



Son équilibre vaut -1 car la hauteur de son sous-arbre gauche vaut 1, la hauteur de son sous-arbre droit vaut 2 et 1 – 2 = -1
Un arbre est dit équilibré si son équilibre vaut -1, 0 ou 1.
L'arbre précédent est donc équilibré.

2. Recopier et compléter l'arbre de ce dernier exemple avec les noms FLAUBERT, BALZAC, PROUST, SAND, WOOLF, COLETTE, CHRISTIE et AUDIARD quitte à modifier l'ordre d'insertion de manière à ce que cet arbre reste équilibré.

L'éditeur souhaite utiliser une fonction récursive `recherche_auteur(ABR, NOM)` qui prend en paramètres `ABR` un arbre binaire de recherche et `NOM` un nom d'auteur. La fonction renvoie `TRUE` si `NOM` est une étiquette de l'arbre `ABR` et `FALSE` dans le cas contraire.

On donne la fonction suivante :

```
Fonction mystere(ABR, t) :
 SI ABR = NULL :
   RENVOYER FAUX
 SINON SI valeur(ABR) = t :
   RENVOYER VRAI
 SINON :
   RENVOYER mystere(fils_gauche(ABR),t) OU mystere(fils_droit(ABR),t)
```

3. Que renvoie l'appel `mystere(A2,'SIMENON')` ? Justifier la réponse.

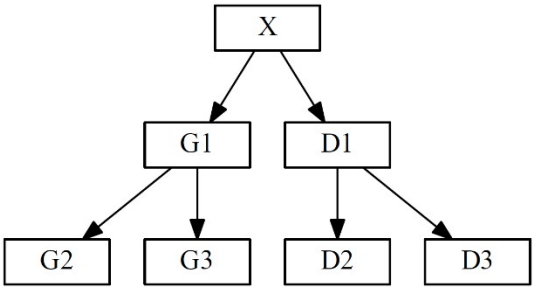
L'éditeur souhaite utiliser une fonction récursive `hauteur(ABR)` qui prend en paramètre un arbre binaire `ABR` et renvoie la hauteur de cet arbre.

4. Ecrire un algorithme de la fonction `hauteur(ABR)` qui prend en entrée `ABR` un arbre binaire de recherche et renvoie sa hauteur. On pourra avoir recours aux fonctions `MIN(val1, val2)` et `MAX(val1, val2)` qui renvoient respectivement la plus petite et la plus grande valeur entre `val1` et `val2`.

EXERCICE 4

Cet exercice porte sur les arbres binaires de recherche.

Un arbre binaire est soit vide, soit un nœud qui a une valeur et au plus deux fils (le sous-arbre gauche et le sous-arbre droit). Dans la représentation ci-contre,



`X` est un nœud
`G1` est le fils gauche de `X`
`D1` est le fils droit de `X`

Un arbre binaire de recherche est ordonné de la manière suivante :

Pour **chaque** nœud,

- les valeurs de **tous les nœuds** du sous-arbre gauche sont **strictement inférieures** à la valeur du nœud
- les valeurs de **tous les nœuds** du sous-arbre droit sont **supérieures ou égales** à la valeur du nœud

Ainsi, par exemple, toutes les valeurs des nœuds `G1`, `G2` et `G3` sont strictement inférieures à la valeur du nœud `X` et toutes les valeurs des nœuds `D1`, `D2` et `D3` sont supérieures ou égales à la valeur du nœud `X`.

Au fur et à mesure que les billets d'une loterie sont vendus, les numéros inscrits sur les billets sont insérés dans un arbre binaire de recherche. Chaque nœud de l'arbre est un couple, la 1ère valeur correspond au numéro du billet et la seconde est une référence permettant de connaître le lieu de vente du billet.

Par exemple au couple (45,'AZ60') correspond le billet n°45 vendu par le commerçant 'AZ60'. Seul le numéro du billet est utilisé pour positionner le nœud dans l'arbre.

1.

a. Dessiner l'arbre binaire de recherche dont la racine est (45,'AZ60') et dans lequel on insère dans cet ordre les nœuds (70,'AZ60'), (22,'AZ60'), (65,'BB54'), (58,'BC25') et (67,'BC25')

b. Pour construire **la liste triée** de tous les numéros de billets vendus, préciser quel type de parcours de cet arbre faut-il programmer parmi les propositions ci-dessous :

- un parcours en largeur ;
- un parcours en profondeur infixe ;
- un parcours en profondeur préfixe ;
- un parcours en profondeur suffixe.

La fonction `filsgauche(a)` retourne le sous arbre gauche du nœud `a` et `null` si le nœud `a` n'a pas de fils gauche. Il en est de même pour la fonction `filsdroit(a)` avec le fils droit.

2. On rappelle que la taille d'un arbre est le nombre de nœuds. Recopier et compléter les [.....] de l'algorithme suivant pour que la fonction taille(a) renvoie la taille d'un arbre a.

```
fonction taille(a)
     si a est null
        alors renvoyer [.....]
     sinon
        renvoyer 1 + [.....] + [.....]
```

3. Si `a` n'est pas `null`, la fonction `billet(a)` retourne la première valeur du nœud `a` (c'est-à-dire le numéro du billet) et `reference(a)` retourne la deuxième valeur du nœud `a` (c'est-à-dire le lieu de vente du billet `billet(a)`).

a. Que fait la fonction récursive écrite en pseudo-langage suivante, où le paramètre `a` est un nœud et `n` un nombre entier ?

```
fonction mystere(a, n)
  si a est null
    alors renvoyer Faux
  sinon, si billet(a) vaut n
    alors renvoyer Vrai
  sinon
    renvoyer mystere(filsgauche(a)) OU mystere(filsdroit(a))
```

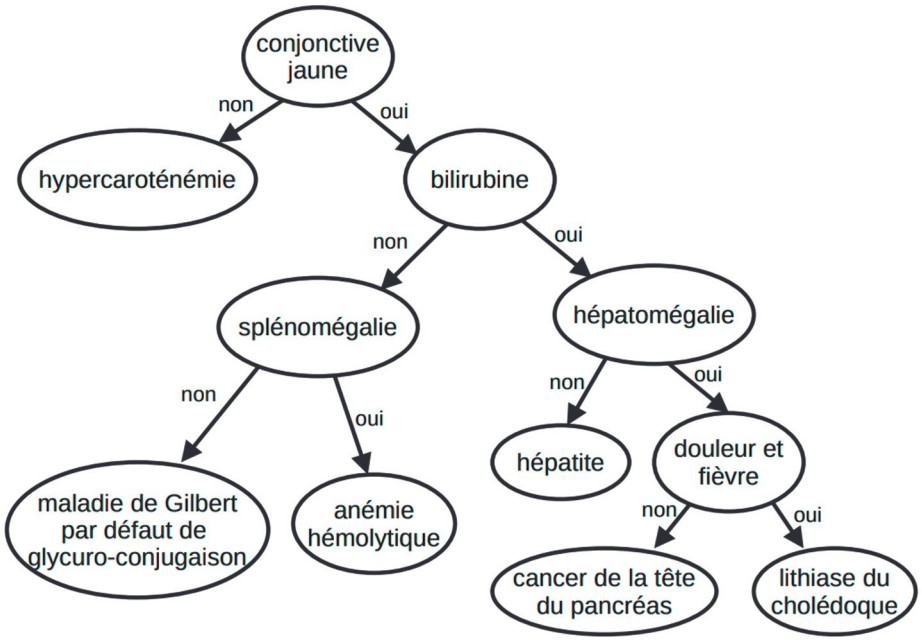
b. La fonction précédente est applicable à tout arbre binaire. L'arbre construit à la première question est un arbre binaire de recherche. Recopier et compléter le [.....] de la ligne 6 de l'algorithme ci-dessous pour améliorer la fonction `mystere` en tenant compte de cette remarque.

```
1. fonction mystereABR(a, n)
2.      si a est null
3.         alors renvoyer Faux
4.      sinon, si billet(a) vaut n
5.         alors renvoyer Vrai
6.      sinon si [.....]
7.        renvoyer mystereABR(filsgauche(a))
8.      sinon
9.         renvoyer mystereABR(filsdroit(a))
```

Exercice 3 (4 points)

Cet exercice porte sur les arbres binaires.

Les premiers travaux concernant l'aide à la décision médicale se sont développés pendant les années soixante-dix parallèlement à l'avènement de l'informatique dans le secteur médical. L'arbre de décision est une technique décisionnelle fréquemment employée pour rechercher la meilleure stratégie thérapeutique. L'arbre de décision de cet exercice, présenté ci-dessous, est un arbre binaire que l'on nommera `arb_decision`.



Arbre de décision en présence d'une jaunisse (peau anormalement jaune) chez un patient.

Rappels :

- ✓ Un **arbre binaire** est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément, appelé **nœud**, porte une étiquette.
- ✓ Le nœud initial est appelé **racine**.
- ✓ Chaque nœud d'un arbre binaire possède au plus deux **sous-arbres**.
- ✓ Chacun de ces sous-arbres est un arbre binaire, appelés sous-arbre gauche et sous-arbre droit.
- ✓ Un nœud dont les sous-arbres sont vides est appelé une **feuille**.
- ✓ Dans cet exercice, on utilisera la convention suivante : la **hauteur** d'un arbre binaire ne comportant qu'un nœud est égale à **1**.

Dans l'arbre de décision en présence d'une jaunisse chez un patient,

- ✓ un **nœud** représente un symptôme dont le médecin doit étudier la présence ou l'absence ; la réponse ne peut être que **oui** ou **non** ;
- ✓ le sous-arbre gauche d'un nœud donné décrit la démarche à adopter si le symptôme est **absent** ;
- ✓ le sous-arbre droit d'un nœud donné décrit la démarche à adopter si le symptôme est **présent** ;
- ✓ l'étiquette d'une feuille est la maladie induite par le chemin parcouru.

1. Déterminer la taille et la hauteur de l'arbre donné en exemple en introduction (arbre de décision en présence d'une jaunisse).

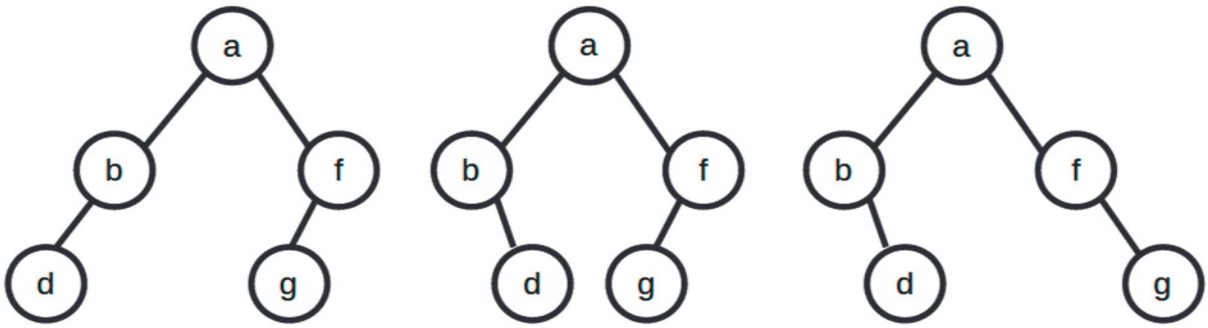
2. On choisit d'implémenter un arbre binaire à l'aide d'un dictionnaire.

```
arbre_vide = {}
arbre = {'etiquette': 'valeur' ,
         'sag': sous_arbre_gauche ,
         'sad': sous_arbre_droit }
```

Le code ci-dessous représente un arbre selon le modèle précédent.

```
{'etiquette' :'a',
 'sag':{'etiquette' :'b',
        'sag':{},
        'sad' : {'etiquette' : 'd',
                 'sag' : {},
                 'sad' : {}}},
 'sad': {'etiquette' :'f',
         'sag' : {'etiquette' : 'g',
                  'sag' : {},
                  'sad' : {}},
         'sad' : {} }}
```

a. À quelle représentation graphique correspond la structure implémentée ci-dessus ?



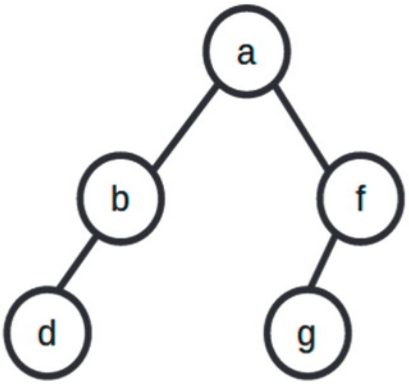
b. Représenter graphiquement l'arbre correspondant au code ci-dessous.

```
{'etiquette' :'H',
 'sag':{ 'etiquette' :'G',
        'sag': {'etiquette' : 'E',
                'sag' : {},
                'sad' : {}},
        'sad' : {'etiquette' : 'D',
                'sag' : {},
                'sad' : {'etiquette' : 'B',
                         'sag' : {},
                        'sad' : {} }}},
 'sad': {'etiquette' :'F',
         'sag' : {'etiquette' : 'C',
                  'sag' : {},
                  'sad' : {'etiquette' : 'A',
                           'sag' : {} ,
                           'sad' : {}}},
          'sad' : {} }}
```

3. La fonction `parcours(arb)` ci-dessous permet de réaliser le parcours des nœuds d'un arbre binaire `arb` donné en argument.

```
def parcours(arb):
    if arb == {}:
        return None
    parcours(arb['sag'])
    parcours(arb['sad'])
    print(arb['etiquette'])
```

a. Donner l'affichage après l'appel de la fonction `parcours` avec l'arbre dont une représentation graphique est ci-contre.



b. Écrire une fonction `parcours_maladies(arb)` qui n'affiche que les feuilles de l'arbre binaire non vide `arb` passé en argument, ce qui correspond aux maladies possiblement induites par l'arbre de décision.

4. On souhaite maintenant afficher l'ensemble des symptômes relatifs à une maladie. On considère la fonction `symptomes(arbre, mal)` avec comme argument `arbre` un arbre de décision binaire et `mal` le nom d'une maladie.
L'appel de cette fonction sur l'arbre de décision `arb_decision` de l'introduction fournit les affichages suivants.

```
>>> symptomes(arb_decision, "anémie hémolytique")
symptômes de anémie hémolytique
splénomégalie
pas de bilirubine
conjonctive jaune
```

Pour cela, on modifie la structure précédente en ajoutant une clé `surChemin` qui sera un booléen indiquant si le nœud est sur le chemin de la maladie.
La clé `surChemin` est initialisée à `False` pour tous les nœuds.

```
arbre = {'etiquette': 'valeur' ,
         'surChemin ': False ,
         'sag': 'sous-arbre gauche' ,
         'sad': 'sous-arbre droit' }
```

Recopier et compléter les lignes 6, 8, 14 et 18 du code suivant sur votre copie.

```
01    def symptomes(arb, mal):
02        if arb['sag'] != {} :
03            symptomes(arb['sag'],mal)
04
05        if arb['sad'] != {} :
06            symptomes(.....................)
07
08        if ............................. :
09            arb['surChemin'] = True
10            print('symptômes de', arb['etiquette'],':')
11
12        else :
13            if arb['sad'] != {} and arb['sad']['surChemin'] :
14                print(.....................)
15                arb['surChemin'] = True
16
17            if arb['sag'] != {} and arb['sag']['surChemin'] :
18                print(.....................)
19                arb['surChemin'] = True
```

Exercice 5 (4 points)

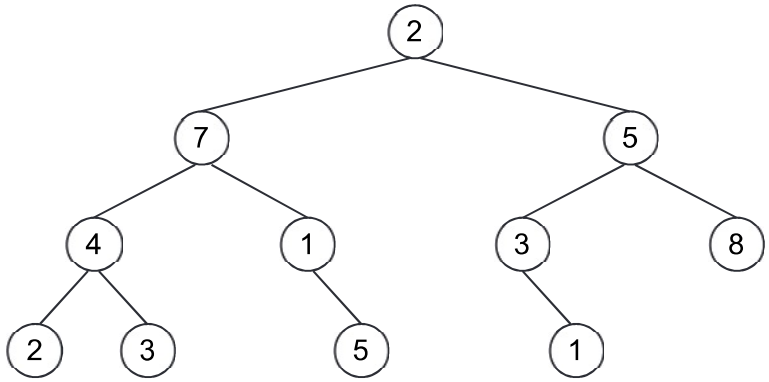
Cet exercice porte sur les arbres binaires, la programmation orienté objet et la récursivité

Dans un arbre binaire, chaque nœud admet au plus deux enfants, appelés sous-arbre gauche et sous-arbre droit. On considère dans cet exercice des arbres binaires étiquetés avec des nombres entiers.
On définit un chemin racine-feuille dans un tel arbre comme une liste ordonnée de nœuds telle que

- le premier nœud est la racine ;
- chaque nœud suivant est enfant du précédent ;
- le dernier nœud est une feuille.

On appellera somme d'un chemin racine-feuille la somme des étiquettes des nœuds du chemin.
Enfin, la plus grande somme racine-feuille d'un arbre est la plus grande somme qu'il est possible d'obtenir en considérant tous les chemins racine-feuille de l'arbre.

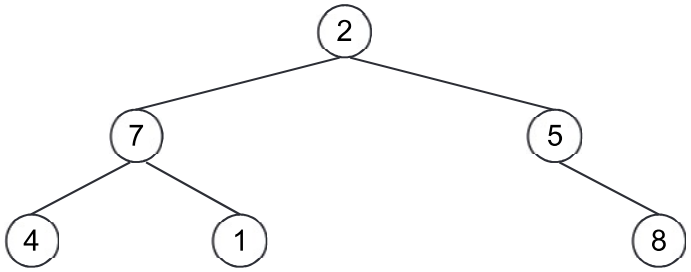
1. Déterminer la plus grande somme racine-feuille de l'arbre représenté ci-dessous.



2. La classe `Noeud` ci-dessous implémente le type abstrait d'arbre binaire.

```
class Noeud:

    def __init__(self, v):
        self.etiquette = v
        self.sag = None
        self.sad = None

    def niveau(self):
        if self.sag!=None and self.sad!=None:
            hg = self.sag.niveau()
            hd = self.sad.niveau()
            return 1+max(hg, hd)
        if self.sag!=None:
            return self.sag.niveau()+1
        if self.sad!=None:
            return self.sad.niveau()+1
        return 0

    def modifier_sag(self, nsag) :
        self.sag = nsag

    def modifier_sad(self, nsad) :
        self.sad = nsad
```

a. Écrire une suite d'instructions utilisant la classe `Noeud` permettant de représenter l'arbre ci-dessous.

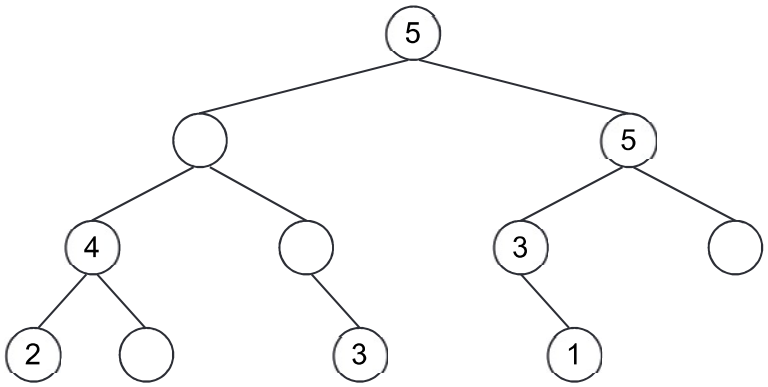


b. Que renvoie l'appel de la méthode `niveau` sur l'arbre ci-dessus ?

3. S'inspirer du code de la méthode `niveau` pour écrire une méthode récursive `pgde_somme` qui renvoie la plus grande somme racine-feuille d'un arbre.

4. On appelle arbre magique un arbre binaire dont toutes les sommes des chemins racine-feuille sont égales.

a. Recopier et compléter l'arbre ci-dessous pour qu'il soit magique.



b. Un arbre est magique si ses sous-arbres sont magiques et qu'ils ont de plus la même plus grande somme racine-feuille. Écrire une méthode récursive `est_magique` qui renvoie `True` si l'arbre est magique et `False` sinon.