Sujet BAC 11 : Algorihtmes Arbres Binaires

1. Amérique du Sud 2022 J2

Exo

Dans un arbre binaire, chaque noeud 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 noeuds telle que :

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

On appellera somme d'un chemin racine-feuille la somme des étiquettes des noeuds du chemin. Enfin, la plus graande 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.

Question 1

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

```
graph TD
    A("2") --> B("7")
    B --> D("4")
    B --> E("1")
    A --> C("5")
    C --> C1("3")
    C --> C2("8")
    D --> D1("2")
    D --> D2("3")
    E --> F(" ")
    E --> E1("5")
    linkStyle 8 stroke-width:0px;
    style F opacity:0;
    C1 --> C3(" ")
    C1 --> C4("1")
    C2 --> C5(" ")
    C2 --> C6(" ")
    linkStyle 10 stroke-width:0px;
    style C3 opacity:0;
    linkStyle 12 stroke-width:0px;
    style C5 opacity:0;
    linkStyle 13 stroke-width:0px;
    style C6 opacity:0;
```

Réponse

La plus grand somme racine-feuille de cette arbre est **16**, elle est obtenu pour la branche en rouge dans le schéma suivant :

```
graph TD
A["2"] --> B["7"]
A --> C["5"]
B --> D["4"]
B --> E["1"]
C --> F["3"]
C --> G["8"]
D --> H["2"]
D --> I["3"]
E --> V1[" "]
E --> J["5"]
F --> V2[" "]
F --> K["1"]
style V1 opacity:0;
style V2 opacity:0;
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
linkStyle 0 stroke:#FF0000,stroke-width:2px
linkStyle 2 stroke:#FF0000,stroke-width:2px
linkStyle 7 stroke:#FF0000,stroke-width:2px
```

Question 2

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

🐍 **Script Python**

```
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. Ecrire une suite d'instructions utilisant la class `Noeud` permettant de représenter l'arbre ci-dessous :

```
graph TD
    A("2") --> B("7")
    B --> D("4")
    B --> E("1")
    A --> C("5")
    C --> C1(" ")
    C --> C2("8")
    D --> D2(" ")
    linkStyle 6 stroke-width:0px;
    style D2 opacity:0;
    linkStyle 4 stroke-width:0px;
    style C1 opacity:0;
```

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

Réponse

a. On peut écrire la suite d'instructions suivante :

Script Python

```
s2 = Noeud(2)
s7 = Noeud(7)
s5 = Noeud(5)
s2.modifier_sag(s7)
s2.modifier_sad(s5)
s4 = Noeud(4)
s1 = Noeud(1)
s7.modifier_sag(s4)
s7.modifier_sad(s1)
s8 = Noeud(8)
s5.modifier_sad(s8)
```

b. L'appel à niveau sur cet arbre renvoie 3.

Question 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.

Réponse

Script Python

```
def pgde_somme(self):
    if self.sag != None and self.sad!=None:
        pgde_gauche = self.sag.pgde_somme()
        pgde_droite = self.sad.pgde_somme()
        return self.etiquette + max(pgde_gauche,pgde_droite)
    if self.sag != None:
        return self.etiquette + self.sag.pgde_somme()
    if self.sad != None:
        return self.etiquette + self.sad.pgde_somme()
    return self.etiquette
```

Question 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.

```
graph TD
    A("5") --> B("...")
    B --> D("4")
    B --> E("...")
    A --> C("5")
    C --> C1("3")
    C --> C2("...")
    D --> D1("2")
    D --> D2("...")
    E --> F("...")
    E --> E1("3")
    linkStyle 8 stroke-width:0px;
    style F opacity:0;
    C1 --> C3("...")
    C1 --> C4("1")
    C2 --> C5("...")
    C2 --> C6("...")
    linkStyle 10 stroke-width:0px;
    style C3 opacity:0;
    linkStyle 12 stroke-width:0px;
    style C5 opacity:0;
    linkStyle 13 stroke-width:0px;
    style C6 opacity:0;
```

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

Réponse

a. Arbre complété :

```
graph TD
A["2"] --> B["3"]
A --> C["5"]
B --> D["4"]
B --> E["3"]
C --> F["3"]
C --> G["4"]
D --> H["2"]
D --> I["2"]
E --> V1[" "]
E --> J["5"]
F --> V2[" "]
F --> K["1"]
style V1 opacity:0;
style V2 opacity:0;
style B stroke:#FF0000
style E stroke:#FF0000
style G stroke:#FF0000
style I stroke:#FF0000
linkStyle 8 stroke:#FFFFFF,stroke-width:0px
linkStyle 10 stroke:#FFFFFF,stroke-width:0px
```

b.

🐍 **Script Python**

```python
def est_magique(self):
    if self.sad is not None and self.sag is not None:
        return self.sad.est_magique() and self.sag.est_magique() and self.sag.pgde_somme() ==
self.sad.pgde_somme()
    elif self.sad is not None:
        return self.sad.est_magique()
    elif self.sag is not None:
        return self.sag.est_magique()
    else:
        return True
```

2. 2022 Mayotte J1

Exo

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.

```
graph TD
   A("Albert <br> Normand") --> B("Jules <br> Normand <br> Père")
   B --> D("Michel <br> Normand <br> (Père)")
   B --> E("Hélène <br> Breton <br> (Mère)")
   A --> C("Marie <br> Comtois <br> (Mère)")
   C --> C1("Thibaut <br> Comtois <br> (Père)")
   C --> C2("Gabrielle <br> Savoyard <br> (Mère)")
   D --> D1("Jules <br> Normand <br> (Père)")
   D --> D2("Odile <br> Picard <br> (Mère)")
   E --> F("Evariste <br> Breton <br> (Père)")
   E --> E1("Camèlia <br> Charentais <br> (Mère)")
   C1 --> C3("Léo <br> Comtois <br> (Père)")
   C1 --> C4("Eulalie <br> Lorrain <br> (Mère)")
   C2 --> C5("Guillaume <br> Savoyard <br> (Père)")
   C2 --> C6("Janet <br> Chesterfield <br> (Mère)")
```

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`.

Question 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) ?

Réponse

a. Un arbre binaire est un arbre d'arité 2, c'est à dire un arbre dans lequel chaque noeud possède au plus deux fils. C'est bien le cas ici, une personne ayant au maximum deux parents connus.
b. Dans un arbre binaire de recherche, on dispose d'une relation d'ordre entre les clés associées à chaque noeud et pour tout noeud, sa clé est supérieure aux clés du sous arbre gauche et inférieure aux clés du sous arbre droit. Ici les clés sont des personnes sur lesquelles on n'a pas de relation d'ordre.

Question 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 :

🐍 **Script Python**

```
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**.

Réponse

a. On rappelle que dans un parcours en *profondeur préfixe*, on liste en premier la racine puis récursivement les clés du sous arbre gauche et du sous arbre droit. Ce qui donne ici :
Albert Normand
–
Jules Normand
–
Michel Normand
–
Jules Normand
–
Odile Picard
–
Hélène Breton
–
Evariste Breton

b. Dans le parcours en *profondeur infixe*, on liste récursivement les clés du sag puis la racine puis les clés du sad. Ce qui donne ici : Jules Normand
–
Michel Normand
–
Odile Picard
–
Jules Normand
–
Evariste Breton
–
Hélène Breton
–
Camélia Charentais

c. En parcours **prefixe** on insère l'affichage du tuple `(prenom,nom)` **avant** de relancer les parcours récursifs sur les deux sous arbres.

🐍 **Script Python**

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
    noeud_actuel = racine_de_l_arbre
    print(noeud_actuel.identite)
```

```
    parcours(noeud_actuel.gauche)
    parcours(noeud_actuel.droite)
```

d. En parcours **infixe** on insère l'affichage du tuple `(prenom,nom)` **entre** les parcours récursifs sur les deux sous arbres.

🐍 **Script Python**

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

Question 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.

🐍 **Script Python**

```
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.

🐍 **Script Python**

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

Réponse

a.

🐍 **Script Python**

```
class Noeud() :
    def __init__(self, prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
        self.generation = 0
```

Bug

Dans l'énoncé, `self` ne figure pas dans les paramètres de `__init__` (ajouté dans cette correction)

b.

🐍 **Script Python**

```
def numerotation(racine_de_l_arbre, num_gen=0) :
    if racine_de_l_arbre != None:
        racine_de_l_arbre.generation = num_gen
        numerotation(racine_de_l_arbre.gauche,num_gen+1)
        numerotation(racine_de_l_arbre.droit,num_gen+1)
```

Question 4

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

🐍 **Script Python**

```
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

Réponse

Cette fonction parcourt l'arbre en préfixe mais affiche seulement les noeuds droit, ce qui donne :
Odile Picard
–
Hélène Breton
–
Camélia Charentais
–
Marie Comtois
–
Eulalie Lorrain
–
Gabrielle Savoyard
–
Janet Chesterfield

3. 2021 France J2

Exo

Une agence immobilière développe un programme pour gérer les biens immobiliers qu'elle propose à la vente.
Dans ce programme, pour modéliser les données de biens immobiliers, on définit une classe Bim avec les attributs suivants :

- `nt` de type `str` représente la nature du bien (appartement, maison, bureau, commerces,...);
- `sf` de type `float` est la surface du bien ;
- `pm` de type `float` est le prix moyen par m² du bien qui dépend de son emplacement.

La classe `Bim` possède une méthode `estim_prix` qui renvoie une estimation du prix du bien. Le code (incomplet) de la classe `Bim` est donné ci-dessous :

🐍 **Script Python**

```
class Bim:
    def __init__(self, nature, surface, prix_moy):
        ...

    def estim_prix(self):
        return self.sf * self.pm
```

Question 1

Recopier et compléter le code du constructeur de la classe `Bim`.

Réponse

🐍 **Script Python**

```
def __init__(self, nature, surface, prix_moy):
    self.nt = nature
    self.sf = surface
    self.pm = prix_moy
```

Question 2

On exécute l'instruction suivante :

Script Python

```
b1 = Bim('maison', 70.0, 2000.0)
```

Que renvoie l'instruction `b1.estim_prix()` ?
Préciser le type de la valeur renvoyée.

Réponse

L'instruction `b1.estim_prix()` renvoie 140000.0 (estimation du prix de b1) ; la valeur renvoyée est de type flottant, car `self.sf` et `self.pm` sont de type flottant.

Question 3

On souhaite affiner l'estimation du prix d'un bien en prenant en compte sa nature :

- pour un bien dont l'attribut `nt` est 'maison' la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 1,1 ;
- pour un bien dont l'attribut `nt` est 'bureau' la nouvelle estimation du prix est le produit de sa surface par le prix moyen par m² multiplié par 0,8 ;
- pour les biens d'autres natures, l'estimation du prix ne change pas.

Modifier le code de la méthode `estim_prix` afin de prendre en compte ce changement de calcul.

Réponse

🐍 **Script Python**

```python
def estim_prix(self):
    prix_brut = self.sf * self.pm
    if self.nt=='maison':
        return prix_brut*1.1
    elif self.nt=='bureau':
        return prix_brut*0.8
    else :
        return prix_brut
```

Question 4

Écrire le code Python d'une fonction `nb_maison(lst)` qui prend en argument une liste Python de biens immobiliers de type `Bim` et qui renvoie le nombre d’objets de nature 'maison' contenus dans la liste `lst`.

Réponse

🐍 **Script Python**

```python
def nb_maison(lst):
    compteur = 0
    for b in lst :
        if b.nt=='maison':
            compteur = compteur + 1
    return compteur
```

Question 5

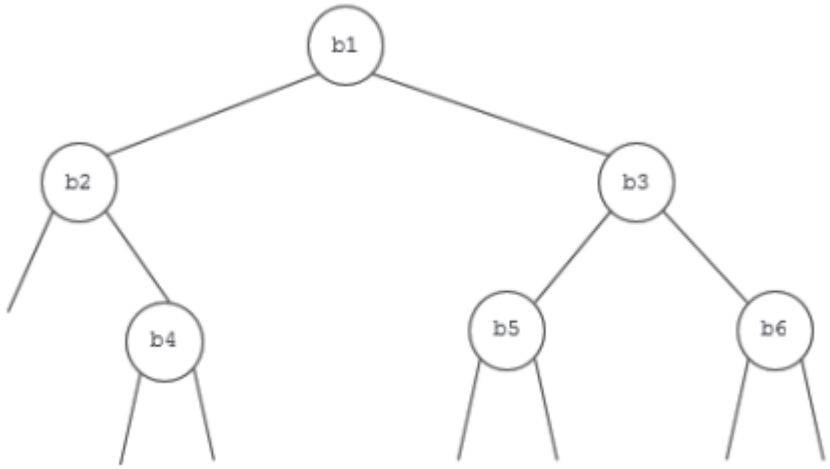
Pour une recherche efficace des biens immobiliers selon le critère de leur surface, on stocke les objets de type `Bim` dans un arbre binaire de recherche, nommé `abr`. Pour tout nœud de cet arbre :

- tous les objets de son sous-arbre gauche ont une surface inférieure ou égale à la surface de l'objet contenue dans ce nœud ;
- tous les objets de son sous-arbre droit ont une surface strictement supérieure à la surface de l'objet contenue dans ce nœud.

L'objet abr dispose des méthodes suivantes :

- `abr.est_vide()` : renvoie True si abr est vide et False sinon.
- `abr.get_v()` : renvoie l'élément (de type `Bim`) situé à la racine de `abr` si `abr` n'est pas vide et `None` sinon.
- `abr.get_g()` : renvoie le sous-arbre gauche de `abr` si `abr` n'est pas vide et `None` sinon.
- `abr.get_d()` : renvoie le sous-arbre droit de abr si `abr` n'est pas vide et `None` sinon.

a. Dans cette question, on suppose que l'arbre binaire abr a la forme ci-dessous :



Donner la liste les biens b1, b2, b3, b4, b5, b6 triée dans l'ordre croissant de leur surface.
b. Recopier et compléter le code de la fonction récursive contient donnée ci-dessous, qui prend en arguments un nombre surface de type float et un arbre binaire de recherche abr contenant des éléments de type Bim ordonnés selon leur attribut de surface sf. La fonction `contient(surface, abr)` renvoie `True` s'il existe un bien dans abr d'une surface supérieure ou égale à surface et `False` sinon.

🐍 Script Python

```
def contient(surface, abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= ...:
        return True
    else:
        return contient( surface,... )
```

Réponse

a. on effectue un parcours infixe : b2 - b4 - b1 - b5 - b3 - b6

b.

🐍 Script Python

```
def contient(surface,abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= surface:
        return True
    else :
        return contient(surface, abr.get_d())
```