545. Boundary of Binary Tree

March 27, 2017 | 33.6K views

Average Rating: 4.58 (24 votes)

Given a binary tree, return the values of its boundary in **anti-clockwise** direction starting from root. Boundary includes left boundary, leaves, and right boundary in order without duplicate **nodes**. (The values of the nodes may still be duplicates.)

Left boundary is defined as the path from root to the **left-most** node. **Right boundary** is defined as the path from root to the **right-most** node. If the root doesn't have left subtree or right subtree, then the root itself is left boundary or right boundary. Note this definition only applies to the input binary tree, and not applies to any subtrees.

The **left-most** node is defined as a **leaf** node you could reach when you always firstly travel to the left subtree if exists. If not, travel to the right subtree. Repeat until you reach a leaf node.

The **right-most** node is also defined by the same way with left and right exchanged.

Example 1

```
Input:
  1
   \
    2
   / \
  3   4

Ouput:
[1, 3, 4, 2]

Explanation:
The root doesn't have left subtree, so the root itself is left boundary.
The leaves are node 3 and 4.
The right boundary are node 1,2,4. Note the anti-clockwise direction means you should
So order them in anti-clockwise without duplicates and we have [1,3,4,2].
```

Example 2

```
Input:
    ____1_____
   /          \
  2            3
 / \          /
4   5        6
   / \      / \
  7   8    9  10

Ouput:
[1,2,4,7,8,9,10,6,3]

Explanation:
The left boundary are node 1,2,4. (4 is the left-most node according to definition)
The leaves are node 4,7,8,9,10.
The right boundary are node 1,3,6,10. (10 is the right-most node).
So order them in anti-clockwise without duplicate nodes we have [1,2,4,7,8,9,10,6,3].
```

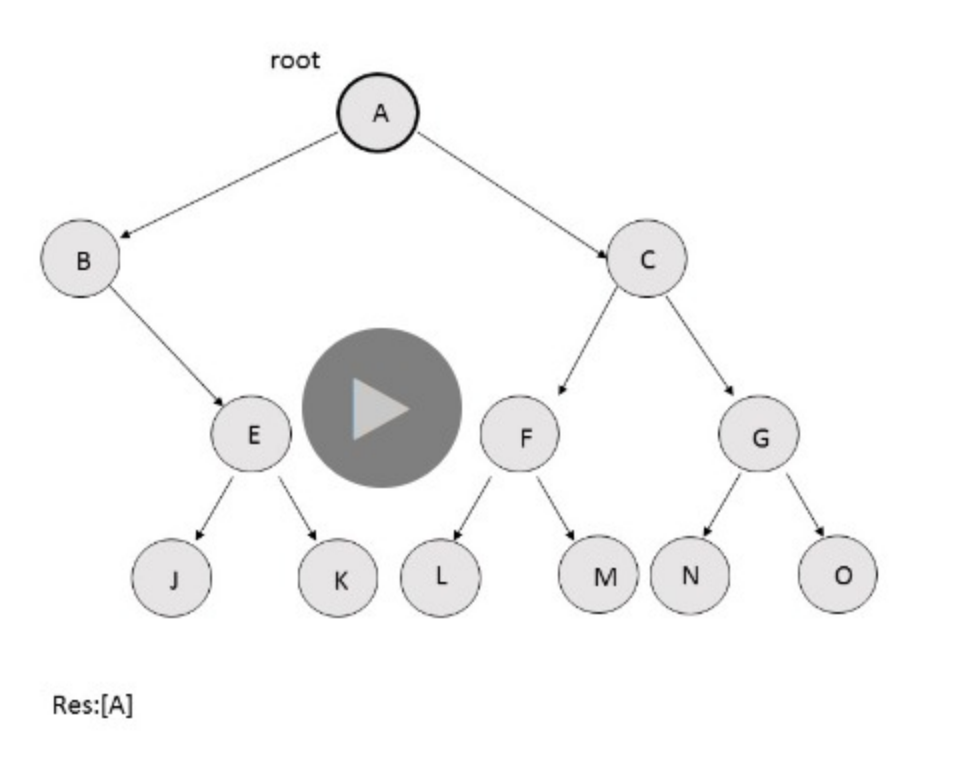
Solution

Approach #1 Simple Solution [Accepted]

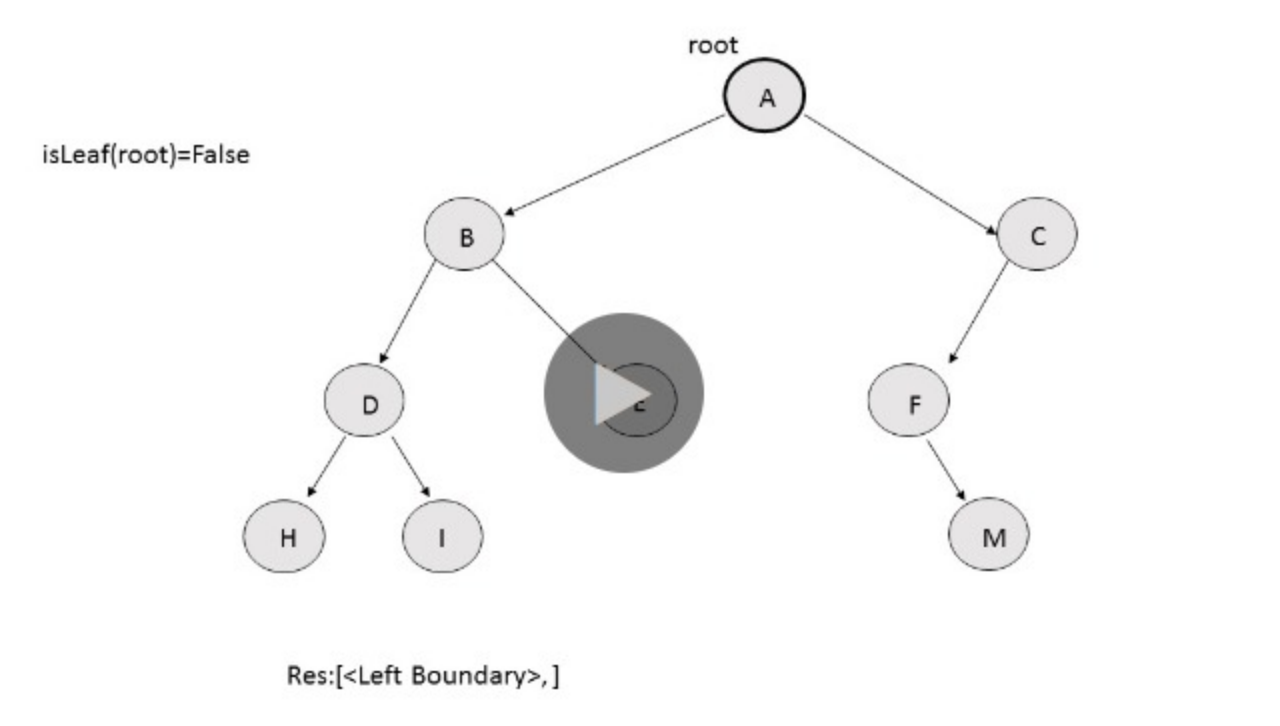
Algorithm

One simple approach is to divide this problem into three subproblems- left boundary, leaves and right boundary.

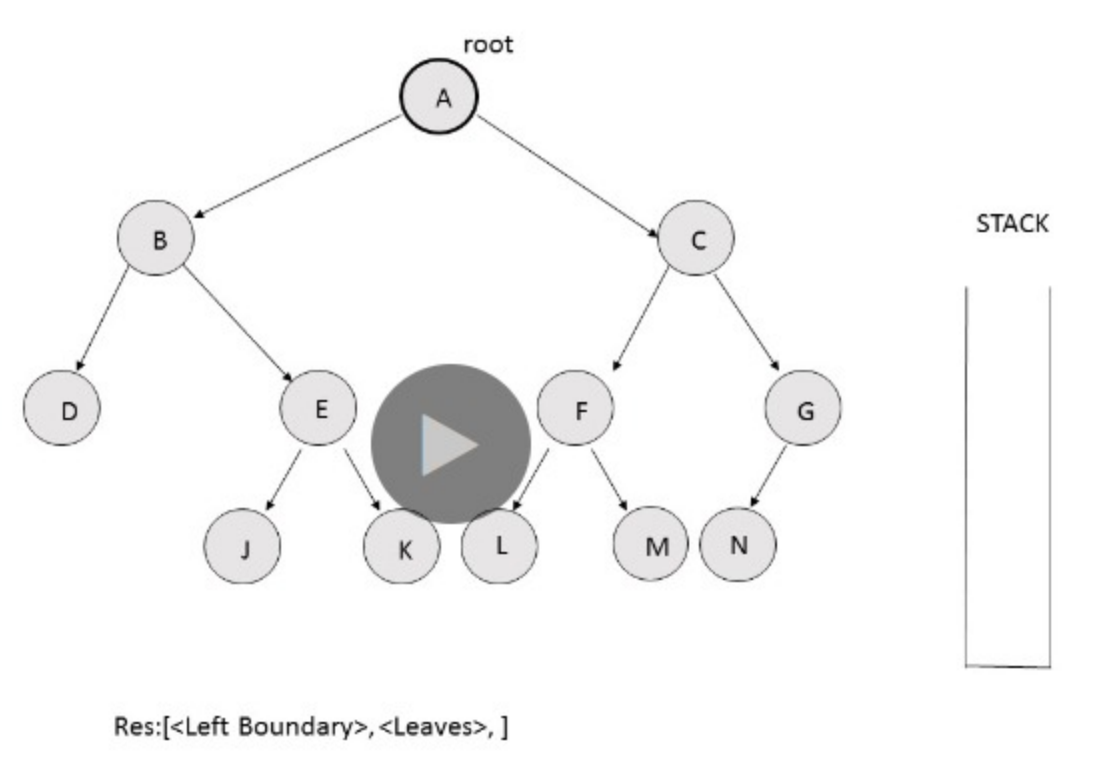
- Left Boundary: We keep on traversing the tree towards the left and keep on adding the nodes in the res array, provided the current node isn't a leaf node. If at any point, we can't find the left child of a node, but its right child exists, we put the right child in the res and continue the process. The following animation depicts the process.



- Leaf Nodes: We make use of a recursive function `addLeaves(res,root)`, in which we change the root node for every recursive call. If the current root node happens to be a leaf node, it is added to the res array. Otherwise, we make the recursive call using the left child of the current node as the new root. After this, we make the recursive call using the right child of the current node as the new root. The following animation depicts the process.



- Right Boundary: We perform the same process as the left boundary. But, this time, we traverse towards the right. If the right child doesn't exist, we move towards the left child. Also, instead of putting the traversed nodes in the res array, we push them over a stack during the traversal. After the complete traversal is done, we pop the element from over the stack and append them to the res array. The following animation depicts the process.



Java

```java

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {

    public boolean isLeaf(TreeNode t) {
        return t.left == null && t.right == null;
    }

    public void addLeaves(List<Integer> res, TreeNode root) {
        if (isLeaf(root)) {
            res.add(root.val);
        } else {
            if (root.left != null) {
                addLeaves(res, root.left);
            }
            if (root.right != null) {
                addLeaves(res, root.right);
            }
        }
```

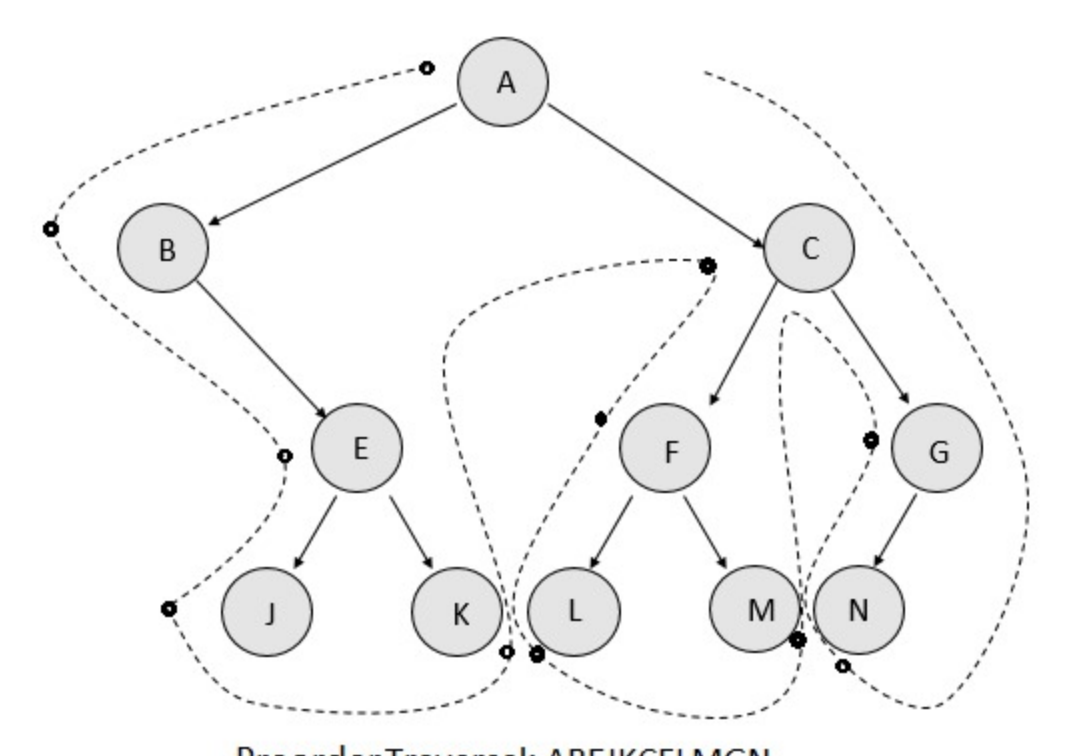
Complexity Analysis

- Time complexity : $O(n)$ One complete traversal for leaves and two traversals upto depth of binary tree for left and right boundary.
- Space complexity : $O(n)$ res and $stack$ is used.

Approach #2 Using PreOrder Traversal [Accepted]

Algorithm

Before we dive into this approach, let's look at the preorder traversal of a simple Binary Tree as shown below:



Preorder Traversal: ABEJKCFLMGN

From the above figure, we can observe that our problem statement is very similar to the Preorder traversal. Actually, the order of traversal is the same(except for the right boundary nodes, for which it is the reverse), but we need to selectively include the nodes in the return result list. Thus, we need to include only those nodes in the result, which are either on the left boundary, the leaves or the right boundary.

In order to distinguish between the various kinds of nodes, we make use of a $flag$ as follows:

- Flag=0: Root Node.
- Flag=1: Left Boundary Node.
- Flag=2: Right Boundary Node.
- Flag=3: Others(Middle Node).

We make use of three lists *[Math Processing Error]*, *[Math Processing Error]*, $leaves$ to store the appropriate nodes and append the three lists at the end.

We go for the normal preorder traversal, but while calling the recursive function for preorder traversal using the left child or the right child of the current node, we also pass the $flag$ information indicating the type of node that the current child behaves like.

For obtaining the flag information about the left child of the current node, we make use of the function `leftChildFlag(node, flag)`. In the case of a left child, the following cases are possible, as can be verified by looking at the figure above:

- The current node is a left boundary node: In this case, the left child will always be a left boundary node. e.g. relationship between E & J in the above figure.
- The current node is a root node: In this case, the left child will always be a left boundary node. e.g. relationship between A & B in the above figure.
- The current node is a right boundary node: In this case, if the right child of the current node doesn't exist, the left child always acts as the right boundary node. e.g. G & N. But, if the right child exists, the left child always acts as the middle node. e.g. C & F.

Similarly, for obtaining the flag information about the right child of the current node, we make use of the function `rightChildFlag(node, flag)`. In the case of a right child, the following cases are possible, as can be verified by looking at the figure above:

- The current node is a right boundary node: In this case, the right child will always be a right boundary node. e.g. relationship between C & G in the above figure.
- The current node is a root node: In this case, the right child will always be a left boundary node. e.g. relationship between A & C in the above figure.
- The current node is a left boundary node: In this case, if the left child of the current node doesn't exist, the right child always acts as the left boundary node. e.g. B & E. But, if the left child exists, the left child always acts as the middle node.

Making use of the above information, we set the $flag$ appropriately, which is used to determine the list in which the current node has to be appended.

Java

```java

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List < Integer > boundaryOfBinaryTree(TreeNode root) {
        List < Integer > left_boundary = new LinkedList < > (), right_boundary = new LinkedList < > (),
leaves = new LinkedList < > ();
        preorder(root, left_boundary, right_boundary, leaves, 0);
        left_boundary.addAll(leaves);
        left_boundary.addAll(right_boundary);
        return left_boundary;
    }

    public boolean isLeaf(TreeNode cur) {
        return (cur.left == null && cur.right == null);
    }

    public boolean isRightBoundary(int flag) {
        return (flag == 2);
    }
```

Complexity Analysis

- Time complexity : $O(n)$ One complete traversal of the tree is done.
- Space complexity : $O(n)$ The recursive stack can grow upto a depth of n. Further, *[Math Processing Error]*, *[Math Processing Error]* and $leaves$ combined together can be of size n.

Analysis written by: @vinod23

Rate this article: ★★★★★

Previous Next

Comments: 23

Sort By

Type comment here... (Markdown is supported)

Preview Post

Zbeeee ★49 December 7, 2019 2:47 AM

This should be a hard problem.

35 Share Reply

SHOW 1 REPLY

paolomaldini ★23 January 19, 2020 6:04 AM

Lol Oracle asked me this in phone interview along with another medium level question to be answered in 45 mins. MAD

23 Share Reply

SHOW 3 REPLIES

theOtherWC ★55 January 17, 2020 9:04 AM

Such a horrendous question and description.

13 Share Reply

gg2529 ★16 March 31, 2017 11:41 PM

The first algorithm will not work for [1, null, 2, 3, 4] as this algorithm will include 2 element in both left traversal as well as right traversal. Or I am missing something?

12 Share Reply

SHOW 3 REPLIES

leetcode_deleted_user ★247 December 27, 2017 2:44 AM

Just making "Approach #2 Using PreOrder Traversal" a bit simple as below (accepted):

```
public class Solution {
    enum Flag {ROOT, LEFT, RIGHT, MIDDLE};
    public List<Integer> boundaryOfBinaryTree(TreeNode root) {
```

Read More

6 Share Reply

dwayne ★5 August 6, 2017 11:09 AM

Small typo in "The current node is a root node: In this case, the right child will always be a left boundary node. e.g. relationship between A & C in the above figure."
It should say "The current node is a root node: In this case, the right child will always be a RIGHT boundary node. e.g. relationship between A & C in the above figure."

4 Share Reply

ganeshdudwadkar ★61 April 22, 2017 2:49 AM

Pls check out my recursive solution. It does this in one iteration.
https://discuss.leetcode.com/topic/87069/java-recursive-solution-beats-94

1 Share Reply

LearningFromLeetcode ★2 August 6, 2018 7:55 PM

from geeksforgeek link : https://www.youtube.com/watch?v=7yFf4nKsamU
Similar idea but simple code
idea :

Read More

0 Share Reply

SHOW 1 REPLY

tyuan73 ★1116 July 14, 2017 9:31 PM

Please explain why the output of this test case:
[1,2,null,3,4,null,null,5,6,null,null,null,null]
is [1,2,3,5,6] not [1,2,3,5,6,4]?

[4] is not on the right boundary?

Read More

0 Share Reply

SHOW 1 REPLY

vinod23 ★425 April 1, 2017 2:56 AM

No, 2 will be included only in the right boundary.

0 Share Reply

‹ 1 2 3 ›