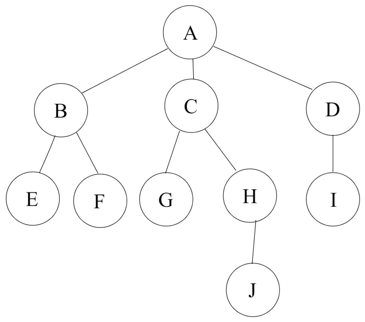
CSC148 - Trees and Nested Lists

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 sub-nested-list is the root of the tree, and each other sub-nested-list is the nested list representation of one of the tree's subtrees. 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]`.

For example, consider the following tree:



Its nested list representation is

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

In this representation, we require that the *first sub-nested-list is not a list*, but instead a simple type like an integer or string. This is to help us visually distinguish the root value of the tree from the tree's sublists. It also means that while every tree has a nested list representation, not every nested list is a valid representation of a tree.

1. Follow the recursive design recipe to write the method `Tree.to_nested_list`, which returns the nested list representation of a given tree.

   Reminder: for the recursive step, before writing any code, it'll be useful to draw a tree with around three subtrees, and then write the nested list representation of each subtree.

2. Next, follow the recursive design recipe to implement the *top-level function* `to_tree`, which takes a nested list and returns the tree that it represents, or `None` if the given nested list is not a valid representation of a tree.

   Note that the only `Tree` method you should need to use here is the initializer; since `to_tree` is a top-level function, you should not access the private instance attributes of `Tree` objects.