


// <https://leetcode.com/problems/flatten-binary-tree-to-linked-list/> (check last solution with O(1) space using 'Morris Traversal' based approach)

 Explore Problems Mock Contest Discuss Store

Description

Solution

Discuss (999+)

Submissions

C#

114. Flatten Binary Tree to Linked List

Medium 3228 352 Add to List Share

Given a binary tree, flatten it to a linked list in-place.

For example, given the following tree:



```
graph TD
    1((1)) --> 2((2))
    1 --> 5((5))
    2 --> 3((3))
    2 --> 4((4))
    5 --> 6((6))
```


The flattened tree should look like:


```
graph TD
    1((1)) --> 2((2))
    2 --> 3((3))
    3 --> 4((4))
    4 --> 5((5))
    5 --> 6((6))
```

Accepted 371,716 Submissions 743,479

Seen this question in a real interview before?

Companies  

Related Topics 

Similar Questions 

```
1 /**
2  * Definition for a binary tree node.
3  * public class TreeNode {
4  *     public int val;
5  *     public TreeNode left;
6  *     public TreeNode right;
7  *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
8  *         this.val = val;
9  *         this.left = left;
10 *         this.right = right;
11 *     }
12 * }
13 */
14 public class Solution {
15     public void Flatten(TreeNode root) {
16         ConvertTreeToLinkedList(root);
17     }
18     public TreeNode ConvertTreeToLinkedList(TreeNode root)
19     {
20         if(root==null) return null;
21         if(root.left==null && root.right==null) return root;
22
23         TreeNode LeftTail = ConvertTreeToLinkedList(root.left);
24         TreeNode RightTail = ConvertTreeToLinkedList(root.right);
25
26         // base condition
27         if(root.left!=null)
28         {
29             LeftTail.right=root.right;
30             root.right=root.left;
31             root.left=null;
32         }
33
34         return RightTail!=null? RightTail : LeftTail;
35     }
36 }
37
```

Your previous code was restored from your local storage. [Reset to default](#)

/**

* Definition for a binary tree node.

* public class TreeNode {

* public int val;

* public TreeNode left;

* public TreeNode right;

* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {

* this.val = val;

* this.left = left;

* this.right = right;

* }

* }

```

*/
public class Solution {
    public void Flatten(TreeNode root) {
        ConvertTreeToLinkedList(root);
    }
    public TreeNode ConvertTreeToLinkedList(TreeNode root)
    {
        if(root==null) return null;
        if(root.left==null && root.right==null) return root;

        TreeNode LeftTail = ConvertTreeToLinkedList(root.left);
        TreeNode RightTail = ConvertTreeToLinkedList(root.right);

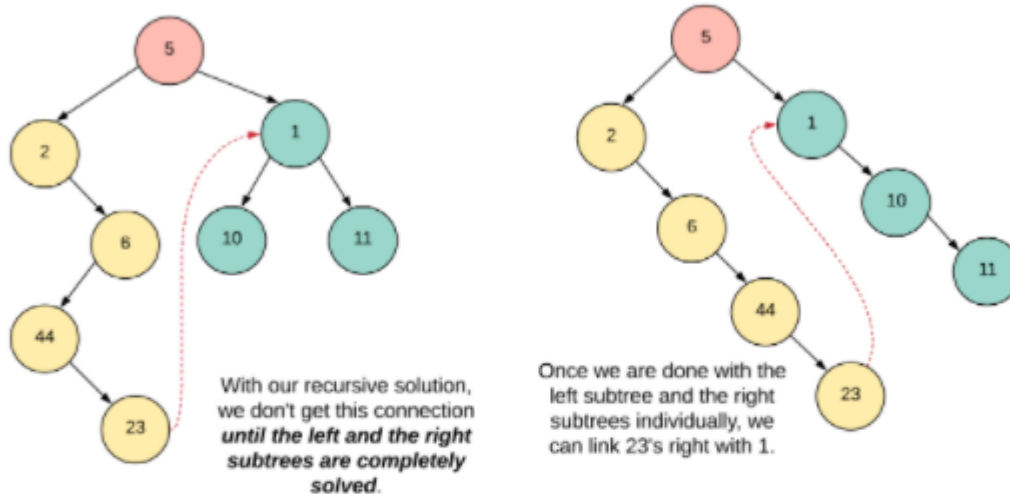
        // base condition
        if(root.left!=null)
        {
            LeftTail.right=root.right;
            root.right=root.left;
            root.left=null;
        }

        return RightTail!=null? RightTail : LeftTail;
    }
}

// Morris Traversal based soln

```

With recursion, we only re-wire the connections for the "current node" once we are already done processing the left and the right subtrees *completely*. Let's see what that looks like in a figure.



However, the **postponing** of rewiring of connections on the current node until the left subtree is done, is basically what recursion is. Recursion is all about postponing decisions until something else is completed. In order for us to be able to postpone stuff, we need to use the stack. However, in our current approach we want to get rid of the stack altogether. So, we will have to come up with a **greedy** way that will be costlier in terms of time, but will be space efficient in achieving the same results.

For a current node, we will check if it has a left child or not. If it does, we will find the last node in the rightmost branch of the subtree rooted at this left child. Once we find this "rightmost" node, we will hook it up with the right child of the current node.

Let's look at this idea on our current sample tree.

Let's say our current node is the root node of the tree. This node does have a left child. So, ***we will find the final node in the rightmost branch***



Once we do find this node, ***we will hook un***

```
class Solution {
```

```
    public void flatten(TreeNode root) {
```

```

// Handle the null scenario
if (root == null) {
    return;
}

TreeNode node = root;

while (node != null) {

    // If the node has a left child
    if (node.left != null) {

        // Find the rightmost node
        TreeNode rightmost = node.left;
        while (rightmost.right != null) {
            rightmost = rightmost.right;
        }

        // rewire the connections
        rightmost.right = node.right;
        node.right = node.left;
        node.left = null;
    }

    // move on to the right side of the tree
    node = node.right;
}
}

```

}