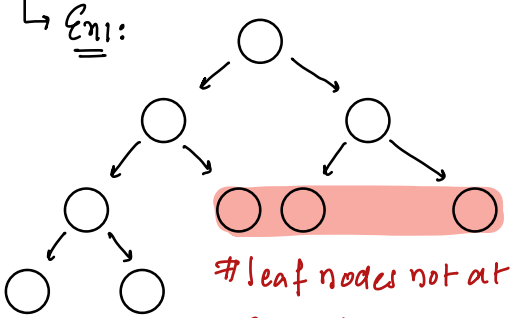
Todays Content: 7:05 AM

- → Perfect Binary Tree
- → fill right in Perfect binary Tree
- → CDLL
- → Merge 2 CDLL
- → BST → CDLL
- → Morris Inorder traversal

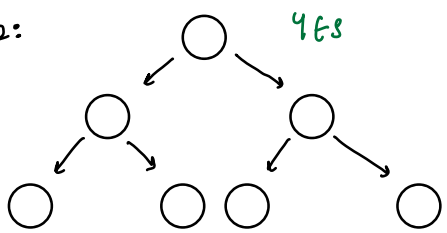
Perfect binary Tree:

A binary tree is a perfect binary tree in which all non leaf nodes have 2 children & all leaf nodes are at same level

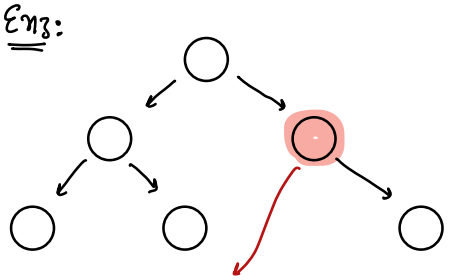
Ex1:



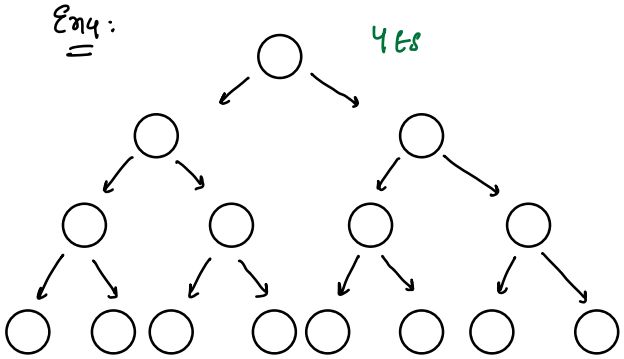
Ex2: Yes



Ex3:



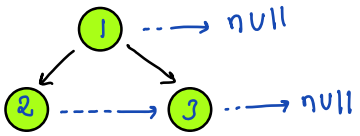
Ex4: Yes



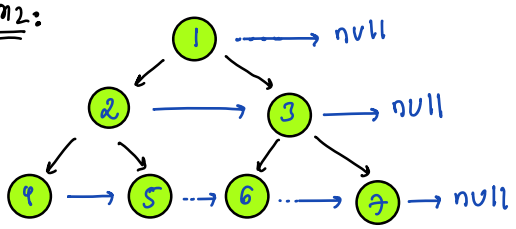
Fill next in Perfect Binary Tree : Expected SC: O(1)

```
class Node {
    int data
    Node left, right, next;
    Node(n) {
        data = n
        left = null
        right = null
        next = null
    }
}
```

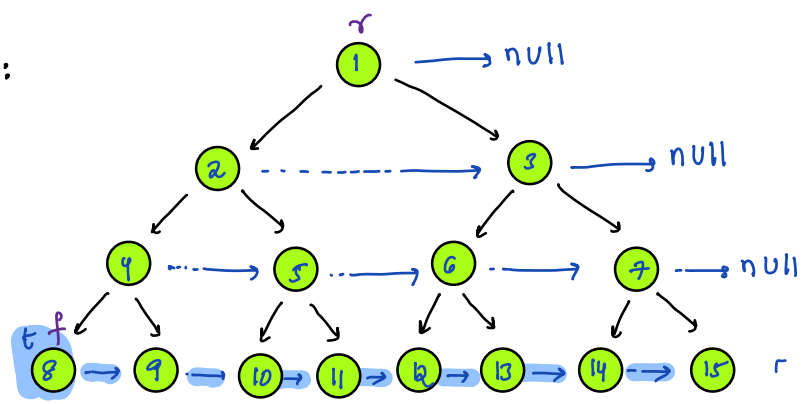
Ex1:



Ex2:



Ex3:



Node FillNext(Node root) { TC: O(N) SC: O(1)

```
Node t = root;
while( t.left != null ) {
    Node f = t
    while (t != null) {
        t.left.next = t.right ✓
        if (t.next != null) {
            t.right.next = t.next.left ✓
        }
        t = t.next ✓
    }
    t = f.left
}
```

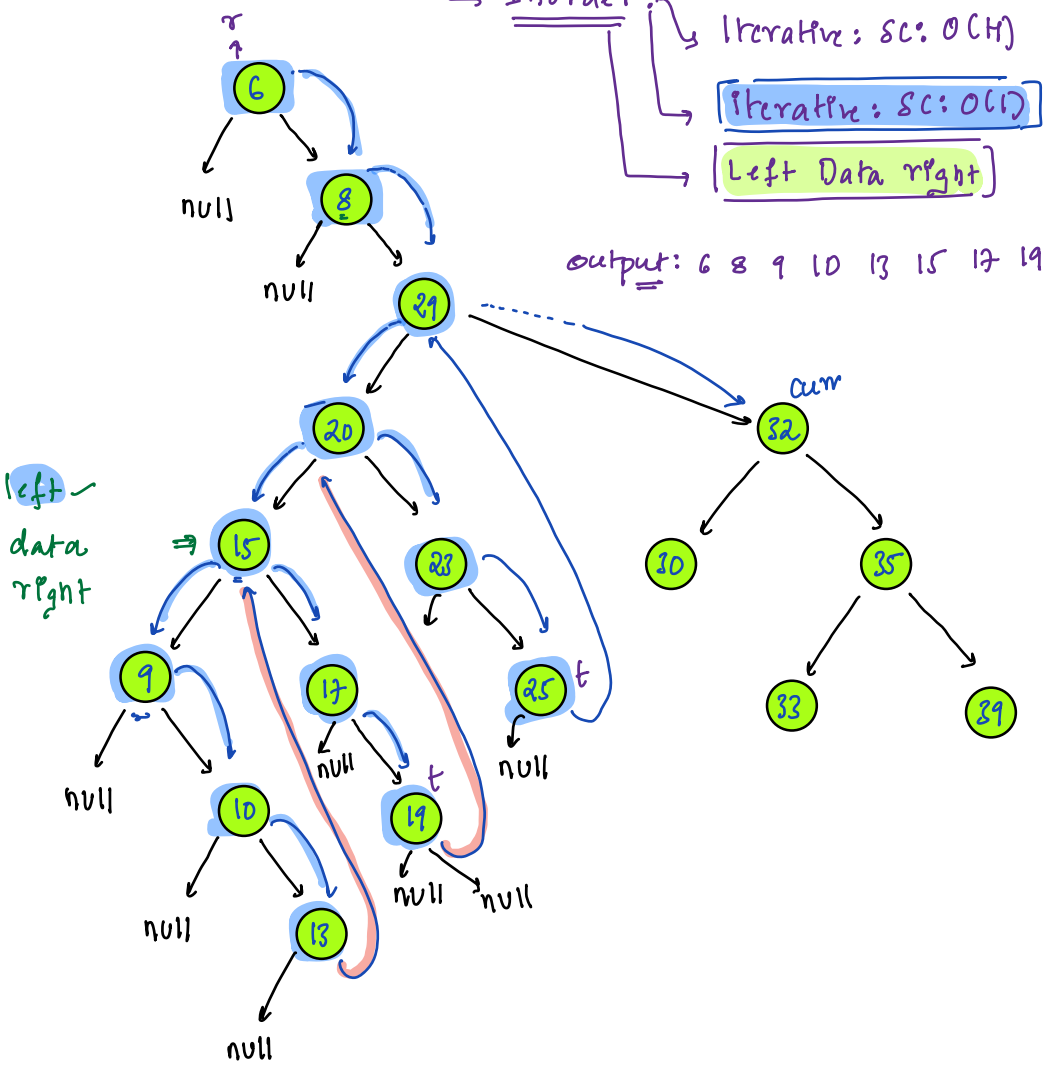
TODO: If not a perfect binary Tree how to do?

// fill next in a particular,

TODO: In above question after fill next for every node print level order traversal, without extra space

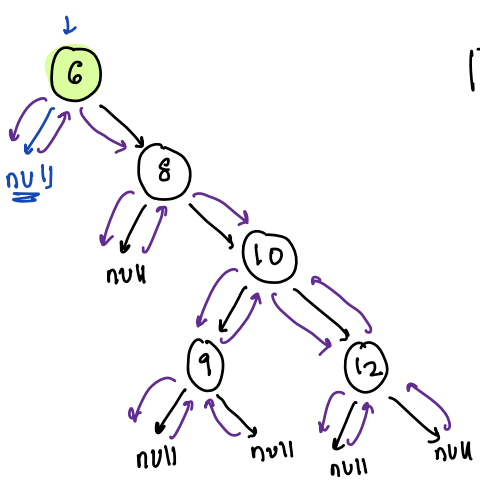
→ Morris Inorder traversal :

↳ Inorder : → Recusive : SC: O(H)
→ Iterative : SC: O(H)
→ Iterative : SC: O(1)
→ Left Data right



output: 6 8 9 10 13 15 17 19 20 23 25 29

→ Ex:



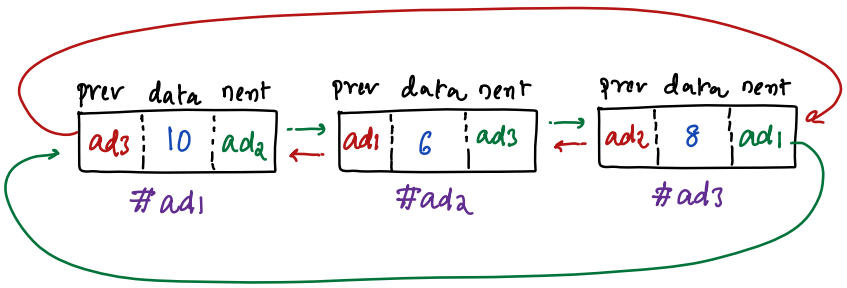
8:48 am

```
void inorder(Node root){ TC: O(N) SC: O(1)
    Node curr = root
    while (curr != null){
        if (curr.left == null){ // Inorder left data right
            print(curr.data)
            curr = curr.right
        }
        else{
            obs1: If we are visit node 1st time
                  → Man of hist, right will point = null
            obs2: If we are visit node 2nd time
                  → Man of hist, right will point = curr node itself
            Node t = curr.left
            while (t.right != null && t.right != curr){
                t = t.right
            }
            if (t.right == null){ // curr we visited 1st time
                t.right = curr      left node 1st right
                curr = curr.left
            }
            else{ // curr we visited 2nd time
                print(curr.data) left node 2nd right
                curr = curr.right
                t.right = null // deleting link we created
            }
        }
    }
}
```

Obs: faster than recursive code

```
class Node {
    int data
    Node(int x) {
        data = x
        left = null
        right = null
    }
}
```

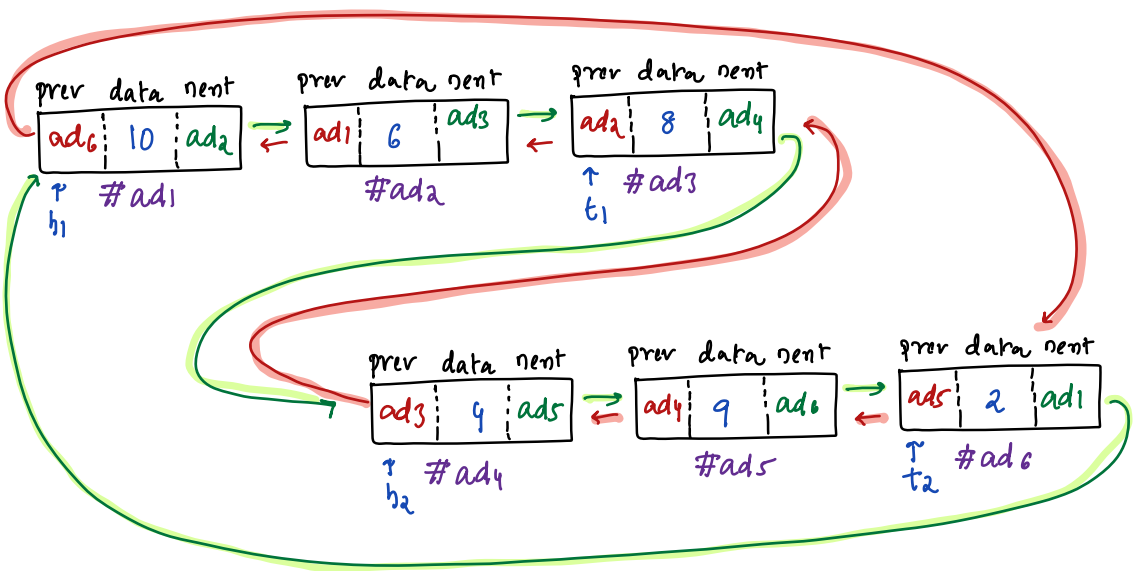
Circular double linked list:



Note: In Circular double linked list

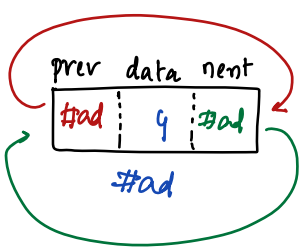
→ 1st node prev = last node & last node next = first node

Ex1: Given 2 circular double linked list combine them

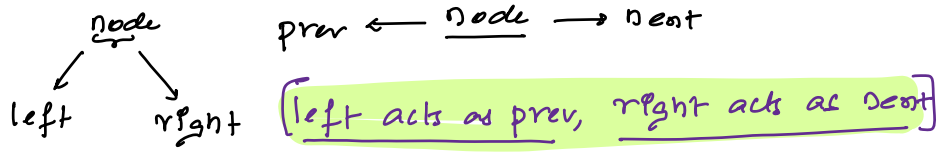



```
Node combine(Node h1, Node h2) {        TC: O(1) SC: O(1)
    [if (h1 == null) { return h2 }
    if (h2 == null) { return h1 }]
    Node t1 = h1.prev
    Node t2 = h2.prev
    t1.next = h2   h1.prev = t2
    h2.prev = t1   t2.next = h1
}
```

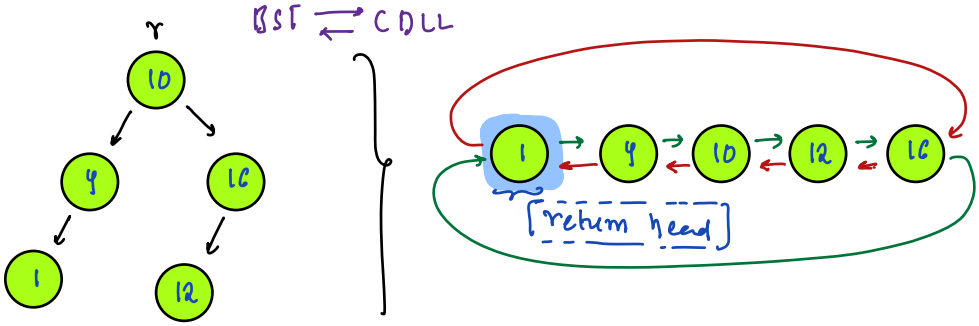
Note: Single node in
↳ Circular Double linked list

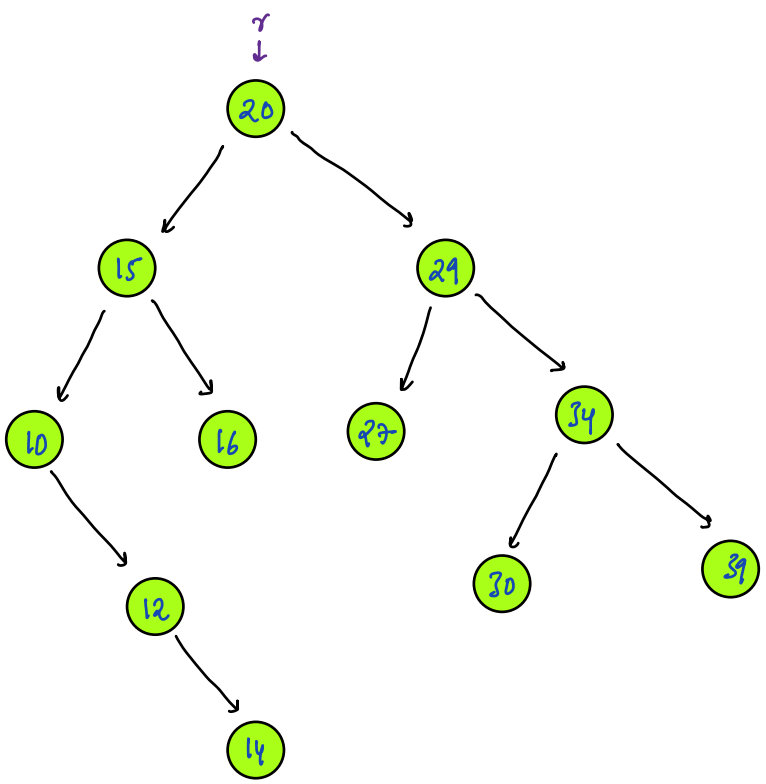


38) Given a BST convert into a sorted circular double linked & return head node



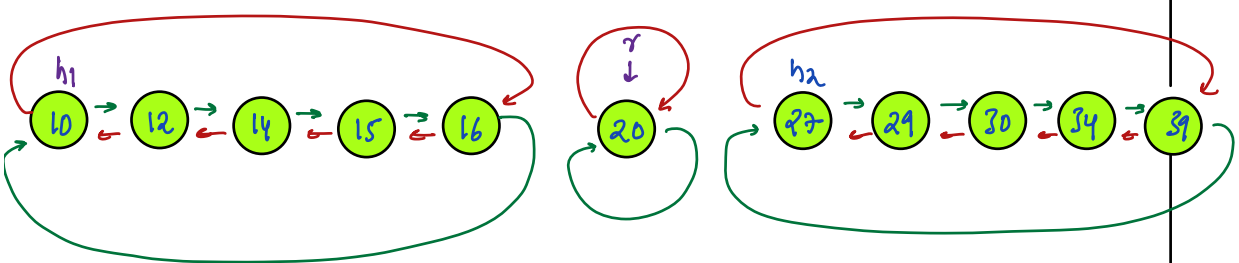
Eg1:

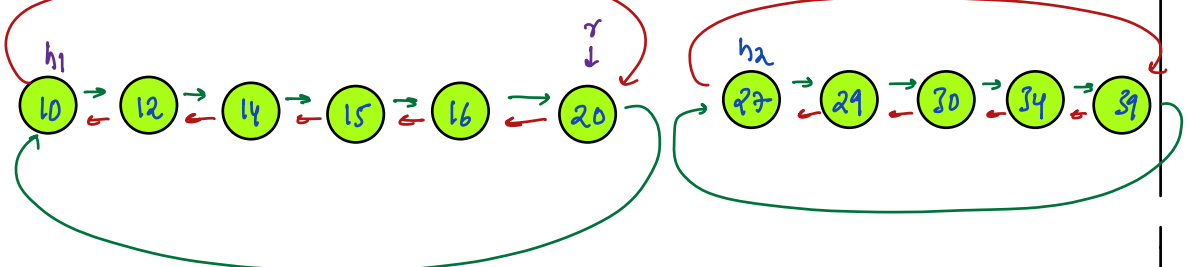


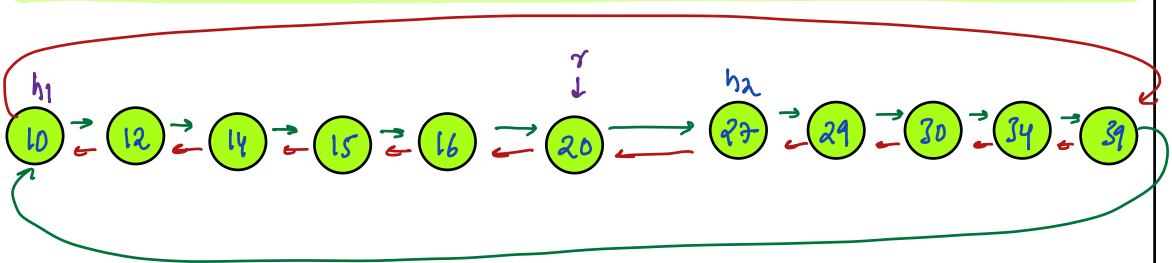


Ass: Given root of BST, convert into CDLL & return head node.

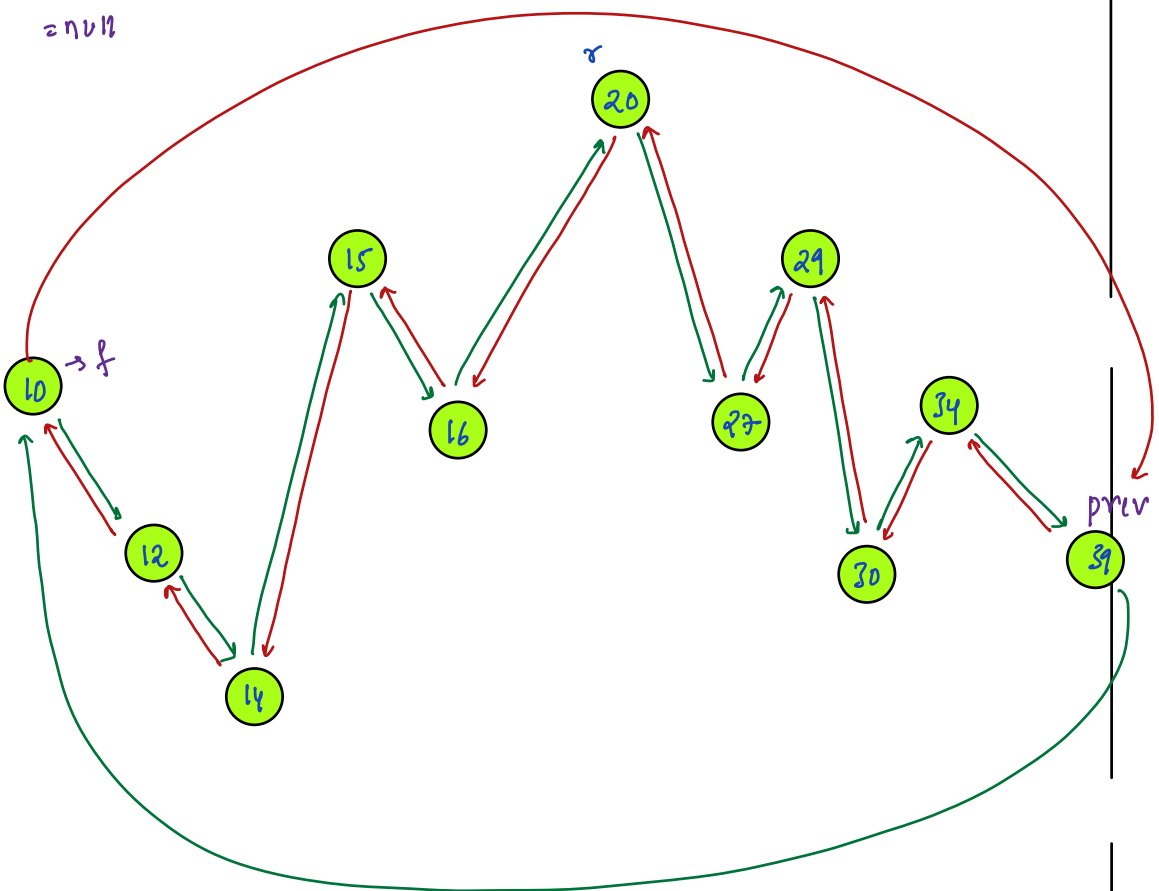
```
Node BST2CDLL(Node root){ TC: O(N) SC: O(H) // height of tree
    if(root == null){return null}
    Node h1 = BST2CDLL(root.left) // return head of linked list
    Node h2 = BST2CDLL(root.right) // return head of linked list
    root.left = root, root.right = root
    h1 = merge(h1, r) // head of merged
    h1 = merge(h1, h2) // head of merged
    return h1
}
```







→ Inorder traversal :



```
Node BST2CDLL Iterative (Node root) { TC:O(N) SC:O(1)
    Node curr = root
    Node prev = null
    while (curr != null) {
        if (curr.left == null) { // Inorder left data right
            print(curr.data) ⟶ curr.left = prev
                               if (prev != null) {
            curr = curr.right      prev.right = curr
        }                      }
                               prev = curr
        else {
            Node t = curr.left
            while (t.right != null && t.right != curr) {
                t = t.right
            }
            if (t.right == null) { // curr we visited 1st time
                t.right = curr      left node right
                curr = curr.left    1st
            }
            else // curr we visited 2nd time left node right
                 //                          2nd
                print(curr.data) ⟶ curr.left = prev
                curr = curr.right  if (prev != null) {
                t.right = null         prev.right = curr
                          // deleting link we created
            }                      prev = curr
        }
    }
    Node f = prev
    while (f.left != null) { f = f.left }
    f.left = prev, prev.right = f
    return f
}
```