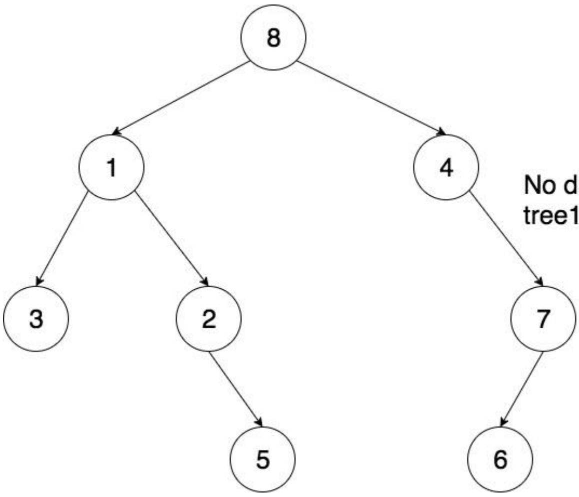
CSCI-SHU 210 Data Structures

Homework Assignment 6 Binary Trees

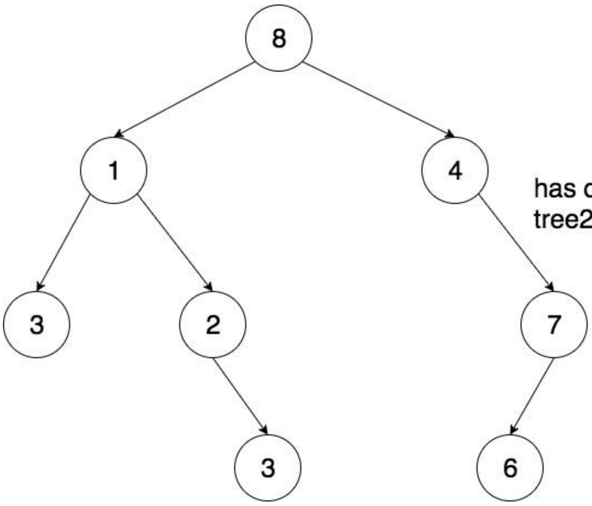
Problem 1: has duplicate

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:



No duplicate.
tree1.has_duplicate() --> False



has duplicate.
tree2.has_duplicate() --> True

Important:

- You may want to declare additional functions with extra parameters, then use/call the new function from `has_duplicate(self)` to perform task
- You should not use Python lists or any other data storage (O(1) memory complexity)
- Time Complexity O(n^2), n is the number of nodes

Problem 2: isomorphic trees

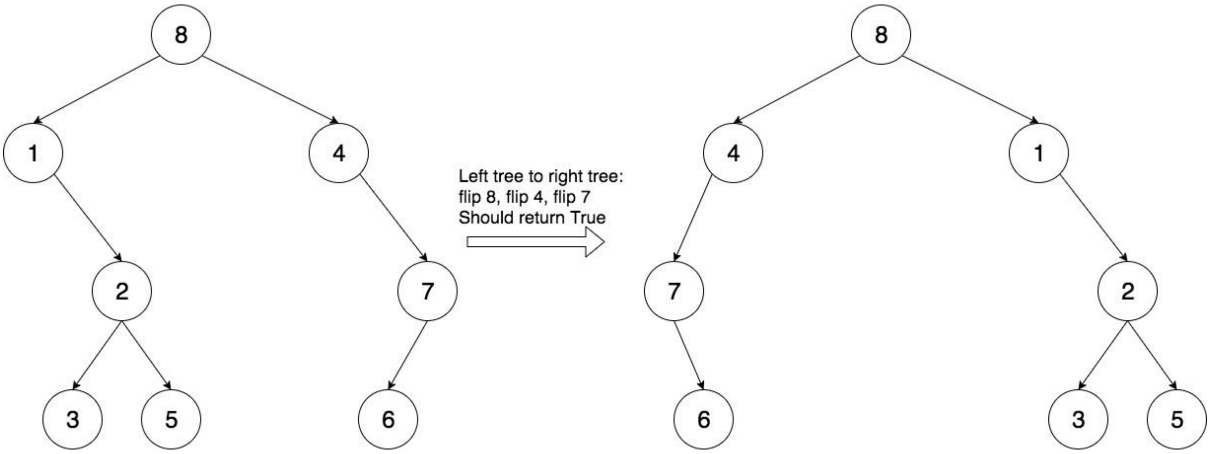
Implement function `is_isomorphic(tree1, tree2).`

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

Definition of isomorphic:

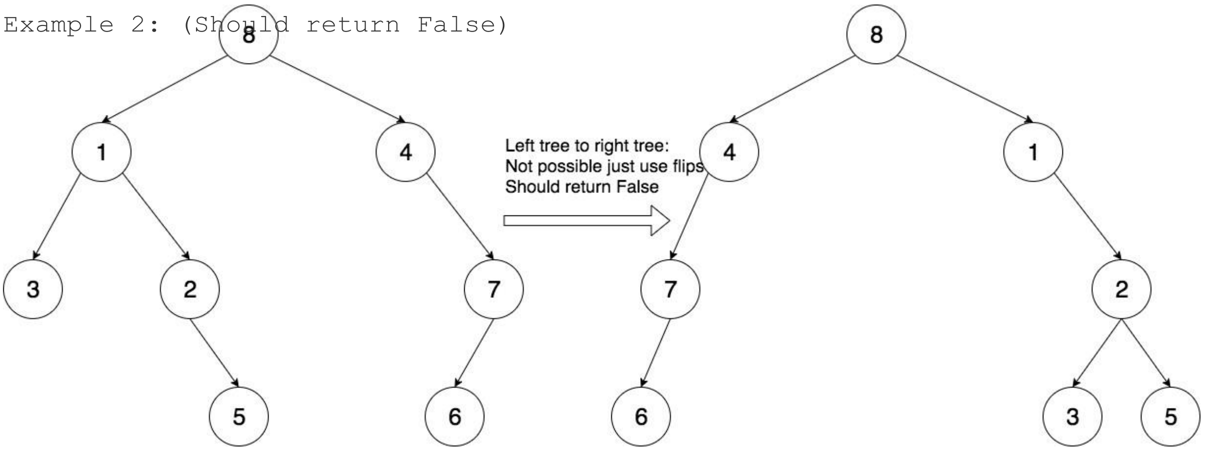
Two trees are called isomorphic if one of them can be obtained from other by a series of flips on multiple nodes.

`Example 1: (Should return True)`



Left tree to right tree:
flip 8, flip 4, flip 7
Should return True

`Example 2: (Should return False)`



Left tree to right tree:
Not possible just use flips
Should return False

Important:

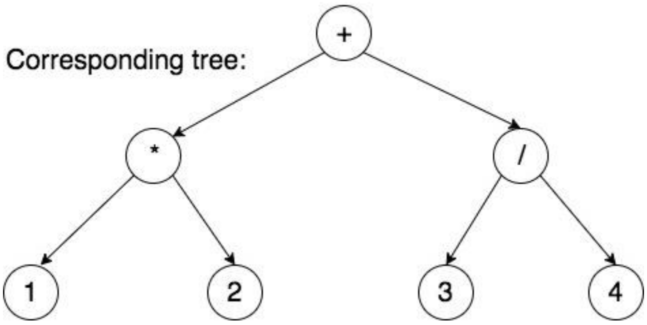
- This function is not a method(self) function of class Tree.
  - In other words, you have no access to `self`.
  - `tree1, tree2` are two trees
- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.

Problem 3: Expression Tree

Your task is to build an `Expression Tree` from `infix 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;

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



The aim is to build an expression tree using **infix** expression.
Implement function `build_expression_tree(infix)`. When called, it should return a class `Tree` object that represents the `Expression Tree` for the given `infix input string`.

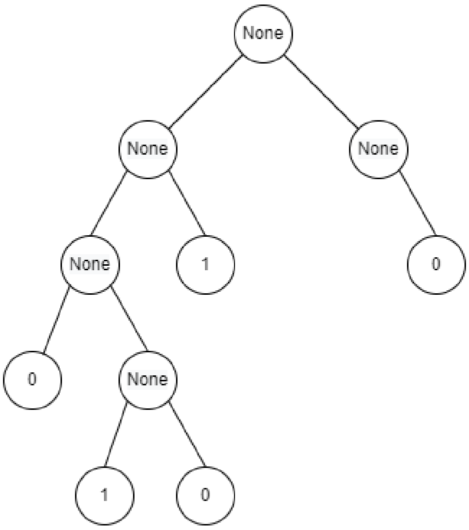
```
Example function call:

>>> tree = build_expression_tree(“1 * 2 + 3 / 4”)

## then the variable tree, is the expression tree above.
```

Important:

- This function is not a method(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 `infix input string.`
- Input infix string contains spaces between each operand/operator.
    - For simplicity, there will not be brackets in the infix expression.
- 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 infix expressions.
- Hint: Use two stacks (two python lists as stacks)
    - For same-priority operators, we follow the left-to-right rule. E.g., for “1 + 2 – 3”, we build the subtree “1 + 2” first, and then link this as the left child to “–“.

Problem 4: One-zero Tree

Given a binary tree T that has `0` or `1` stored as the element in its leaves. For internal nodes, the elements are initialized as `None.`

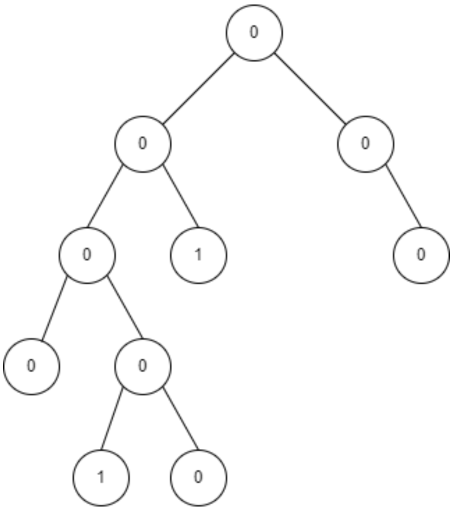


The aim is to update the element of each node following Dr. X's rule:

- If the node has depth 0, 2, 4, …, its element equals the **maximum** of all its children's element
- If the node has depth 1, 3, 5, …, its element equals the **minimum** of all its children's element

Implement function `zero_one_update(T)` that updates **all** node's element in T according to Dr X's rule. Recall depth is defined as the number of edges needed from the root to a certain node. For example, root has depth 0.

After running `zero_one_update(T)` passing in the above tree T, it should be updated to:



Important:

- This function is not a method(self) function of class Tree.
  - In other words, you have no access to `self`.
  - tree `T` (`class Tree`) is given to you and its elements are already initialized as described
  - Return T with all elements updated.
- No Python lists allowed
- Your runtime complexity should be O(n), n is the number of nodes
- You may want to declare additional functions with extra parameters, then use the new function to perform recursion task.