Module 2 Application Exercises

2.5.32) Find LCA of a tree

For this problem we'll need to be able to identify when a parent node contains both target nodes and in addition be able to identify the lowest parent node that contains those (since we are looking for LCA). If we can store parent nodes that fit this criteria in an array while using BFS we should be able to just return the last node stored in the array which should theoretically be the node that is the LCA of the two target nodes.

Using python syntax for this since it's easier to write out. Intuition should be the same across languages though. Assuming we are given a root and the two target values (t1 and t2 parameters in this function definition). Tested on several tree variations and all seems okay.

```
#start code

# python syntax
def find_LCA(t1: str, t2:str, root: Node):
    ans = [] # array to store nodes that have target nodes as children

    #helper function to find target nodes
    def find_node(target_val: str, node: Node):
        if node.val == target_val:
            return True

        if node.children:
            for child in node.children:
                if find_node(target_val, child):
                    return True
        return False

    # will use a queue for BFS to cover all nodes starting with the root
    queue = [root]

    while queue:
        curr_node = queue.pop(0)
        if curr_node.children:
            for child_node in curr_node.children:
                queue.append(child_node)
        # need to find both target nodes to be valid
        found_target_1 = find_node(t1, curr_node)
        found_target_2 = find_node(t2, curr_node)
        if found_target_1 and found_target_2:
            ans.append(curr_node.val) # storing the node vals instead of node
```

```
            #return last node of arr or None if empty
            return ans[-1] if ans else None
```

Run time:

O(n * m)

every node has its children visited twice while checking for targets. n will be the tree nodes and m is the children for each node n. Most likely not the most efficient.