Question Two: Path to Leaves

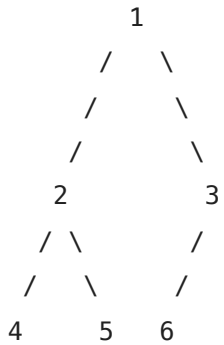
Given the root of a binary tree, return all root to leaf paths in any order.

Section 1: Paraphrase the problem in your own words

In this problem, we are given a binary tree and are required to create a function that returns us all of the different paths you can take down the tree. A binary tree is a non-linear data structrue that has a maximum of 2 children nodes for each parent node. The important nomenclautre to understand for this problem are root and leaf. The root is the topmost node in a binary tree and a leaf is the node at the bottom of the tree. Leaf nodes have no children. Therefore, a binary tree can be visualized as a hierarchial structure with the root at the top and the leaves at the bottom. In this problem we are trying find all of the different paths that lead from the topmost root to a leaf. It's important to note that movement on a tree is unidirectional, so when creating a path you cannot move back up the tree.

Section 2: Create 2 new examples that demonstrate you understand the problem.

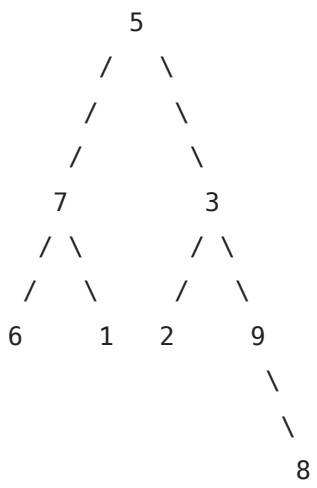
Example 1:



Input: `root = [1, 2, 4, 5, 3, 6]`
Output: [[1,2,4], [1,2,5], [1,3,6]]

In order to generate this tree, you must use preorder traversal

Example 2:



Input: `root = [5, 7, 6, 1, 3, 2, 9, 8]`
Output: [[5,7,6], [5,7,1], [5,3,2], [5,3,9,8]]

In order to generate this tree, you must use preorder traversal

Section 3: Code the solution to your assigned problem in Python (code chunk).

```
from collections import deque


#Definition for a binary tree node.
class TreeNode(object):
  def __init__(self, val = 0, left = None, right = None):
    self.val = val
    self.left = left
    self.right = right

# create a global variable to store a list of all paths
binaryTreePaths = []

# The main function that will print paths from the root node to every leaf node
def binary_tree_path(root):
    # list to store root-to-leaf path
    path = deque()
    binary_tree_path_recur(root, path)
    print(binaryTreePaths)

# Function to check if a given node is a leaf node or not
def isLeaf(TreeNode):
    return TreeNode.left is None and TreeNode.right is None

# The recursive helper function to find paths from the root node to every leaf node
def binary_tree_path_recur(node, path):
    # base case
    if node is None:
        return
    # include the current node to the path
    path.append(node.val)
    # if a leaf node is found, store the path as a list in a global variable
    if isLeaf(node):
        binaryTreePaths.append(list(path))
    # recursively use function for the left and right subtree
    binary_tree_path_recur(node.left, path)
    binary_tree_path_recur(node.right, path)
    # backtrack: remove the current node after the left, and right subtree are done
    path.pop()

# example tree to test the function
root = TreeNode(6)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

# print all root-to-leaf paths
binary_tree_path(root)
```

Section 4: Explain why your solution works

My solution works by taking advantage of recursive functions. The idea is to traverse the tree in a preorder fashion - we visit the left leaf before the right leaf. The function starts from the root and stores every encountered node. The helper function is called recursively until we reach a leaf node, when the function saves the completed path to a global list variable. This solution also takes advantage of queues. The current path is generated using a stack, and the current node is popped off once the left and right subtree have been visited.

Section 5: Explain the problem's time and space complexity

Time Complexity: The time complexity of my solution is O(n) where n is the number of nodes in the binary tree. We traverse each node once, and thus the time complexity is O(n).
Space Complexity: The space complexity of my solution is O(h) where h is the height of the tree. This solution uses a recursion stack to store the path information and therefore the space complexity is O(h).

Section 6: Explain the thinking to an alternative solution

An alternative implementation of this solution could involve using an interative approach instead of recursion. We could store the path from the root-to-leaf in a string as we traverse the tree iteratively using a while loop. We could still take advantage of stacks in order to save the path, and once again print the path whenever we encounter any leaf node.