Exercise 5: Trees

Due date: Sunday November 5 before 10pm

Starter code

- ex5.py
- ex5_test.py

Task 1: Another tree method

In this task, you'll get another chance to practice using recursion on trees.

In the starter code, write a recursive method `__eq__` that tests whether two trees are equal. Two trees are equal if and only if their root values are equal, they have the same number of subtrees, and each of the corresponding subtrees are equal. *Order matters*.

Note that `__eq__` is another built-in Python method, and can be called either using the regular method call syntax `tree1.__eq__(tree2)`, or the more convenient syntax `tree1 == tree2`.

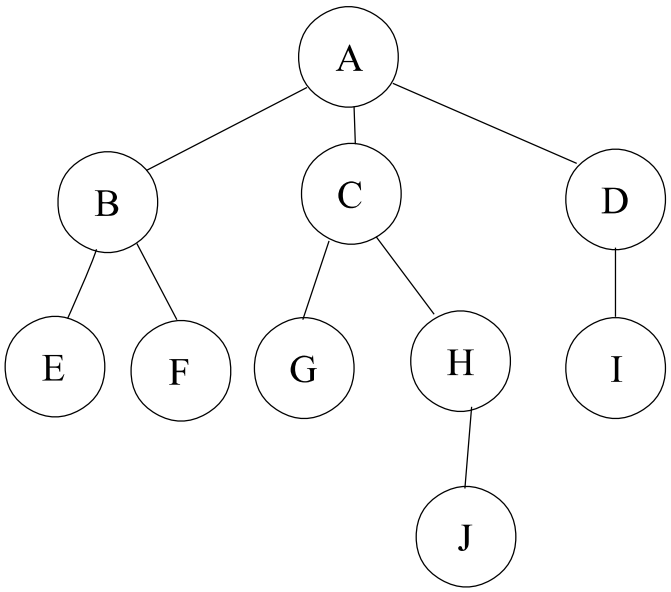
A corner case: if `tree1` is empty, then `tree1 == tree2` is `True` if and only if `tree2` is also empty. Don't forget about this!

You may not use any other Tree methods here, other than `is_empty` and helpers you defined yourself. You may access all Tree attributes.

Task 2: Nested lists and trees

We have already noted the structural similarities between trees and nested lists in lecture. In fact, we can represent every tree as a nested list, where the first item of the nested list is the root of the tree, and each other item in the nested list is a nested list representation of one of the tree's subtrees.

For example, consider the following tree:



Its nested list representation is

```
['A', ['B', ['E'], ['F']],
      ['C', ['G'], ['H', ['J']]],
      ['D', ['I']]]
```

Note: we've done some extra whitespace formatting to make the structure clearer; you won't see this when you run the code yourself.

The nested list representation of an empty tree is simply the empty list. The nested list representation of a tree with a single item `x` is `[x]`.

Your task is to write a Tree method `to_nested_list`, which returns a nested list representation of a tree, and the function `to_tree`, which takes a nested list and returns the Tree that it represents.

You may access all Tree attributes in `to_nested_list`, but *not* in `to_tree`. You may not use any Tree methods here other than the initializer, `is_empty`, and any helpers you define yourself.

Task 3: Binary trees

In this task, you'll preview a particularly useful application of trees for storing data called Binary Search Trees. We'll return to this in great detail next week!

A **binary tree** is a tree where every node has *at most two children*—or put another way, a tree that has at most two subtrees, and these two subtrees each have at most two subtrees, etc.

This structural restriction means that rather than having a list of subtrees, the `BinaryTree` class found in the starter code has two attributes `_left` and `_right`, representing the left and right subtrees.

These always have `BinaryTree` values (including possibly empty `BinaryTree`s), unless the tree is empty, in which case all three attributes are `None`.

Binary trees have three common traversals: *preorder*, *inorder*, and *postorder*. Your task is to complete the methods `preorder`, `inorder`, and `postorder`—note that each of these methods *returns a list*, but doesn't print anything.

Hint: this is a pretty straightforward exercise as long as you've been following along and are thinking recursively! The descriptions themselves describe precisely how your recursive functions should behave; the trickiest part is dealing with empty trees. In this case, all three traversals should return an empty list.

Preorder traversal

Return the root, the preorder traversal of the left subtree, and the preorder traversal of the right subtree, in that order. (The root appears *before* the two subtree traversals.)

Inorder traversal

Return the inorder traversal of the left subtree, then the root, then the inorder traversal of the right subtree, in that order. (The root appears *between* the two subtree traversals.)

Postorder traversal

Return the postorder traversal of the left subtree, then the postorder traversal of the right subtree, then the root, in that order. (The root appears *after* the two subtree traversals.)

For course-related questions, please contact **csc14817f@cs.toronto.edu**.