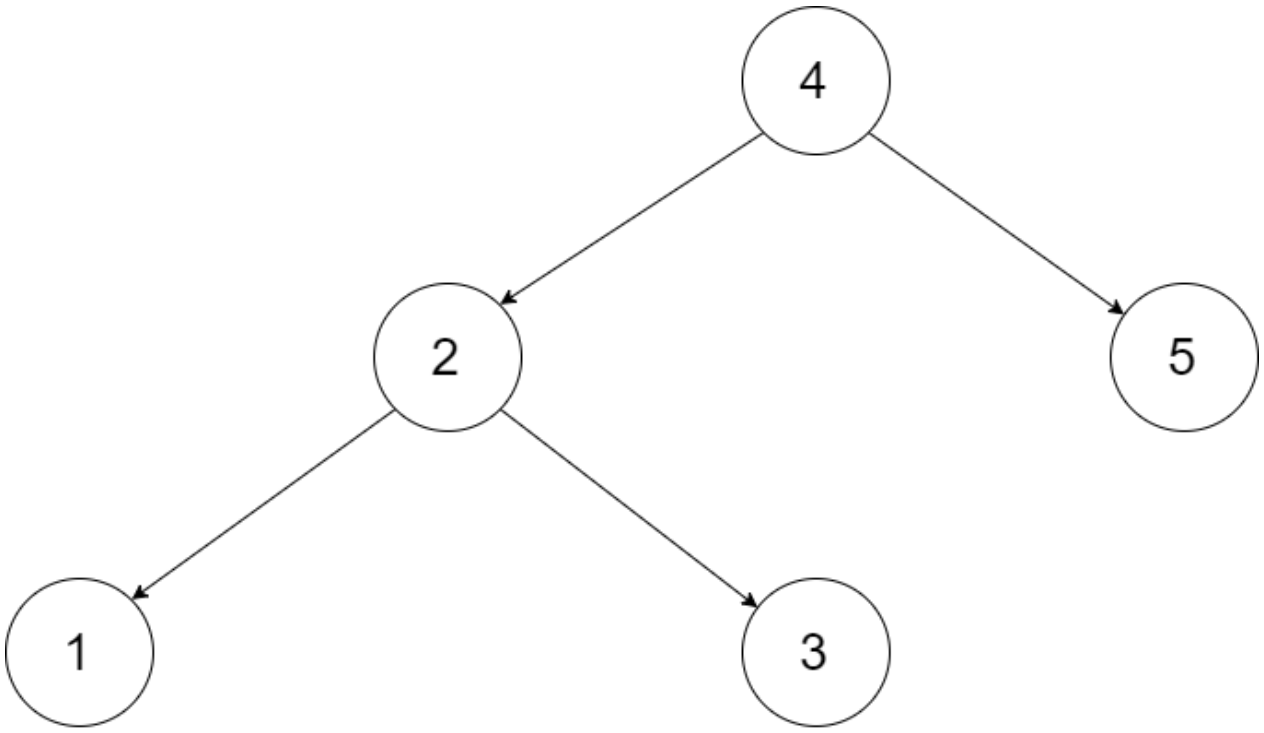
426. Convert Binary Search Tree to Sorted Doubly Linked List (/problems/convert-binary-search-tree-to-sorted-doubly-linked-list/)

March 27, 2019 | 13.9K views

Average Rating: 4.71 (17 votes)

Convert a BST to a sorted circular doubly-linked list in-place. Think of the left and right pointers as synonymous to the previous and next pointers in a doubly-linked list.

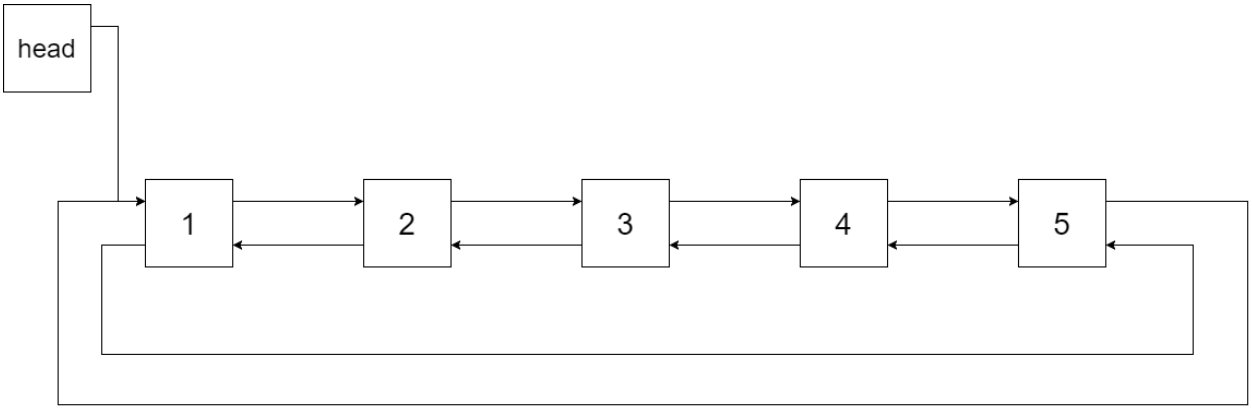
Let's take the following BST as an example, it may help you understand the problem better:



We want to transform this BST into a circular doubly linked list. Each node in a doubly linked list has a predecessor and successor. For a circular doubly linked list, the predecessor of the first element is the last element, and the successor of the last element is the first element.

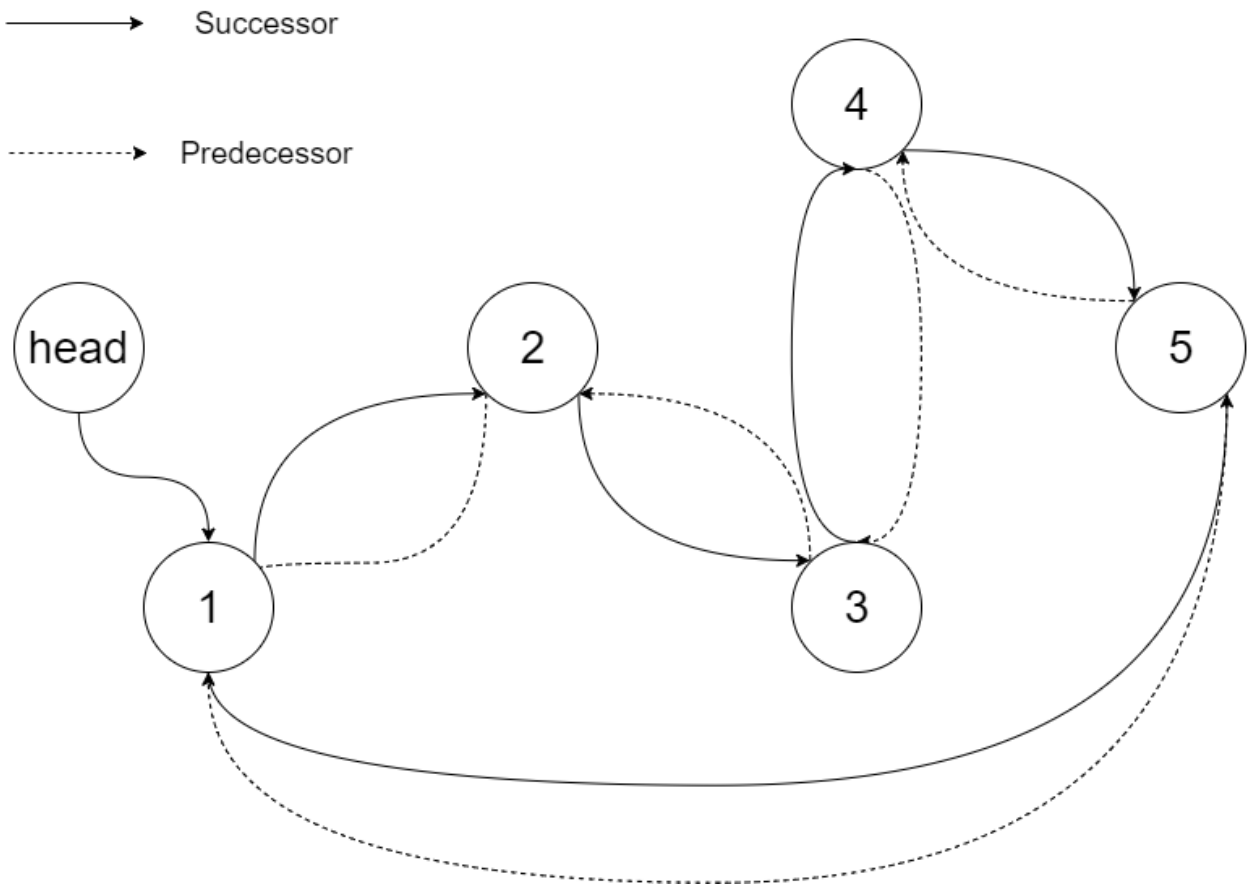
The figure below shows the circular doubly linked list for the BST above. The "head" symbol means

the node it points to is the smallest element of the linked list.



Specifically, we want to do the transformation in place. After the transformation, the left pointer of the tree node should point to its predecessor, and the right pointer should point to its successor. We should return the pointer to the first element of the linked list.

The figure below shows the transformed BST. The solid line indicates the successor relationship, while the dashed line means the predecessor relationship.



Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Depth First Search* (`DFS`)

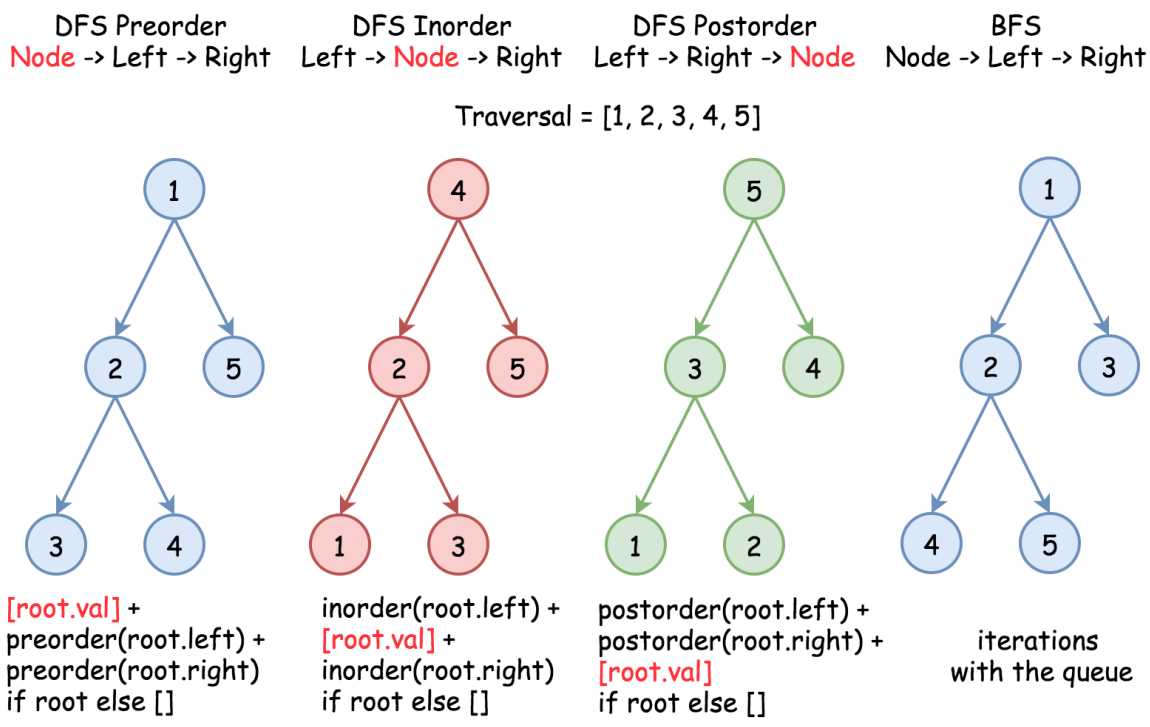
  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

  The DFS strategy can further be distinguished as `preorder` , `inorder` , and `postorder` depending on the relative order among the root node, left node and right node.

- *Breadth First Search* (`BFS`)

  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

On the following figure the nodes are numerated in the order you visit them, please follow `1-2-3-4-5` to compare different strategies.



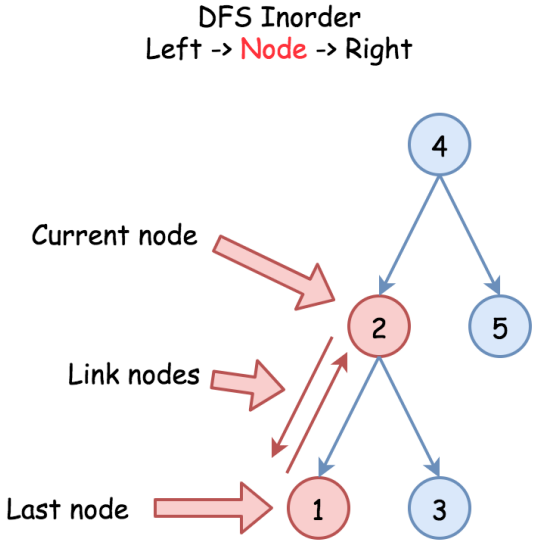
Here the problem is to implement DFS inorder traversal in a textbook recursion way because of in-place requirement.

Approach 1: Recursion

Algorithm

Standard inorder recursion follows `left -> node -> right` order, where `left` and `right` parts are the recursion calls and `node` part is where all processing is done.

Processing here is basically to link the previous node with the current one, and because of that one has to track the last node which is the largest node in a new doubly linked list so far.

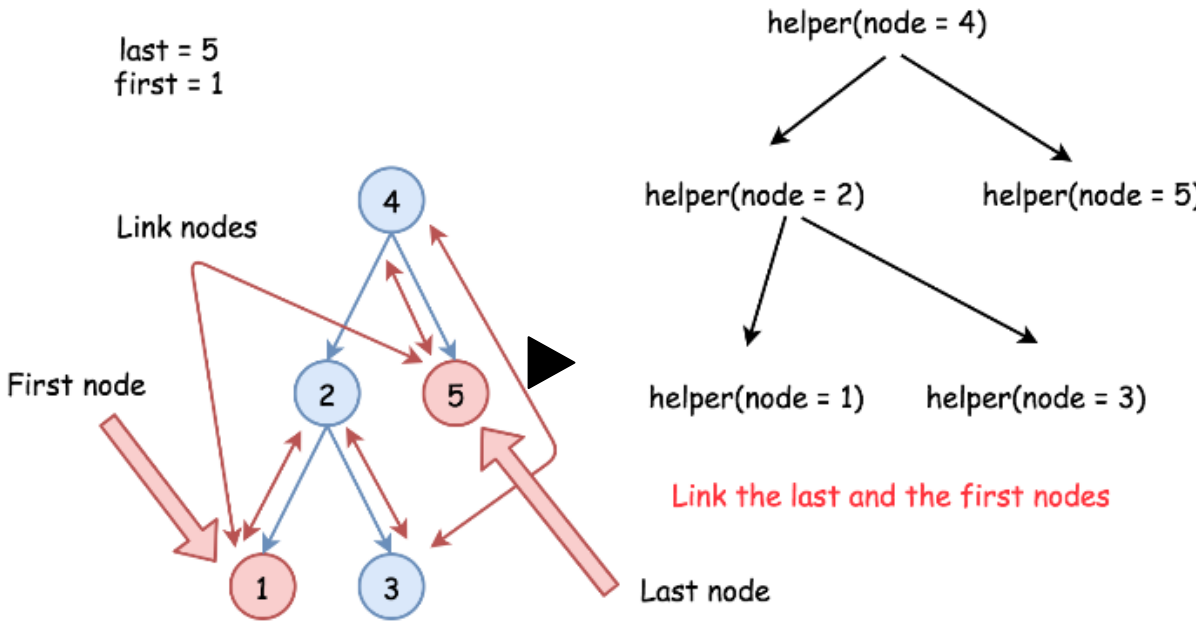


One more detail : one has to keep the first, or the smallest, node as well to close the ring of doubly linked list.

Here is the algorithm :

- Initiate the `first` and the `last` nodes as nulls.
- Call the standard inorder recursion `helper(root)` :
    - If node is not null :
        - Call the recursion for the left subtree `helper(node.left)` .
        - If the `last` node is not null, link the `last` and the current `node` nodes.
        - Else initiate the `first` node.
        - Mark the current node as the last one : `last = node` .
        - Call the recursion for the right subtree `helper(node.right)` .
- Link the first and the last nodes to close DLL ring and then return the `first` node.

Implementation



8 / 9

C++ Java Python Copy

```python
class Solution:
    def treeToDoublyList(self, root: 'Node') -> 'Node':
        def helper(node):
            """
            Performs standard inorder traversal:
            left -> node -> right
            and links all nodes into DLL
            """
            nonlocal last, first
            if node:
                # left
                helper(node.left)
                # node
                if last:
                    # link the previous node (last)
                    # with the current one (node)
                    last.right = node
                    node.left = last
                else:
                    # keep the smallest node
                    # to close DLL later on
                    first = node
                last = node
                # right
                helper(node.right)

        if not root:
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since each node is processed exactly once.
- Space complexity : $\mathcal{O}(N)$. We have to keep a recursion stack of the size of the tree height, which is $\mathcal{O}(\log N)$ for the best case of completely balanced tree and $\mathcal{O}(N)$ for the worst case of completely unbalanced tree.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/combinations/) Next (/articles/maximal-rectangle/)

Comments: 5

Sort By

Type comment here... (Markdown is supported)

Preview Post

(/coder_xyzzy)

coder_xyzzy (coder_xyzzy) ★ 42 March 27, 2019 3:18 PM

This is not O(1) space. It is O(tree depth), so O(n) worst case and O(log(n)) average case.

Keep in mind that the recursive call stack takes space, and there will be (at maximum depth) one stack from for each level in the tree.

7 Share Reply

SHOW 5 REPLIES

(/stonedcoders)

StonedCoders (stonedcoders) ★ 4 August 8, 2019 10:47 PM

Java iterative solution.

```
public Node treeToDoublyList(Node root) {
   if( root == null) return root;
   Node dummy = new Node(0);
```

Read More

4 Share Reply

SHOW 1 REPLY

(/alazemif)

alazemif (alazemif) ★ 2 April 12, 2019 6:41 PM

A simple way is to put all node's pointers in after an in order traversal. Then link adjacent nodes and the first and last.
Complexity N for inorder traversal + N to go through the loop = O(2N) = O(N)
Space, You need an Array to hold N node's pointers -> O(log N) recursive stack and O(N) for array of pointers.

Read More

2 ▲ ▼ | Share | Reply

SHOW 1 REPLY

(/cheney1993)

Cheney1993 (cheney1993) ★ 21 August 20, 2019 4:40 PM

Approach 1: Iteration
Python3

```
def treeToDoublyList(self, root: 'Node') -> 'Node':
```

Read More

1 ▲ ▼ | Share | Reply

(/czhangaegean)

czhangaegean (czhangaegean) ★ 197 October 6, 2019 1:34 PM

The question itself states "in-place", but it does not seems possible.

0 ▲ ▼ | Share | Reply

SHOW 1 REPLY