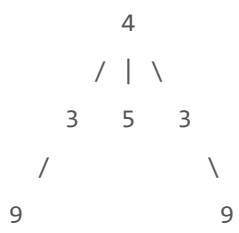
Daily Coding Problem #237

Problem

This problem was asked by Amazon.

A tree is symmetric if its data and shape remain unchanged when it is reflected about the root node. The following tree is an example:



Given a k-ary tree, determine whether it is symmetric.

Solution

When solving problems with k-ary trees, it is often helpful to consider the simpler case of a binary tree first. Let's analyze the example below.

```
      4
    /   \
  3      3
 / \    / \
9   4  5   9
```

Here are the checks we would perform:

- `root == root`
- `root.left == root.right`
- `root.left.left == root.right.right`
- `root.left.right == root.right.left`

For the last comparison, since `4 != 5`, we would return `False`.

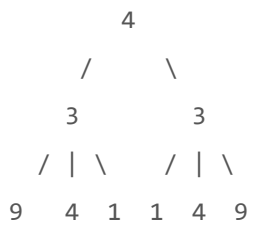
We can turn this into a recursive solution. For each `Node` we traverse, starting with the root, we check that the values of its left and right child nodes are equal, and that the grandchild nodes form mirror images.

```
class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_symmetric(left, right):
    if not left and not right:
        return True
    elif not left or not right:
        return False
    return left.val == right.val and \
        is_symmetric(left.left, right.right) and is_symmetric(left.right,
right.left)

assert is_symmetric(root, root)
```

For a k-ary tree, we will need to compare up to k children, but a similar principle applies. Suppose the number of children for two nodes we are comparing is k. Then we can loop through the list of children for each node, comparing `left[0]` to `right[k - 1]`, `left[1]` to `right[k - 2]`, and so on.



For the example above, each of the root's child nodes have three children. Comparing them in the way described above, we find that `9 == 9`, `4 == 4`, and `1 == 1`, so this tree is indeed symmetric.

A recursive implementaton of the above is as follows:

```
class Node:
    def __init__(self, val, children=[]):
        self.val = val
        self.children = children

def is_symmetric(left, right):
    if left.val != right.val:
        return False

    if not left.children and not right.children:
        return True

    if len(left.children) != len(right.children):
        return False

    k = len(left.children)
    for i in range(k):
        if not is_symmetric(left.children[i], right.children[k - 1 - i]):
            return False
```

```
    return True

assert is_symmetric(root, root)
```

The complexity of this algorithm is `O(N)` for both binary and k-ary trees, since in either case we only examine each node once.
