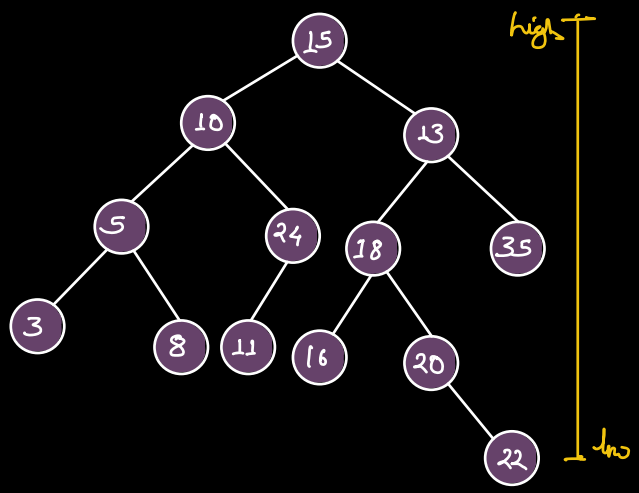
LCA : Lowest common ancestor

An(3) ⇒ 3, 5, 10, 15

An(22) ⇒ 22, 20, 18, 13, 15

An(18) ⇒ 18, 13, 15

An(24) ⇒ 24, 10, 15



Given 2 nodes in a BT. Return their LCA.

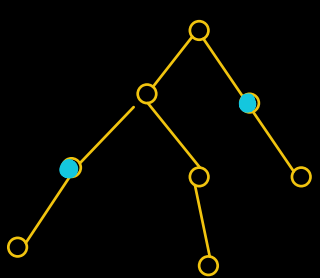
Both nodes are always present in the tree.

LCA(3, 22) ⇒ 15

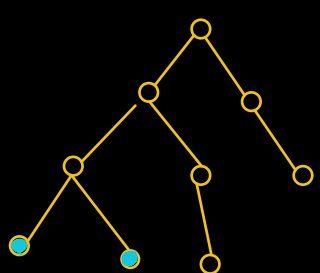
LCA(22, 18) ⇒ 18

LCA(3, 24) ⇒ 10

LCA(root, n_1, n_2)

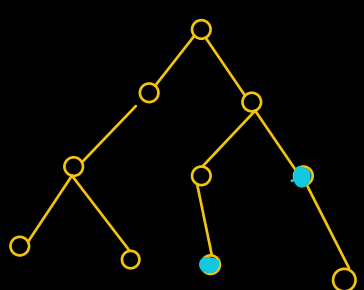


Root



LCA(root, n_1, n_2)

⇒ LCA(root.left, n_1, n_2)



LCA(root, n_1, n_2)

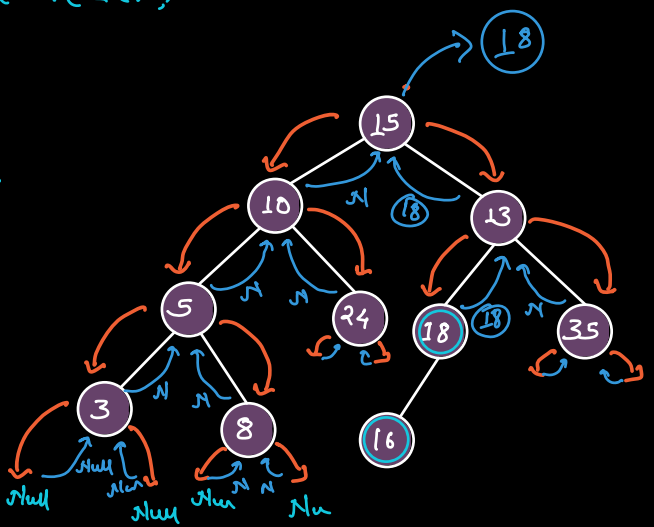
⇒ LCA(root.right, n_1, n_2

```
TreeNode LCA(root, n1, n2) {
    if (root == n1 || root == n2)
        ret root;

    boolean n1Present = find(root.left, n1); => O(N)
    boolean n2Present = find(root.left, n2); => O(N)
    if (n1Present && n2Present) {
        ret LCA(root.left, n1, n2);
    }
    if (n1Present || n2Present) {
        ret root;
    }
    else {
        ret LCA(root.right, n1, n2);
    }
}
```

TC: $O(N^2)$

```
TreeNode LCA (root, n1, n2) {

    if (root == null) {
            ret null;
    }
    if( root == n1 || root == n2) {
            ret root;
    }

    TreeNode lLCA = LCA(root.left, n1, n2);
    TreeNode rLCA = LCA(root.right, n1, n2);

(15) ⇒  if (lLCA != Null && rLCA != null) {
            ret root;
    }

    else if (lLCA == null) {
            ret rLCA;
    }

    else
            lLCA;

}
```

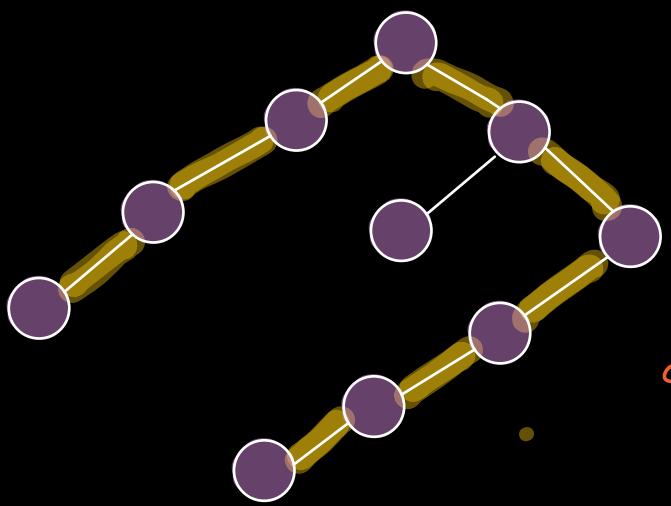
TC : O(N)
SC : O(N)



Amazon
MS
Adobe
⋮

Q Given a BT. Find the diameter of the tree.

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

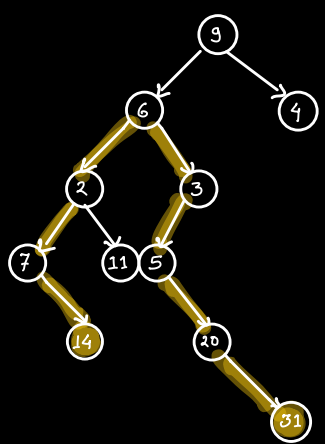


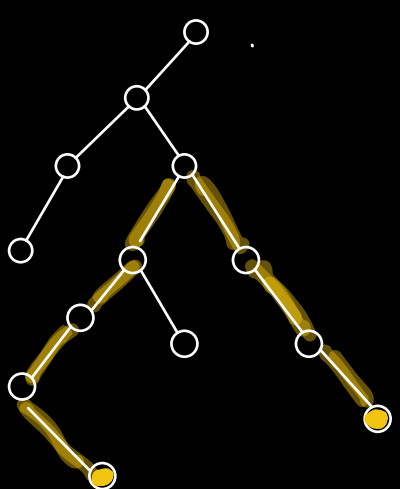
⇒ 8

dia = Ht(LST) + Ht(RST)

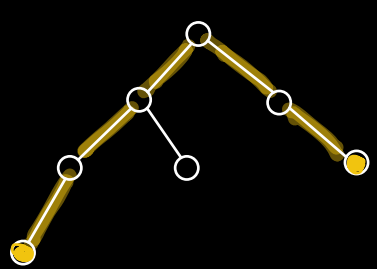
✗ Ht(LST) + Ht(RST) + 1

✓ Ht(LST) + Ht(RST) + 2

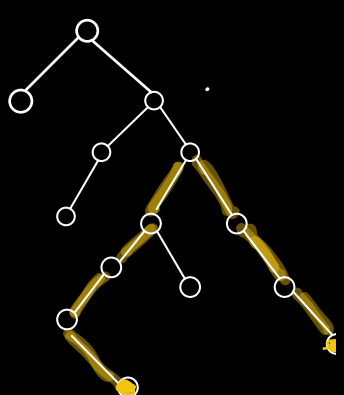




Dia (root.left)



Ht(LST) + Ht(RST) + 2



Dia (root.right)

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

    int lh = height(root.left); ⇒ O(N)
    int rh = height(root.right); ⇒ O(N)
    int ld = dia(root.left);
    int rd = dia(root.right);

    ret max(ld, rd, (lh+rh+2));
}
```

TC: $O(N^2)$

```
TreeInfo dia(root) {
    if(root == null)
        ret new TreeInfo(-1, -1);

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

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

height

dia

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

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

TC: $O(N)$

SC: $O(N)$

Height Balanced Tree

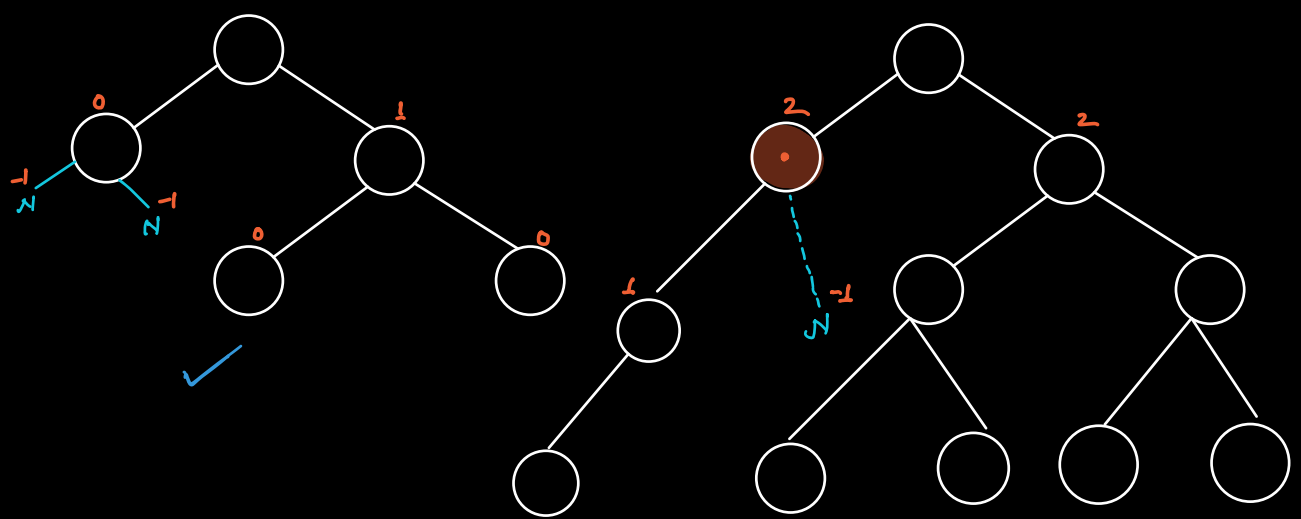
Ht(Height Balanced tree) ⇒ logN

Balanced BST

TC : O(logN)

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

This holds true for all the nodes in the tree.

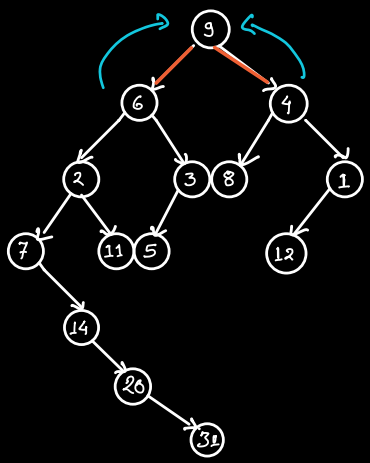



```
boolean isBalanced(root){
    if(root == null) ret true;
    int lh = height(root.left); ⇒ O(N)
    int rh = height(root.right); ⇒ O(N)
    if(abs(lh - rh) > 1)
        ret false;
    ret isBalanced(root.left)
        &&
        isBalanced(root.right);
}
```

TC : $O(N^2)$

```
TreeInfo isBal (root) {
    if (root == null) {
        ret new TreeInfo(true, -1);
    }

    TreeInfo Linfo = isBal (root.left);
    TreeInfo Rinfo = isBal (root.right);

    if ( Linfo.isBal && Rinfo.isBal
            && abs (Linfo.ht - Rinfo.ht) <= 1) {
        ret new TreeInfo (true, max (Linfo.ht, Rinfo.ht) + 1);
    }
    else {
        ret new TreeInfo (false, max (Linfo.ht, Rinfo.ht) + 1);
    }
}
```

```
class TreeInfo {
    boolean isBal;
    int ht;
    :
}
```


9
6
4
2
3
8
1
7
11
5
12
14
26
31