CSCI-SHU 210 Data Structures

Assignment 7 Binary Trees

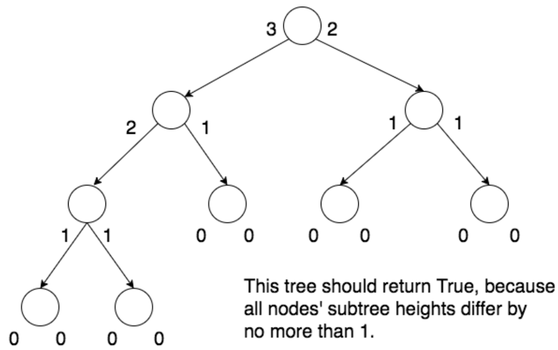
Please download LinkedBinaryTree.py, the starting point of this assignment.
Your tasks are located at line 398.

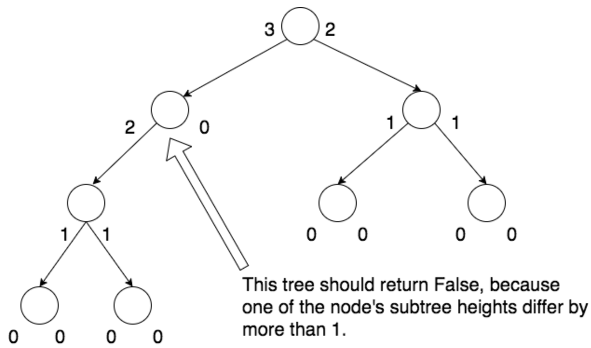
Problem 1: is the tree height balanced?

Implement function `is_height_balanced(self)`
When called on a tree, it will return True if the tree is height balanced, or False otherwise.

Let's define some definitions:

- None has height 0
- Leaf node has height 1
- Other nodes have height max(height of left child, height of right child) + 1.
- We consider a tree is height balanced, if for every node within this tree, the height of this node's left child, height of this node's right child, differ by no more than 1.






```
Example function call:
>>> T1.is_height_balanced()   # Suppose T1 is the left tree above
True
>>> T2.is_height_balanced()   # Suppose T2 is the right tree above
False
```

Important:

- Bonus 5 points if you solved with runtime O(n) respect to tree size.
- You may want to declare additional functions with extra parameters, then use/call the new function to perform recursion task.

Problem 2: Same Tree

Implement function `sameSame(self, other)`.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

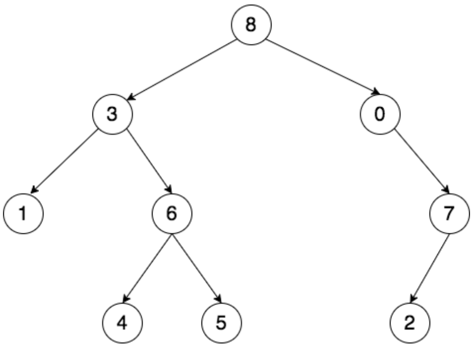
```
Input:     1         1              Input:     1           1
          / \       / \                       /             \
         2   3     2   3                     2               2
return: True                        return: False
```

Important:

- If both trees are None, return True.
- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.

Problem 3: Sum of leaves

Implement function `sum_of_leaves(self).`
This function returns the sum value of leaves. For example, if the following tree is given:



We have 4 leaf nodes within this tree. Their sum is: 1 + 4 + 5 + 2 = 12.

```
Example function call:
>>> T.sum_of_leaves()
12
```

Important:

- If root is None, return 0.
- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.

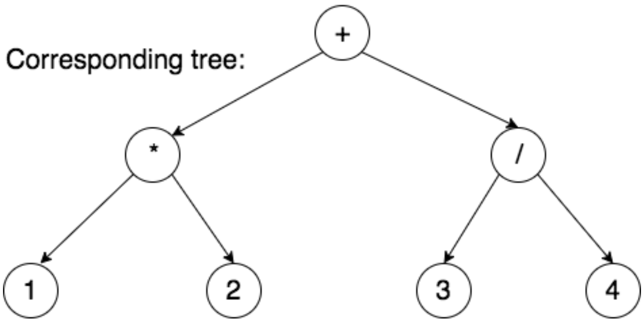
Problem 4: Expression Tree

In problem 4, your task is to build an `Expression Tree` from `postfix input`. Benefits of the expression tree are:
If we perform preorder traversal on the tree, we get the prefix expression;
If we perform inorder traversal on the tree, we get the infix expression;
If we perform postorder traversal on the tree, we get the postfix expression;

Our textbook introduced building an expression tree using infix expression inputs. However, using infix input, unnecessary parentheses are required.

Textbook infix Example: ((1 * 2) + (3 / 4))
For our postfix question: 1 2 * 3 4 / +

Corresponding tree:



Preorder traversal:	+ * 1 2 / 3 4
Inorder traversal:	1 * 2 + 3 / 4
Postorder traversal:	1 2 * 3 4 / +

More info:

Implement function `build_expression_tree(postfix)`. When called, it should **return** a LinkedBinaryTree object that represents the `Expression Tree` for the `postfix input string`.

```
Example function call:
>>> tree = build_expression_tree("1 2 * 3 4 / +")
## then the variable tree, is the expression tree above.
```

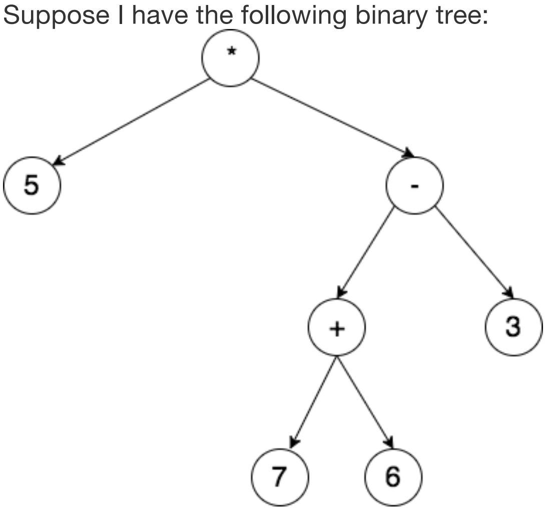
Important:

- This function is not a member(self) function of class LinkedBinaryTree.
  - In other words, you have no access to `self` Tree
  - Create (and return) a new instance of `LinkedBinaryTree`
  - Use the `postfix input string`
- Input postfix string contains spaces between each operand/operator.
- For data storage within our expression tree,
  - Operators are stored as string. Example: "+"
  - Numbers are stored as integer. Example: 9
- Test cases will only include valid postfix expressions.

Problem 5: Evaluating Expression Tree

Your task is to implement function `evaluate(self)`. This function evaluates the numeric result of expression trees.

Suppose I have the following binary tree:




```
Example function call:
>>> T.evaluate()     # Suppose T is the expression tree above
50
>>> build_expression_tree("5 7 6 + 3 - *").evaluate()   # Task4 + Task5
50
```

Important:

- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.
- Test cases will only include valid expression trees.