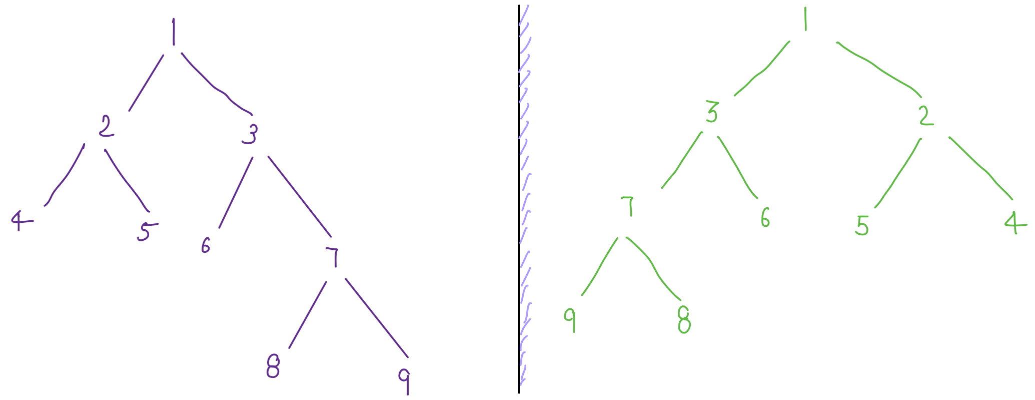
Lecture : Trees-5

Agenda

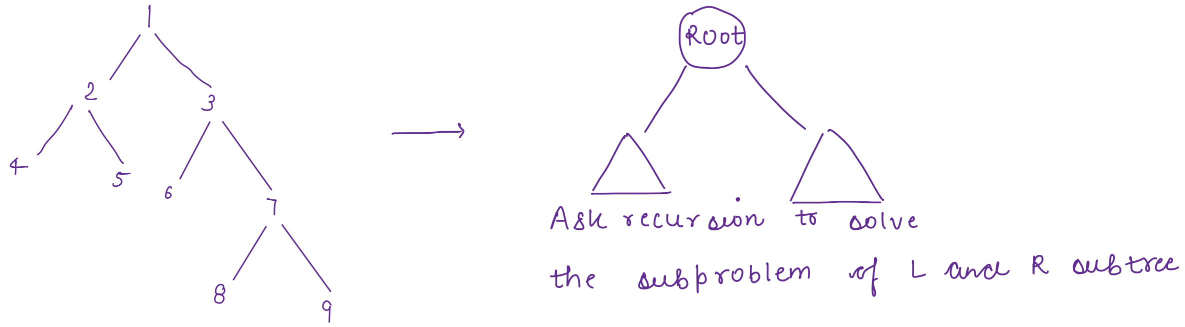
- Invert binary tree
- Equal tree partition
- Next pointer binary tree
- Root to leaf path sum=K
- Diameter of binary tree.

Qu1 Invert a binary tree

ip

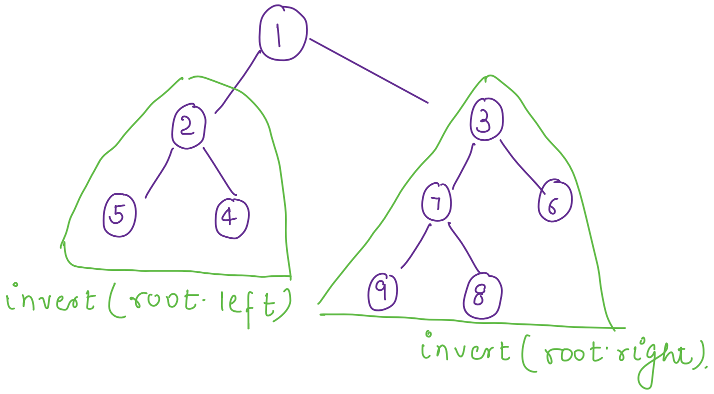


Logic:

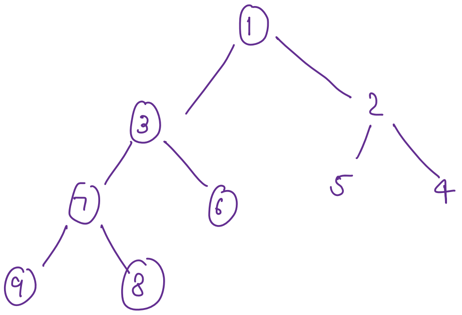


Ask recursion to solve the subproblem of L and R subtree

step1:



step2: Find our own ans. [swapping L and R subtree]

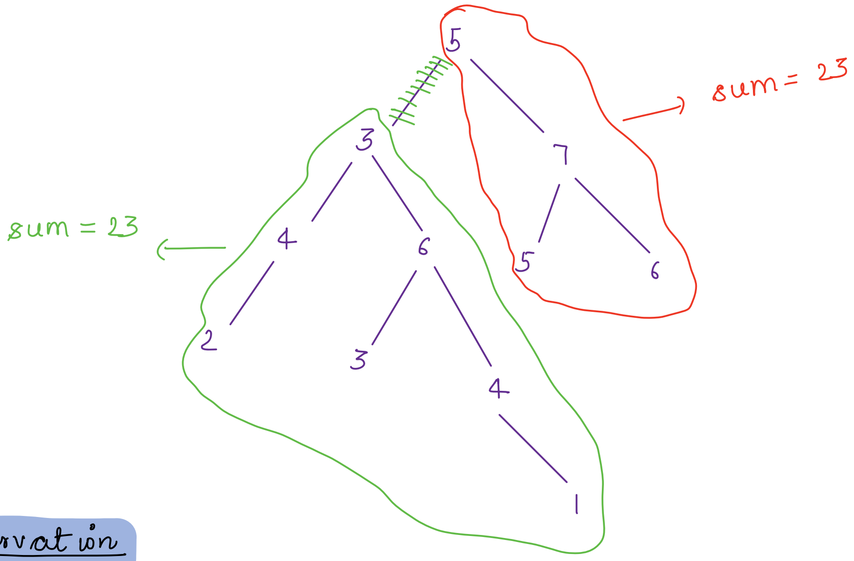


Code:

```
void invert( root) {
        if(root == null) {
            return;
        }
        invert (root.left);

        invert (root.right);

        Node temp = root.left;

        root.left = root.right;

        root.right = temp;
}
```

Qu2 **Equal avg. partition**

check if it is possible to remove an edge from B.T. such that sum of resultant 2 trees are equal.



true.

observation

If sum of tree is odd — ans = false

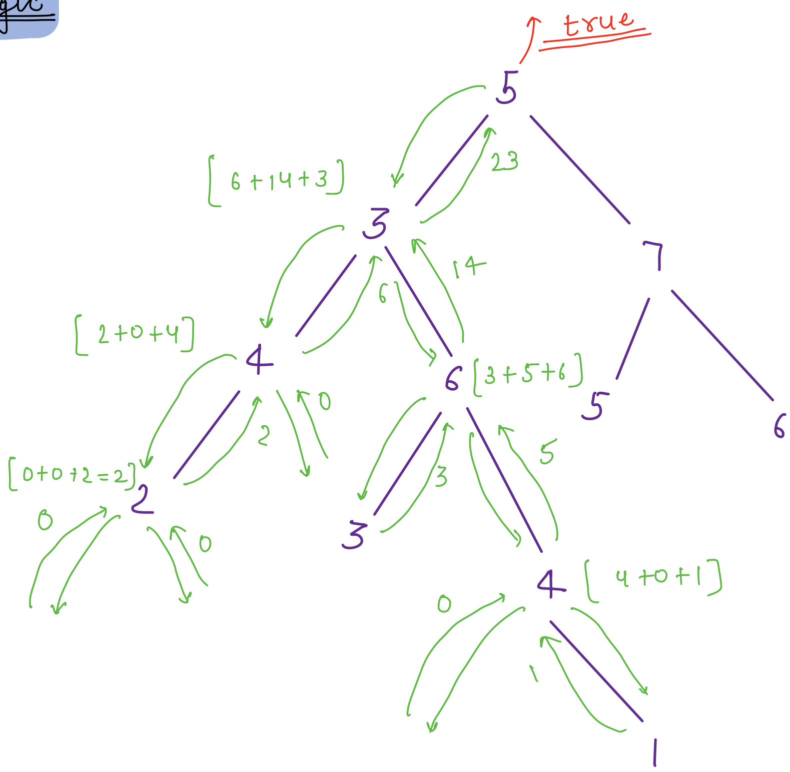
Lets say sum (tree) = S

Modified: check if there is any subtree of sum = $\frac{S}{2}$

if found — true

else — false.

Logic



sum = 46

check = $\frac{sum}{2}$ = 23

Code:

```
boolean ans = false;
boolean check( root) {
        sum = getsum(root);
        if(sum % 2 != 0) {
              return false;
        }
        check( root, sum/2);
        return ans;
}

int check( root, sum) {
        if( root == null) {
              return 0;
        }
        leftsum = check( root.left, sum);
        rightsum = check( root.right, sum);
        if ( leftsum == sum || rightsum == sum) {
              ans = true;
        {
        return leftsum + rightsum + root.data;
}
```

1. Sum Of a tree.

2. Question. ←

Qu3 Next pointer in a BT. [Medium]

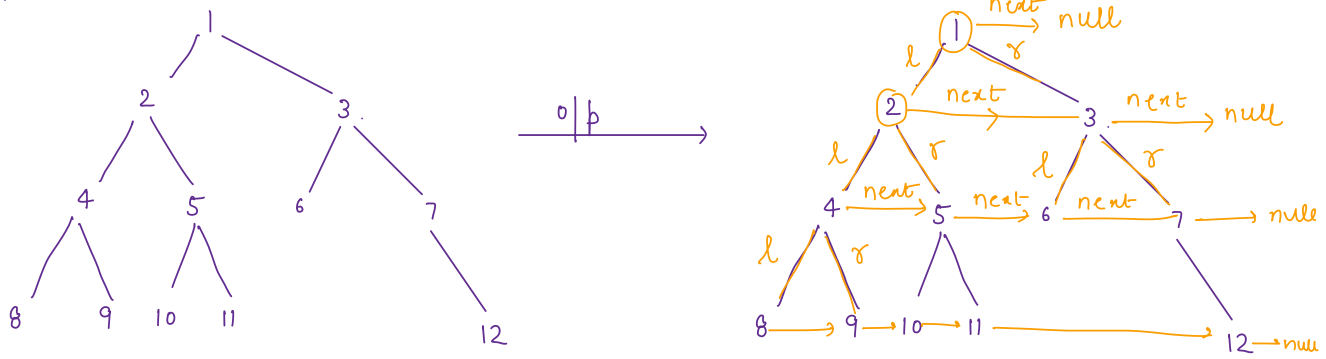
Initially each node next pointer points to null. Update each node next pointer to point to next node in same level.

Current structure

```
class Node{
    int data;
    Node left;
    Node right;
    Node(x){
        data=x;
    }
}
```

Question structure

```
Node{
 int data;
 Node left;
 Node right;
 Node next;
}
```



Code:

```
void levelorder( Node root) {
    Queue < Node > q = new LinkedList<>();
    q. add(root);
    while( ! q. is empty() ) {
        currLevel = q. size();
                        currlevel
        for(i=1; i<= ~~n~~; i++) {
            Node curr = q.poll();
            if( i != currLevel) {
                curr. next = q. peek();
            }
            if( curr.left != null) {
                q. add( curr.left);
            }
            if( curr. right != null) {
                q. add( curr. right);
            }
        }
    }
}
```

TC: O(n)

SC: O(n)

Break: 8:11 — 8:22 AM

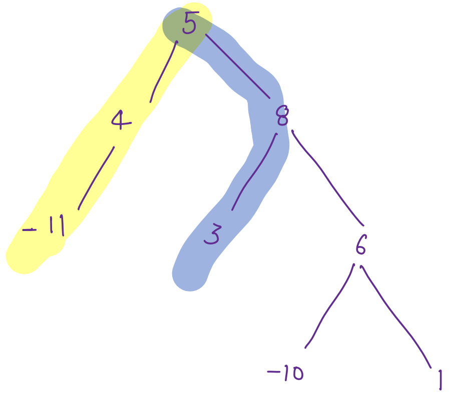
Qu: Check if B.T. has only one root to leaf path = K [sum]

K = -2 true

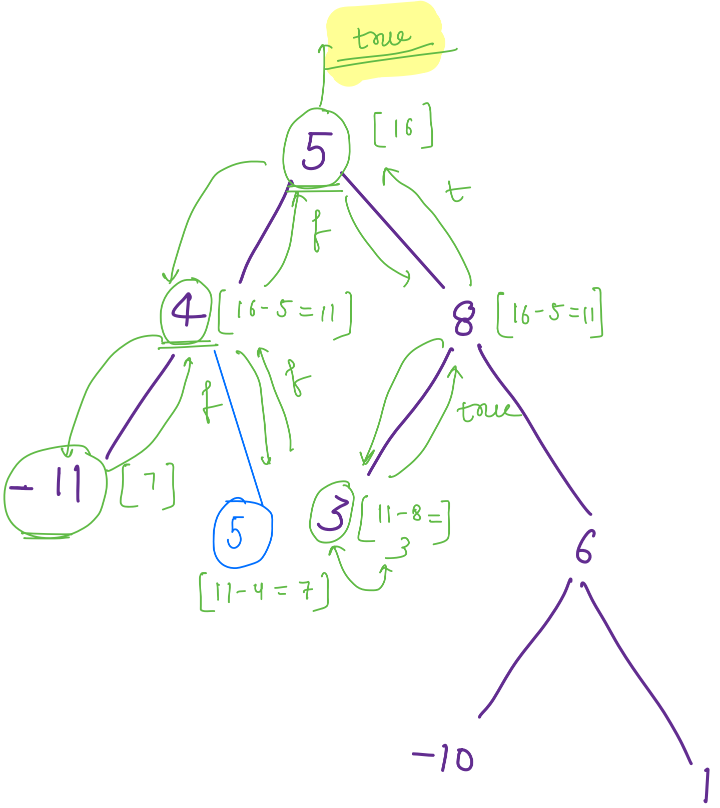
k = 16 true

k = 13 - false

K = 10



k = 16



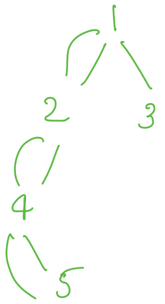
code:

```
          boolean  check(root, k)  {
              if(root == null)  {
                    return false;
              }
k=5  true
5  k=6 false  if(root.left==null && root.right==null) {

                    if(root.data==k) {
                          return true;
                    }
                    return false;
              }

              boolean l = check(root.left, k-root.data);
              if( l == true) {
                     return true;
              }
              boolean r = check( root.right, k-root.data);

              return r;
       }
```

TC: o(n)

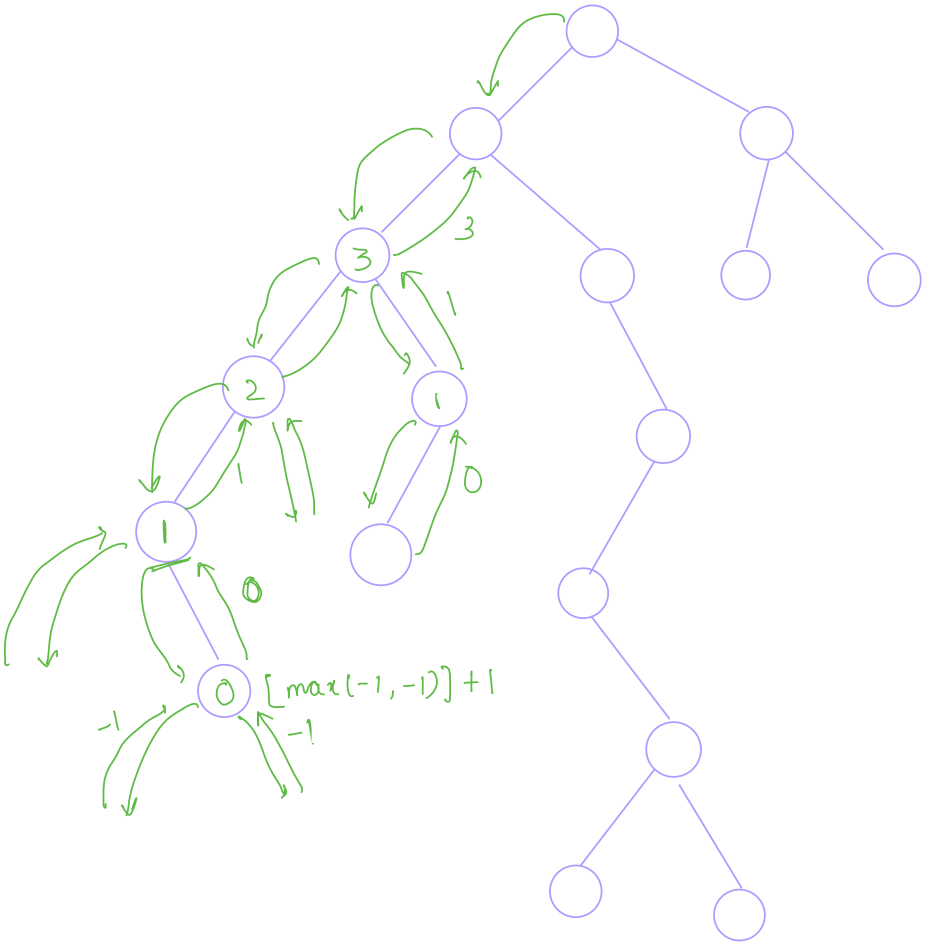
SC: o(h)

Qu: Height of B.T. [in terms of edges]

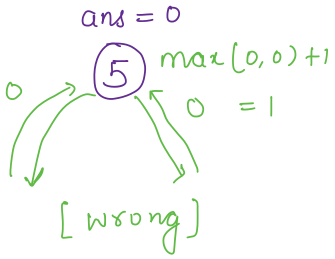


: h = 4
↑
nodes

h = 3
↑
edges



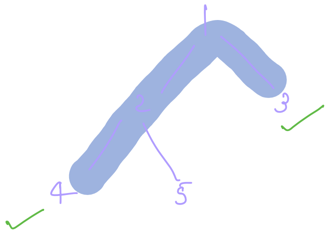
code:



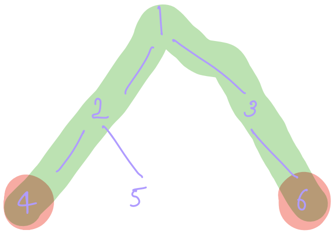
```
int  height(root) {
        if(root == null) {
                return -1;  — edges.
        }
        lh = height(root.left);

        rh = height(root.right);

        return max(lh, rh) +1;
}
```

Qu: Diameter of binary tree [Medium-hard]

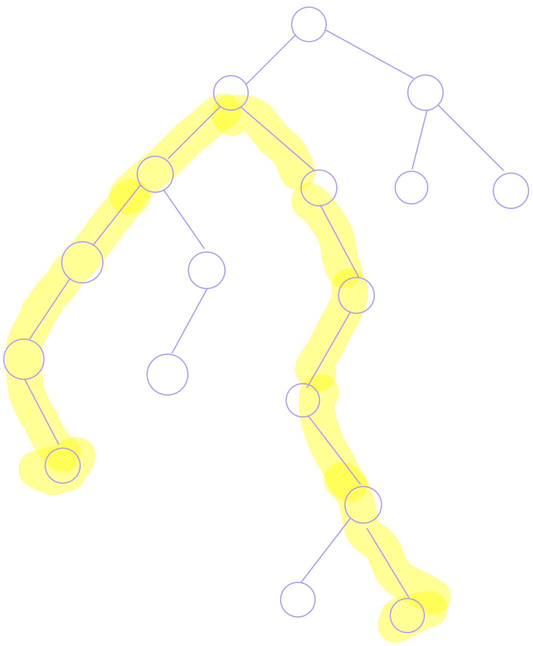
No. of edges of longest path b/w any 2 nodes of binary tree



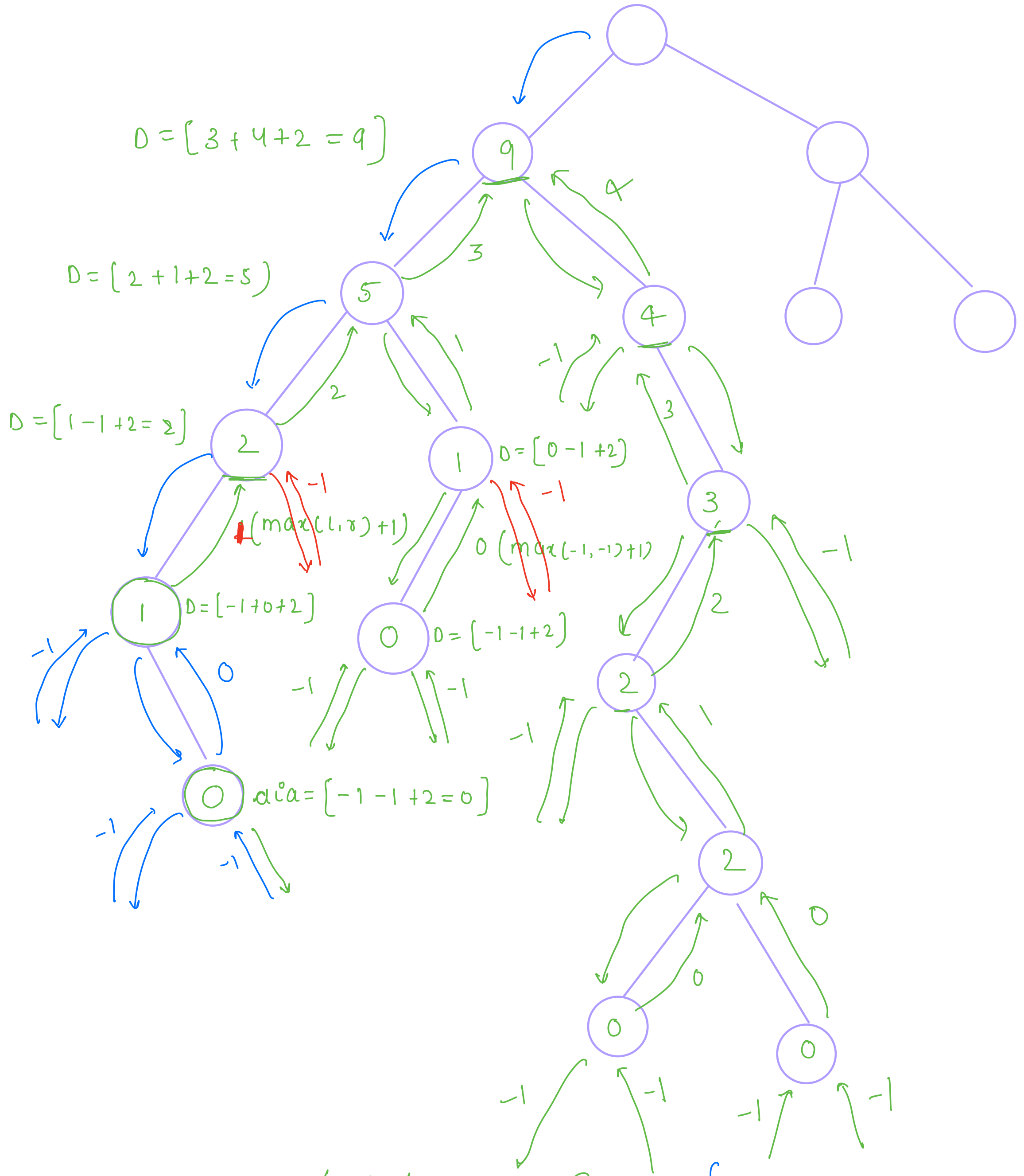
ans = 3



— ans = 4



ans = 9



maxDiameter: ~~0~~ ~~1~~ ~~2~~ 3 . . . 9

Calc: diameter
return: height

Code:

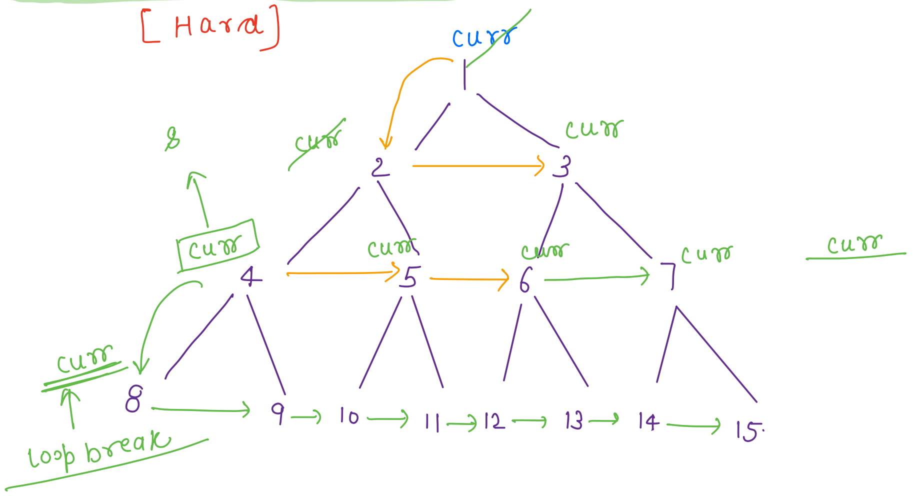
```
int diameter = -1;

int height(root) {

    if(root == null) {

        return -1;
    }

    lh = height(root.left);

    rh = height(root.right);

    diameter = max(diameter, lh + rh + 2);

    return max(lh, rh) + 1;
}
```

TC: O(n)

SC: o(h)

Qu: fill next in perfect binary tree [optional] SC: O(1)

[Hard]



Algo:

```
curr = root;

while(curr.left != null) {
        Node s = curr;
        while(curr != null){
              curr.left.next = curr.right

              if( curr.next != null ){
                 curr.right.next = curr.next.left;
              }
              curr = curr.next;
        }

   curr = s.left;
}
```

H/W: Is it possible to apply this logic in non-perfect binary tree? → No

Thankyou :)