<https://leetcode.com/problems/sum-of-left-leaves/description/>

Given the root of a binary tree, return **the sum of all left leaves.**

A **leaf** is a node with no children. A **left leaf** is a leaf that is the left child of another node.

**Example 1:**



**Input:** root = [3,9,20,null,null,15,7]

**Output:** 24

**Explanation:** There are two left leaves in the binary tree, with values 9 and 15 respectively.

**Example 2:**

**Input:** root = [1]

**Output:** 0

**Constraints:**

The number of nodes in the tree is in the range [1,1000].

-1000 <= Node.val <= 1000

**Attempt 1: 2024-08-06**

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

/\*\*

 \* 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 sumOfLeftLeaves(TreeNode root) {

        return helper(root, false);

    }

    private int helper(TreeNode root, boolean isLeft) {

        if(root == null) {

            return 0;

        }

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

            return isLeft ? root.val : 0;

        }

        int leftSum = helper(root.left, true);

        int rightSum = helper(root.right, false);

        return leftSum + rightSum;

    }

}

**Solution 2: BFS (10 min)**

/\*\*

 \* 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 sumOfLeftLeaves(TreeNode root) {

        int result = 0;

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

        q.offer(root);

        while(!q.isEmpty()) {

            TreeNode node = q.poll();

            if(node.left != null) {

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

                    result += node.left.val;

                }

                q.offer(node.left);

            }

            if(node.right != null) {

                q.offer(node.right);

            }

        }

        return result;

    }

}

**Refer to chatGPT**

**DFS solution:**

// Definition for a binary tree node.

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;

}

}

public class Solution {

public int sumOfLeftLeaves(TreeNode root) {

return sumOfLeftLeavesHelper(root, false);

}

private int sumOfLeftLeavesHelper(TreeNode node, boolean isLeft) {

if (node == null) {

return 0;

}

// Check if it is a leaf node

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

return isLeft ? node.val : 0;

}

// Recursively sum the left and right subtrees

int leftSum = sumOfLeftLeavesHelper(node.left, true);

int rightSum = sumOfLeftLeavesHelper(node.right, false);

return leftSum + rightSum;

}

}

**sumOfLeftLeavesHelper Method:**

Takes two parameters: the current TreeNode and a boolean isLeft indicating if the current node is a left child.

If the node is null, it returns 0.

If the node is a leaf node (both left and right children are null), it returns the node's value if isLeft is true, otherwise returns 0.

Recursively calculates the sum of left leaves in the left and right subtrees.

Sums up the values from the left and right subtrees and returns the result.

**BFS solution**

import java.util.LinkedList;

import java.util.Queue;

// Definition for a binary tree node.

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;

}

}

public class Solution {

public int sumOfLeftLeaves(TreeNode root) {

if (root == null) {

return 0;

}

int sum = 0;

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

queue.offer(root);

while (!queue.isEmpty()) {

TreeNode node = queue.poll();

// Check the left child

if (node.left != null) {

// If the left child is a leaf

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

sum += node.left.val;

} else {

// Otherwise, add the left child to the queue for further processing

queue.offer(node.left);

}

}

// Add the right child to the queue for further processing

if (node.right != null) {

queue.offer(node.right);

}

}

return sum;

}

}

**Refer to**

<https://leetcode.com/problems/sum-of-left-leaves/solutions/88950/java-iterative-and-recursive-solutions/>

**Recursive method.**

For given node we check whether its left child is a leaf. If it is the case, we add its value to answer, otherwise recursively call method on left child. For right child we call method only if it has at least one nonnull child.

public int sumOfLeftLeaves(TreeNode root) {

if(root == null) return 0;

int ans = 0;

if(root.left != null) {

if(root.left.left == null && root.left.right == null) ans += root.left.val;

else ans += sumOfLeftLeaves(root.left);

}

ans += sumOfLeftLeaves(root.right);

return ans;

}

**Iterative method.**

Here for each node in the tree we check whether its left child is a leaf. If it is true, we add its value to answer, otherwise add left child to the stack to process it later. For right child we add it to stack only if it is not a leaf.

public int sumOfLeftLeaves(TreeNode root) {

if(root == null) return 0;

int ans = 0;

Stack<TreeNode> stack = new Stack<TreeNode>();

stack.push(root);

while(!stack.empty()) {

TreeNode node = stack.pop();

if(node.left != null) {

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

ans += node.left.val;

stack.push(node.left);

}

if(node.right != null) {

stack.push(node.right);

}

}

return ans;

}