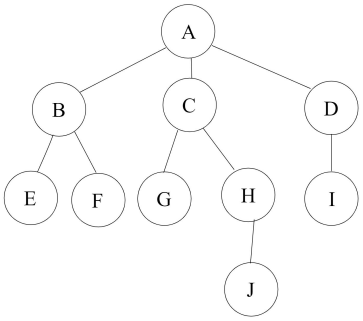
CSC148 - Trees and Nested Lists

We can represent any tree as a nested list, where the first sub-nested-list is the root of the tree, and every 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]`.

1. Here's a bigger tree:



   Its nested list representation is:

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

   Match the opening and closing brackets, and then make sure you see the correspondence between this nested list and the tree above.

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

   Reminder: for the recursive step, before writing any code, it'll be useful to draw a tree with around three subtrees, and for each, to write the expected return value when we recurse on that subtree.

3. In our nested list representation of a tree, we require that the *first sub-nested-list is not a list*, but instead a simple type like an integer or string — something that can be the root value for a tree. This means that while every tree has a nested list representation, not every nested list is a valid representation of a tree. Create a small nested list that is not a valid representation of a tree. Try to make a tree out of your nested list, and observe the problem with doing so.

4. 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. And since `to_tree` is a top-level function, you should not access the private instance attributes of `Tree` objects.