Sum of Path Numbers (medium)

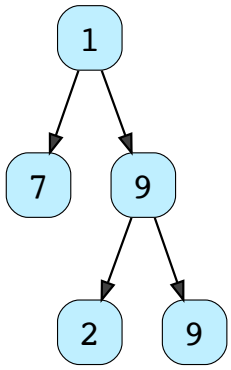
We'll cover the following

Problem Statement#

Given a binary tree where each node can only have a digit (0-9) value, each root-to-leaf path will represent a number. Find the total sum of all the numbers represented by all paths.

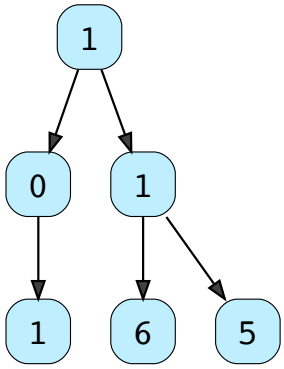
Example 1:

Output: 408
Explanation: The sum of all path numbers: 17 + 192 + 199



Example 2:

Output: 332
Explanation: The sum of all path numbers: 101 + 116 + 115



Try it yourself#

Try solving this question here:

Java | Python3 | JS | C++

```
import java.util.*;

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

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

class SumOfPathNumbers {
  public static int findSumOfPathNumbers(TreeNode root) {
    // TODO: Write your code here
    return -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("Total Sum of Path Numbers: " + SumOfPathNumbers.findSumO
  }
}
```

Run Save Reset

Solution#

This problem follows the Binary Tree Path Sum (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5642684278505472/) pattern. We can follow the same **DFS** approach. The additional thing we need to do is to keep track of the number representing the current path.

How do we calculate the path number for a node? Taking the first example mentioned above, say we are at node '7'. As we know, the path number for this node is '17', which was calculated by: `1 * 10 + 7 => 17`. We will follow the same approach to calculate the path number of each node.

Code#

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

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

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

class SumOfPathNumbers {
  public static int findSumOfPathNumbers(TreeNode root) {
    return findRootToLeafPathNumbers(root, 0);
  }

  private static int findRootToLeafPathNumbers(TreeNode currentNode, int pathSum
    if (currentNode == null)
      return 0;

    // calculate the path number of the current node
    pathSum = 10 * pathSum + currentNode.val;

    // if the current node is a leaf, return the current path sum.
    if (currentNode.left == null && currentNode.right == null) {
      return pathSum;
    }

    // traverse the left and the right sub-tree
    return findRootToLeafPathNumbers(currentNode.left, pathSum) +
           findRootToLeafPathNumbers(currentNode.right, pathSum);
  }

  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("Total Sum of Path Numbers: " + SumOfPathNumbers.findSumO
  }
}
```

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).

Back

Next

All Paths for a Sum (medium)

Path With Given Sequence (medium)

Mark as Completed

Report an Issue