

CSCI-SHU 210 Data Structures

Assignment 7 Binary Trees

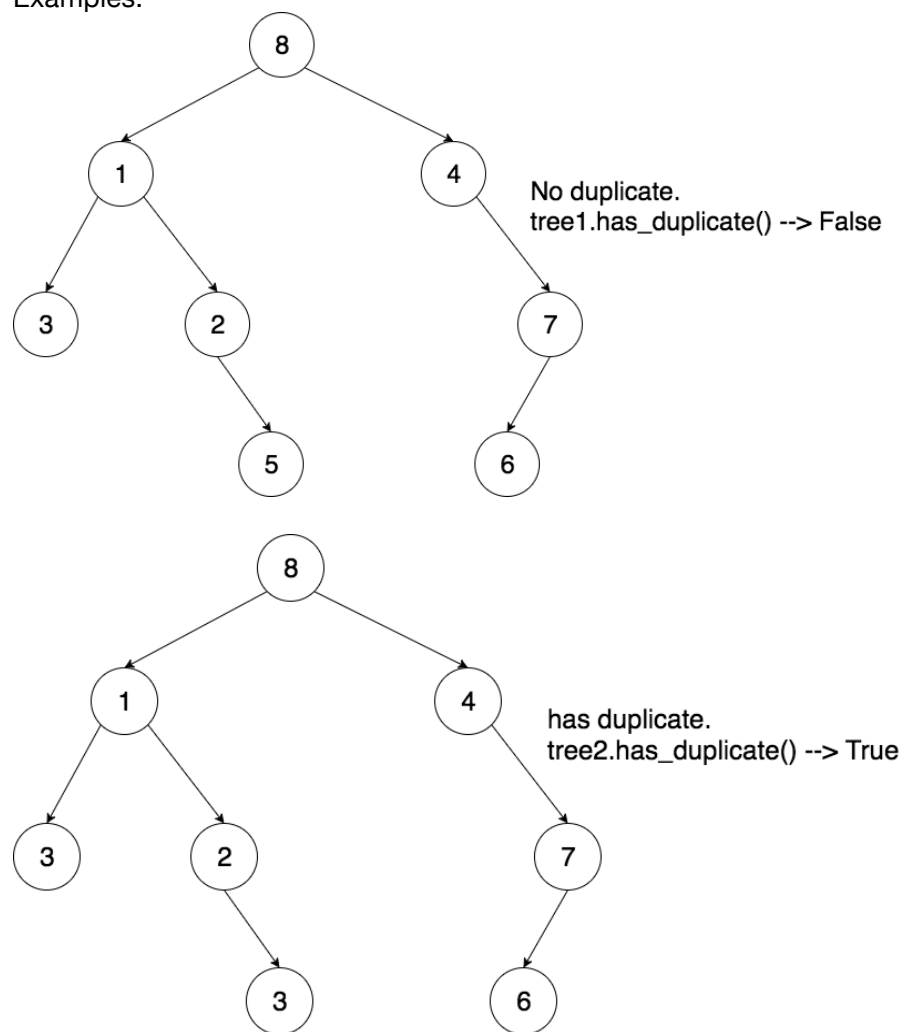
Please download Assignment7.py, the starting point of this assignment.

Problem 1: has duplicate

Implement function `has_duplicate(self)`

When called on a tree, it will return True if self binary tree contains duplicate values. Returns False otherwise.

Examples:



Important:

- You may want to declare additional functions with extra parameters, then use/call the new function from `has_duplicate(self)` to perform recursion task.

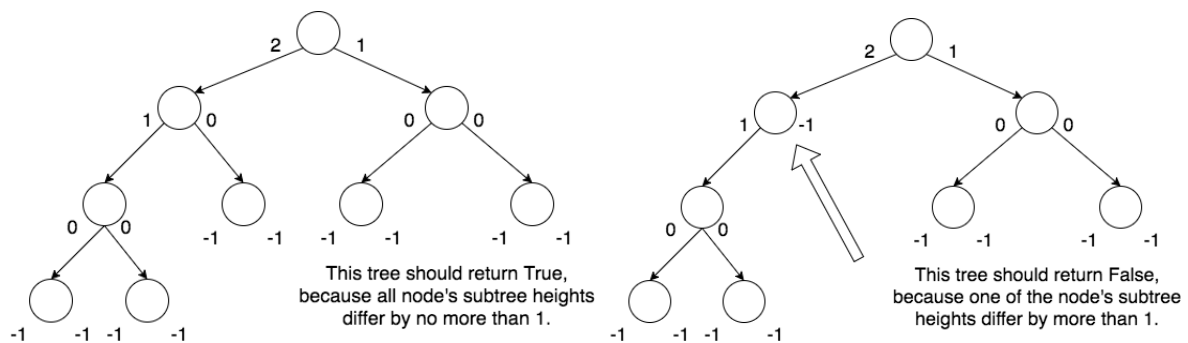
Problem 2: is the tree height balanced?

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 -1
- Leaf node has height 0
- Other nodes have height $\max(\text{height of left child, height of right child}) + 1$.
- 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
```

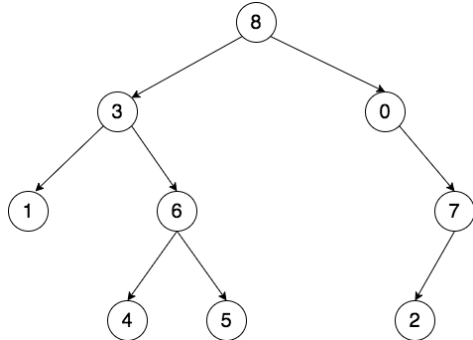
Important:

- Bonus 5 points if you solved with runtime $O(N)$ (suppose tree size = N).
- You may want to declare additional functions with extra parameters, then use/call the new function from `is_height_balanced(self)` 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: isomorphic trees

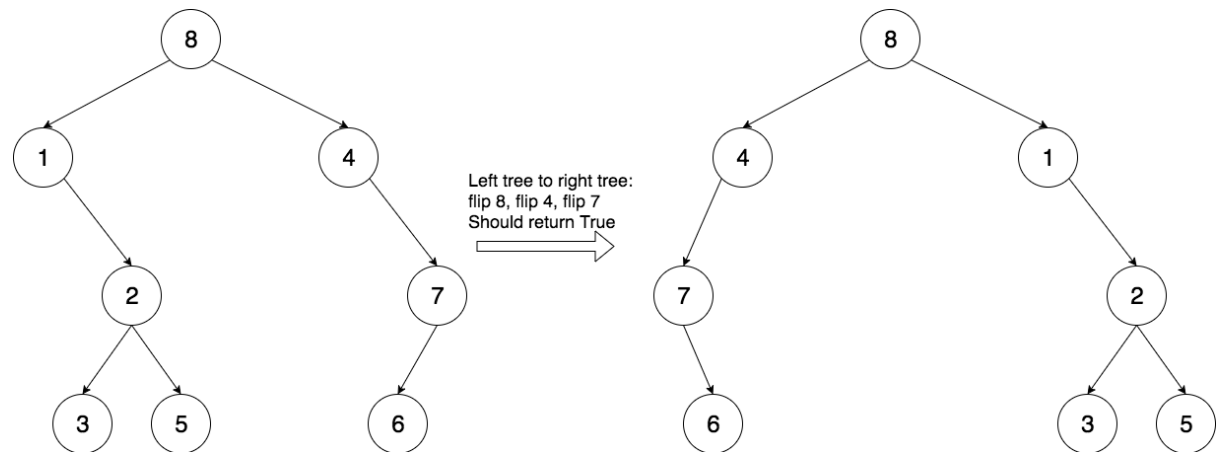
Implement function `is_isomorphic(tree1, tree2)`.

This function returns `True` if binary tree `tree1` and binary tree `tree2` are isomorphic. Returns `False` otherwise.

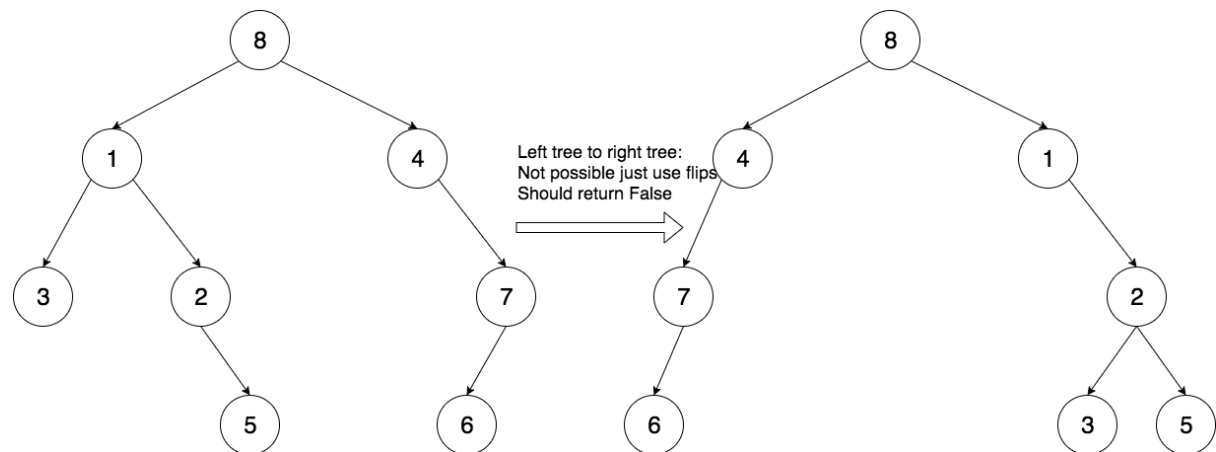
Definition of isomorphic:

Two trees are called isomorphic if one of them can be obtained from other by a series of flips on multiple nodes.

Example 1: (Should return True)



Example 2: (Should return False)



Important:

- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.

Problem 5: Expression Tree

In problem 5, 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;

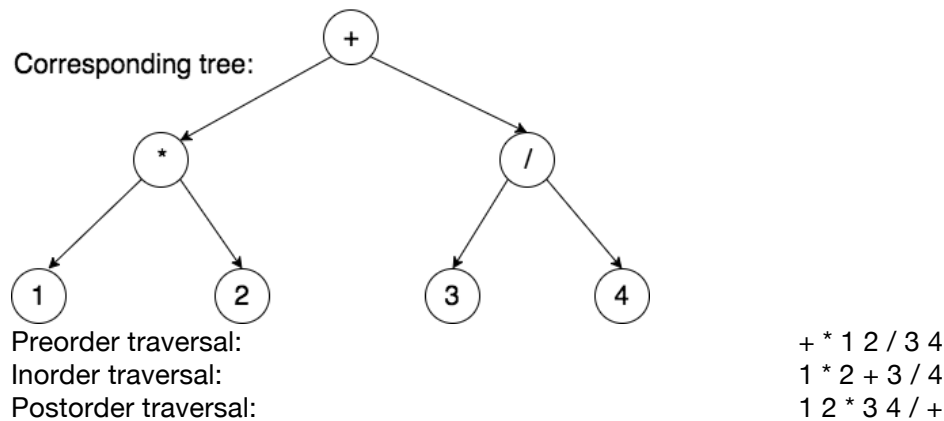
If we perform inorder traversal on the tree, we get the infix expression;

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: $1\ 2\ *\ 3\ 4\ /\ +$



More info:

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

Example function call:

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

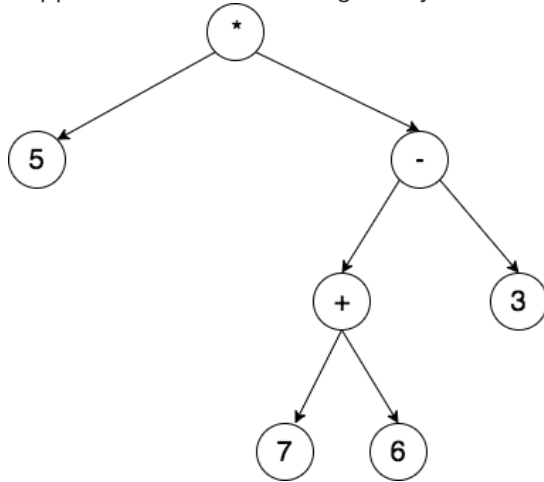
Important:

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

Problem 6: 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()  # Task5 + Task6
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.