Flatten Binary Tree to LinkedList

114. Flatten Binary Tree to Linked List

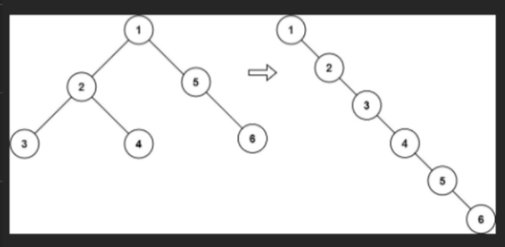
Solved

Medium | Topics | Companies | Hint

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** of the binary tree.

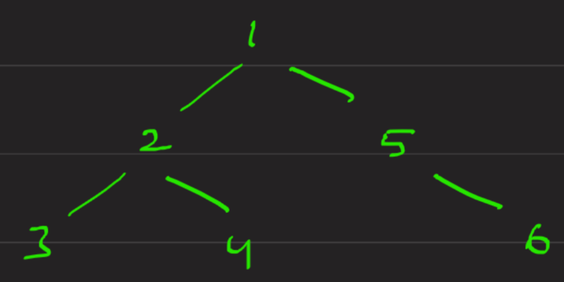
Example 1:



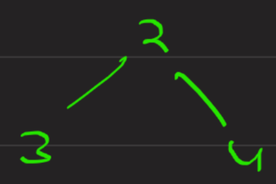
Input: root = [1,2,5,3,4,null,6]
Output: [1,null,2,null,3,null,4,null,5,null,6]

Example 2:

Input: root = []
Output: []



If you observe carefully, the nodes which are apearing as left child are added first to the root as right pointers

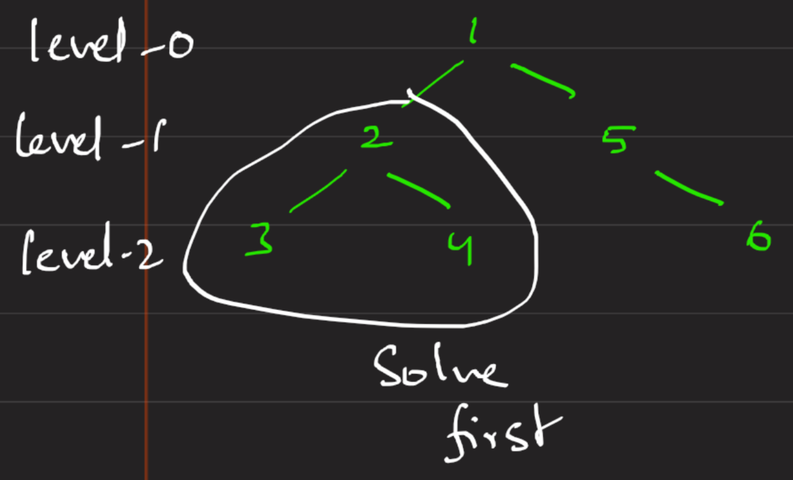


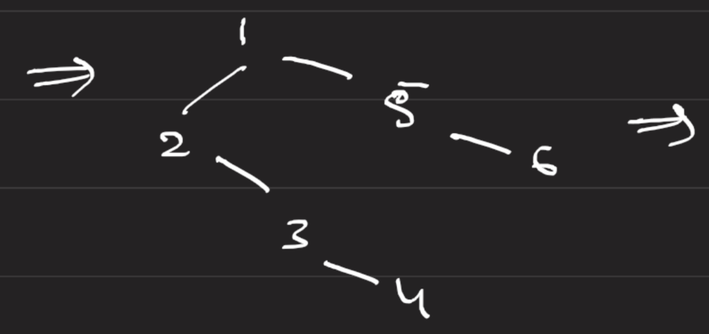
take this as a sub problem

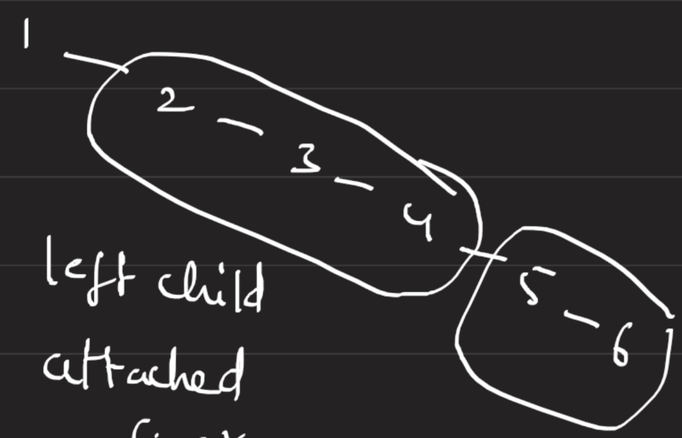
→ this will be our linked list

Intution

left child processes first and attach to the root and attached child's right node will be parents previous right child







* So, we need to process left child first

if we use a queue → in 1st level (we add 2, 5) to queue

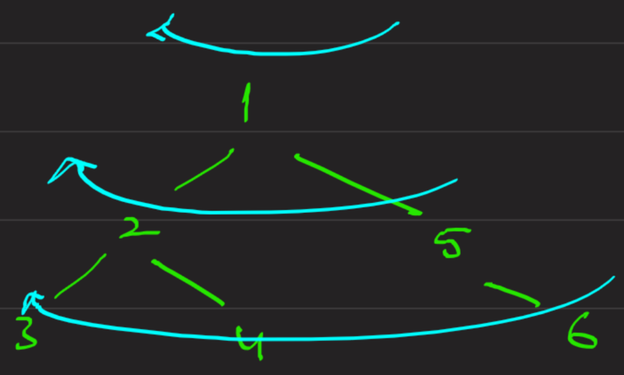
But when we remove from queue node (2) will come first

* But we are mainly focusing one first node we are appending to the queue

So, for that we uses stack instead

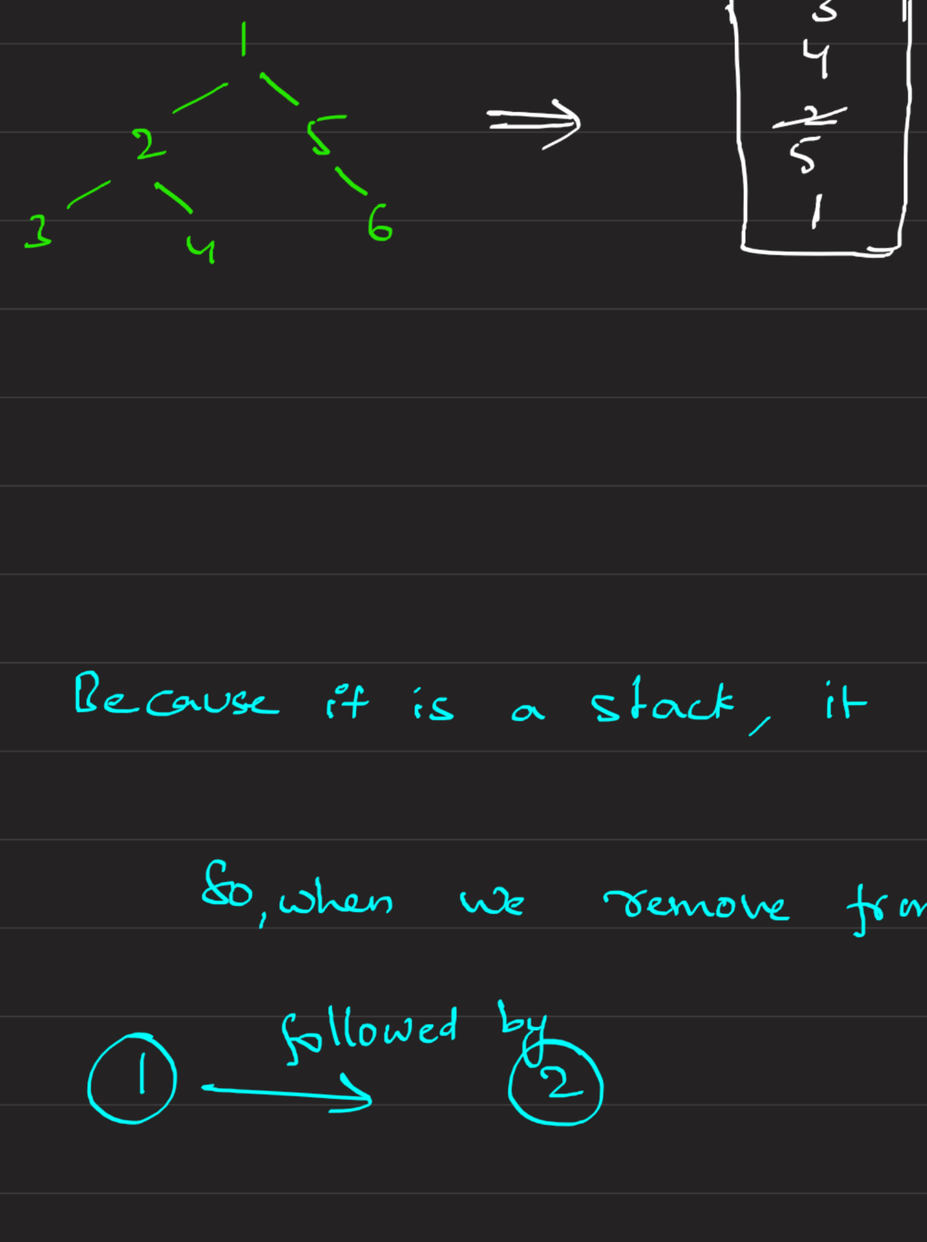
Note: This can be solved using queue aswell, but this technique will be handy for you in Others problems aswell

*



If I use stack for BFS, & traverse from right to left

it solves our problem of getting left children inOrder first



See, At first we added node (1)

&

then by right to left we add (5) then (2)

Because it is a stack, it works on LIFO (we added 2 last)

So, when we remove from stack for new level BFS (2 will be removed)

(1) followed by (2)

Then, Because we are traversing from right to left

children of 2 {3, 4}

(4) added to stack first & then (3)

Because, it is a stack, on new BFS level, we remove stacks top which (3)

∴ (1) —followed by→ (2) —followed by→ (3) —followed by→ (4)

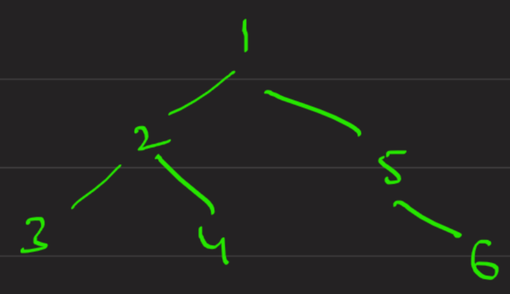
this is the Order we are getting when we use (stack & traverse from right to left)

```
if(root == null)return;
Stack<TreeNode> st = new Stack();
st.push(root);

while(!st.isEmpty()){
    TreeNode cur = st.pop();

    if(cur.right != null){
        st.push(cur.right);
    }
    if(cur.left != null){
        st.push(cur.left);
    }
    if(st.size()>0){
        cur.right = st.peek();
    }
    cur.left = null;
}

}
```

Simple BFS

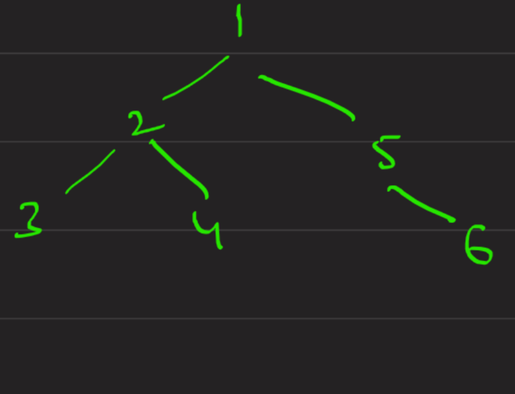
→ Traverse right first & then left

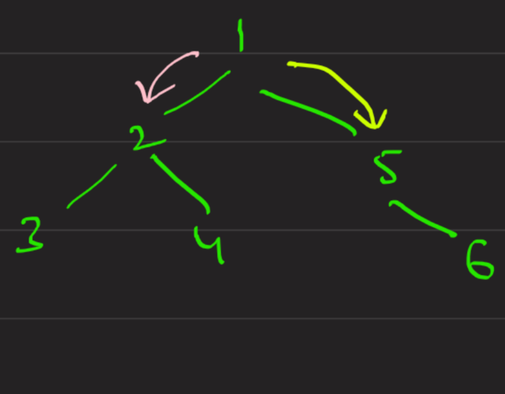
make sure, you connect the links for linked list

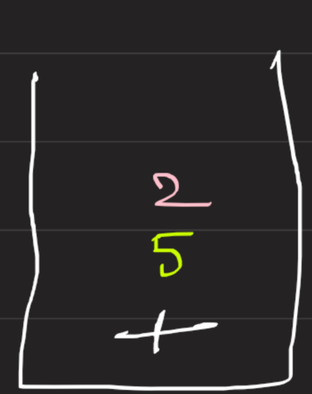


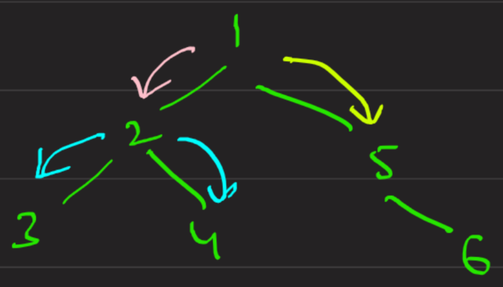

```java
if(root == null)return;
Stack<TreeNode> st = new Stack();

st.push(root);

while(!st.isEmpty()){
    TreeNode cur = st.pop();

    if(cur.right != null){
        st.push(cur.right);
    }
    if(cur.left != null){
        st.push(cur.left);
    }
    if(st.size()>0){
        cur.right = st.peek();
    }
    cur.left = null;
}
}
```

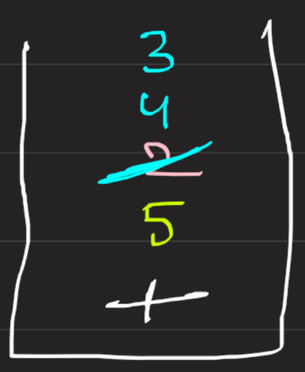
1

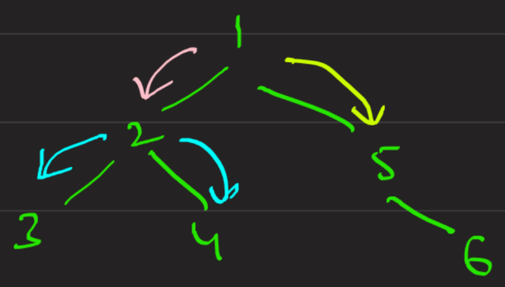


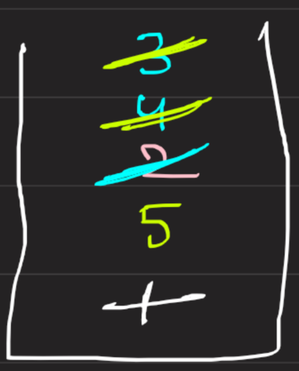


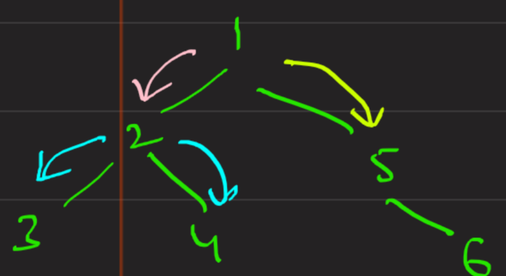


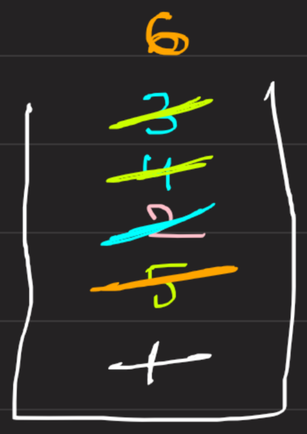


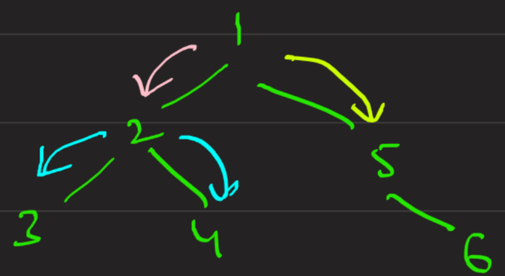


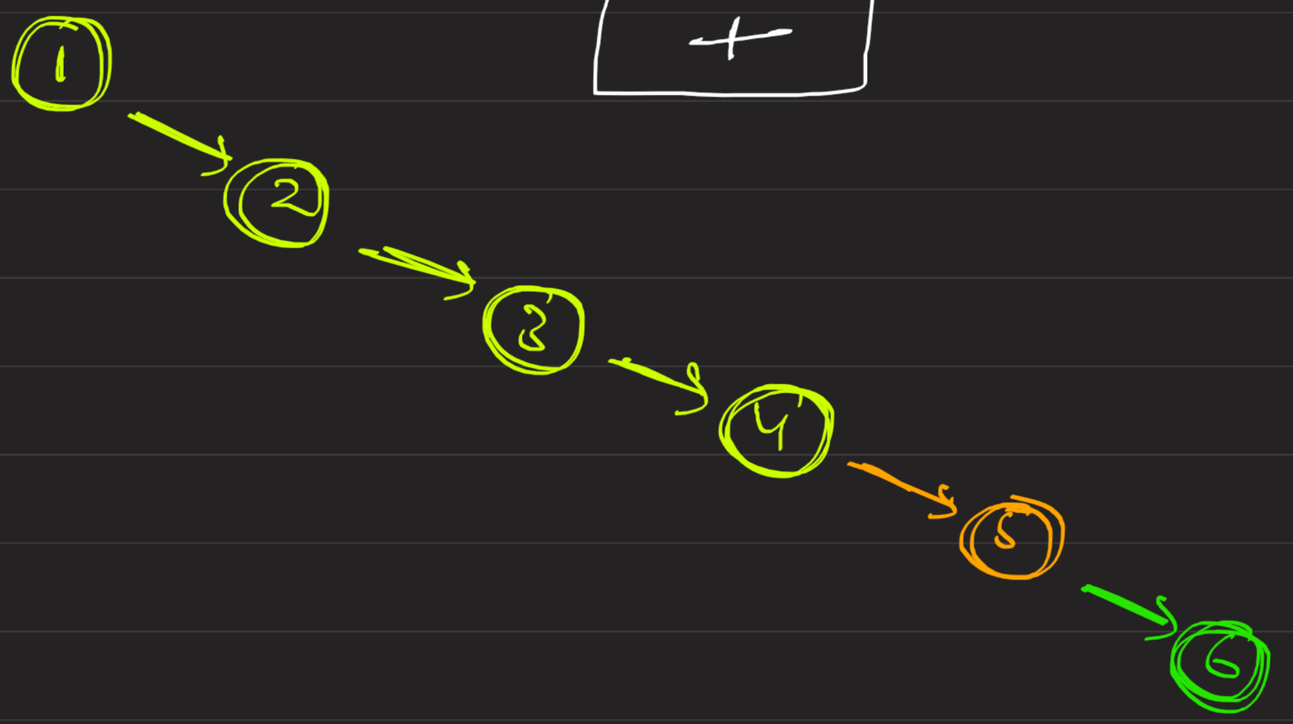












Attaching peak ~~node to~~ Cur → right

* Make Sure On every iteration, you will definetely attach a node,

```java
if(st.size()>0){
    cur.right = st.peek();
}
cur.left = null;
```
→ at that time, make left pointer to null