Recursion and Trees

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`.

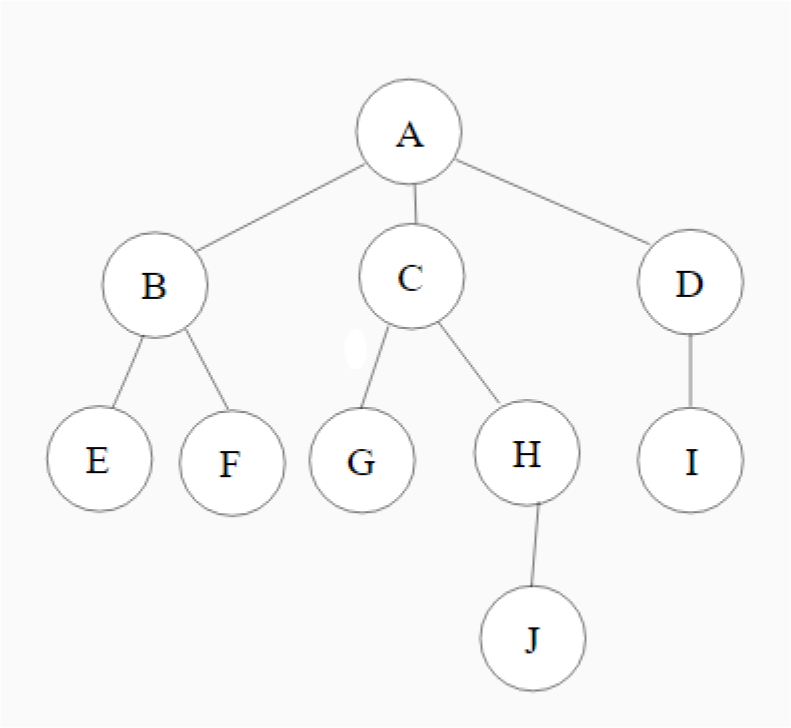
A corner case: if `tree1` is empty, then `tree1 == tree2` is `True` if and only if `tree2` is also empty. Don't forget about this!

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