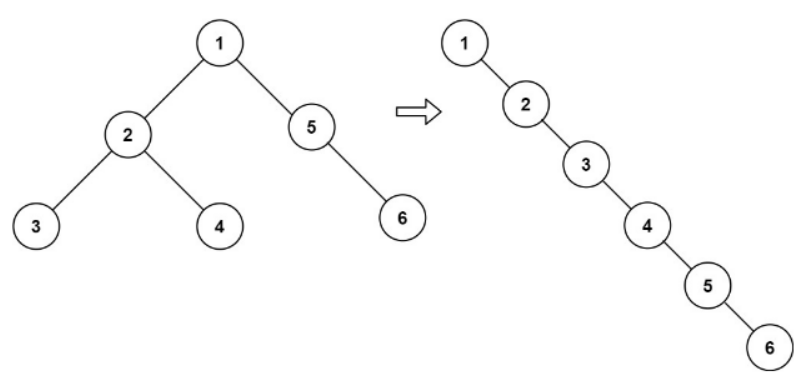
<https://leetcode.com/problems/flatten-binary-tree-to-linked-list/>

Given the root of a binary tree, flatten the tree into a "linked list":

* The "linked list" should use the same TreeNode class where the right child pointer points to the next node in the list and the left child pointer is always null.
* The "linked list" should be in the same order as a **[pre-order traversal](https://en.wikipedia.org/wiki/Tree_traversal" \l "Pre-order,_NLR)** of the binary tree.

**Example 1:**



Input: root = [1,2,5,3,4,null,6]

Output: [1,null,2,null,3,null,4,null,5,null,6]

**Example 2:**

Input: root = []

Output: []

**Example 3:**

Input: root = [0]

Output: [0]

**Constraints:**

* The number of nodes in the tree is in the range [0, 2000].
* -100 <= Node.val <= 100

**Follow up:** Can you flatten the tree in-place (withO(1)extra space)?

**Attempt 1: 2022-12-27**

**Solution 1:  Reverse preorder traversal (60 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

TreeNode prev = null;

public void flatten(TreeNode root) {

if(root == null) {

return;

}

flatten(root.right);

flatten(root.left);

root.right = prev;

root.left = null;

prev = root;

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Refer to**

<https://leetcode.com/problems/flatten-binary-tree-to-linked-list/solutions/36977/my-short-post-order-traversal-java-solution-for-share/comments/150699>

Great solution of the Reverse preorder traversal in (right, left, root) order!

Basically, the traversing order after flattening is pre order traversal in (root, left, right), like

1

\

2

\

3

\

4

\

5

\

6

from the original tree:

1

/ \

2 5

/ \ \

3 4 6

If we traverse the flattened tree in the reverse way, we would notice that [6->5->4->3->2->1] is in (right, left, root) order of the original tree. So the reverse order after flattening is reverse preorder traversal in (right, left, root) order like [6->5->4->3->2->1].

The idea is to traverse the original tree in this order by

public void flatten(TreeNode root) {

if (root == null)

return;

flatten(root.right);

flatten(root.left);

}

and then set each node's right pointer as the previous one in [6->5->4->3->2->1], as such the right pointer behaves similar to a link in the flattened tree(though technically, it's still a right child reference from the tree data structure's perspective) and set the left child as null before the end of one recursion by

private TreeNode prev = null;

public void flatten(TreeNode root) {

if (root == null)

return;

flatten(root.right);

flatten(root.left);

root.right = prev;

root.left = null;

prev = root;

}

**Refer to**

<https://leetcode.com/problems/flatten-binary-tree-to-linked-list/solutions/36977/my-short-post-order-traversal-java-solution-for-share/comments/146292>

draw a picture for understanding iterative process

1

/ \

2 5

/ \ \

3 4 6

-----------

pre = 5

cur = 4

1

/

2

/ \

3 4

\

5

\

6

-----------

pre = 4

cur = 3

1

/

2

/

3

\

4

\

5

\

6

-----------

cur = 2

pre = 3

1

/

2

\

3

\

4

\

5

\

6

-----------

cur = 1

pre = 2

1

\

2

\

3

\

4

\

5

\

6

**Solution 2: Stack (30 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public void flatten(TreeNode root) {

if(root == null) {

return;

}

Stack<TreeNode> stack = new Stack<TreeNode>();

stack.push(root);

while(!stack.isEmpty()) {

TreeNode node = stack.pop();

if(node.right != null) {

stack.push(node.right);

}

if(node.left != null) {

stack.push(node.left);

}

node.left = null;

if(stack.isEmpty()) {

node.right = null;

} else {

node.right = stack.peek();

}

}

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Step by step process**

1

/ \

2 5

/ \ \

3 4 6

push [1] --> pop [1] --> node = 1 --> push [5] --> push [2](peek) --> 1 right connect 2 --> on stack [2]

pop [2] --> node = 2 --> push [4] --> push [3](peek) --> 2 right connect 3 --> on stack [3, 4, 5]

pop [3] --> node = 3 --> 4 is peek now --> 3 right connect 4 --> on stack [4, 5]

pop [4] --> node = 4 --> 5 is peek now --> 4 right connect 5 --> on stack [5]

pop [5] --> node = 5 --> push [6](peek) --> 5 right connect 6 --> on stack [6]

pop [6] --> node = 6 --> stack is empty --> while loop end

Now 1 --> 2 --> 3 --> 4 --> 5 --> 6 concatenate finished