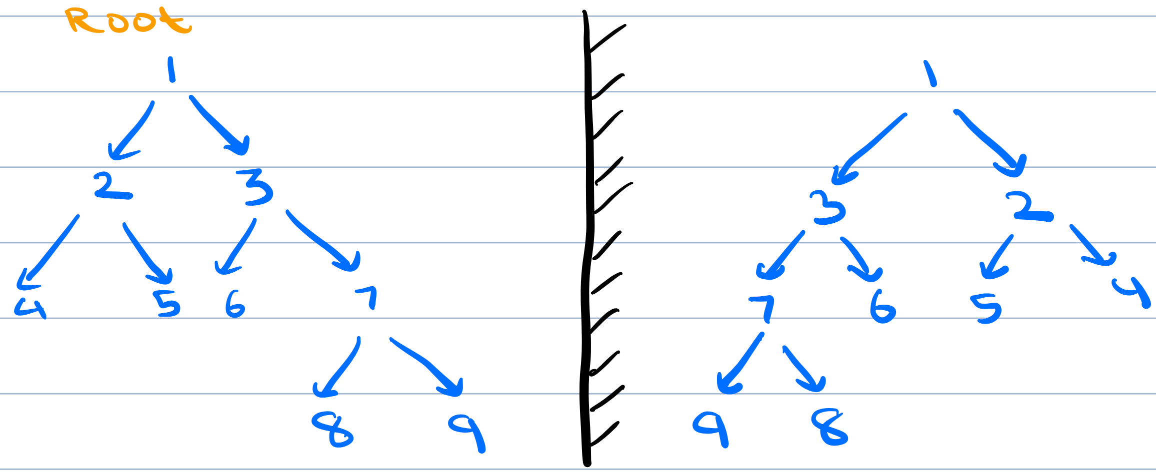
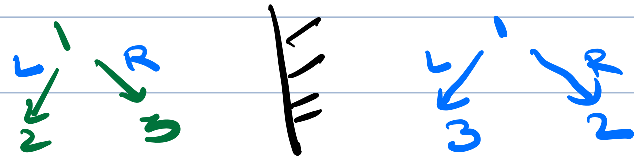
Agenda

1. Invert Binary Tree
2. Equal Tree Partition
3. Next Pointer Binary Tree
4. Root to Leaf Path Sum $= k$
5. Diameter of Binary Tree

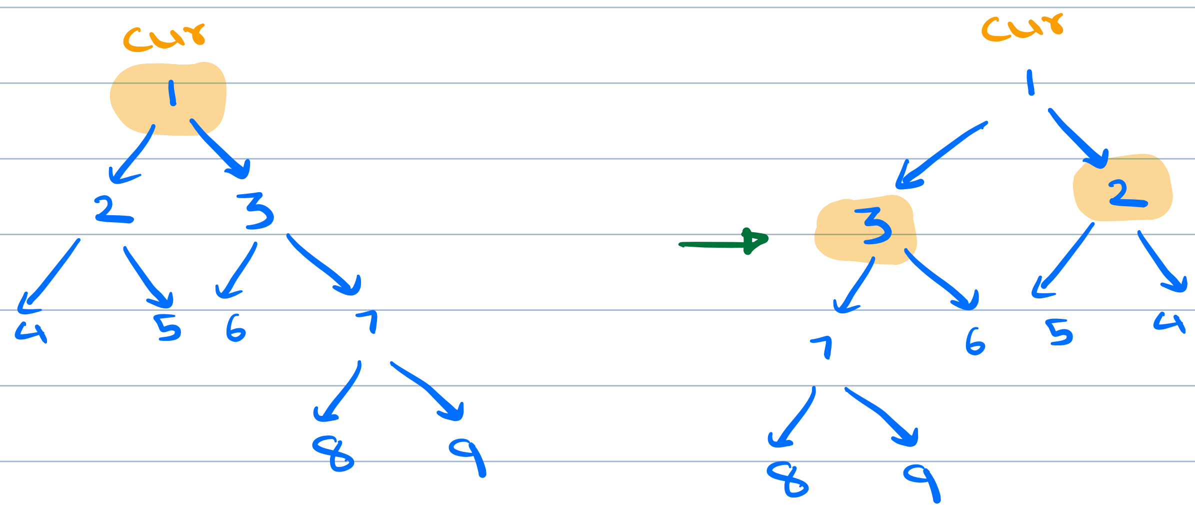
1. Invert a binary tree.

Input:

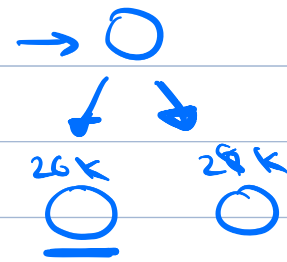




Observation: For every node, swap left and right child



```
void invert (Node root) {
    if (root == NULL)
        return
    Node temp = root.left
    root.left = root.right
    root.right = temp
    invert (root.left)
    invert (root.right)
}
```



LC

Temp

26 K

TC: O(N)

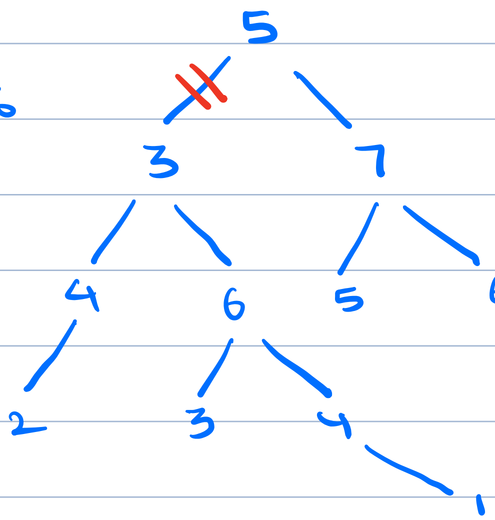
SC: O(H)

$\log_2 N \rightarrow N$

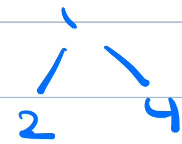
2. Check if it is possible to remove an edge from Binary Tree s.t. sum of resultant two trees is equal.

Total sum=46

tree
↓
sum=23



Ans= True

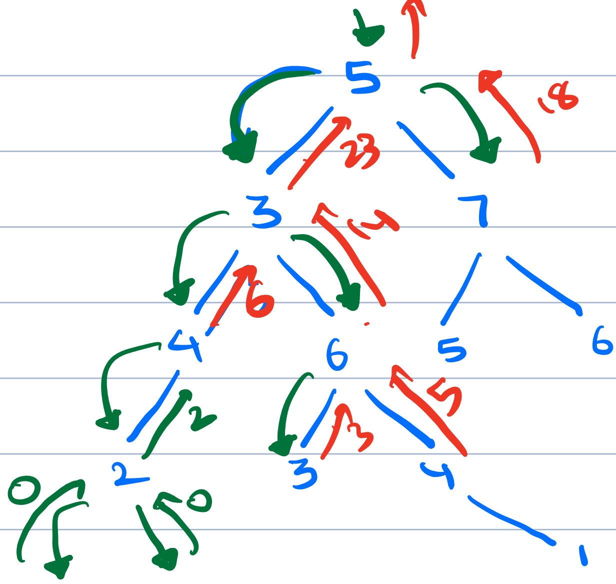


Ans = false

Obs 1 : If total sum is odd, return false

Obs 2 : If totalsum is even (s) then find a subtree with sum = s/2

46

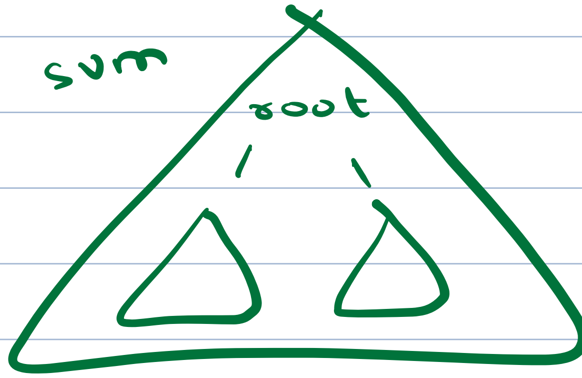


ans = ~~false~~ true

Total sum = 46

sum = 23

sum



sum (root) = sum (root.left) + sum (root.right) + root.val

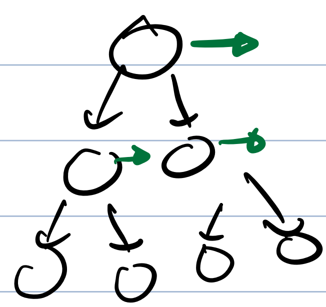
```
int totalsum = 0

void preorder (Node root) {
    if (root == NULL)
        return

    totalsum += root.val

    preorder (root.left)
    preorder (root.right)
}
```

TC : O(N)

SC : O(H)

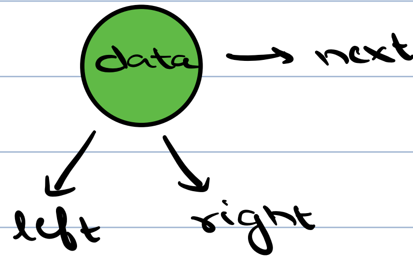
```
bool ans = false
int dsum = totalsum/2

int subtreesum(Node root){
    if (root == NULL)
            return 0

    int l = subtreesum(root.left)
    int r = subtreesum(root.right)
    int rootsum = root.val + l + r
        if (rootsum == dsum)
                ans = true

    return rootsum
}
```

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



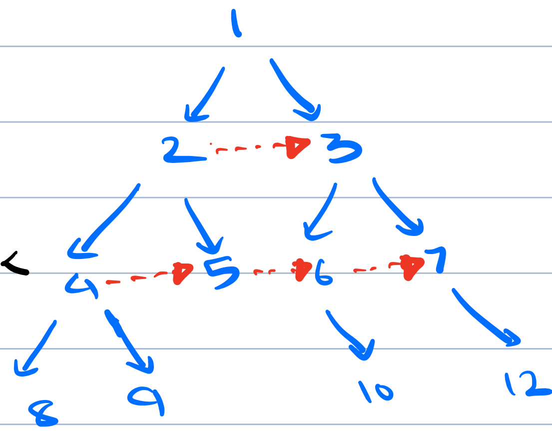
3. a) Populate next pointer in BT

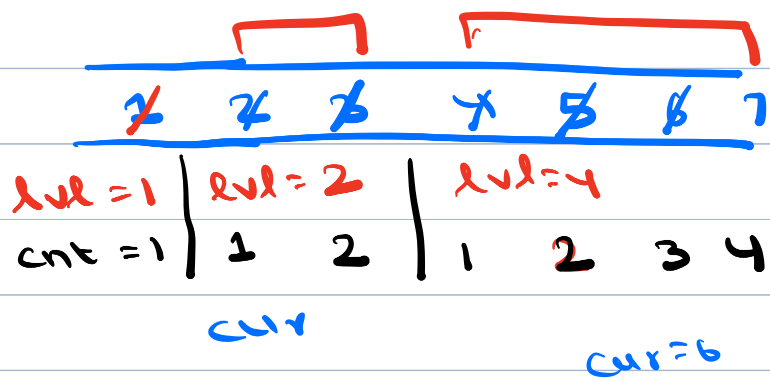
```
class Node {
    int data
    Node left, right, next
}
```



Initially each node's next points to NULL. Update each node's next to store address of next node in same level.

```
Queue <Node> q
q.enqueue (root)
while (!q.empty()) {
    int levelsize = q.size()
    for (cnt = 1 ; cnt ≤ levelsize ; cnt++) {
        Node cur = q.front()
        q.dequeue()
        if (cnt != levelsize)
            cur next = q.front()
        if (cur.left != NULL)
            q.enqueue (cur.left)

        if (cur.right != NULL)
            q.enqueue (cur.right)
    }
}
```





TC : O(N)

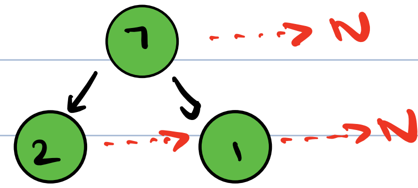
SC : O(N)

3 b) Populate next pointer in Perfect BT

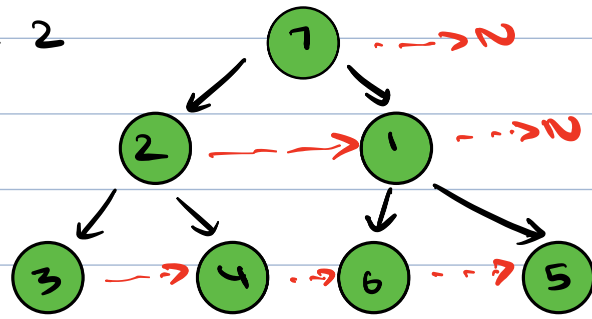
Expected SC : O(1)

```
class Node {
    int data
    Node left, right, next

    Node(x) {
        data = x
        left = NULL
        right = NULL
        next = NULL
    }
}
```

Ex 1

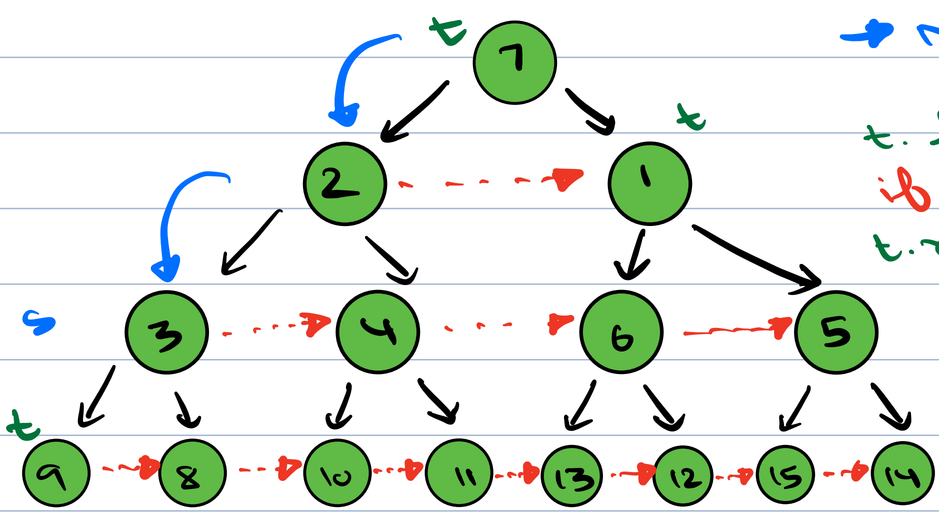


Ex 2



t = root

Ex 3



→ Node s = t

t.left.next = t.right
if (t.next != NULL)
t.right.next = t.next.left

// Move in level
t = t.next

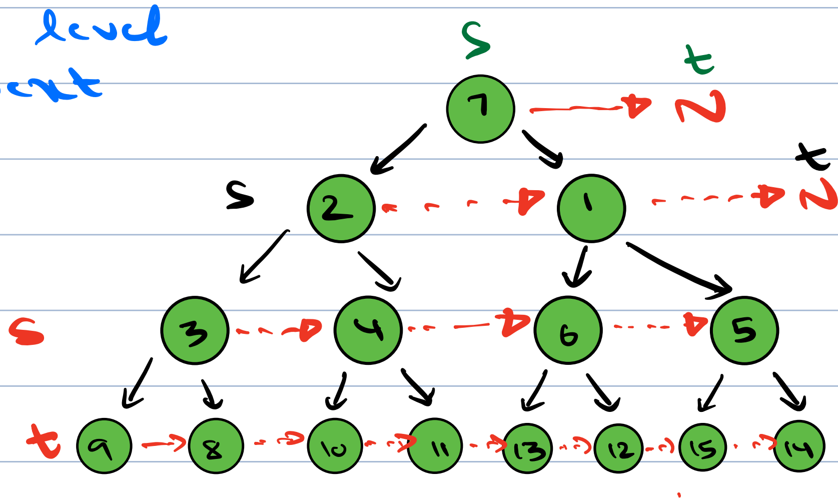
Make t jump to next level

t = s.left

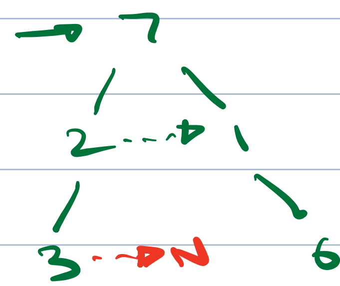
```
Node t = root

while (t.left != NULL) {
    Node s = t
    while (t != NULL) {

        t.left.next = t.right
        if (t.next != NULL)
        t.right.next = t.next.left

            // Move in level
                t = t.next
    }

    t = s.left
}
```

TC : O(N)

SC : O(1)

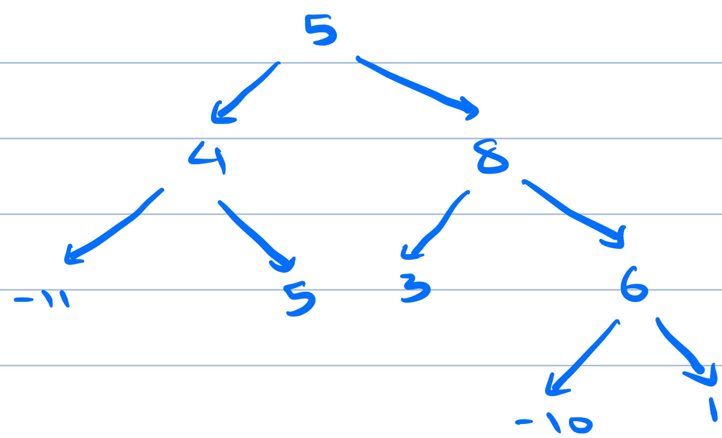


10:30

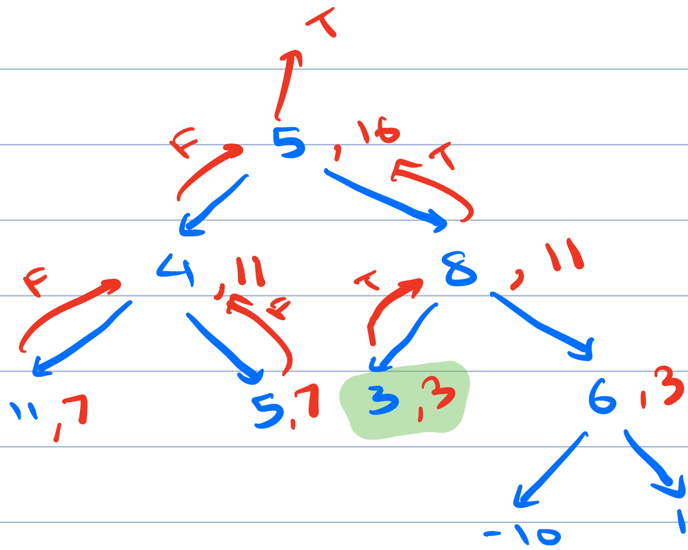


4. Check if given binary tree has any root to leaf path sum = k.

k = 16	True
k = -2	True
k = 9	True



k = 16



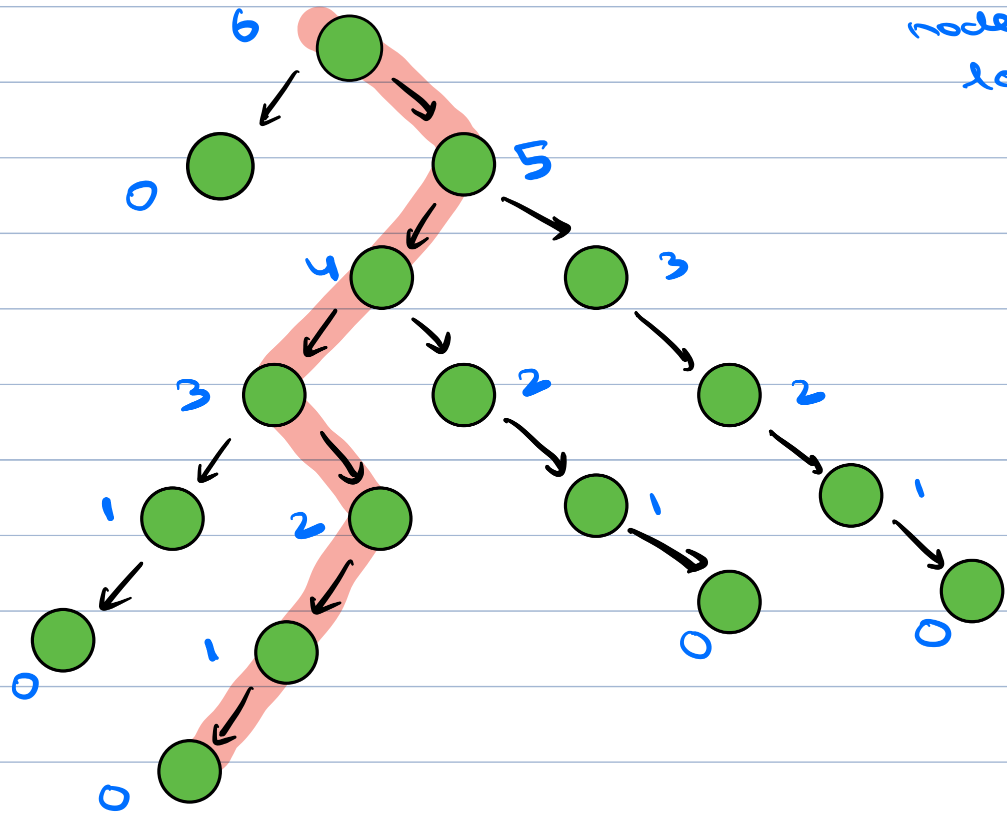
```
// Given root node, check whether sum k
   can be formed from root to leaf
bool Check (Node root, int k) {
        if (root == NULL)
              return false
      // root can contribute root.val -> k
      // more -> k - root.val
      if (root.left == NULL && root.right == NULL)
              return root.val == k
   return check (root.left, k - root.val) ||
          check (root.right, k - root.val)
}
```

TC : O(N)

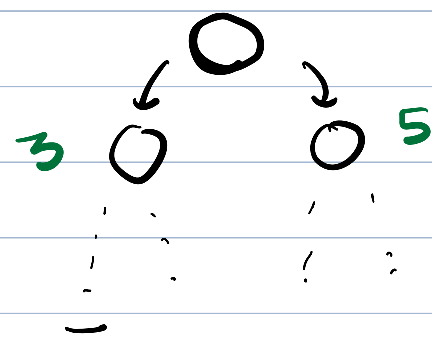
SC : O(H)

Concept: Height of BT (edges)

longest path node to leaf



$h(node) = \max(h(LC), h(RC)) + 1$



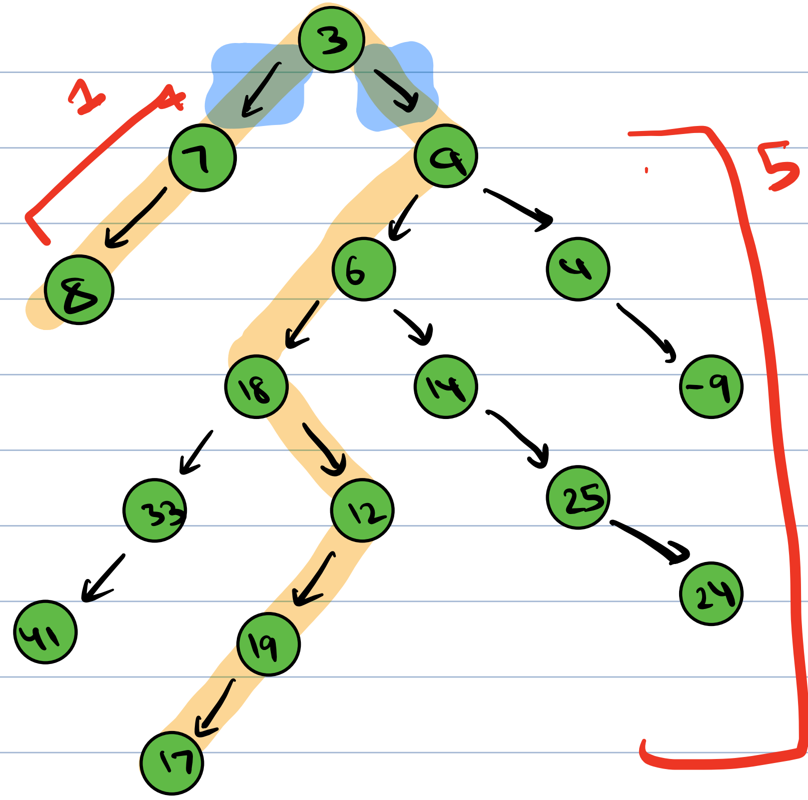
$h(NULL) = -1$

```
int height(Node root){
    if (root == NULL)
                return -1
    int lh = height(root.left)
    int rh = height(root.right)
    return   max(rh, lh)+1
}
```

TC: O(N)

SC: O(H)

5 a) Longest path across root (count edges)



ans = 8

$ans = \underset{LC}{1} + \underset{RC}{5} + 2$

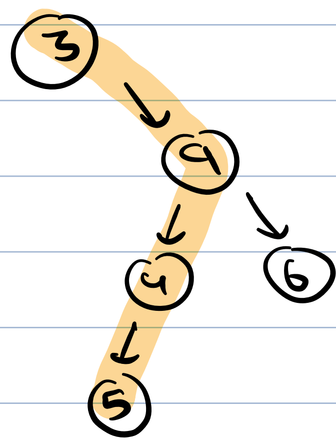
$= 8$

3

$1 = h(7) \quad h(9) = 5$

Longest path across root $= h(\text{root.left}) + h(\text{root.right}) + 2$

ans = 0



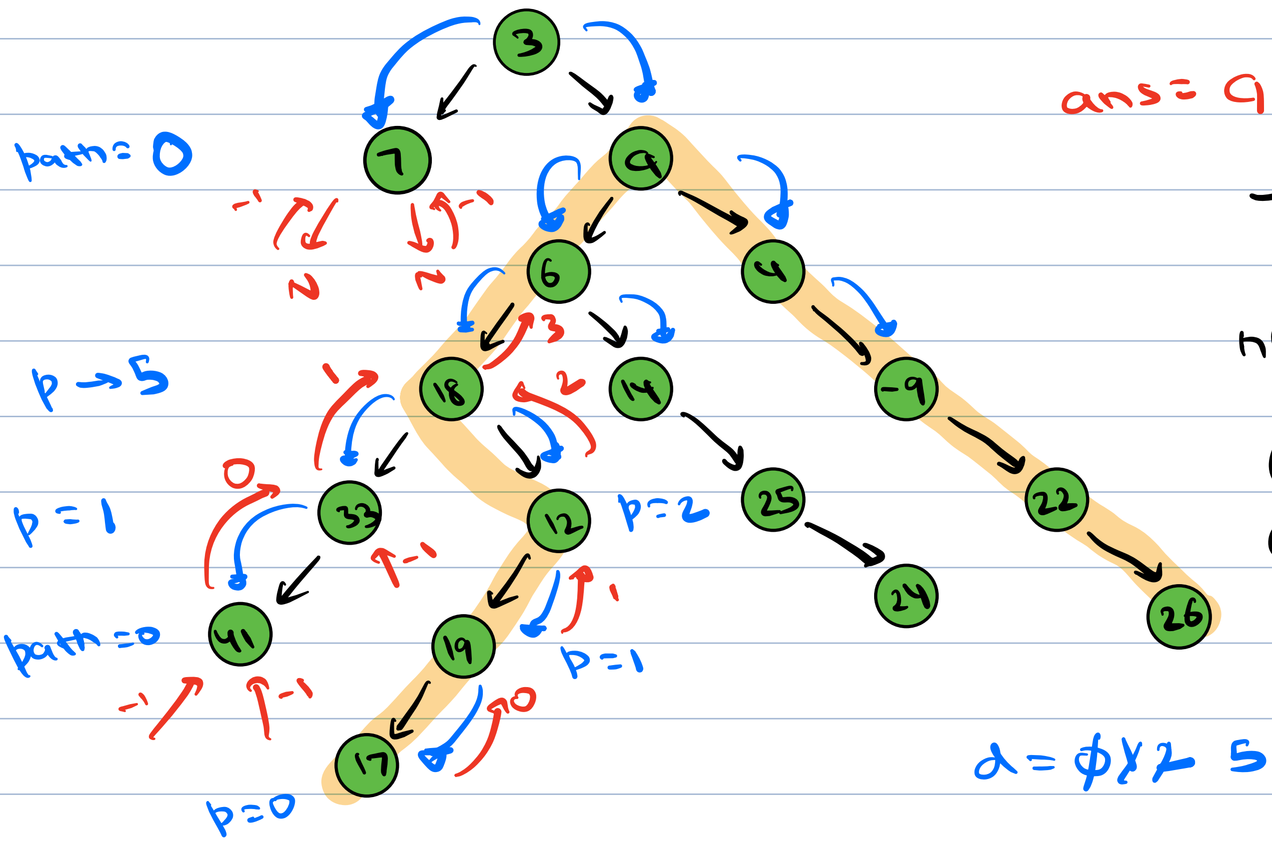
ans = 3

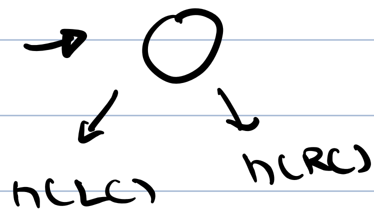
$h(LC) + h(RC) + 2$

$-1 + 2 + 2$

$= 3$

5. b) Longest Path b/w any 2 nodes in a tree → Diameter of tree





1. h(Node)
2. Longest path across Node

int diam = 0

```
int height (Node root) {
    if (root == NULL)
        return -1

    int lh = height (root.left)
    int rh = height (root.right)
    diam = max (diam, lh + rh + 2)
    return max (rh, lh) + 1
}
```

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