Solution: Flatten Binary Tree to Linked List

Let's solve the Flatten Binary Tree to Linked List problem using the Tree Depth-First Search pattern.

We'll cover the following

Statement

Given the `root` of a binary tree, the task is to flatten the tree into a linked list using the same `TreeNode` class. The left child pointer of each node in the linked list should always be NULL, and the right child pointer should point to the next node in the linked list. The nodes in the linked list should be in the same order as that of the preorder traversal of the given binary tree.

Constraints:

- $-100 \leq$ `Node.data` ≤ 100.
- The tree contains nodes in the range $[1, 500]$.

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

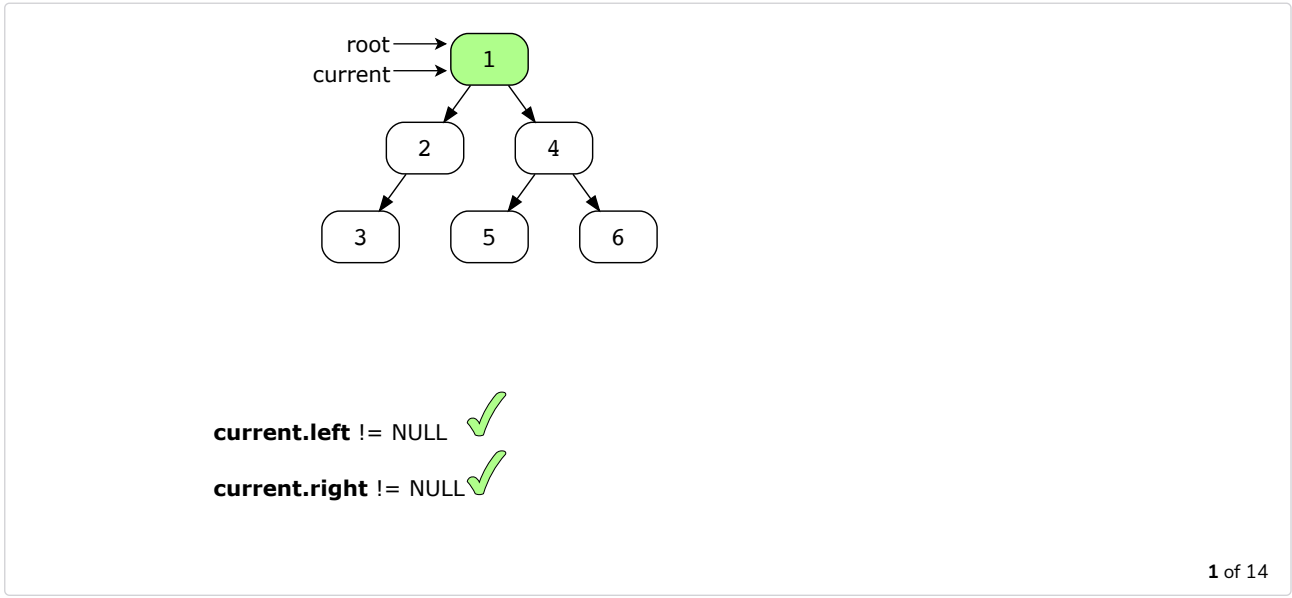
The naive approach to flatten a binary tree into a linked list is to perform a preorder traversal of the tree and store the visited nodes in a Queue. After the traversal, start dequeuing the nodes and set the pointers of each node such that: the right pointer of the dequeued node is set to the previously dequeued node, and the left pointer is set to NULL.

However, this naive approach requires extra memory because it uses a Queue. The space complexity would be $O(n)$. However, can the problem be solved without additional data structures?

Optimized approach using depth-first search

To flatten the binary tree, we will follow a depth-first search approach. We start at the root node and for each node and find the right-most node in its left subtree. We set the right pointer of the right-most node to the current node's right pointer. After that, we set the current node's right pointer to the current node's left

pointer. Finally, we set the current node's left pointer to NULL. We will repeat this process for all nodes in the binary tree.



> **Note:** In the following section, we will gradually build the solution. Alternatively, you can skip straight to just the code.

Step-by-step solution construction

Starting from the tree's root, we traverse the tree in a depth-first search manner. At each node, we check if it has a left child. If it does, we follow a path down to the rightmost node of the left subtree. This can be achieved by repeatedly moving to the right child of each node in the left subtree until we reach a node that does not have a right child.

Java

Main.java

BinaryTree.java

TreeNode.java

```java
public class Main {
    public static TreeNode<Integer> flattenTree(TreeNode<Integer> root) {
        if (root == null) {
            return null;
        }
        //Assign current to root
        TreeNode<Integer> current = root;
        TreeNode<Integer> last = null;
        // Traversing the whole tree
        System.out.println("\n\tTraversing the tree:");
        while (current != null) {
            // printing the tree
            Print.displayTree(root, current);
            if (current.left != null) {
                System.out.println("\n\tThe current node has a left child.");
                last = current.left;
                // printing the tree
                Print.displayTree(root, last);
                // If the last node has a right child
```

```
                while (last.right != null) {
                    System.out.println("\n\tThe current node has a right chil
                    // printing the tree
                    Print.displayTree(root, last.right);
                    last = last.right;
                }

                System.out.println("\n\tThe current node does not have a righ
                System.out.println("\tWe'll move back");
```

Flatten Binary Tree to Linked List

Once we reach the rightmost node, we point the right pointer of this node to the right child of the current node. After making this connection, we point the current node's right pointer to the current node's left child. Finally, we set the current node's left pointer to NULL. We repeat this process until all nodes of the tree have been traversed.

Java

Main.java

BinaryTree.java

TreeNode.java

```
public class Main {
    public static TreeNode<Integer> flattenTree(TreeNode<Integer> root) {
        if (root == null) {
            return null;
        }
        //Assign current to root
        TreeNode<Integer> current = root;
        // Traversing the whole tree
        System.out.println("\n\tTraversing the tree:");
        while (current != null) {
            // printing the tree
            Print.displayTree(root, current);
            if (current.left != null) {
                System.out.println("\n\tThe current node has a left child.");
                TreeNode<Integer> last = current.left;
                // printing the tree
                Print.displayTree(root, last);
                // If the last node has a right child
                while (last.right != null) {
                    System.out.println("\n\tThe current node has a right chil
                    // printing the tree
                    Print.displayTree(root, last.right);
                    last = last.right;
                }

                System.out.println("\n\tThe current node does not have a righ
                System.out.println("\tWe'll merge it with the right subtree."
                last.right = current.right;
```

Flatten Binary Tree to Linked List

Just the code

Here's the complete solution to this problem:

Java

Main.java

BinaryTree.java

TreeNode.java

```java
public class Main {
    public static TreeNode<Integer> flattenTree(TreeNode<Integer> root) {
        if (root == null) {
            return null;
        }

            if (current.left != null) {

                TreeNode<Integer> last = current.left;
                while (last.right != null) {
                    last = last.right;
                }

                last.right = current.right;
                current.right = current.left;
                current.left = null;

            }
            current = current.right;
        }
        return root;
    }

    public static void main(String[] args) {
```

Flatten Binary Tree to Linked List

Solution summary

- Traverse the binary tree, and for each node, check if it has a left child.
- If the left child exists, find the rightmost node in the left subtree.
- Point the right pointer of the rightmost node to the right child of the current node.
- Set the current node's right pointer to the current node's left pointer.
- Set the current node's left child to NULL.
- Repeat the steps above until the entire binary tree has been traversed.

Time complexity

The time complexity is $O(n)$, where n is the number of nodes in the tree.

Space complexity

The space complexity will be $O(1)$ for this problem.

Back

Flatten Binary Tree to ...

Next

Diameter of Binary Tree

Mark as Completed