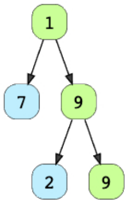
Path With Given Sequence (medium)

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:

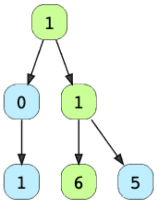
Sequence: [1, 9, 9]
Output: true
Explanation: The tree has a path 1 -> 9 -> 9.



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++

```java
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class PathWithGivenSequence {
  public static boolean findPath(TreeNode root, int[] sequence) {
    // TODO: Write your code here
    return false;
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(0);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(1);
    root.right.left = new TreeNode(6);
    root.right.right = new TreeNode(5);

    System.out.println("Tree has path sequence: " + PathWithGivenSequence.findPath(root, new int[] { 1, 0
    System.out.println("Tree has path sequence: " + PathWithGivenSequence.findPath(root, new int[] { 1, 1
```

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.

MW
Explore
Tracks
My Courses
Edpresso
Refer a Friend
Create

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
    if (currentNode == null)
      return false;

    if (sequenceIndex >= sequence.length || 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 == null && currentNode.right == null && sequenceIndex == sequence.length - 1)
      return true;

    // recursively call to traverse the left and right sub-tree
    // return true if any of the two recusrive call return true
    return findPathRecursive(currentNode.left, sequence, sequenceIndex + 1)
        || findPathRecursive(currentNode.right, sequence, sequenceIndex + 1);
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(0);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(1);
    root.right.left = new TreeNode(6);
    root.right.right = new TreeNode(5);

    System.out.println("Tree has path sequence: " + PathWithGivenSequence.findPath(root, new int[] { 1, 0
    System.out.println("Tree has path sequence: " + PathWithGivenSequence.findPath(root, new int[] { 1, 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).

☑ MARK AS COMPLETED

← Back
Sum of Path Numbers (medium)

Next →
Count Paths for a Sum (medium)

Report an Issue   Ask a Question