426. Convert Binary Search Tree to Sorted Doubly Linked List

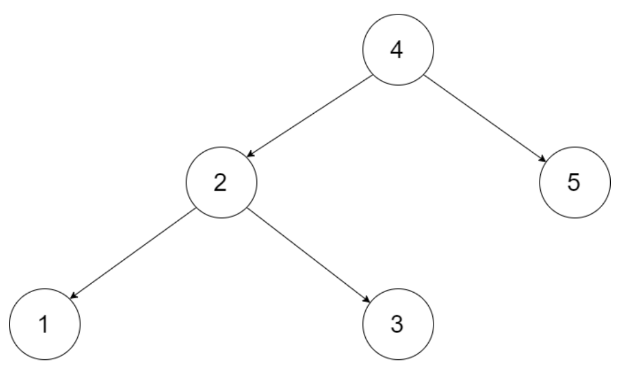
Medium 1805 149 Add to List Share

Convert a **Binary Search Tree** to a sorted **Circular Doubly-Linked List** in place.

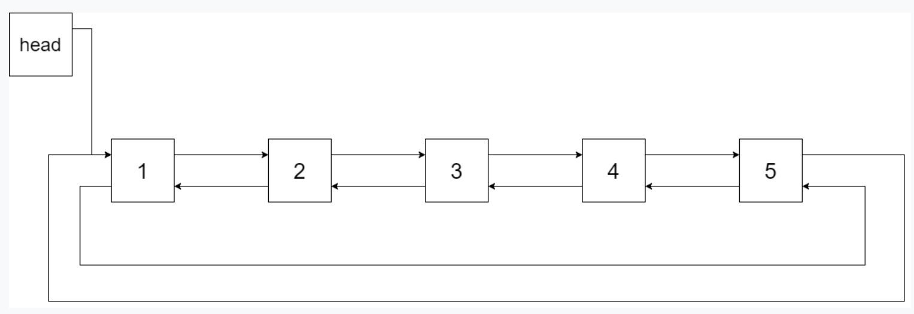
You can think of the left and right pointers as synonymous to the predecessor and successor pointers in a doubly-linked list. For a circular doubly linked list, the predecessor of the first element is the last element, and the successor of the last element is the first element.

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. You should return the pointer to the smallest element of the linked list.

Example 1:

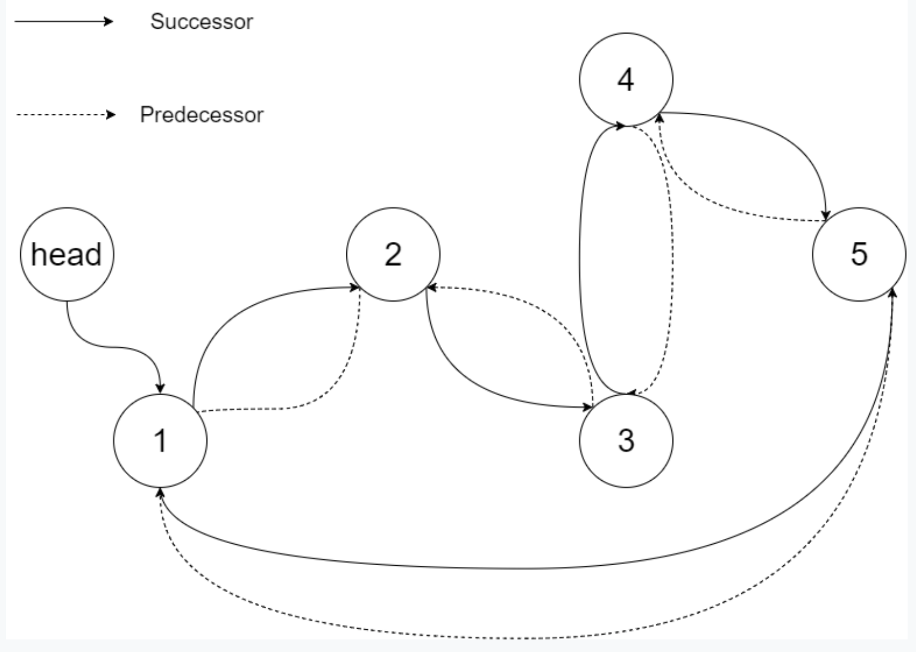


```
Input: root = [4,2,5,1,3]
```



```
Output: [1,2,3,4,5]

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



Example 2:

```
Input: root = [2,1,3]
Output: [1,2,3]
```

Example 3:

```
Input: root = []
Output: []
Explanation: Input is an empty tree. Output is also an empty Linked List.
```

Example 4:

```
Input: root = [1]
Output: [1]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 2000]`.
- `-1000 <= Node.val <= 1000`
- All the values of the tree are **unique**.

Accepted 164,363 Submissions 258,952

Seen this question in a real interview before? Yes No

Companies *i*

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Facebook 58 Amazon 5 Microsoft 4 ByteDance 2

Related Topics

Linked List Stack Tree Depth-First Search Binary Search Tree Binary Tree Doubly-Linked List

Similar Questions

Binary Tree Inorder Traversal Easy

i Java Autocomplete

```java
/*
// Definition for a Node.
class Node {
    public int val;
    public Node left;
    public Node right;

    public Node() {}

    public Node(int _val) {
        val = _val;
    }

    public Node(int _val,Node _left,Node _right) {
        val = _val;
        left = _left;
        right = _right;
    }
};
*/

class Solution {
  // the smallest (first) and the largest (last) nodes
  Node first = null;
  Node last = null;

  public void helper(Node node) {
    if (node != null) {
      // left
      helper(node.left);
      // node
      if (last != null) {
        // 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);
    }
  }

  public Node treeToDoublyList(Node root) {
    if (root == null) return null;

    helper(root);
    // close DLL
    last.right = first;
    first.left = last;
    return first;
  }
}
```