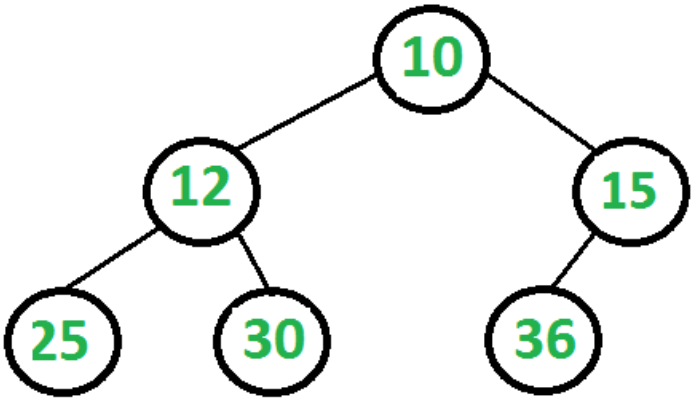
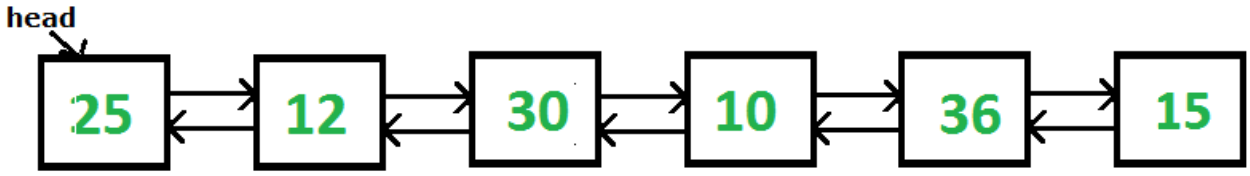
Binary Tree to DLL

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.



The above tree should be in-place converted to following Doubly Linked List(DLL).



Example 1:

```
Input:
      1
    /  \
   3    2
Output: 3 1 2
2 1 3
Explanation: DLL would be 3<=>1<=>2
```

Example 2:

```
Input:
       10
      /   \
     20   30
   /   \
  40   60
```

```
Output: 40 20 60 10 30
30 10 60 20 40 Explanation: DLL would be
40<=>20<=>60<=>10<=>30.
```

Your Task:
You don't have to take input. Complete the function **bToDLL()** that takes **root** node of the tree as a parameter and returns the head of DLL . The driver code prints the DLL both ways.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(H).
Note: H is the height of the tree and this space is used implicitly for the recursion stack.

Constraints:

1 ≤ Number of nodes ≤ 10^5

1 ≤ Data of a node ≤ 10^5

```
class Solution:
    def bToDLL(self,root):
        # do Code here
        self.dummyHead = Node(-1)
        self.prev = self.dummyHead
        self.helper(root)
        head = self.dummyHead.right
        self.dummyHead.right = None
        head.left = None
        return head

    def helper(self,root):
        if root is None:
            return
        self.helper(root.left)
        self.prev.right = root
        root.left = self.prev
        self.prev = root
        self.helper(root.right)
```

```
class Solution:
    def bToDLL(self,root):
        # do Code here
        stack = []
        self.addAllLeft(root,stack)

        dummyHead = Node(-1)
        prev = dummyHead
```

```
        while len(stack)>0:
            curr = stack.pop()
            prev.right = curr
            curr.left = prev
            prev = curr

            self.addAllLeft(curr.right,stack)
        head = dummyHead.right
        dummyHead.right = None
        head.left = None
        return head



    def addAllLeft(self,curr,stack):
        while curr!=None:
            stack.append(curr)
            curr = curr.left
```