

GeeksforGeeks

A computer science portal for geeks

Placements Practice GATE CS IDE Q&A
GeeksQuiz

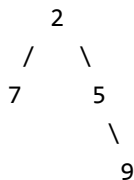
Find multiplication of sums of data of leaves at same levels

Given a Binary Tree, return following value for it.

- 1) For every level, compute sum of all leaves if there are leaves at this level. Otherwise ignore it.
- 2) Return multiplication of all sums.

Examples:

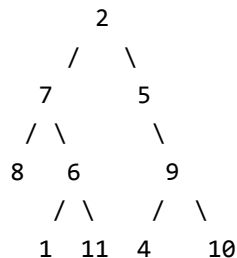
Input: Root of below tree



Output: 63

First level doesn't have leaves. Second level has one leaf 7 and third level also has one leaf 9. Therefore result is $7 \times 9 = 63$

Input: Root of below tree



Output: 208

First two levels don't have leaves. Third level has single leaf 8. Last level has four leaves 1, 11, 4 and 10. Therefore result is $8 \times (1 + 11 + 4 + 10)$

We strongly recommend you to minimize your browser and try this yourself first.

One **Simple Solution** is to recursively compute leaf sum for all level starting from top to bottom. Then multiply sums of levels which have leaves. Time complexity of this solution would be $O(n^2)$.

An **Efficient Solution** is to use Queue based level order traversal. While doing the traversal, process all different levels separately. For every processed level, check if it has a leaves. If it has then compute sum of leaf nodes. Finally return product of all sums.

C++

```
/* Iterative C++ program to find sum of data of all leaves
   of a binary tree on same level and then multiply sums
   obtained of all levels. */
#include <bits/stdc++.h>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// helper function to check if a Node is leaf of tree
bool isLeaf(Node *root)
{
    return (!root->left && !root->right);
}

/* Calculate sum of all leaf Nodes at each level and returns
   multiplication of sums */
int sumAndMultiplyLevelData(Node *root)
{
    // Tree is empty
    if (!root)
        return 0;

    int mul = 1;    /* To store result */

    // Create an empty queue for level order traversal
    queue<Node *> q;

    // Enqueue Root and initialize height
    q.push(root);

    // Do level order traversal of tree
    while (1)
    {
```

```
// NodeCount (queue size) indicates number of Nodes
// at current level.
int NodeCount = q.size();

// If there are no Nodes at current level, we are done
if (NodeCount == 0)
    break;

// Initialize leaf sum for current level
int levelSum = 0;

// A boolean variable to indicate if found a leaf
// Node at current level or not
bool leafFound = false;

// Dequeue all Nodes of current level and Enqueue all
// Nodes of next level
while (NodeCount > 0)
{
    // Process next Node of current level
    Node *Node = q.front();

    /* if Node is a leaf, update sum at the level */
    if (isLeaf(Node))
    {
        leafFound = true;
        levelSum += Node->data;
    }
    q.pop();

    // Add children of Node
    if (Node->left != NULL)
        q.push(Node->left);
    if (Node->right != NULL)
        q.push(Node->right);
    NodeCount--;
}

// If we found at least one leaf, we multiply
// result with level sum.
if (leafFound)
    mul *= levelSum;
}

return mul; // Return result
}

// Utility function to create a new tree Node
Node* newNode(int data)
{

```

```

    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
    Node *root = newNode(2);
    root->left = newNode(7);
    root->right = newNode(5);
    root->left->right = newNode(6);
    root->left->left = newNode(8);
    root->left->right->left = newNode(1);
    root->left->right->right = newNode(11);
    root->right->right = newNode(9);
    root->right->right->left = newNode(4);
    root->right->right->right = newNode(10);

    cout << "Final product value = "
         << sumAndMultiplyLevelData(root) << endl;

    return 0;
}

```

Java

```

/* Iterative Java program to find sum of data of all leaves
of a binary tree on same level and then multiply sums
obtained of all levels. */

/* importing the necessary class */
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

/* Class containing left and right child of current
node and key value*/
class Node {

    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

```

```
class BinaryTree {

    Node root;

    // helper function to check if a Node is leaf of tree
    boolean isLeaf(Node node)
    {
        return ((node.left == null) && (node.right == null));
    }

    /* Calculate sum of all leaf Nodes at each level and returns
       multiplication of sums */
    int sumAndMultiplyLevelData()
    {
        return sumAndMultiplyLevelData(root);
    }

    int sumAndMultiplyLevelData(Node node)
    {
        // Tree is empty
        if (node == null) {
            return 0;
        }

        int mul = 1; /* To store result */

        // Create an empty queue for level order traversal
        LinkedList<Node> q = new LinkedList<Node>();

        // Enqueue Root and initialize height
        q.add(node);

        // Do level order traversal of tree
        while (true) {

            // NodeCount (queue size) indicates number of Nodes
            // at current level.
            int NodeCount = q.size();

            // If there are no Nodes at current level, we are done
            if (NodeCount == 0) {
                break;
            }

            // Initialize leaf sum for current level
            int levelSum = 0;

            // A boolean variable to indicate if found a leaf
            // Node at current level or not
            boolean leafFound = false;
```

```

// Dequeue all Nodes of current level and Enqueue all
// Nodes of next level
while (NodeCount > 0) {
    Node node1;
    node1 = q.poll();

    /* if Node is a leaf, update sum at the level */
    if (isLeaf(node1)) {
        leafFound = true;
        levelSum += node1.data;
    }

    // Add children of Node
    if (node1.left != null) {
        q.add(node1.left);
    }
    if (node1.right != null) {
        q.add(node1.right);
    }
    NodeCount--;
}

// If we found at least one leaf, we multiply
// result with level sum.
if (leafFound) {
    mul *= levelSum;
}
}

return mul; // Return result
}

public static void main(String args[]) {

    /* creating a binary tree and entering
    the nodes */
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(2);
    tree.root.left = new Node(7);
    tree.root.right = new Node(5);
    tree.root.left.left = new Node(8);
    tree.root.left.right = new Node(6);
    tree.root.left.right.left = new Node(1);
    tree.root.left.right.right = new Node(11);
    tree.root.right.right = new Node(9);
    tree.root.right.right.left = new Node(4);
    tree.root.right.right.right = new Node(10);
    System.out.println("The final product value : ")

```

```
        + tree.sumAndMultiplyLevelData());  
    }  
}  
  
// This code is contributed by Mayank Jaiswal
```

Output:

Final product value = 208

This article is contributed by [Mohammed Raqeeb](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



 Gmail for Work

[12 Comments](#) Category: [Trees](#) Tags: [Microsoft](#)

Related Posts:

- [Check sum of Covered and Uncovered nodes of Binary Tree](#)
- [Lowest Common Ancestor in a Binary Tree | Set 2 \(Using Parent Pointer\)](#)
- [Construct a Binary Search Tree from given postorder](#)
- [BFS vs DFS for Binary Tree](#)
- [Maximum difference between node and its ancestor in Binary Tree](#)
- [Inorder Non-threaded Binary Tree Traversal without Recursion or Stack](#)
- [Check if leaf traversal of two Binary Trees is same?](#)
- [Closest leaf to a given node in Binary Tree](#)

([Login](#) to Rate and Mark)

2.4 Average Difficulty : **2.4/5.0**
Based on **17** vote(s)

☐

Add to TODO List

☐

Mark as DONE

[Like](#) [Share](#) 14 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

@geeksforgeeks, [Some rights reserved](#)

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)