<https://leetcode.com/problems/sum-root-to-leaf-numbers/>

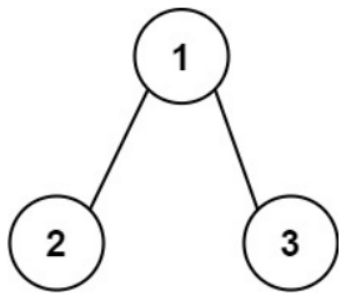
You are given the root of a binary tree containing digits from 0 to 9 only.

Each root-to-leaf path in the tree represents a number.

* For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123.
* Return *the total sum of all root-to-leaf numbers*. Test cases are generated so that the answer will fit in a **32-bit** integer.

A **leaf** node is a node with no children.

**Example 1:**



Input: root = [1,2,3]

Output: 25

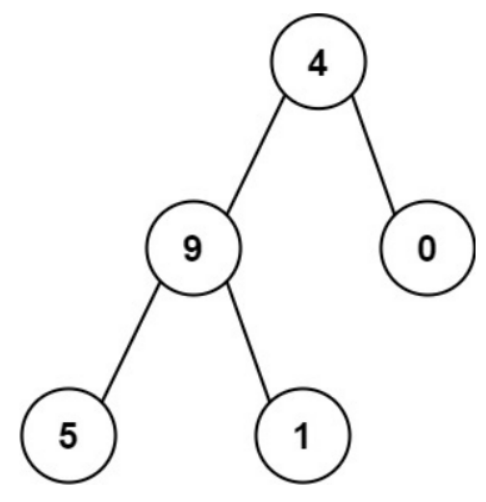
Explanation:

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Therefore, sum = 12 + 13 = 25.

**Example 2:**



Input: root = [4,9,0,5,1]

Output: 1026

Explanation:

The root-to-leaf path 4->9->5 represents the number 495.

The root-to-leaf path 4->9->1 represents the number 491.

The root-to-leaf path 4->0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

**Constraints:**

* The number of nodes in the tree is in the range [1, 1000].
* 0 <= Node.val <= 9
* The depth of the tree will not exceed 10.

**Attempt 1: 2022-12-22**

**Solution 1: DFS (10 min)**

**Style 1: Recursive traversal with O(N) array to store each path sum**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int sumNumbers(TreeNode root) {

List<Integer> list = new ArrayList<Integer>();

helper(root, list, 0);

int result = 0;

for(int i : list) {

result += i;

}

return result;

}

private void helper(TreeNode root, List<Integer> list, int sum) {

if(root == null) {

return;

}

sum = sum \* 10 + root.val;

if(root.left == null && root.right == null) {

list.add(sum);

return;

}

helper(root.left, list, sum);

helper(root.right, list, sum);

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Style 2: Recursive traversal with O(1) global variable to store final sum**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

int allPathsSum = 0;

public int sumNumbers(TreeNode root) {

helper(root, 0);

return allPathsSum;

}

private void helper(TreeNode root, int currPathSum) {

if(root == null) {

return;

}

currPathSum = currPathSum \* 10 + root.val;

if(root.left == null && root.right == null) {

allPathsSum += currPathSum;

return;

}

helper(root.left, currPathSum);

helper(root.right, currPathSum);

}

}

Time Complexity : O(N)

Space Complexity : O(N), the O(N) is not because of global variable O(1), its because of recursion stack take O(N)

**Style 3: Divide and Conquer**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int sumNumbers(TreeNode root) {

return helper(root, 0);

}

private int helper(TreeNode root, int currPathSum) {

if(root == null) {

return 0;

}

currPathSum = currPathSum \* 10 + root.val;

// Base case

if(root.left == null && root.right == null) {

return currPathSum;

}

// Divide

int left = helper(root.left, currPathSum);

int right = helper(root.right, currPathSum);

// Conquer

return left + right;

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Solution 2: BFS (30 min)**

**Style 1: Modify node's original value (NOT suggest)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int sumNumbers(TreeNode root) {

int sum = 0;

Queue<TreeNode> q = new LinkedList<TreeNode>();

q.offer(root);

while(!q.isEmpty()) {

TreeNode node = q.poll();

if(node.left == null && node.right == null) {

sum += node.val;

}

if(node.left != null) {

node.left.val += node.val \* 10;

q.offer(node.left);

}

if(node.right != null) {

node.right.val += node.val \* 10;

q.offer(node.right);

}

}

return sum;

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Style 2: No modify on node's original value but initial a new Node{TreeNode, int}(Suggest)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int sumNumbers(TreeNode root) {

int sum = 0;

Queue<Node> q = new LinkedList<Node>();

q.offer(new Node(root, 0));

while(!q.isEmpty()) {

Node node = q.poll();

TreeNode treeNode = node.treeNode;

int currSum = node.currSum;

currSum = currSum \* 10 + treeNode.val;

if(treeNode.left == null && treeNode.right == null) {

sum += currSum;

}

if(treeNode.left != null) {

q.offer(new Node(treeNode.left, currSum));

}

if(treeNode.right != null) {

q.offer(new Node(treeNode.right, currSum));

}

}

return sum;

}

}

class Node {

TreeNode treeNode;

int currSum;

public Node(TreeNode treeNode, int currSum) {

this.treeNode = treeNode;

this.currSum = currSum;

}

}

Time Complexity : O(N)

Space Complexity : O(N)