



Samedi 13 Avril 2019

OPTION : SCIENCES DU NUMERIQUE

MP / PC / PSI / PT / TSI

Durée : 2 Heures

Condition(s) particulière(s)

Calculatrice interdite

Remettre le QCM avec vos copies d'examen en indiquant votre code candidat

Introduction

Le sujet de l'option informatique contient quatre exercices allant de l'écriture de fonctions à l'analyse de code. Le langage utilisé, dans tous les cas, est Python. Le premier exercice traite de matrice, les exercices 2 et 4 parlent d'arbres binaires, quant au 3ème, il s'agit de déterminer ce que fait une fonction donnée. Ils sont respectivement évalués en 2, 6, 4 et 8 points. Vous trouverez, en dernière page, les différentes fonctions et structures de données autorisées.

A) Matrices : recherche ... (2 points)

Écrire la fonction **searchMatrix(M, x)** qui retourne la position (i, j) de la première valeur **x** trouvée dans la matrice **M** (que l'on supposera non vide). Si **x** n'est pas présent, la fonction retourne $(-1, -1)$.

Exemples d'application avec la matrice
Mat1 ci-contre:

```
>>> searchMatrix(Mat1, -5)
(2, 5)

>>> searchMatrix(Mat1, 5)
(1, 4)

>>> searchMatrix(Mat1, 15)
(-1, -1)
```

Mat1

1	10	3	0	-3	2	8
-1	0	1	8	5	0	-4
10	9	14	1	4	-5	1
10	-3	7	11	6	3	0
7	8	-5	1	5	4	10

B) Arbres binaires : branche de valeur maximum ... (6 points)

Dans un arbre binaire, nous définissons *la valeur d'une branche* comme étant la somme des valeurs des nœuds sur cette branche.

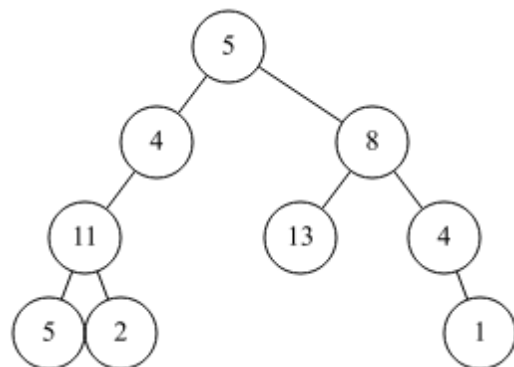
Ecrire la fonction **maxbranch(B)** qui détermine la plus grande valeur des branches de l'arbre binaire **B** (dont les clés sont des entiers).

Les arbres binaires manipulés ici (on supposera la classe BinTree importée) sont :

- **None** pour un arbre vide
- Un arbre binaire non vide possède 3 attributs : *key*, *left*, *right*.

Par exemple, dans l'arbre ci-contre, la somme retournée sera 26.

$$5 + 8 + 13 = 26$$



C) Qu'est-ce que c'est ? ... (4 points)

Soit la fonction **what** ci-dessous :

```
def what (p, v):
    n = len(p)
    if n < 2:
        raise Exception ("not enough ")
    (a, b) = p[0]
    (c, d) = p[1]
    i = 1
    while i < n - 1 and b < v:
        i += 1
        (a, b) = (c, d)
        (c, d) = p[i]
    return b + (d - b) * (v - a) / (c - a)
```

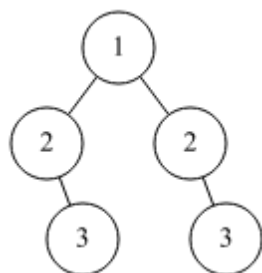
1) Donner les résultats des applications suivantes de **what** :

- a) `what([(0,0), (10,10), (20, 20), (30, 30)], 15)`
- b) `what([(0,0), (10,20), (20,40), (30,60)], 24)`
- c) `what([(0,0), (1, 10), (2,100), (3, 1000)], 2.5)`
- d) `what([(0,3), (1,6), (2,9), (3,10), (4,15)], 20)`

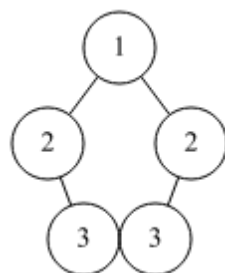
2) Soit **L** la liste de couples d'entiers $[(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})]$ et **Y** une liste numérique. Que calcule **what(L, Y)** ?

D) Arbres binaires: symétrique ... (8 points)

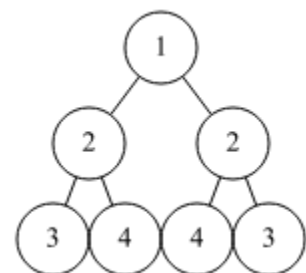
Un arbre binaire est dit *symétrique* s'il est identique à son miroir (arbre renversé selon un axe vertical).



Arbre non symétrique



Arbre symétrique



Arbre symétrique

Ecrire la fonction **symmetric(B)** qui vérifie si un arbre binaire **B** est symétrique. La méthode est laissée à votre imagination, mais bien évidemment, plus elle sera optimisée, mieux ce sera.

Les arbres binaires manipulés ici (on supposera la classe `BinTree` importée) sont :

- **None** pour un arbre vide
- Un arbre binaire non vide possède 3 attributs : *key*, *left*, *right*.

Consignes Python

Tout code doit être écrit dans le langage Python.

- Tout code Python non indenté ne sera pas corrigé.
- Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué ci-dessous.
- Vous pouvez écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.

Fonctions et méthodes autorisées

Vous pouvez utiliser la fonction range :

```
>>> for i in range(10):
...:     print(i, end=' ')

0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...:     print(i, end=' ')

5 6 7 8 9
```

Sur les listes, vous pouvez utiliser la méthode append et la fonction len :

```
>>> help ( list . append )
Help on method_descriptor : append (...)
L.append ( object ) -> None -- append object to end of L

>>> help (len)
Help on built-in function len in module builtins : len (...)
len ( object )
Return the number of items of a sequence or collection .
```

Les matrices sont représentées par des listes de listes comme dans l'exemple ci-dessous :

```
>>> M1 = [[1, 10, 3, 0, 3, 10, 1],
          [1, 0, 1, 8, 1, 0, 1],
          [10, 9, 4, 1, 4, 9, 10],
          [10, 3, 7, 1, 7, 3, 10],
          [7, 8, 5, 1, 5, 8, 7]]
```

Les arbres binaires « classiques » sont représentés par:

```
class BinTree:
    def __init__(self, key, left, right)
        self.key = key
        self.left = left
        self.right = right
```