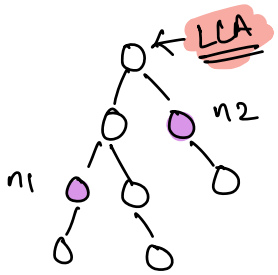
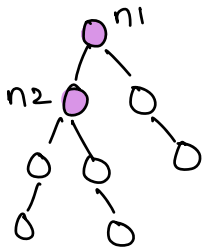
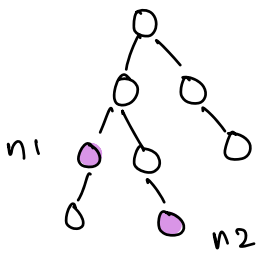


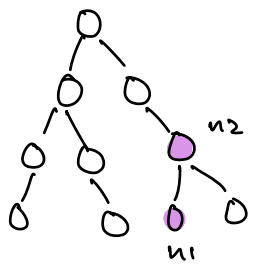


LCA = n1



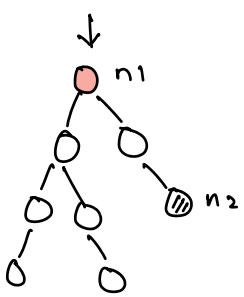
LCA(root, n1, n2) =

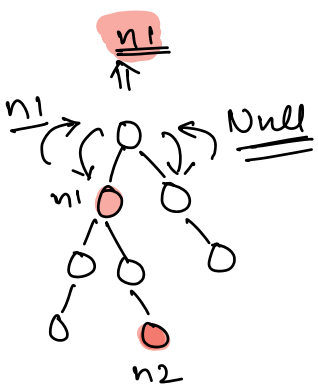
LCA(root.left, n1, n2)



LCA(root, n1, n2) =

LCA(root.right, n1, n2)

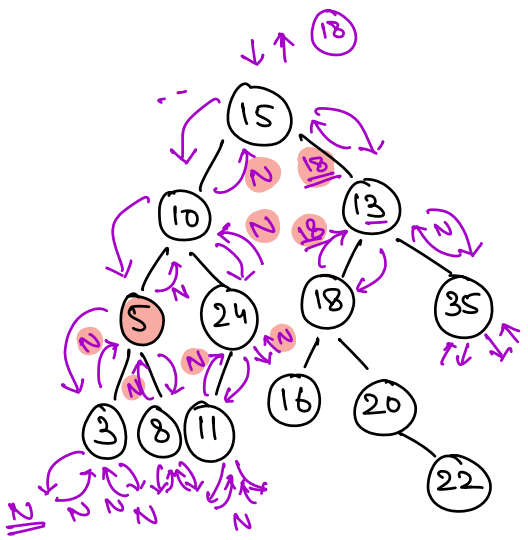




```
TreeNode LCA (root, n1, n2) {
    if (root == Null)
        return null;
    if (root == n1 || root == n2)
        return root;

    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;
```

3

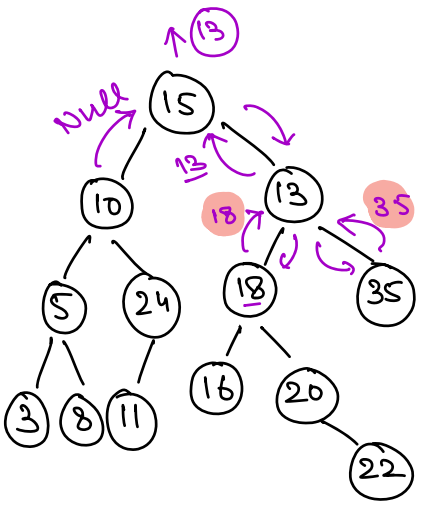


n1 n2

LCA(18, 16) = 18

TC: O(N)

SC: O(N)



LCA (18, 35) = 13

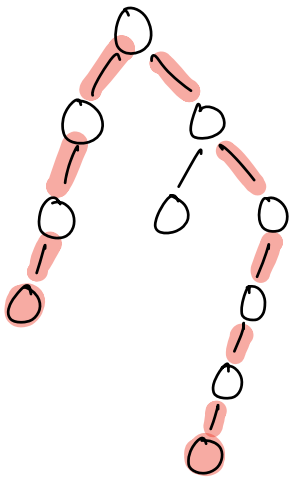
HW :- Try this any nodes are NOT present in the tree.

Q. Given a Binary Tree, find it's diameter.

MS, Amazon, Atlesium, Adobe.

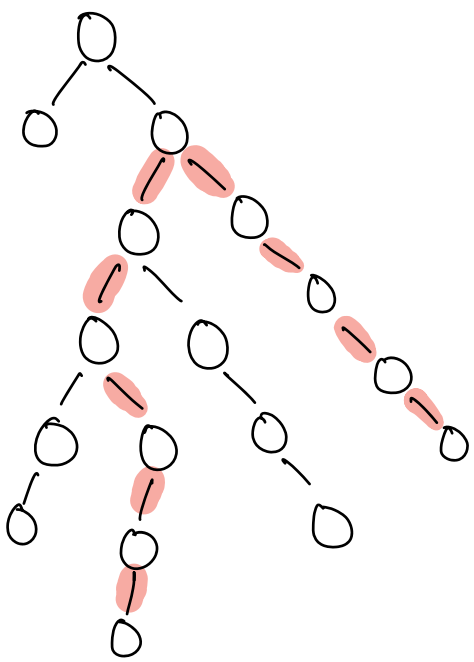
↓

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

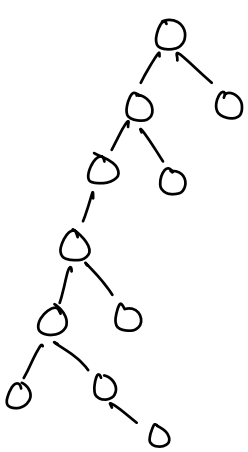


⇒ (8) (Edges)

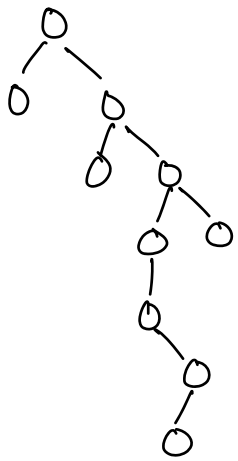
⇒ lh + rh + 2
(2) + (4)



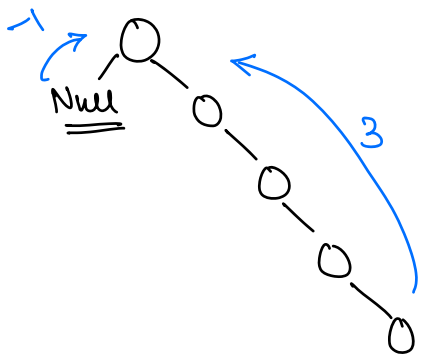
⇒ (9)



⇒ dia(root)
=
dia(root.left)



⇒ dia(root)
=
dia(root.right)



D = 4

-1 + 3 + 2 = 4

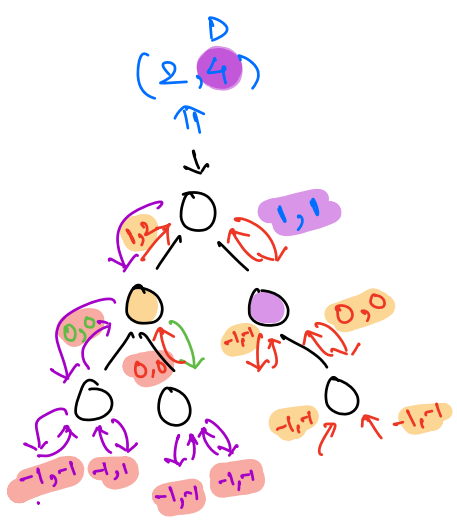
```
int   dia(root) {
      if(root == null)
          return -1;

O(N) ⇐ int lh = height(root.left);
O(N) ⇐ int rh = height(root.right);
      int ld = dia(root.left);
      int rd = dia(root.right);
      return max(ld, rd, lh+rh+2);
}
```

TC: $O(N^2)$

```
class TreeInfo {
    int ht;
    int dia;
    TreeInfo(h, d) {
        this.ht = h;
        this.dia = d;
    }
}
```

```
TreeInfo dia(root) {
    if(root == null) {
        return new TreeInfo(-1, -1);
    }
    TreeInfo l = dia(root.left);
    TreeInfo r = dia(root.right);
    return new TreeInfo(max(l.ht, r.ht) + 1,
        max(l.dia, r.dia, l.ht + r.ht + 2));
}
```



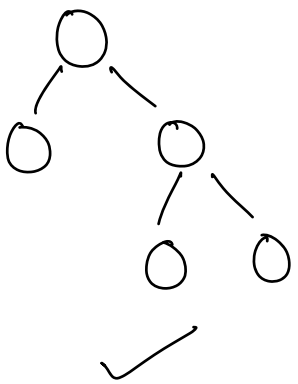
TC : O(N)
SC : O(N)

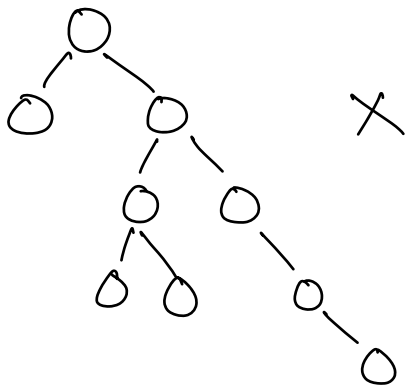
10:31 pm

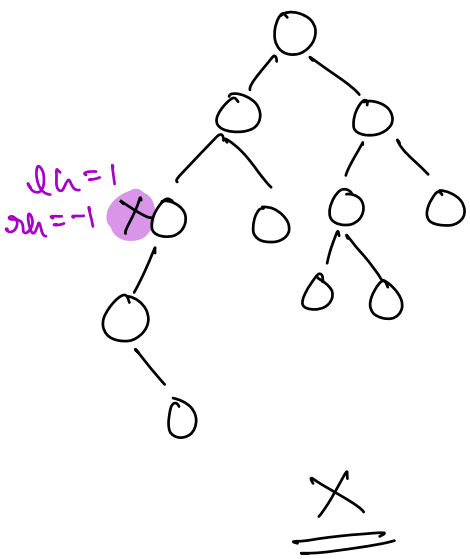
Height Balanced Tree

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

↳ for all the nodes in a tree.







X

Q. Given a Binary Tree, check if it is Height Balanced or not.

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

TC: $O(N^2)$

* Search TC in a Height Balanced B.T = $O(N)$

Ht (Height Balanced B.T) ≈ $O(\log N)$

* Height Balanced BST ≡ Balanced BST

$O(\log N)$

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

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

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

*