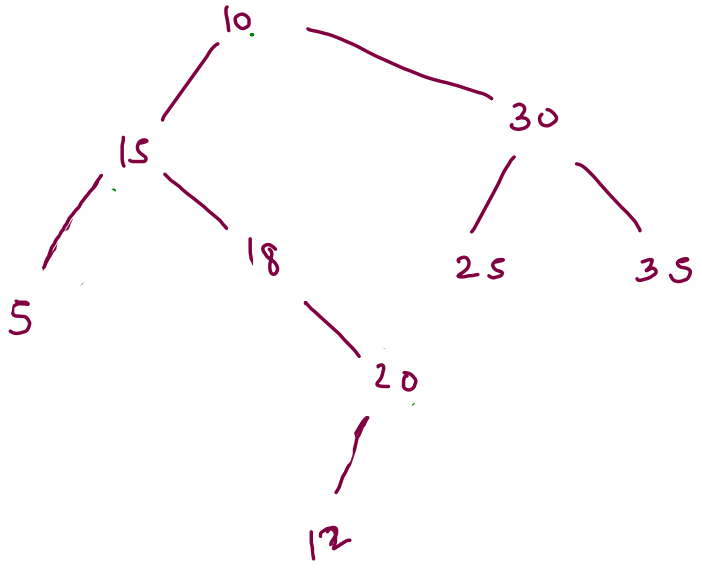
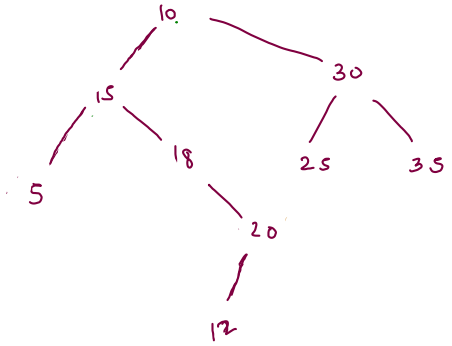
Morris traversal : In-order traversal of a binary tree in O(1) space & O(n) time.




In : 5 15 18 12 20 10 25 30 35




5 15 18 12 20 10 25 30 35

```
curr = root;
while (curr != null) {
    Node ln = curr.left;
    if (ln == null) {
        syso(curr.data);
        curr = curr.right;
    }
    else {
        Node rmn = rightmostnode(ln, curr);
        if (rmn.right == null) {
            rmn.right = curr;
            curr = curr.left;
        }
        else {
            rmn.right = null;
            syso(curr.data);
            curr = curr.right;
        }
    }
}
```

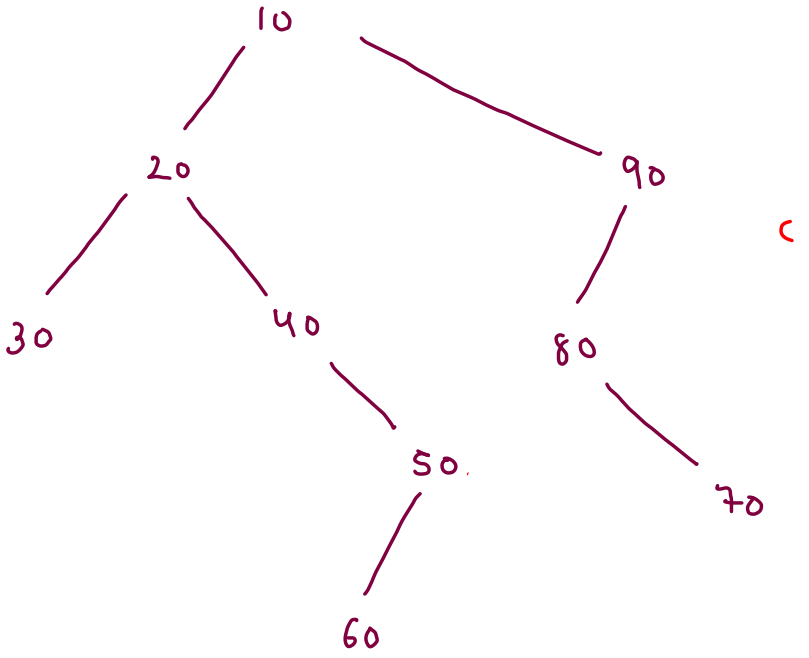
left subtree is not visited

left subtree is visited

```
Node temp = ln;
while (temp.right != null && temp.right != curr) {
    temp = temp.right;
}
```

10 20 30 40 50 60 90 80 70

```java
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer>ans = new ArrayList<>();

    TreeNode curr = root;

    while(curr != null) {
        TreeNode ln = curr.left;

        if(ln == null) {
            ans.add(curr.val);
            curr = curr.right;
        }
        else {
            TreeNode rmn = rightMostNode(ln,curr);

            if(rmn.right == null) {
                //left subtree is unvisited
                ans.add(curr.val);
                rmn.right = curr;
                curr = curr.left;
            }
            else {
                //left subtree is visited
                rmn.right = null;
                curr = curr.right;
            }
        }
    }

    return ans;
}
```



LRN : postorder

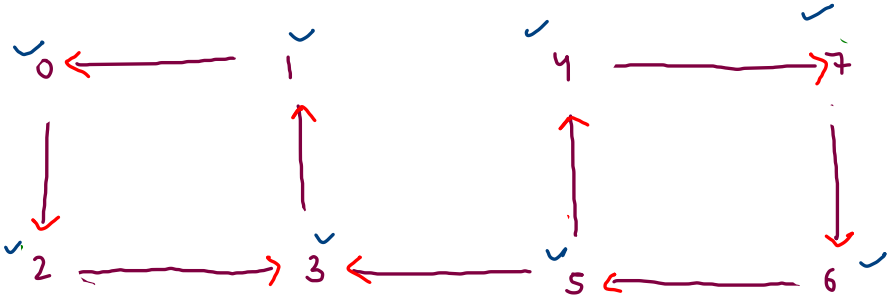
reverse euler preorder : NRL

using morris

( left → right
right → left )

↳ reverse ⟹ LRN (regular euler postorder)

# Kosaraju: strongly connected components

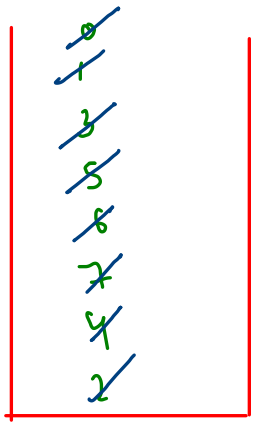


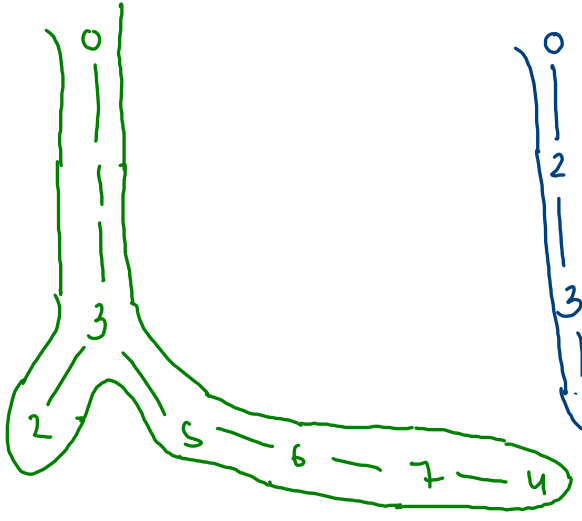

what :

(i) dfs and push in post-order in the stack.

(ii) reverse the graph (edges)

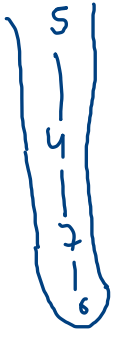
(iii) now apply dfs using stack's order, no. of dfs = no. of strongly conn. compt.









what :

(i) dfs and push in post-order in the stack.

(ii) reverse the graph (edges)

(iii) now apply dfs using stack's order, no. of dfs = no. of Strongly conn. compt.