🔗 LeetCode 114: Flatten Binary Tree to Linked List

🔧 Leetcode 114. Flatten Binary Tree to Linked List — Explanation (Method 1)

✅ Problem Summary

- Convert a binary tree to a "linked list" **in-place**, following **preorder traversal**.
- In the resulting structure:
  - `right` pointer should point to the next node.
  - `left` pointer should always be `NULL`.

🧠 Our Approach (Method 1)

You're solving it using **two main steps**:

1. Store the preorder traversal of nodes in a vector.
2. Rebuild the tree into a right-skewed linked list.

📌 Step 1: Preorder Traversal + Collect Nodes

```
void pre_order(TreeNode* root, vector<TreeNode*>& v) {
    if(root == NULL)return;
    v.push_back(root);
    pre_order(root->left, v);
    pre_order(root->right, v);
    if(root->left) root->left = NULL;
    if(root->right) root->right = NULL;
}
```

- Recursively visits each node in preorder (Root → Left → Right).

- Stores each visited node in a vector `v`.
- Also sets both `left` and `right` pointers to `NULL` to disconnect them early.

📌 Step 2: Rebuild the Flattened Tree

```
void flatten(TreeNode* root) {
    if(root == NULL)return;
    vector<TreeNode*> v;
    pre_order(root, v);
    TreeNode* temp = root;
    root->left = NULL;
    root->right = NULL;
    for(int i = 1; i < v.size(); i++) {
        while(temp != NULL && temp->right != NULL) {
            temp = temp->right;
        }
        temp->right = v[i];
    }
}
```

- Calls `pre_order()` to fill the vector with nodes in preorder.
- Starts with the original root as `temp`.
- Iteratively moves to the end of the current right chain and appends the next node.
- Repeats this for all nodes in the vector, building a right-only chain.

🚨 Space Complexity

- **Time Complexity:** `O(n)`
- **Space Complexity:** `O(n)` for the vector (extra space used)
- Not a true in-place solution due to the vector.

🧾 Summary

Property	Value
Traversal used	Preorder (Root → L → R)
Space used	`O(n)`
In-place	❌ (vector used)
Output format	Right-only chain, Left = NULL
Simplicity	✅ Easy to understand

🔥 Leetcode 114. Flatten Binary Tree to Linked List — Method 2 (Using Direct Recursion)

🧠 Approach Summary

- Use recursion to flatten the tree in-place.
- Flatten left and right subtrees first.
- Then attach the flattened left subtree to the right of the root.
- Finally, append the flattened right subtree at the end of this new chain.
- All left pointers are set to `NULL` as per problem requirement.
- No extra space (vector) is used — fully in-place.

💻 Complete Code with Comments

```
// method 2
// using recursion directly
class Solution {
public:
    void flatten(TreeNode* root) {
        if (root == NULL) return; // base case: empty node
```

```
        TreeNode* left = root->left;   // save left subtree
        TreeNode* right = root->right; // save right subtree

        root->left = NULL; // set left to NULL as required

        flatten(left);  // flatten left subtree recursively
        flatten(right); // flatten right subtree recursively

        root->right = left; // attach flattened left subtree on right

        TreeNode* temp = root;
        // move temp to the end of new right subtree
        while (temp != NULL && temp->right != NULL) {
            temp = temp->right;
        }

        temp->right = right; // attach flattened right subtree at the end
    }
};
```

🔍 Step-by-step Explanation

Step	What happens?
`if(root == NULL) return`	Stops recursion when node is NULL.
`TreeNode* left = root->left;`	Stores left child before breaking links.
`TreeNode* right = root->right;`	Stores right child before breaking links.
`root->left = NULL;`	Sets left pointer to NULL, as required by problem.
`flatten(left); flatten(right);`	Recursively flatten left and right subtrees.
`root->right = left;`	Attaches flattened left subtree to root’s right pointer.
`while(temp->right != NULL) temp = temp->right;`	Finds end of newly attached right subtree.

`temp→right = right;`	Attaches flattened right subtree at the end.

⚙️ Time and Space Complexity

- **Time Complexity:** O(n), each node is visited once.
- **Space Complexity:** O(h), recursion stack space (h = height of tree).
- **In-place:** Yes, no extra data structures like vectors are used.

⚡ Leetcode 114. Flatten Binary Tree to Linked List — Method 3 (Using Morris Traversal)

🧠 Approach Summary

- Morris Traversal technique ka use karke **in-place** tree ko flatten karte hain.
- Is method me **left subtree ko right subtree ke beech mein insert kar dete hain**, bina recursion ya extra space ke.
- Har node ke left subtree ka rightmost node find karke uska `right` pointer current node ke original right child se jod dete hain.
- Phir current node ka right pointer left subtree se connect kar dete hain aur left pointer ko `NULL` kar dete hain.
- Finally, puri tree preorder traversal ke order me flatten ho jati hai.

💻 Complete Code with Comments

```
// Method 3 using morris Traversal
class Solution {
public:
```

```
    void flatten(TreeNode* root) {
        if (root == NULL) return; // base case: empty tree

        TreeNode* c = root;  // current pointer starting from root

        while (c != NULL) {
            if (c->left != NULL) {  // if left child exists
                TreeNode* r = c->right;  // store current right child
                TreeNode* p = c->left;   // pointer to left subtree

                c->right = c->left;      // move left subtree to right

                // find rightmost node of left subtree
                while (p->right != NULL) {
                    p = p->right;
                }

                p->right = r;            // connect original right subtree after it

                c = c->left;             // move current to the new right (old left)
            } else {
                // if no left child, move to right subtree
                c = c->right;
            }
        }

        // After rearrangement, set all left pointers to NULL
        TreeNode* temp = root;
        while (temp != NULL && temp->right != NULL) {
            temp->left = NULL;
            temp = temp->right;
        }
    }
};
```

🔍 Step-by-step Explanation

Step	Description
`if (root == NULL) return;`	If tree is empty, nothing to do.
`TreeNode* c = root;`	Start from root.
`while (c != NULL)`	Iterate until end of tree (rightmost node).
`if (c->left != NULL)`	If left child exists, restructure the tree.
`TreeNode* r = c->right;`	Save the current right subtree.
`TreeNode* p = c->left;`	Move pointer `p` to left child.
`c->right = c->left;`	Move left subtree to the right pointer.
`while (p->right != NULL) p = p->right;`	Find rightmost node of the moved left subtree.
`p->right = r;`	Attach saved right subtree after rightmost node of left subtree.
`c = c->left;`	Move current pointer to new right subtree (originally left subtree).
`else c = c->right;`	If no left child, move to right child.
`while (temp != NULL && temp->right != NULL) { temp->left = NULL; temp = temp->right; }`	Finally, set all left pointers to NULL while traversing right chain.

⚙️ Time and Space Complexity

- **Time Complexity:** O(n) — each node is processed at most twice.
- **Space Complexity:** O(1) — no recursion or extra data structures used.
- **In-place:** ✅ Fully in-place, no extra memory.
