# Q.1 LCA (Lowest Common Ancestor) Binary Tree.

Ancestors(3) = 3, 5, 10, 15

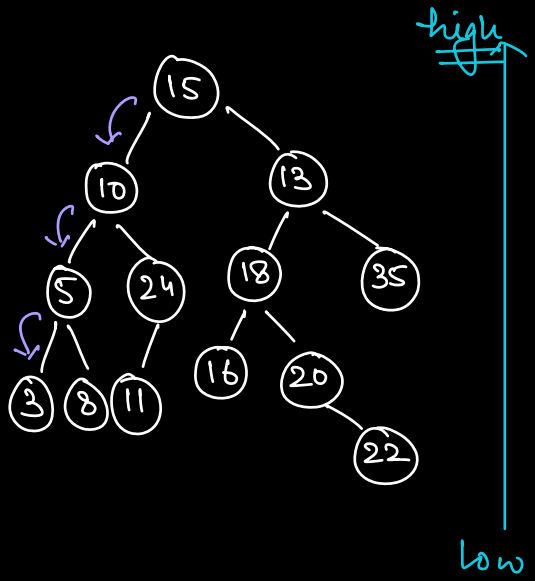
Ancestors(22) = 22, 20, 18, 13, 15

Ancestors(18) = 18, 13, 15

Ancestors(15) = 15

Ancestors(24) = 24, 10, 15

Ancestors(35) = 35, 13, 15




LCA(n1, n2)

LCA(3, 22) = 15

LCA(35, 3) = 15

LCA(22, 35) = 13

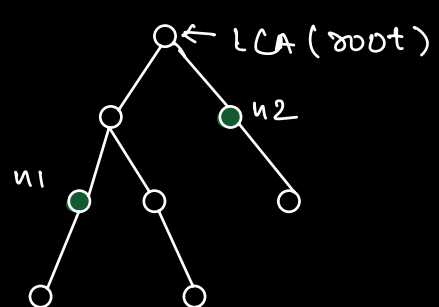
LCA(22, 18) = 18

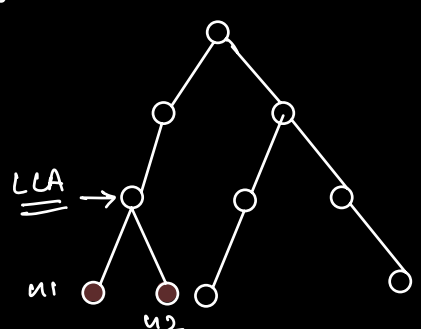
* Assumption:

→ Both the nodes are present in the Tree.

→ No duplicates.

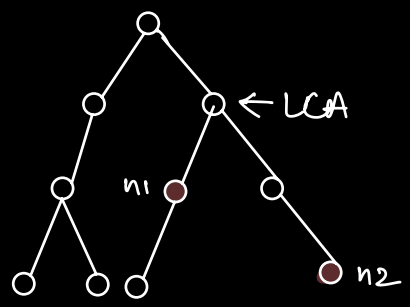
**LCA**: Find the first common node in the paths from the given nodes to the root.

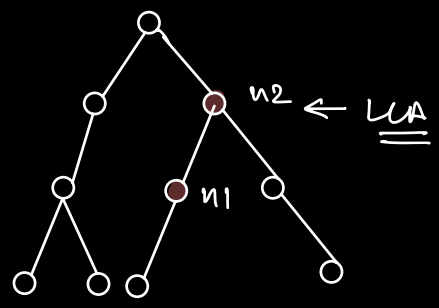


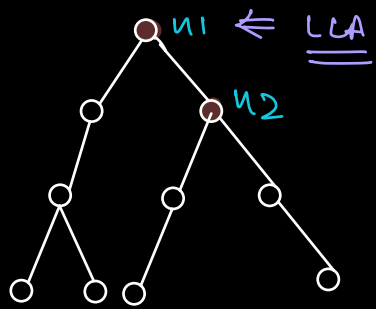

⇒ LCA(root, n1, n2)

= LCA(root.left, n1, n2)




⇒ LCA(root, n1, n2)
= LCA(root.right, n1, n2)

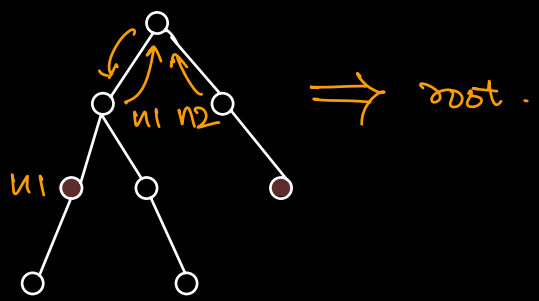



If one of the node (n1 or n2) is root, then root will be LCA.

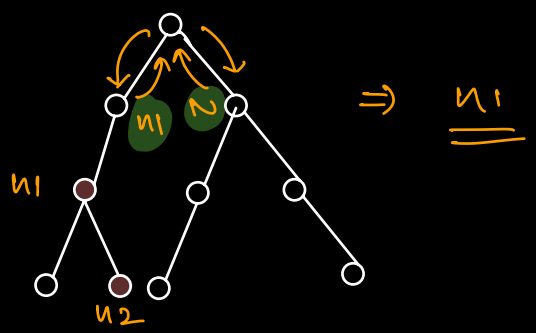
```
TreeNode lca(root, n1, n2) {
    if (root == Null) return null;
    if (root == n1 || root == n2) return root;
O(N)⇐ bool n1present = find(root.left, n1);
O(N)⇐ bool n2present = find(root.left, n2);
    if (n1present && n2present)
        return lca(root.left, n1, n2);
    if (n1present || n2present)
        return root;
    return lca(root.right, n1, n2);
}
```

3

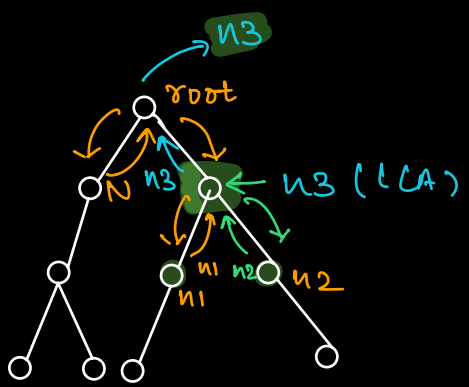
TC : $O(N^2)$




$\Rightarrow$ root.




$\Rightarrow$ n1

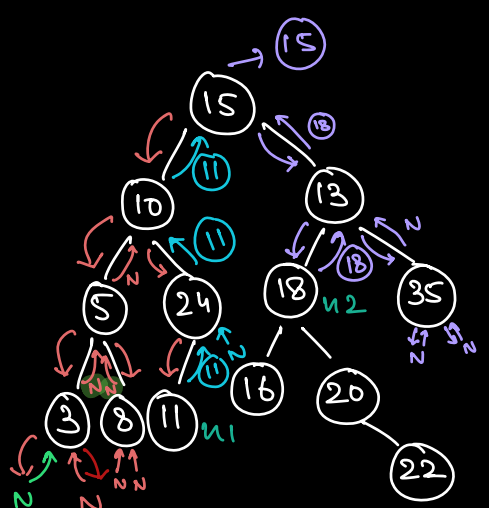


```
TreeNode lca(root, n1, n2) {
    if (root == NULL) return null;
    if (root == n1 || root == n2) return root;
    TreeNode Llca = lca(root.left, n1, n2);
    TreeNode Rlca = lca(root.right, n1, n2);
    if (Llca != Null && Rlca != Null)
        return root;
    else if (LLCA == NULL)
        return Rlca;
    else
        return Llca;
}
```

TC : O(N)
SC : O(N)

⇒ Postorder traversal

lca(root, 11, 18)

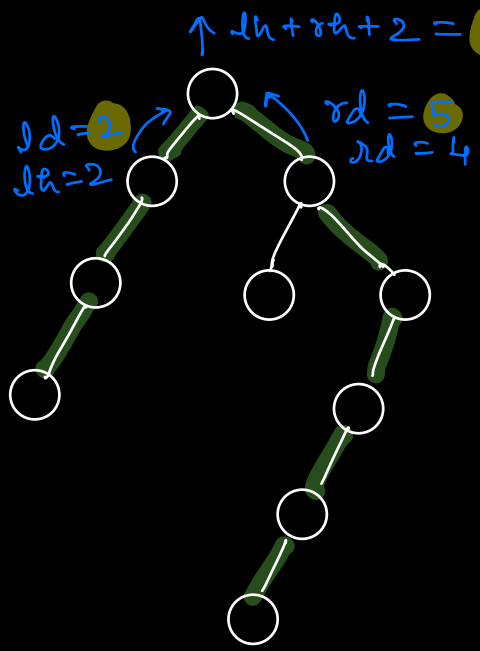


Q.2 Given a B.T, find its diameter.

Amazon
MS|GS

Length of the longest path b/w any 2 nodes in the tree.

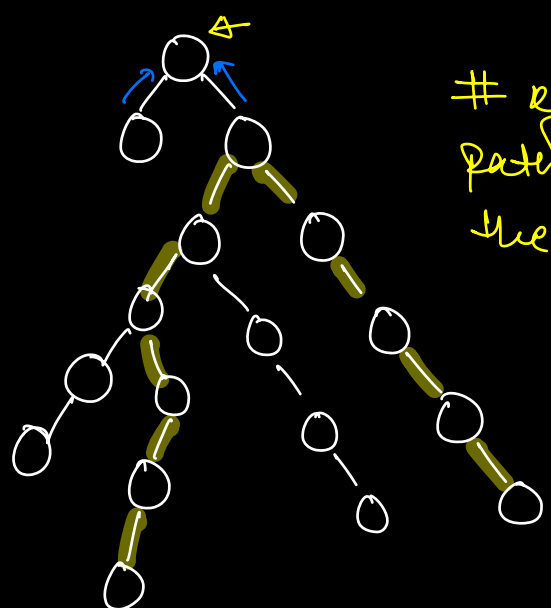
(# of edges)




⇒ (8)

⇒ 8

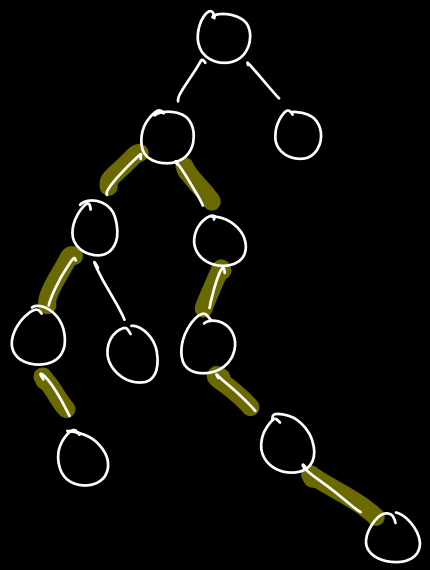
(lh + rh + 2)



# of Edges in the longest path b/w any 2 nodes in the tree = 9

9

Diameter.

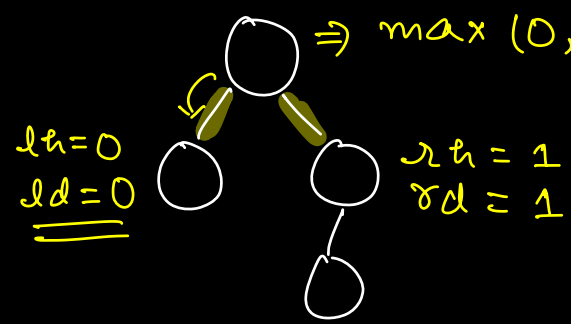


$\Rightarrow 7$

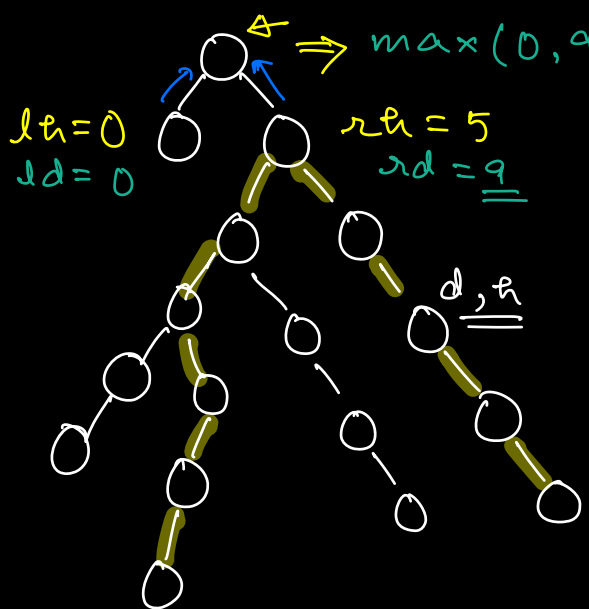
Diameter can

1. be present in LST
2. be present in RST
3. pass through root node.

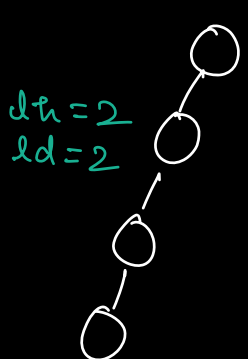
} Max



```
int diameter(root) {
    if(root == NULL) return -1;
    int lh = height(root.left);
    int rh = height(root.right);
    int ld = diameter(root.left);
    int rd = diameter(root.right);
    return max(ld, rd, lh+rh+2);
}
```




$\Rightarrow max(0, 9, 7) = 9$




$\Rightarrow max(2, -1, 2+(-1)+2) = 3$

TC : $O(N^2)$ , SC : $O(N)$

```
Class TreeInfo {
    int ht;
    int dia;
}
```

3

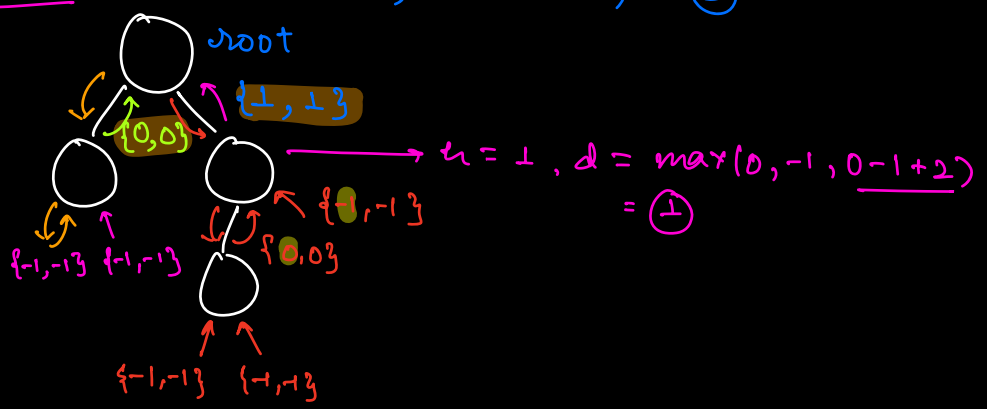
```
TreeInfo diameter(root) {
    if (root == NULL)
        return new TreeInfo(-1, -1);

    TreeInfo l = diameter(root.left);
    TreeInfo r = diameter(root.right);

    return new TreeInfo(max(l.ht, r.ht) + 1,
        max(l.dia, r.dia,
            l.ht + r.ht + 2));
}
```

3

{2,3}

h = max(0,1) + 1 = ②

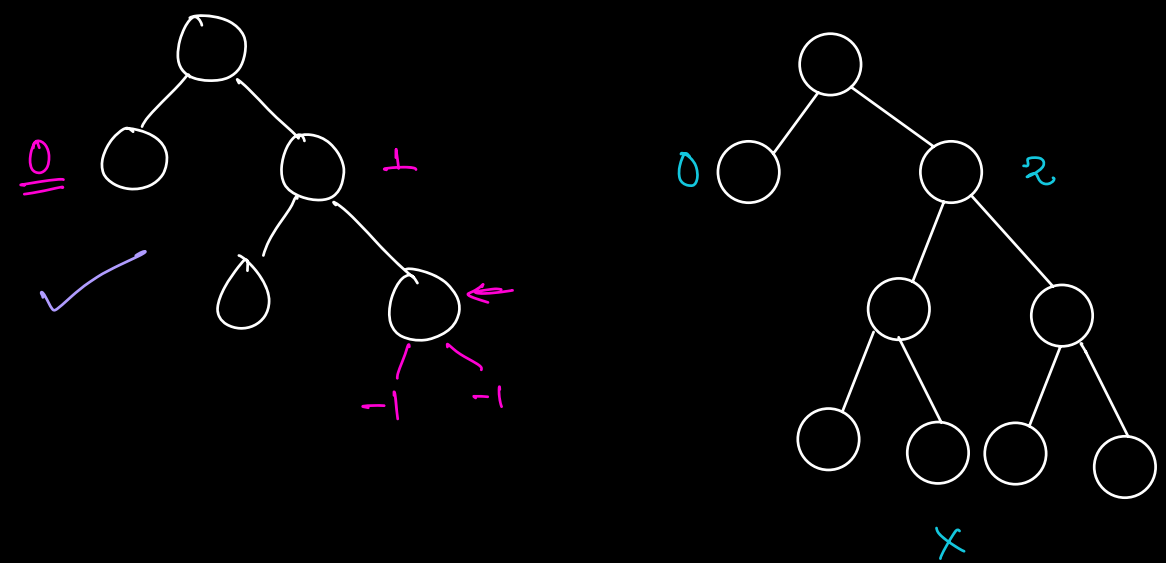
↑ d = max(0, 1, 0+1+2) = ③



# Height Balanced Tree

$$|\ ht(LST) - ht(RST)\ | <= 1$$

→ This should hold true for all the nodes.




- Height of a Height Balanced Tree ≈ Log N
- TC of search Height Balanced Binary Tree ⇒ O(N) *
- TC of search Height Balanced BST ⇒ O(log N)

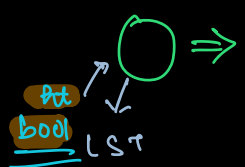
Balanced BST ⇒ Self Balancing BST
(AVL | Red Black Tree)

Ordered HashMap / TreeMap.

Q.8 Given a B-T, check if it is Height Balanced or not.

```
bool isBalanced(root) {
    if (root == NULL)
        return true;
    int lh = height(root.left);  → O(N)
    int rh = height(root.right); → O(N)
    if (abs(lh - rh) > 1)
        return false;
    else {
        return isBalanced(root.left)
                    &&
               isBalanced(root.right);
    }
}
```

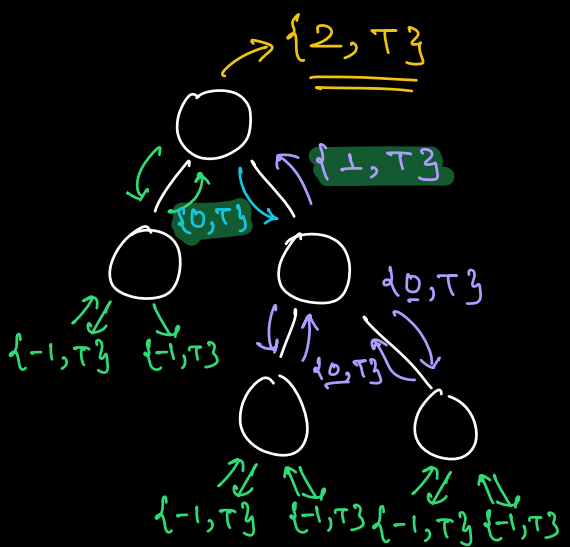
TC : $O(N^2)$
SC : $O(N)$

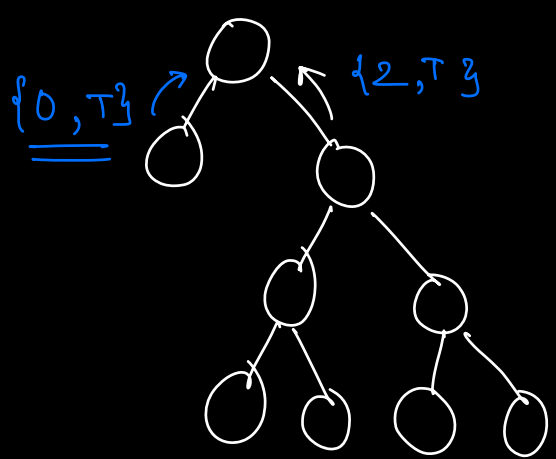


```
class TreeInfo {
    int ht;
    bool isBal;
}

TreeInfo isBalanced(root) {
    if (root == NULL)
        return new TreeInfo(-1, true);
    TreeInfo l = isBalanced(root.left);
    TreeInfo r = isBalanced(root.right);
    if (l.isBal && r.isBal &&
            abs(l.ht - r.ht) <= 1) {
        return new TreeInfo(max(lh, rh)+1,
                            true);
    }
    else {
        return new TreeInfo(max(lh, rh)+1,
                            false);
    }
}
```

{ TC: O(N) {Post order traversal}
  SC: O(N)

```
TreeInfo result = isBalanced(root)
if (result.isBal) print("Tree is Balanced")
```


{2, T}
{1, T}
{0, T}
{0, T}
{-1, T}
{-1, T}
{0, T}
{-1, T}
{-1, T}
{-1, T}
{-1, T}


{0, T}
{2, T}