114. Flatten Binary Tree to Linked List

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

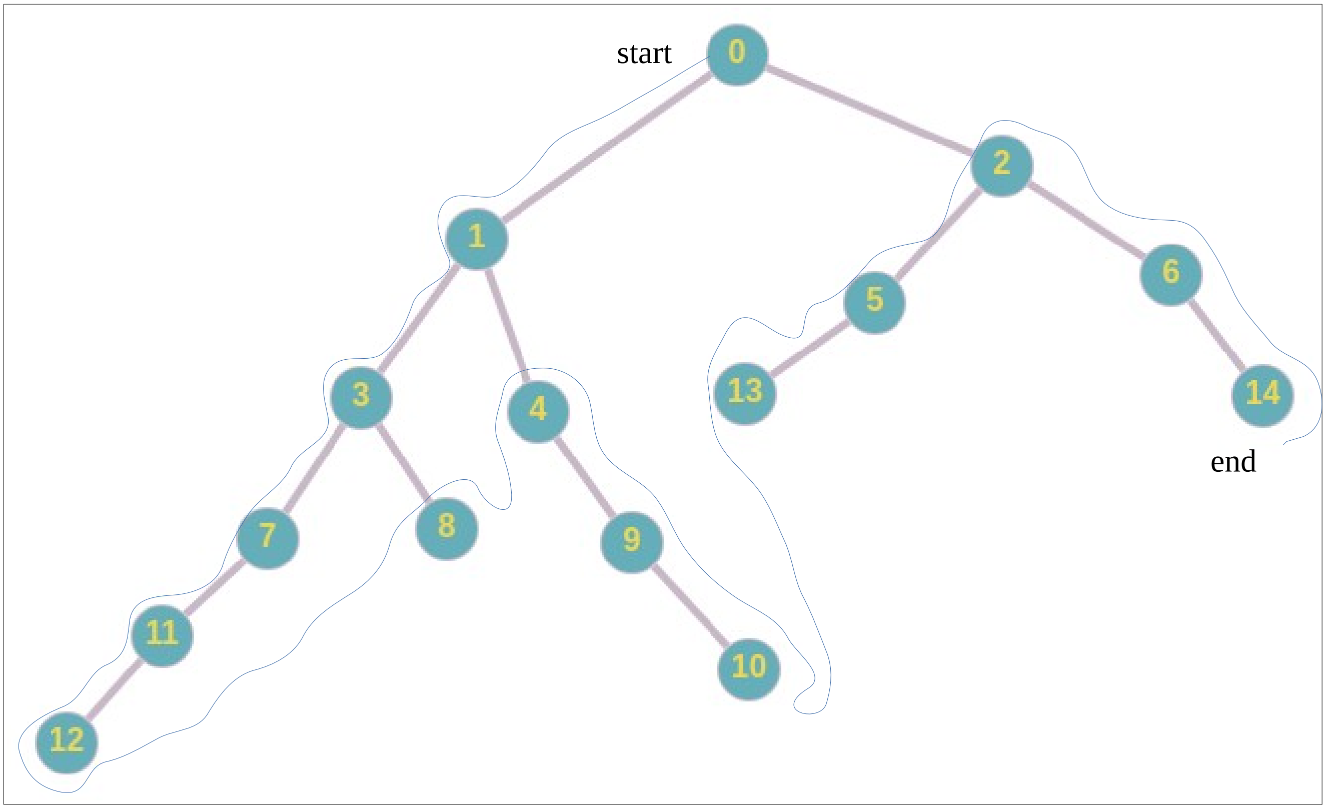
Given the root of a binary tree, flatten the tree into a "linked list":

- 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** of the binary tree.

Wha’s the pre-order traversal?

The pre-order traversal is:

- explore the node first
- then, the left child of the node,
- then, the right child of the node,



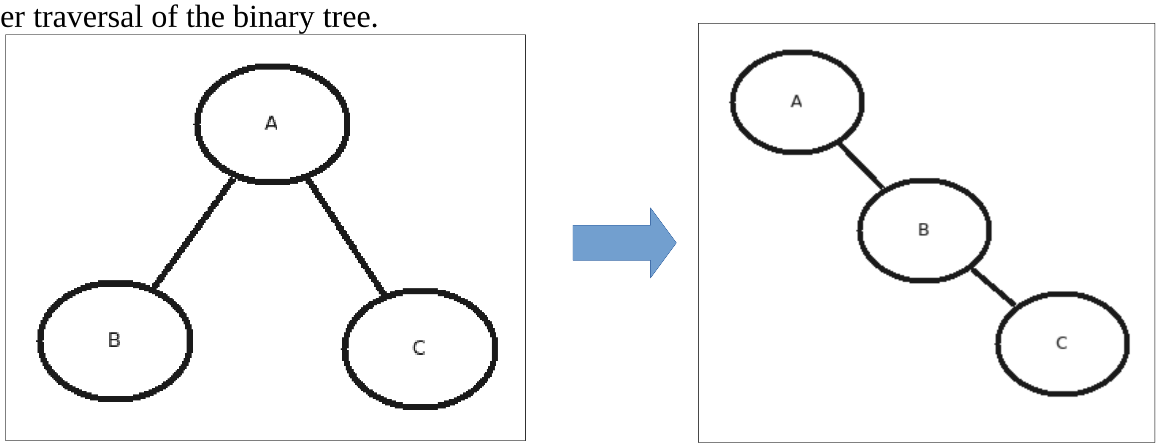
out put will be: 0 1 3 4 11 12 8 4 9 10 13 5 2 6 14

The recursive implementation of pre-order (also in-order, post-order) is very known, which its pseudo-code is:

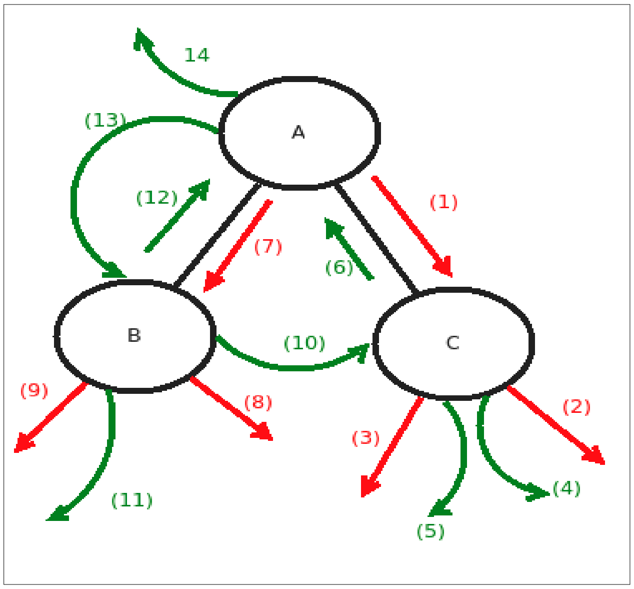
```
pre_order(node)
  print(node value)
  pre_order(node left)
  pre_order(node right)
```

Flatten a binary tree: recursive way

We want to flat a binary tree into a linked list, and the linked list should be in the same order as the pre-order traversal of the binary tree.



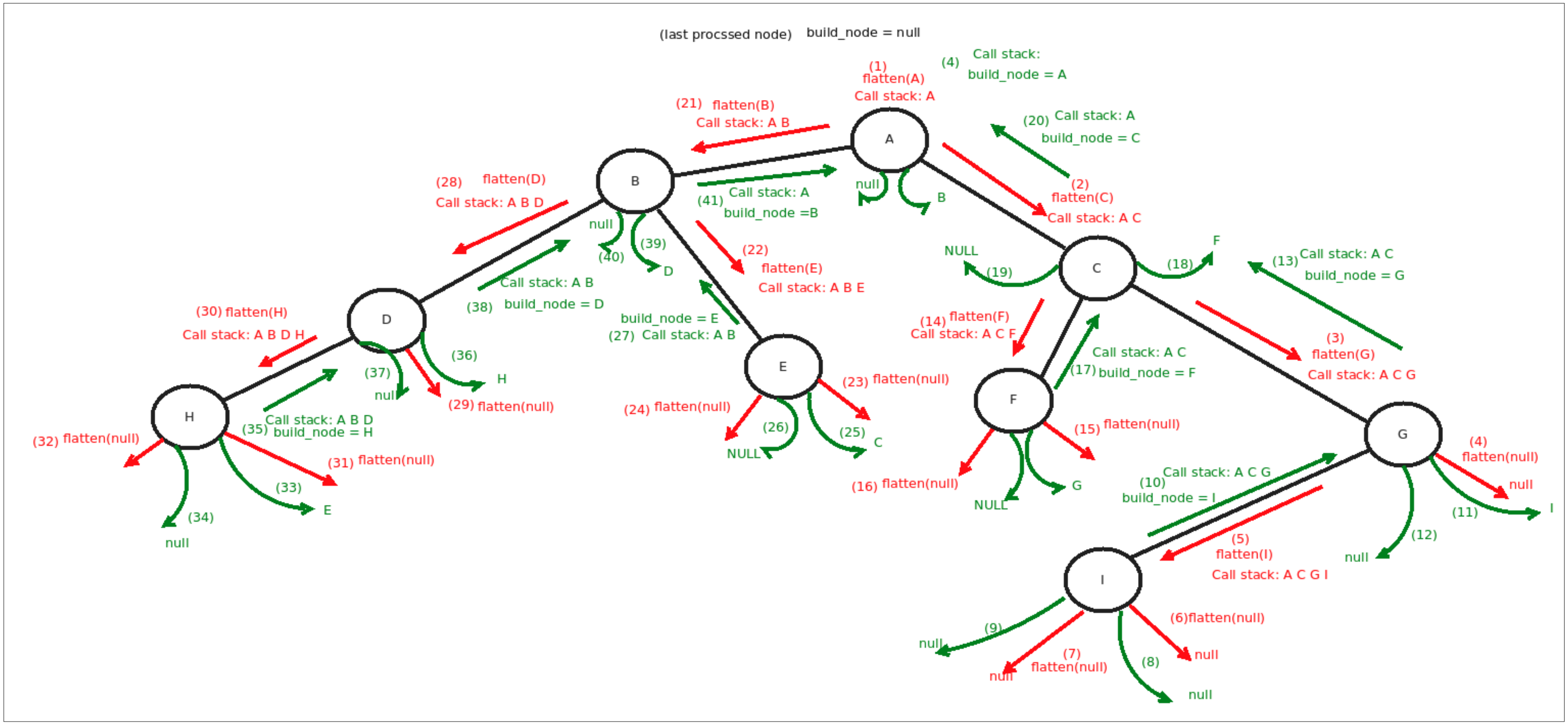
In General, process the right sub tree first, the left sub-tree, at last the node.



(1): go right to process "A" → right: "C"
(2): go right to process "C" → right: "null"
(3): go left to process "C" → left: "null"
(4): "C" → right = last processed node: "null"
(5): "C" → left = "null"
last processed node is "C"
(6) go back to process the left of "A"
(7) go left to process "A" → left: "B"
(8) go right to process "B" → right: "null"
(9) go left to process "B" → left: "null"
(10): "B" → right = last processed node ="C"
(11): "B" → left = "null"
last processed node is "B"
(12): go back the node "A"
"A" is processed
(13): "A → right = last processed node ="B"
(14): "A" → left = "null"

This the post-order traversal, but instead of processing the left sub-tree (left, right, node) first, process the right sub-tree (right, left, node)

Let’s go throw an example:

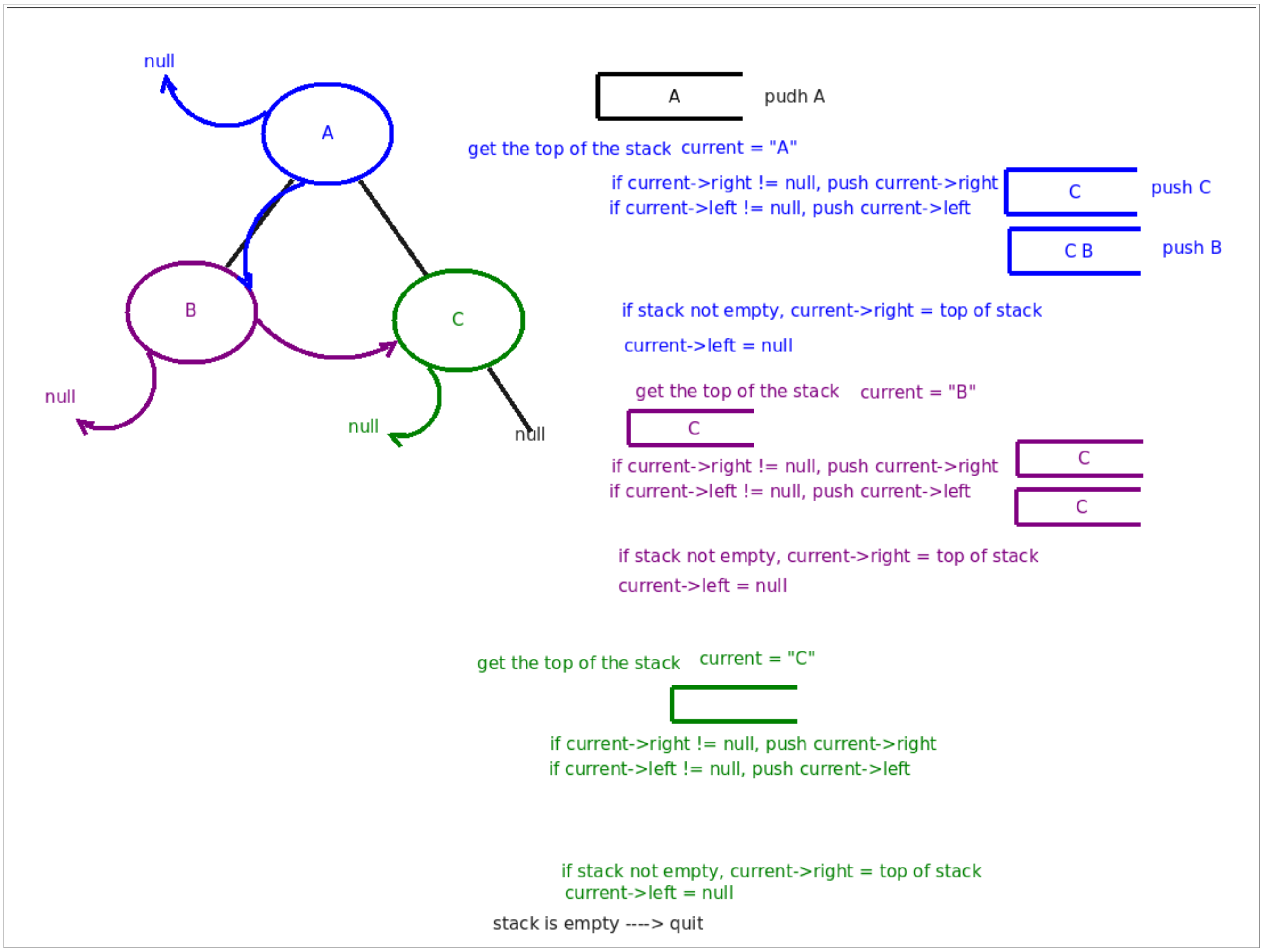

(last procssed node) build_node = null
(1) flatten(A) Call stack: A
(4) Call stack: build_node = A
(21) flatten(B) Call stack: A B
(20) Call stack: A build_node = C
A
null
B
(2) flatten(C) Call stack: A C
B
(41) Call stack: A build_node =B
(28) flatten(D) Call stack: A B D
null
(40)
(39)
D
(22) flatten(E) Call stack: A B E
NULL
(19)
C
(18)
F
(13) Call stack: A C build_node = G
Call stack: A B
(38) build_node = D
build_node = E
(27) Call stack: A B
(14) flatten(F) Call stack: A C F
Call stack: A C
(17) build_node = F
(3) flatten(G) Call stack: A C G
(30) flatten(H) Call stack: A B D H
D
(37)
null
(36)
H
(29) flatten(null)
E
(24) flatten(null)
(26)
NULL
(25)
C
(23) flatten(null)
F
(15) flatten(null)
(16) flatten(null)
NULL
G
G
(4) flatten(null)
null
(11)
I
H
(32) flatten(null)
(35) Call stack: A B D build_node = H
(31) flatten(null)
(33)
E
(34)
null
Call stack: A C G
(10) build_node = I
(5) flatten(I) Call stack: A C G I
null
(12)
I
(9)
null
(7) flatten(null)
null
(8)
null
(6)flatten(null)
null

C++ Code: recursive

```
class Solution {
    public:
        TreeNode* last_processed_node = nullptr;
public:
    void flatten(TreeNode* root) {
        if (root == nullptr) return;
        flatten(root->right);
        flatten(root->left);
        root->right = last_processed_node;
        root->left = nullptr;
        last_processed_node = root;
    }
};
```

Flatten a binary tree: iterative way

The DFS algorithm has the same behavior as pre-order traversal on a tree.



pudh A

get the top of the stack current = "A"

if current->right != null, push current->right

if current->left != null, push current->left

push C

push B

if stack not empty, current->right = top of stack

current->left = null

get the top of the stack current = "B"

if current->right != null, push current->right

if current->left != null, push current->left

if stack not empty, current->right = top of stack

current->left = null

get the top of the stack current = "C"

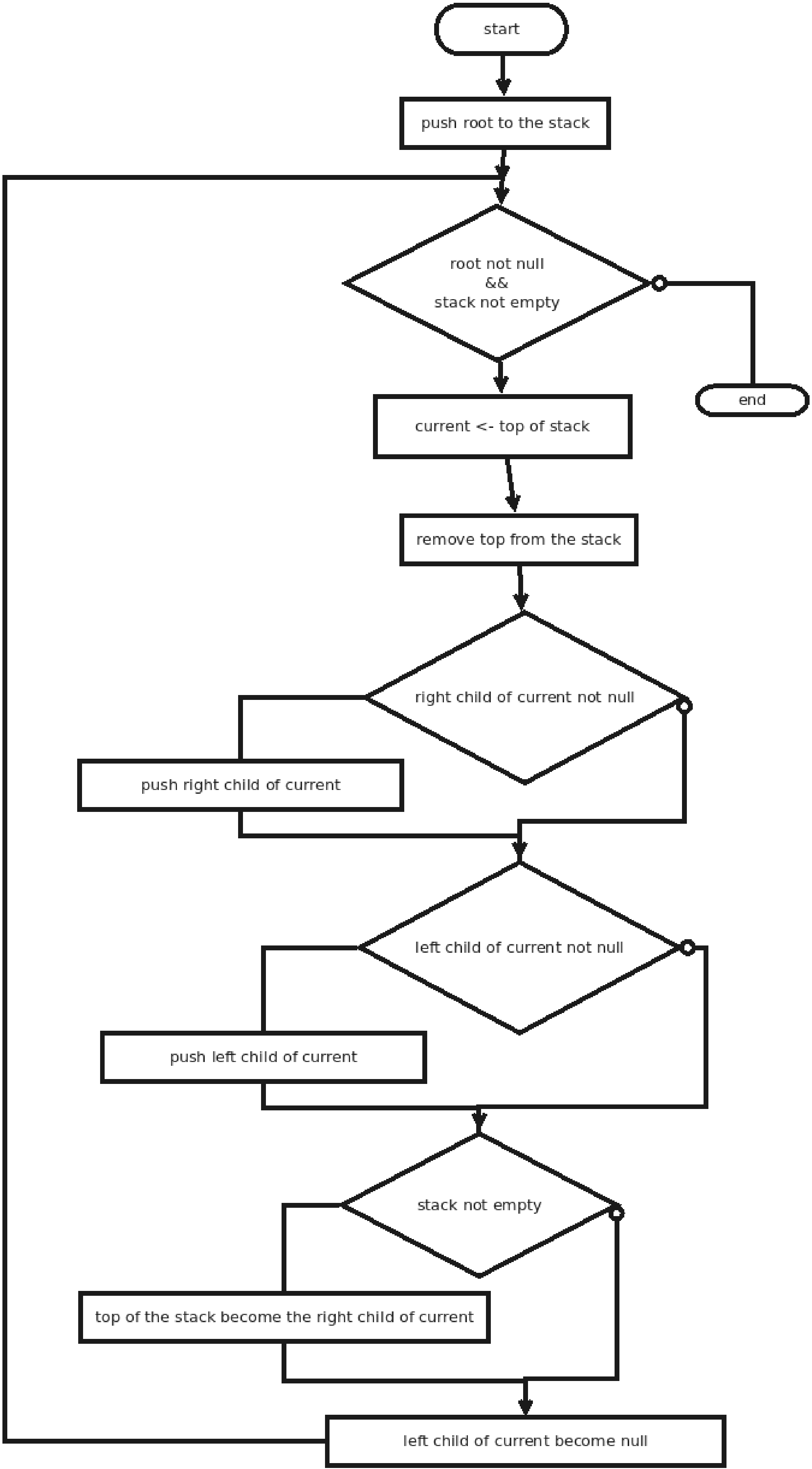
if current->right != null, push current->right

if current->left != null, push current->left

if stack not empty, current->right = top of stack

current->left = null

stack is empty ----> quit


start
push root to the stack
root not null
&&
stack not empty
end
current <- top of stack
remove top from the stack
right child of current not null
push right child of current
left child of current not null
push left child of current
stack not empty
top of the stack become the right child of current
left child of current become null

C++ Code: recursive

```
class Solution {
public:
    void flatten(TreeNode* root) {
        stack<TreeNode*> s ;
        s.push(root);
        while (root != nullptr && !s.empty()){
            TreeNode *current = s.top();
            s.pop();
            if (current->right != nullptr){
                s.push(current->right);
            }
            if (current->left != nullptr){
                s.push(current->left);
            }
            if (!s.empty()) current->right = s.top();
            current->left = nullptr;
        }

    }
};
```