Devoir surveillé n°6 – NSI
19/01/2024

Ce devoir est du type épreuve pratique, à réaliser sur ordinateur, avec le fichier contenant le code Python à compléter fourni.

Exercice 1

La classe `Noeud` ci-dessous permet d'implémenter une structure d'arbre binaire de recherche (ABR). On considère que la clé d'un nœud est un entier et qu'elle est unique dans l'ABR.

```
class Noeud:
    def __init__(self, valeur):
        '''Méthode constructeur pour la classe Noeud.
        Paramètre d'entrée : valeur (int)'''
        self.valeur = valeur
        self.gauche = None
        self.droit = None

    def getValeur(self):
        '''Méthode accesseur pour obtenir la valeur du noeud
        Aucun paramètre en entrée'''
        return self.valeur

    def droitExiste(self):
        '''Méthode renvoyant True si l'enfant droit existe
        Aucun paramètre en entrée'''
        return (self.droit is not None)

    def gaucheExiste(self):
        '''Méthode renvoyant True si l'enfant gauche existe
        Aucun paramètre en entrée'''
        return (self.gauche is not None)

    def inserer(self, cle):
        '''Méthode d'insertion de clé dans un arbre binaire de recherche
        Paramètre d'entrée : cle (int)'''
        if cle < ...:
            # on insère à gauche
            if self.gaucheExiste():
                # on descend à gauche et on retente l'insertion de la clé
                ...
            else:
                # on crée un fils gauche
                self.gauche = ...
        elif cle > ... :
            # on insère à droite
            if ... :
                # on descend à droite et on retente l'insertion de la clé
                ...
            else:
                # on crée un fils droit
                ... = Noeud(cle)
```

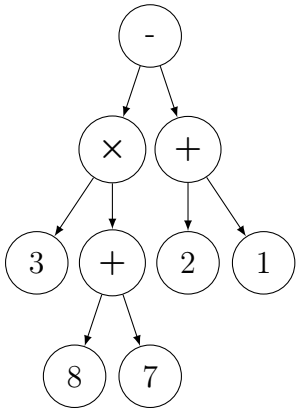
Compléter la fonction récursive `inserer` afin qu'elle permette d'insérer un nœud dans l'arbre binaire de recherche proposé, à l'aide de sa clé. Voici un exemple d'utilisation :

```
>>> arbre = Noeud(7)
>>> for cle in (3, 9, 1, 6):
        arbre.inserer(cle)
>>> arbre.gauche.getValeur()
3
>>> arbre.droit.getValeur()
9
>>> arbre.gauche.gauche.getValeur()
1
>>> arbre.gauche.droit.getValeur()
6
```

Exercice 2

Une expression arithmétique ne comportant que les quatre opérations $+$, $-$,$\times$,$\div$ peut être représentée sous forme d'arbre binaire. Les nœuds internes sont des opérateurs et les feuilles sont des nombres. Dans un tel arbre, la disposition des nœuds joue le rôle des parenthèses que nous connaissons bien.

En parcourant en profondeur infixe l'arbre binaire ci-contre, on retrouve l'expression notée habituellement :
$(3 \times (8 + 7)) - (2 + 1)$.
La classe `Node` ci-après permet d'implémenter une structure d'arbre binaire.



Compléter la fonction récursive `expression_infixe` qui prend en paramètre un objet de la classe `Node` et qui renvoie l'expression arithmétique représentée par l'arbre binaire passé en paramètre, sous forme d'une chaîne de caractères contenant des parenthèses.

Résultat attendu avec l'arbre ci-dessus :

```
>>> e = Node(Node(Node(None, 3, None),
    '*', Node(Node(None, 8, None), '+', Node(None, 7, None))),
    '-', Node(Node(None, 2, None), '+', Node(None, 1, None)))
>>> expression_infixe(e)
'((3*(8+7))-(2+1))'
```

```
class Node:
    '''
    classe implémentant un noeud d'arbre binaire
    '''

    def __init__(self, g, v, d):
        '''
        un objet Node possède 3 attributs :
        - gauche : le sous-arbre gauche,
        - valeur : la valeur de l'étiquette,
        - droit : le sous-arbre droit.
        '''
        self.gauche = g
        self.valeur = v
        self.droit = d
```

```
    def __str__(self):
        '''
        renvoie la représentation du noeud en chaine de caractères
        '''
        return str(self.valeur)

    def est_une_feuille(self):
        '''
        renvoie True si et seulement si le noeud est une feuille
        '''
        return self.gauche is None and self.droit is None


def expression_infixe(e):
    s = ...
    if e.gauche is not None:
        s = '(' + s + expression_infixe(...)
    s = s + ...
    if ... is not None:
        s = s + ... + ...
    return s
```