Diameter of Binary Tree

length of the longest path between any two nodes of a Binary Tree.

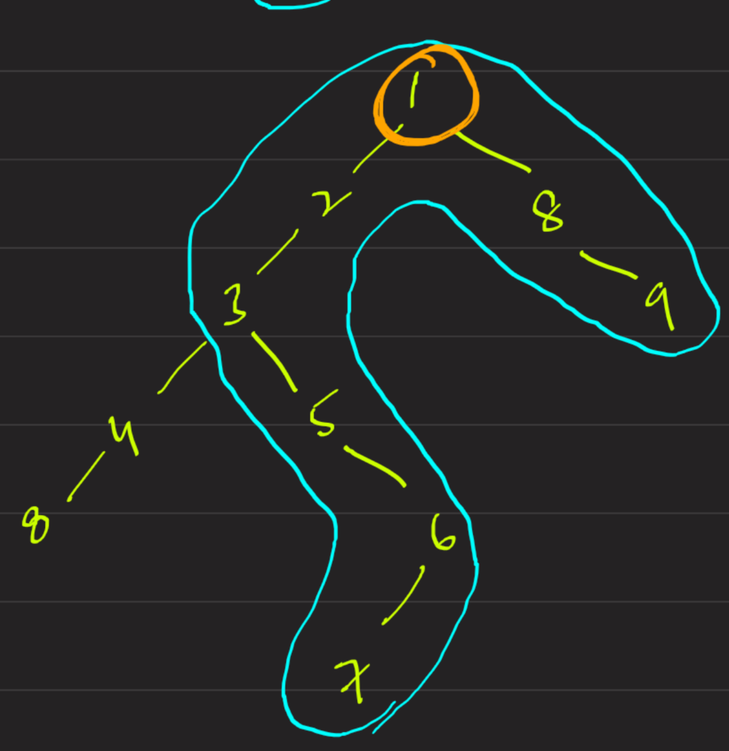
↓

This path may or may not follows root

Intuition

if you consider this tree,

7 Edges path is the longest diameter path
But it is not passing through root



if you consider this tree,

7 Edges path is the longest diameter path
which is passing through root

Observations:

① A BT, can have maximum path diameter by passing from root & not passing as well

So, from this we can conclude that,

→ for every current node

* we can have 3 diameters
  - left subtree's diameter (ld)
  - right subtree's diameter (rd)
  - diameter passing from root (d)
    ↓
    lets name it as self diam

How does diameter can be calculated ? → if you calculate ldiam, rdiam, selfdiam

So, if you do this no. of operations on each node, your will get (TLE)

So, Compute diameter when we are computing height itself

→ for selfdiam → we need left height + right height

for this parent node,
max height of leftSubtree is 4 (Node Count)

for this parent node,
max height of RightSubtree is 3

* from here, if we want to calculate diameter for (1) node

(leftheight + Right Height

lh = 4
rh = 3
diameter pass through root
⇒ lh + rh

×No. of edges passing through root = (lh + rh) → diameter

* While calculating the height, our idea simultaneously calculates the diameter.

* for each node → leftSubtree's diameter (which we hope recursive gives)

rightSubtree's diameter (")

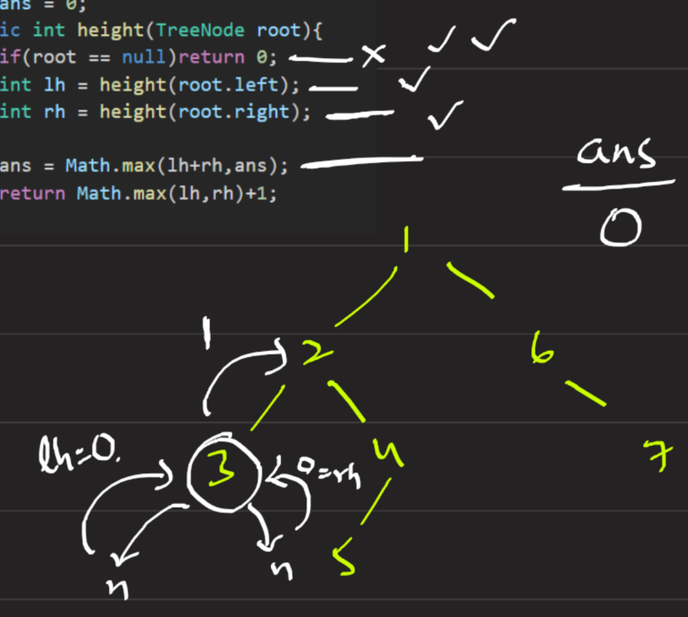
Selfdiam (leftheight + rightheight);

We update max of all diameters

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```


```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```

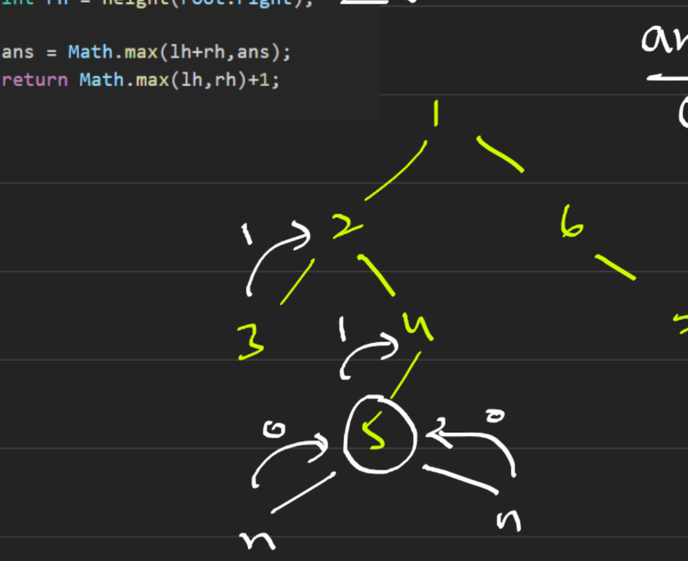

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



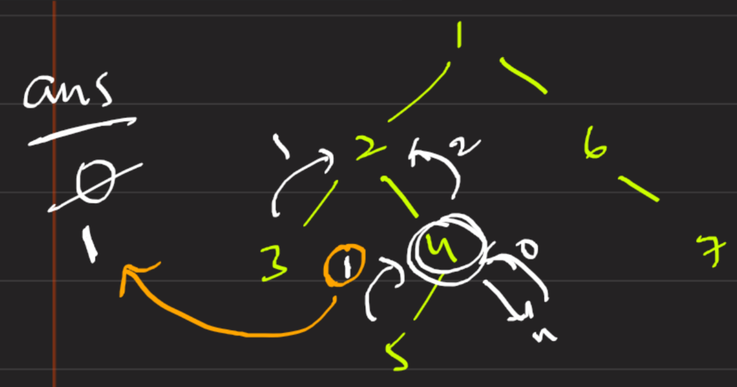

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```


```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```

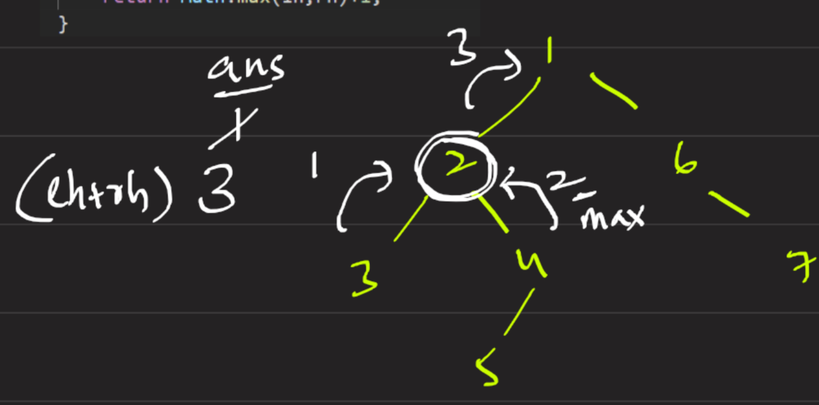

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



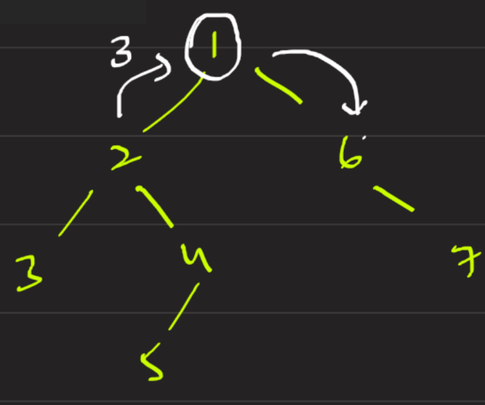

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



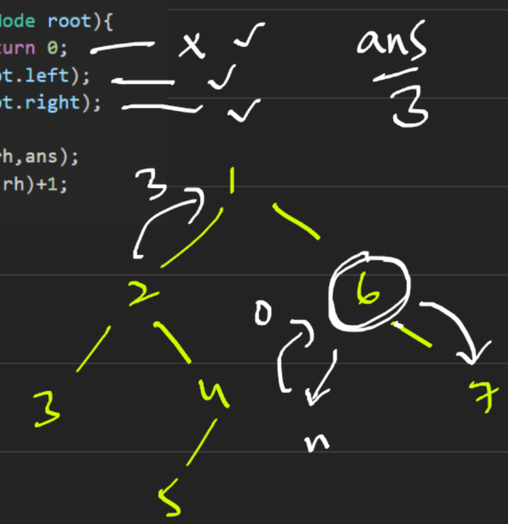

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



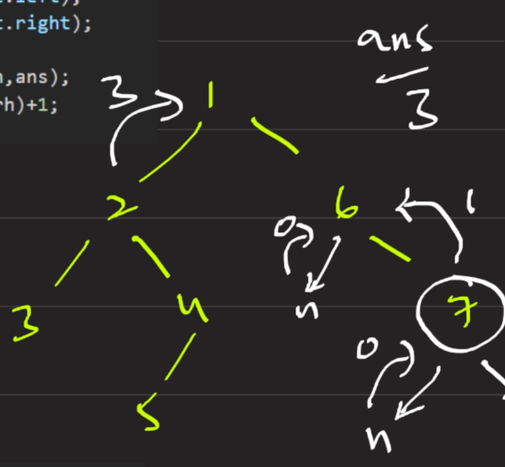

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



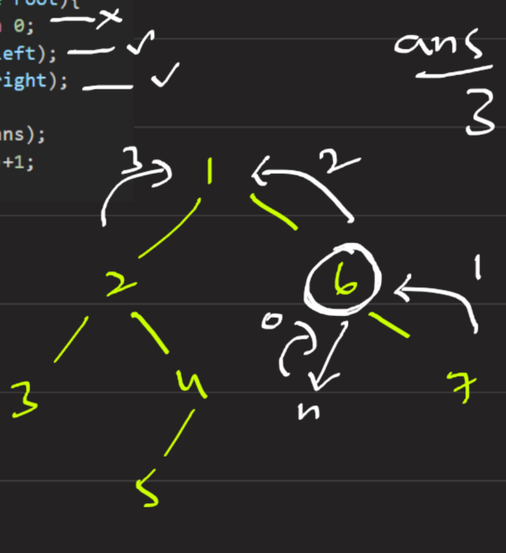

```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```




```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```

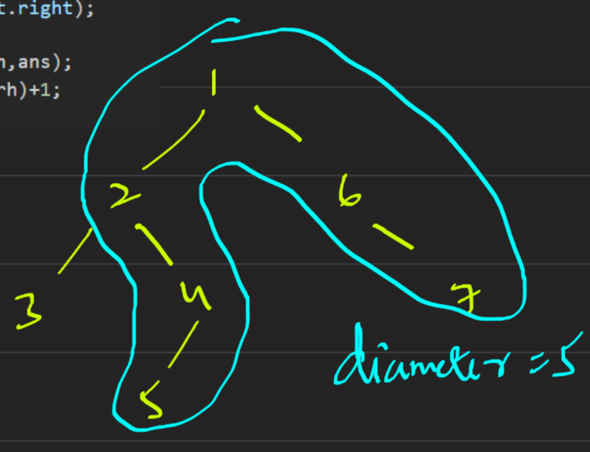



```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```




```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```


```java
int ans = 0;
public int height(TreeNode root){
    if(root == null)return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    ans = Math.max(lh+rh,ans);
    return Math.max(lh,rh)+1;
}
```



* We are calculating height, and meanwhile storing diameter ans as well

diameter ans = (ans, lh+rh) → Max

ans → Without root

lh+rh → With root

* In leetCode, they are asking no. of edges of diameter ⇒ So return ans directly

* In GFG, they are asking no of nodes → So add 1 to our ans;

GFG

```
class Solution {
    // Function to return the diameter of a Binary Tree.
    int ans = 0;

    public int height(Node root){
        if(root == null)return 0;
        int lh = height(root.left);
        int rh = height(root.right);
        ans = Math.max(ans,lh+rh);
        return Math.max(lh,rh)+1;
    }
    int diameter(Node root) {
        // Your code here
        height(root);
        return ans+1;
    }
}
```

leetCode

```
class Solution {
    int ans = 0;
    public int height(TreeNode root){
        if(root == null)return 0;
        int lh = height(root.left);
        int rh = height(root.right);

        ans = Math.max(lh+rh,ans);
        return Math.max(lh,rh)+1;
    }
    public int diameterOfBinaryTree(TreeNode root) {
        height(root);
        return ans;
    }
}
```