IN101 - TD06

Énoncé

Instructions générales

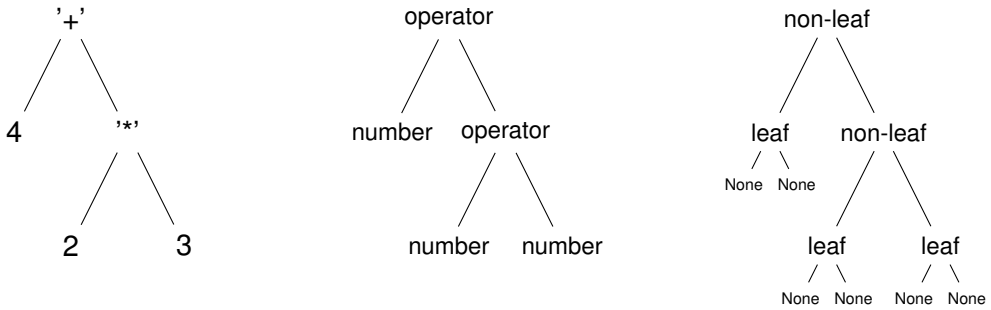
— En cas de problèmes, demandez de l'aide ! Mais n'oubliez pas de vous référer, en premier lieu, au diapositives du cours magistral et aux références bibliographiques. L'index disponible en ligne vous aidera à lier concepts abordés et suports de cours.
— Utilisez l'éditeur de texte de votre choix pour les exercices. En cas d'hésitation, privilégiez gedit.
— Si ce n'est pas encore le cas, créez un dossier `IN101/`.
— Dans ce dossier `IN101/`, créez un nouveau sous-dossier pour la séance de TD, que vous nommerez `TD06/`.
— Pour les questions du type : "Write, in natural language, an algorithm that. . . ", ouvrez un nouveau fichier texte (d'extension ".txt") plutôt qu'un fichier python (d'extension ".py").
Exemple : `helloworld.txt`
— Documentez et commentez précisément votre code : décrivez l'objectif général de l'algorithme, expliquez les points-clefs de son implémentation, détaillez les conditions normales d'éxécution du script (c.-à-d. le fichier ".py") et exposez les cas extrêmes et les erreurs prévus.

Everyone

1 Mathematical Expressions in Infix Notation : Trees

Aims: *Apply a binary tree to a concrete problem : evaluating mathematical expression in infix notation.*

In this exercise, we will represent a mathematical expression, for example $4 + (2 * 3)$, as a tree, as illustrated below. As these trees exemplify, every leaf node should contain a number, and every non-leaf node should contain a mathematical operator.



For this exercise, you *MUST* download the module math_expression_tree (`http://perso.ensta-paristech.fr/~paun/ENSTA_IN101/math_expression_tree.py`), and use it as a basis for your work. This module already contains the names of the functions and appropriate asserts. Do *NOT* change the name of the module or the functions inside.

Several tests have already been provided in the module, but these tests are currently commented out. Once you have written a function, uncomment the tests to see if your function succeeds. Do this *incrementally*, instead of waiting until you have written all the functions.

Q1 Have a look at the existing code, and understand the relationship between the __init__ function and the illustration above. The main idea is that the data field in a MathTreeNode can be either an integer number (1, 2, 3, etc), or a mathematical operator represented as a

character ('+', '-', '*' or '/'). For example, representing the mathematical expression $(2 * 3)$ with a tree consisting of MathTreeNodes would be done as follows :

tree_object = MathTreeNode(MathTreeNode(None,2,None), '$*$', MathTreeNode(None,3,None)))

Q2 Implement the member function is_leaf(self). Uncomment the relevant tests to see if the function works as expected.

Q3 Implement the member function __str__(self) that returns a string representation of the tree (including brackets) in infix notation. For instance, printing the tree above should yield the string '(4+(2*3))'. Uncomment the relevant tests to see if the function works as expected.

Q4 Implement the member function prefix_expression(self) that returns a string representation of the tree in prefix notation, which is also known as the "Polish notation". See the docstring of this function for examples. Uncomment the relevant tests to see if the function works as expected.

Q5 Implement the member function postfix_expression(self) that returns a string representation of the tree in postfix notation, which is also known as the "reverse Polish notation". See the docstring of this function for examples. Uncomment the relevant tests to see if the function works as expected.

Q6 Implement the member function evaluate(self), which evaluates the mathematical expression, i.e. the result of calling tree_object.evaluate() on the above example tree representing $(4 + (2 * 3))$ should yield the number '10'. Uncomment the relevant tests to see if the function works as expected.

```
while number_of_things_i_dont_understand() > 0:
    ask_question_to_mdc()
```

There will be an exam soon on the entire course. If you have any general/specific questions about algorithmics/Python/debugging, this TD is a good opportunity (the last...) to ask your MdC. Also, if you have not completed last week's TD on linked lists, now is a good time to do so.

If you are confident that you have understood everything : below is the last exercise of the this week's TD !

Advanced

2 Searching in trees

Aims: *In the previous exercise, the MathTreeNode represented a node in a binary tree, because it had two subnodes (left and right). In this exercise, we consider trees that can have an arbitrary number of subnodes, and we perform search within this tree.*

Download the following module : `http://perso.ensta-paristech.fr/~paun/ENSTA_IN101/search_tree.py` Several tests have already been provided in the module, but these tests are currently commented out. Once you have written a function, uncomment the tests to see if your

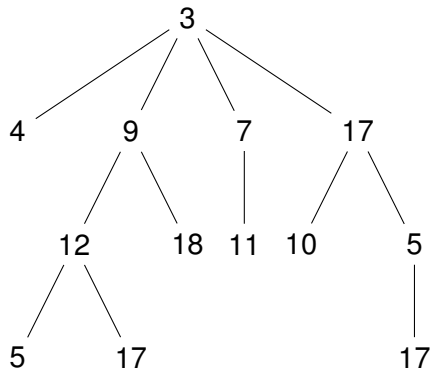
function succeeds.

Q7 Have a look at the __init__ and add_child functions. How is the tree illustrated below represented by the TreeNode ? Why do you think the member variable is called 'children' ? Why is it a list ?

Q8 Implement the member function find(self,data) to TreeNode. This function returns True if a TreeNode or any of its descendants contains data, and False otherwise. In the example tree below, tree.find(9) should return True, but tree.find(8) should return False. Uncomment the relevant tests to see if the function works as expected.

Q9 Implement the member function find_min_depth(self,data) to TreeNode. This function returns the minimum depth at which a value was found, and None if the value could not be found at the tree. In the example tree below, tree.find_depth(3) should return 0, tree.find_depth(17) should return 1 (not 3 !), but tree.find(8) should return None. (why is it not Zen to return False in this function ?). Uncomment the relevant tests to see if the function works as expected.

Q10 Implement the member function count_up_to_depth(self,data,depth) which counts the number of occurences of 'data' in the tree, but only up to a certain depth. In the example tree below, tree.count_up_to_depth(17) should return 0, 1, 1, 3 for depths of 0, 1, 2, 3, 4, 5 respectively. Uncomment the relevant tests to see if the function works as expected.



3 New Tree structure

Aims: *Understand Abstract Data Types*

In a new module, called `dic_tree.py` provide a new full implementation of the Tree Abstract Data Type using the default Python structure `dict`. All methods should be implemented as simple functions that may take as an argument a variable of `dict` type.

Hint

Use a custom key ("data" for example) to store (as the value) the generic type of the information stored in the node. Use an additional key (called "children" for example) to store the list of children.