# Sum of Path Numbers (medium)

We'll cover the following

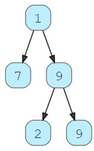


## Problem Statement

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.

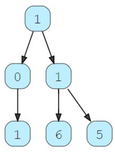
**Example 1:**

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++

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


def find_sum_of_path_numbers(root):
  # TODO: Write your code here
  return -1


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("Total Sum of Path Numbers: " + str(find_sum_of_path_numbers(root)))


main()
```

Run | Save | Reset

## Solution

This problem follows the Binary Tree Path Sum 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

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


def find_sum_of_path_numbers(root):
  return find_root_to_leaf_path_numbers(root, 0)


def find_root_to_leaf_path_numbers(currentNode, pathSum):
  if currentNode is None:
    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 is None and currentNode.right is None:
    return pathSum

  # traverse the left and the right sub-tree
  return find_root_to_leaf_path_numbers(currentNode.left, pathSum) + find_root_to_leaf_path_numbers(currentNode.rig


def main():
  root = TreeNode(1)
  root.left = TreeNode(0)
  root.right = TreeNode(1)
  root.left.left = TreeNode(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).