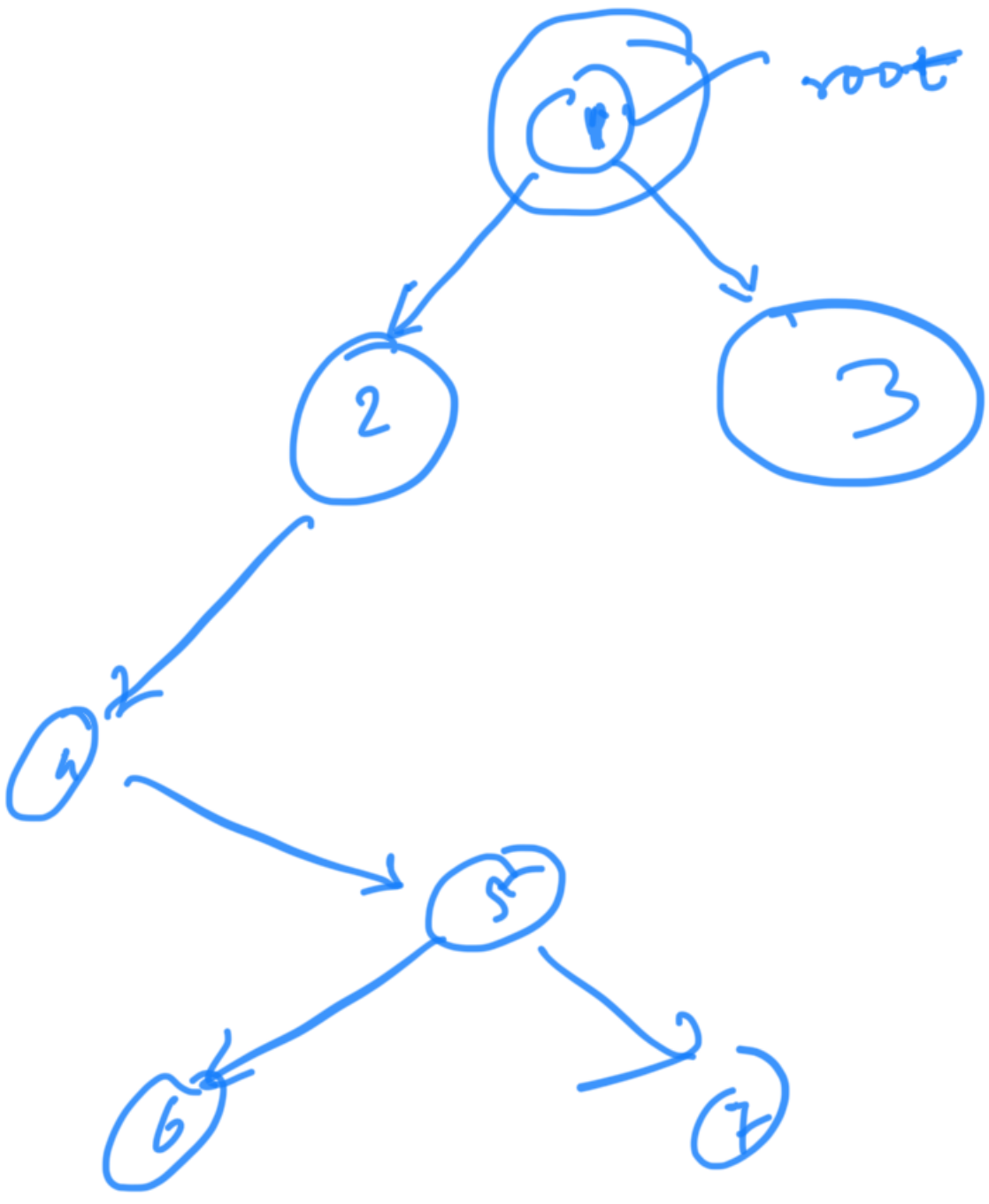
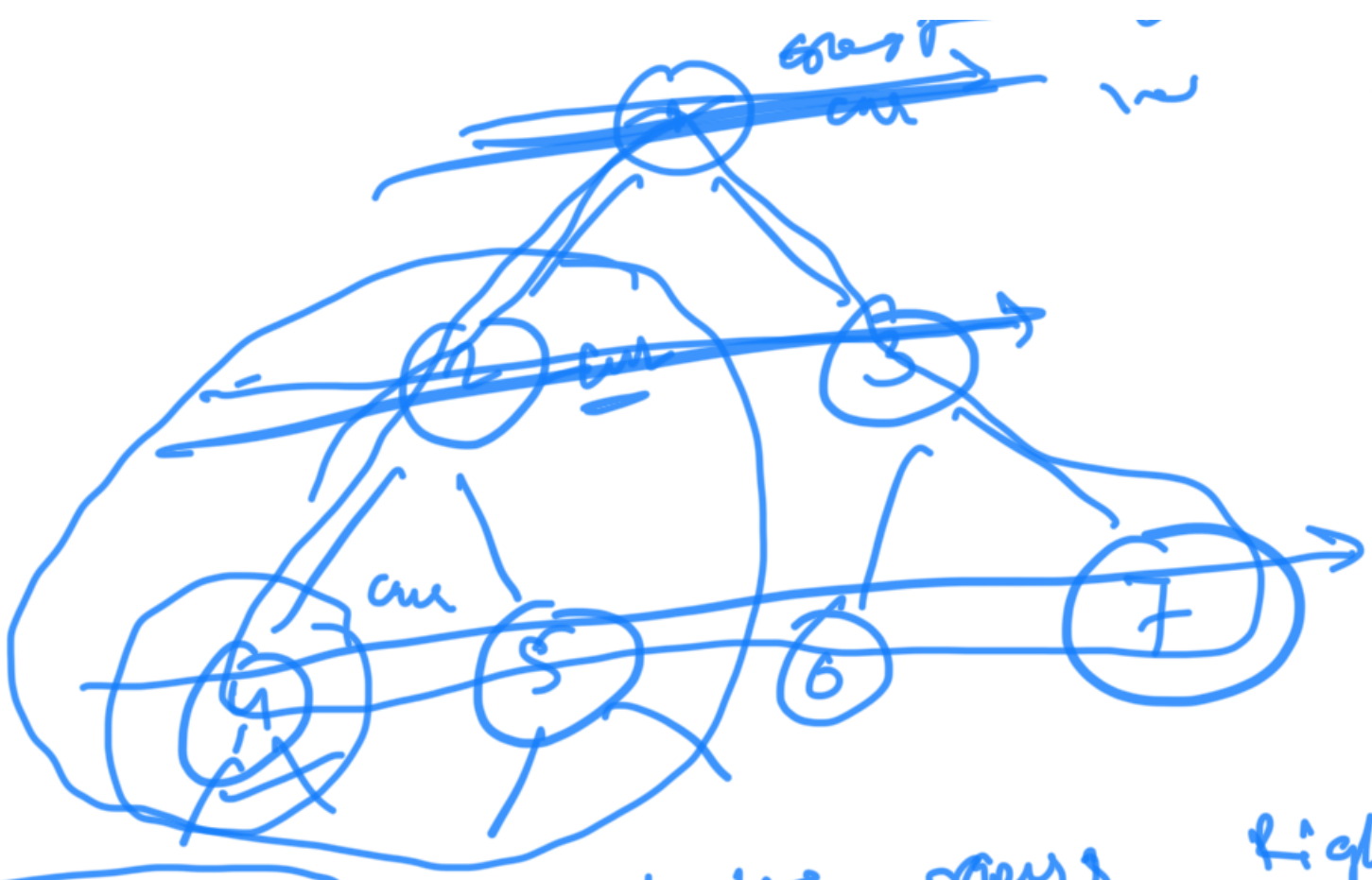
Tree

Binary Tree




```
Class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
}
```



4, 2, 5, 1,

Inorder → Left curr Right

Pre order → curr left Right

Post order → left Right, curr

level order → level wise

Right curr left

Reverse Inorder

Reverse level order

Boundry traversal +

Zig zag traversal
Reverse post order — Right left ans

```
void inorder ( TreeNode root )
{   if ( root == Null ) return;
    inorder ( root.left );   ——— 1
    process ( root );  →     ——— 2
    inorder ( root.right );  ——— 3
}
```

O(1)
O(h)

1, 2, 3 → 3!

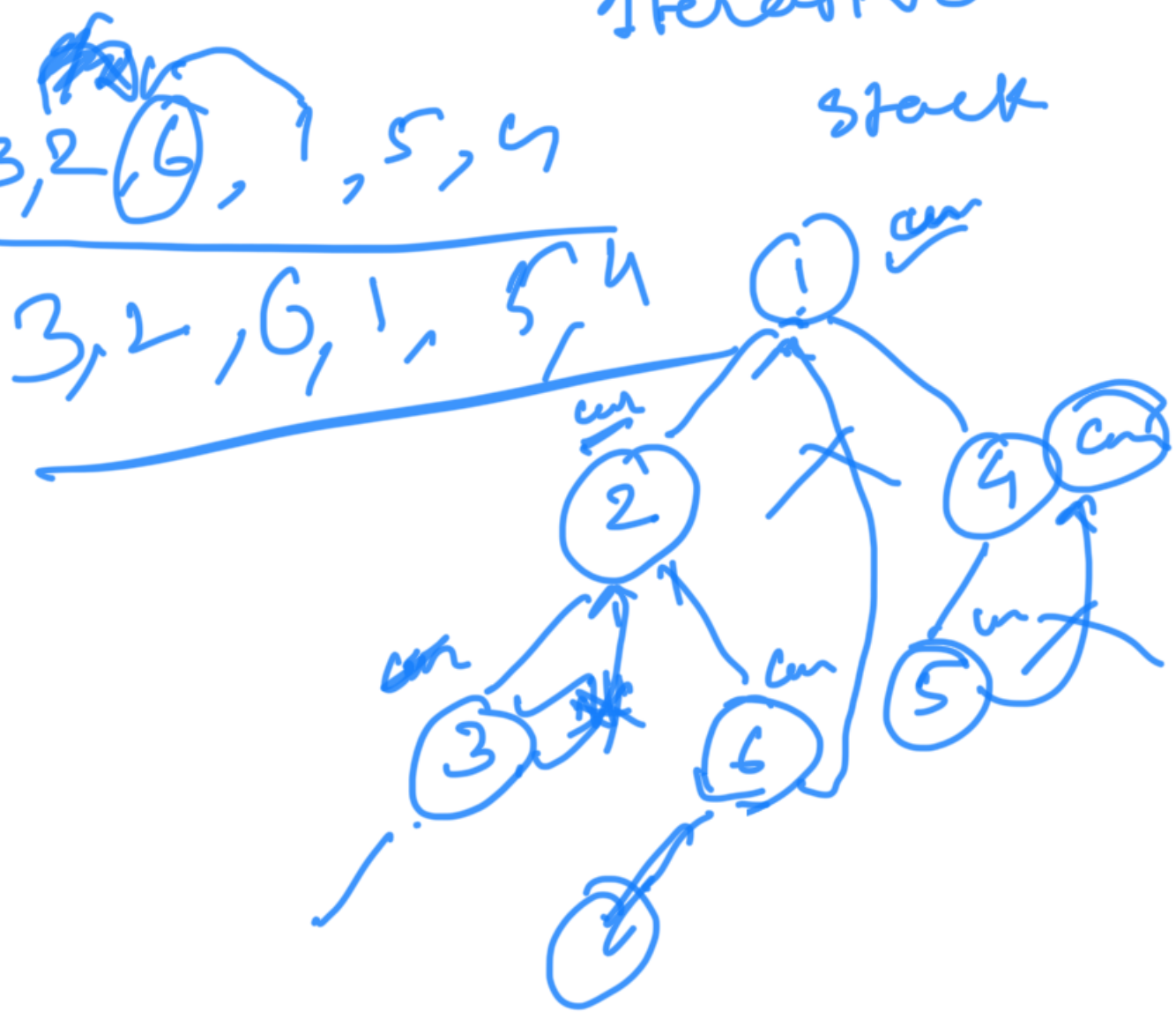
1 2 3
2 1 3
3 1 2

Iterative Version stack

Morris Inorder traversal

Iterative and does not need stack

3, 2, 6, 1, 5, 4

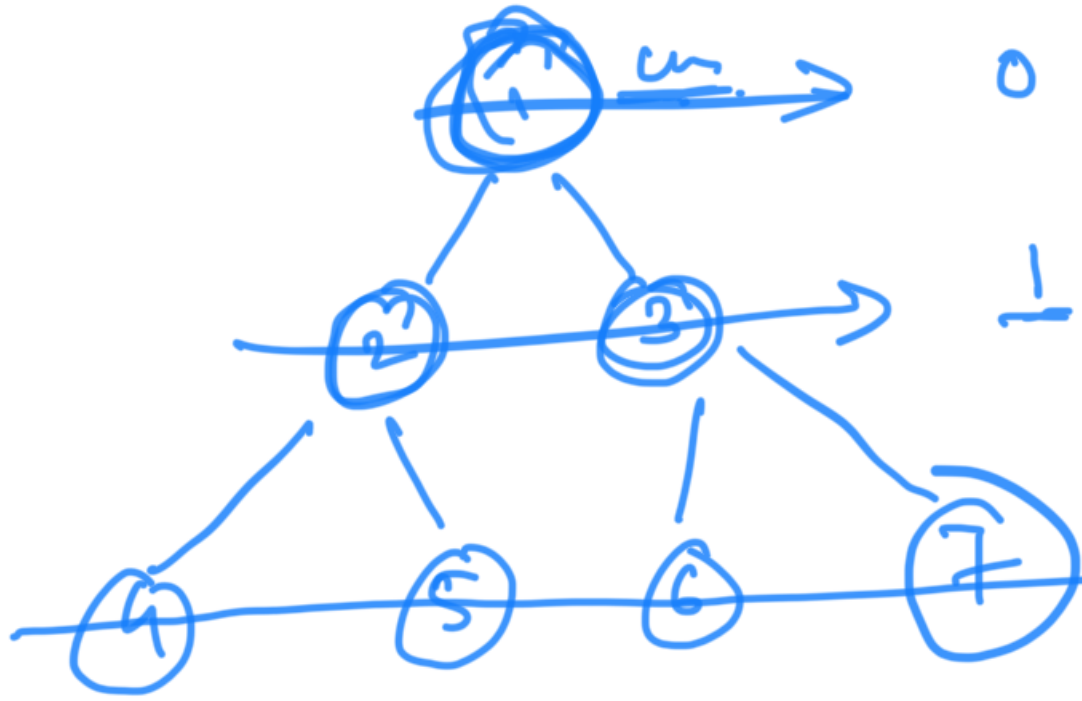


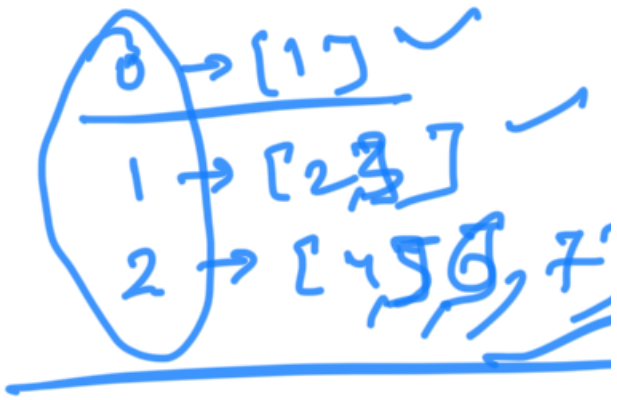
curr

if cur.left == Null
    Print cur

else
    find predecessor of current node
    pr-decessor.right = curr.

level order traversal





✓ HashMap < level, list < TreeNode >)

Queue

Q = [1]

Q = [2, 3]

Q = [3, 4, 5]

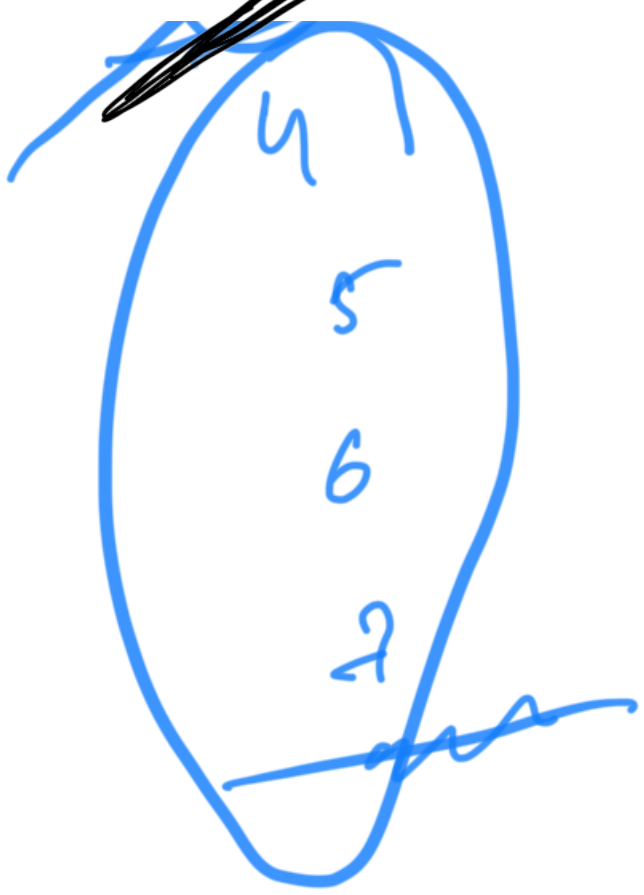
Q = [4, 5, 6, 7]

[1, #]

[2, 3, #]

[3, #, 4, 5]

[#, 4, 5, 6, 7]



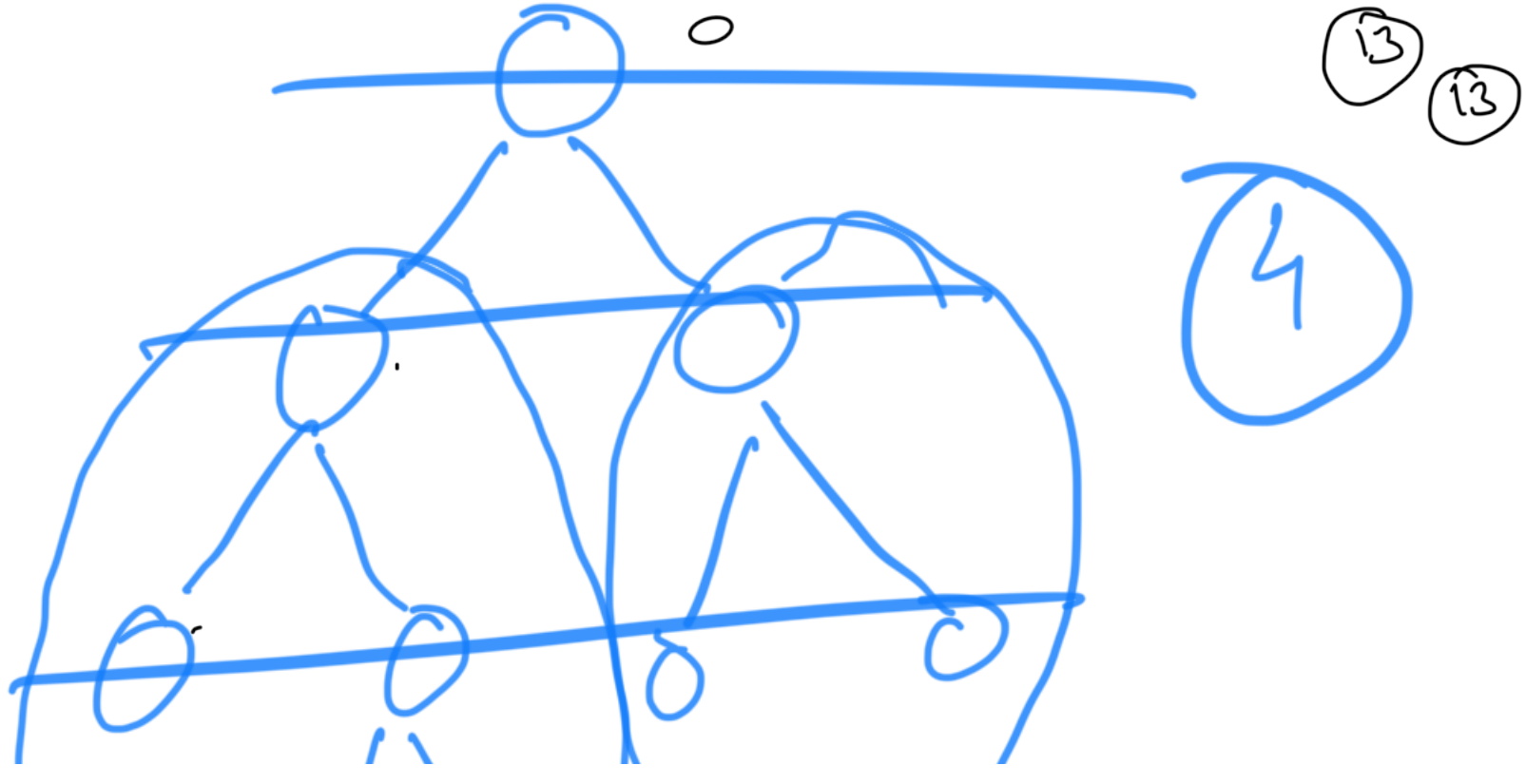
R = [5, 6, 7]

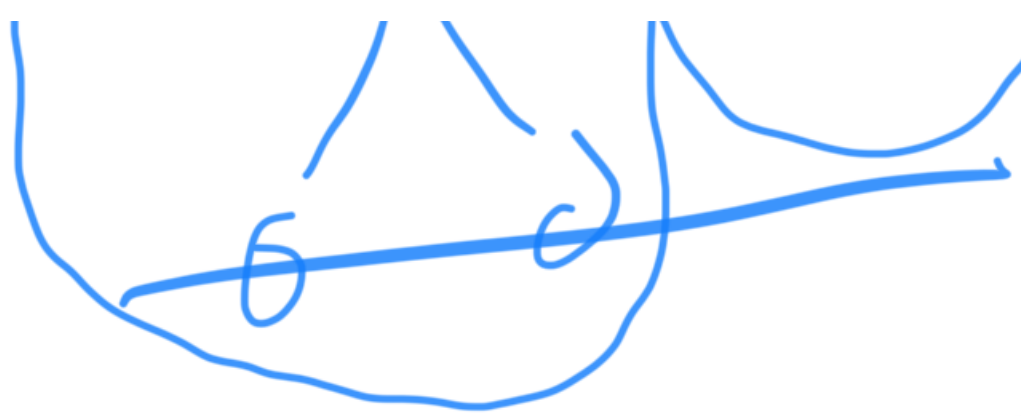
Q = [6, 7]

R = [7]

Q = {}

Height of Tree



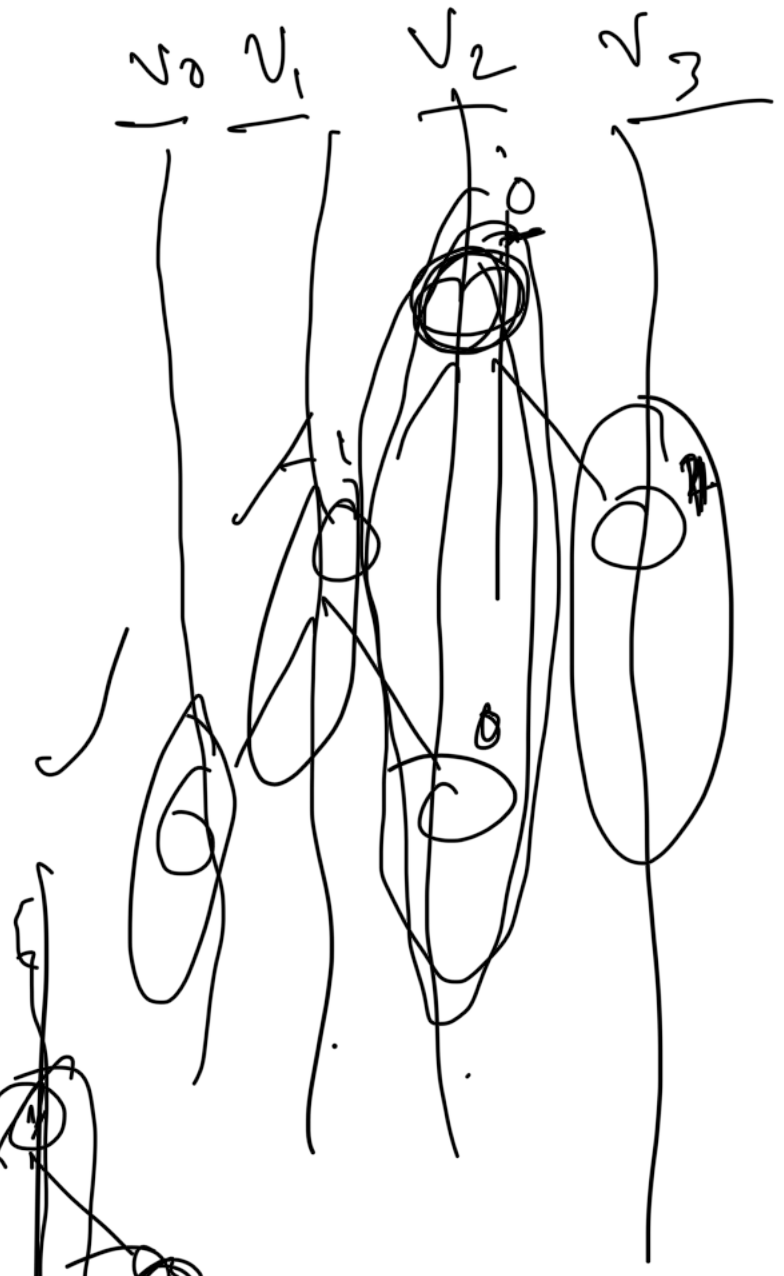


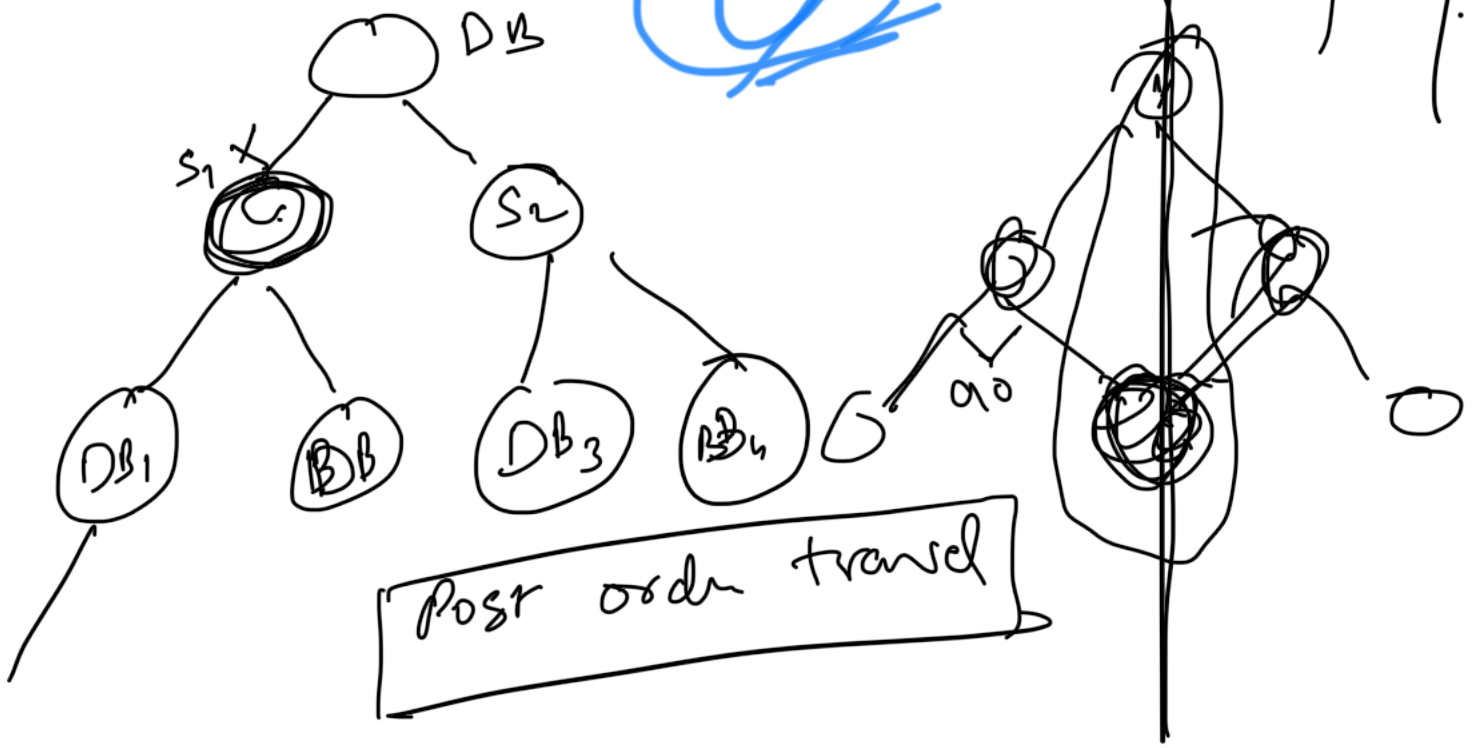
```
int height (TreeNode root)
{
    if (root == null)
        return 0;

    return Math.max (height(root.left),
                     height(root.right));
}
```

LCA least common ancestor


1
3
5
6
7
8


V0
V1
V2
V3


DB
S1
S2
DB1
BB
DB3
BB4
Post order travel