Path With Given Sequence (medium)

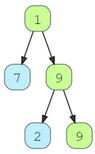
We'll cover the following

Problem Statement

Given a binary tree and a number sequence, find if the sequence is present as a root-to-leaf path in the given tree.

Example 1:

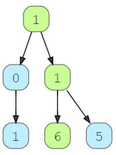
Sequence: [1, 9, 9]
Output: true
Explanation: The tree has a path 1 -> 9 -> 9.



Example 2:

Sequence: [1, 0, 7]
Output: false
Explanation: The tree does not have a path 1 -> 0 -> 7.

Sequence: [1, 1, 6]
Output: true
Explanation: The tree has a path 1 -> 1 -> 6.



Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

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


def find_path(root, sequence):
  # TODO: Write your code here
  return False


def main():

  root = TreeNode(1)
  root.left = TreeNode(0)
  root.right = TreeNode(1)
  root.left.left = TreeNode(1)
  root.right.left = TreeNode(6)
  root.right.right = TreeNode(5)

  print("Tree has path sequence: " + str(find_path(root, [1, 0, 7])))
  print("Tree has path sequence: " + str(find_path(root, [1, 1, 6])))


main()

```

Run | Save | Reset

Solution

This problem follows the Binary Tree Path Sum pattern. We can follow the same **DFS** approach and additionally, track the element of the given sequence that we should match with the current node. Also, we can return `false` as soon as we find a mismatch between the sequence and the node value.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

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


def find_path(root, sequence):
  if not root:
    return len(sequence) == 0

  return find_path_recursive(root, sequence, 0)


def find_path_recursive(currentNode, sequence, sequenceIndex):

  if currentNode is None:
    return False

  seqLen = len(sequence)
  if sequenceIndex >= seqLen or currentNode.val != sequence[sequenceIndex]:
    return False

  # if the current node is a leaf, add it is the end of the sequence, we have found a path!
  if currentNode.left is None and currentNode.right is None and sequenceIndex == seqLen - 1:
    return True

  # recursively call to traverse the left and right sub-tree
  # return true if any of the two recursive call return true
  return find_path_recursive(currentNode.left, sequence, sequenceIndex + 1) or \
      find_path_recursive(currentNode.right, sequence, sequenceIndex + 1)
```

Run | Save | Reset

Time complexity

The time complexity of the above algorithm is $O(N)$, where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

Space complexity

The space complexity of the above algorithm will be $O(N)$ in the worst case. This space will be used to store the recursion stack. The worst case will happen when the given tree is a linked list (i.e., every node has only one child).