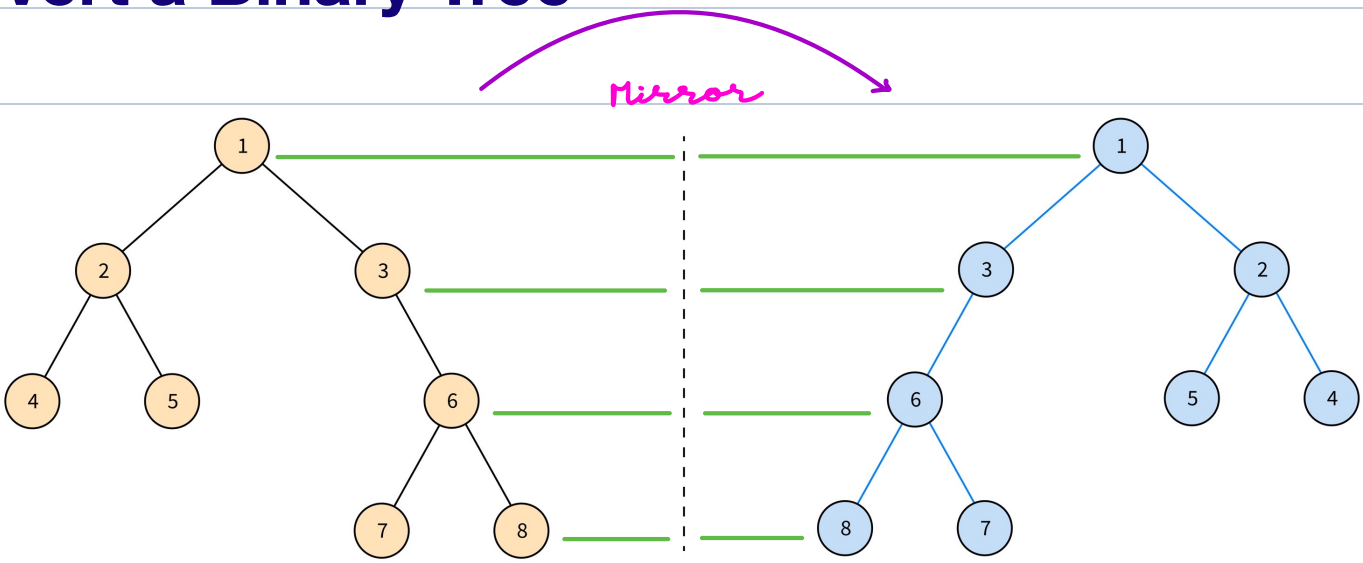
SCALER

Trees - 5

TABLE OF CONTENTS

Invert a Binary Tree



Is the below inversion correct for given binary tree ?

```
Given
     4
    / \
   2   7
  / \   \
 1   3   9

Inverted
     4
    / \
   7   2
  \   / \
   3 1   9
```

No

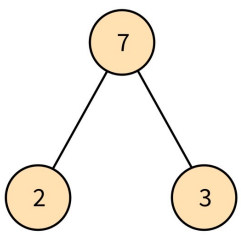
```
void invert (root) {
    if (root == null)
        return
    temp = root.left
    root.left = root.right
    root.right = temp
    invert (root.left)      // Left
    invert (root.right)     // Right
}
```

Node

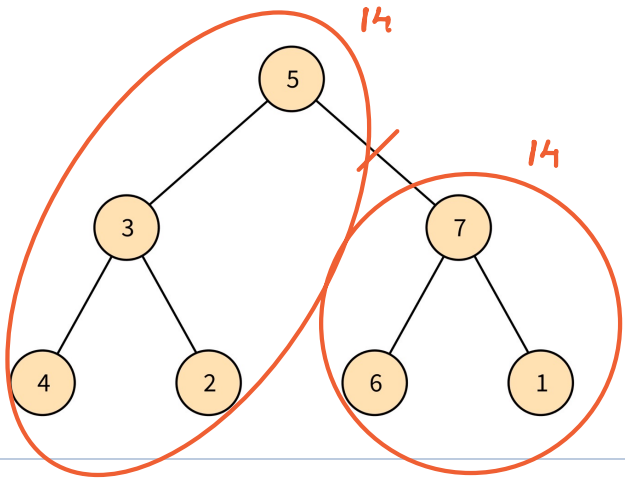
TC = O(N) SC = O(H)

Equal tree Partition

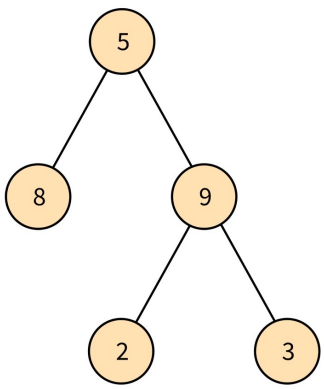
- Check if it is possible to remove an edge from the given binary tree such that sum of resultant two trees is equal. ← half of total.



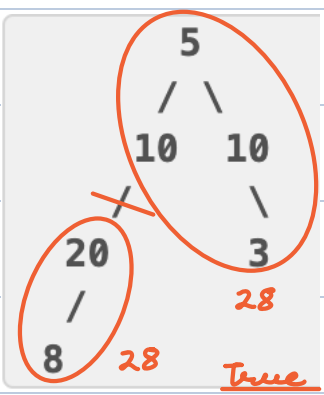
False



True



False




```
sum = getSum(root)
if (sum % 2 == 1) return false
ans = false
checkSum(root, sum/2)
return ans
```

```
int getSum(root) {
    if (root == null) return 0
    return root.val + getSum(root.left) +
                getSum(root.right)
}
```

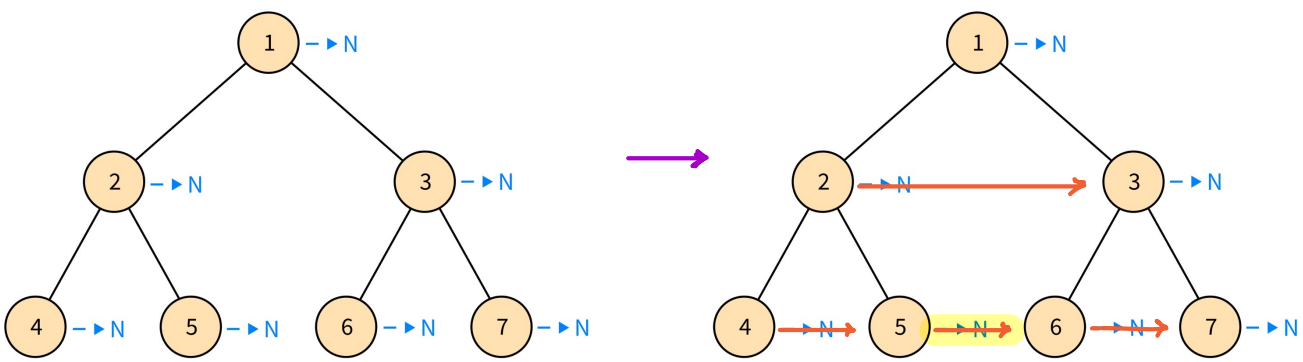
```
int checkSum(root, half_Sum) {
    if (root == null) return 0
    s = root.val + checkSum(root.left, half Sum) +
        checkSum(root.right, half Sum)
    if (s == half Sum) ans = true
    return s
}
```

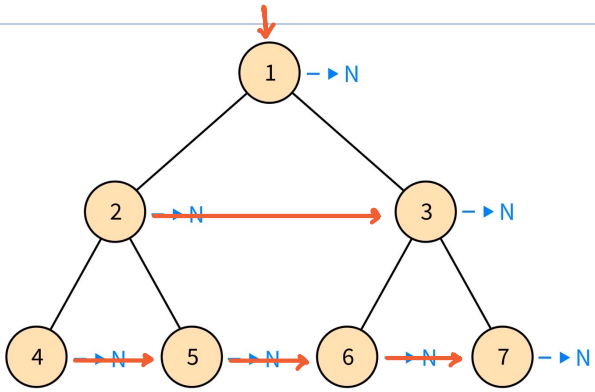
$TC = O(N + N) = O(N)$

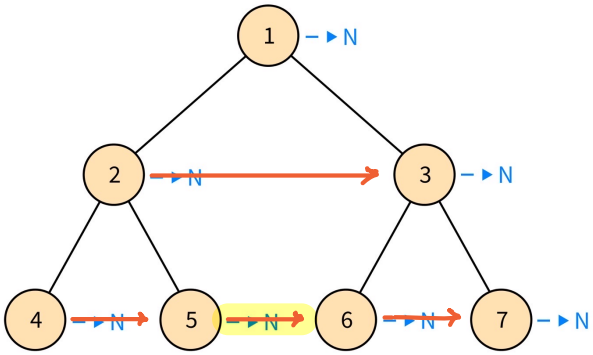
$SC = O(H)$

Next Pointer in Binary Tree

- Given a perfect tree initially with all next pointers set to nullptr, modify the tree in-place to connect each node's next pointer to the next node in the same level from left to right, following an in-order traversal.



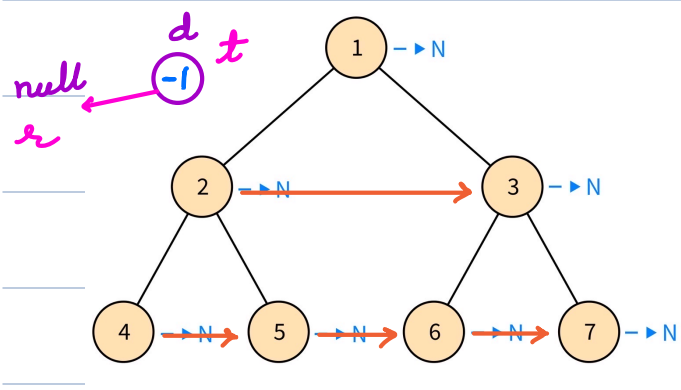


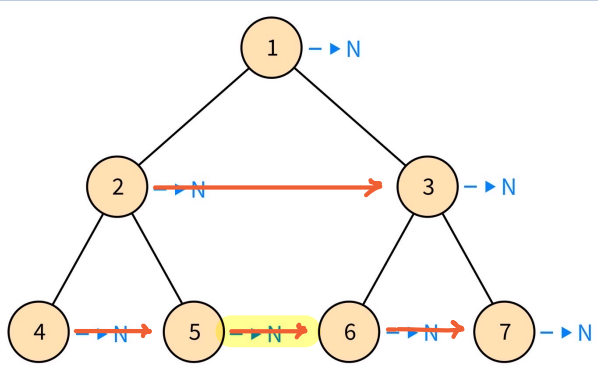


Level Order Traversal → Queue


```
if (root == null) return
// Queue → q
q.enqueue(root)
last = root
while (!q.isEmpty()) {
    cur = q.dequeue()
    if (cur.left != null) q.enqueue(cur.left)
    if (cur.right != null) q.enqueue(cur.right)
    if (cur != last) cur.next = q.front()
    else last = q.rear()
}
```

TC = O(N) SC = O(N)






```
if (root == null) return
dummy = new TreeNode (-1)
temp = dummy
while (root != null) {
    if (root.left != null) {
        temp.next = root.left
        temp = temp.next
    } if (root.right != null) {
        temp.next = root.right
        temp = temp.next
    } root = root.next
if (root == null) {
    root = dummy.next
    dummy.next = null
    temp = dummy
}
}
```

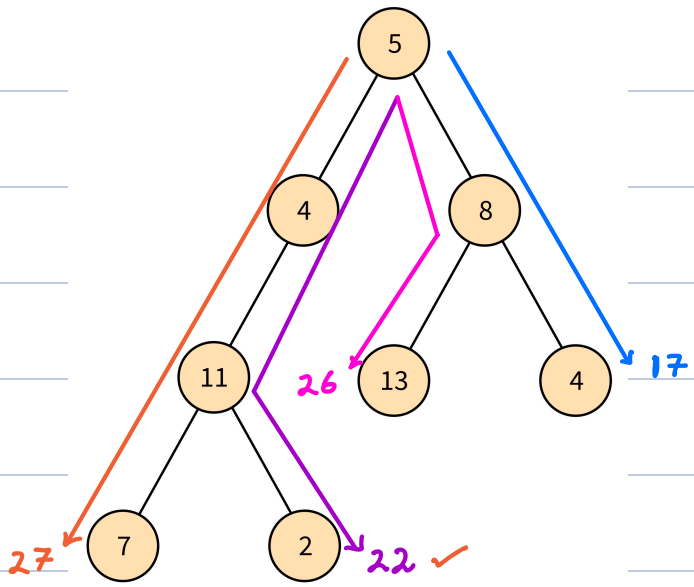
TC = O(N)

SC = O(1)

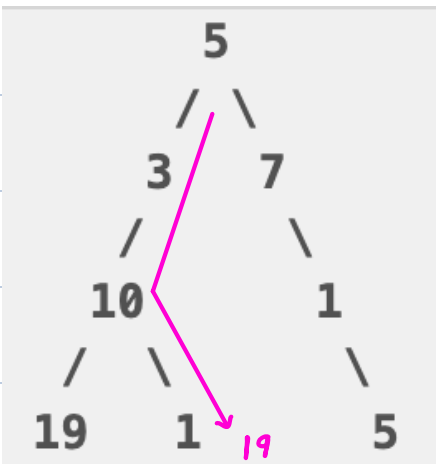
< Question > : Check if root to leaf path sum equals to K.

a. K = 22 → true

b. K = 19 → false




```
boolean check (root, K) {
    if (root == null) return false
    if (root.left == null &&
        root.right == null)
            return (root.val == K)
    return check (root.left, K-root.val) ||
           check (root.right, K-root.val)
}
```

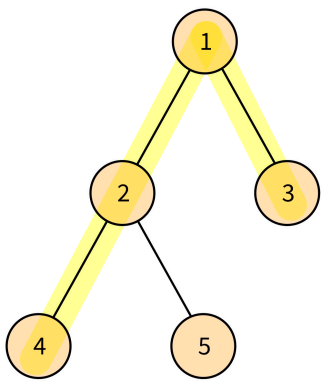


TC = O(N) SC = O(H)

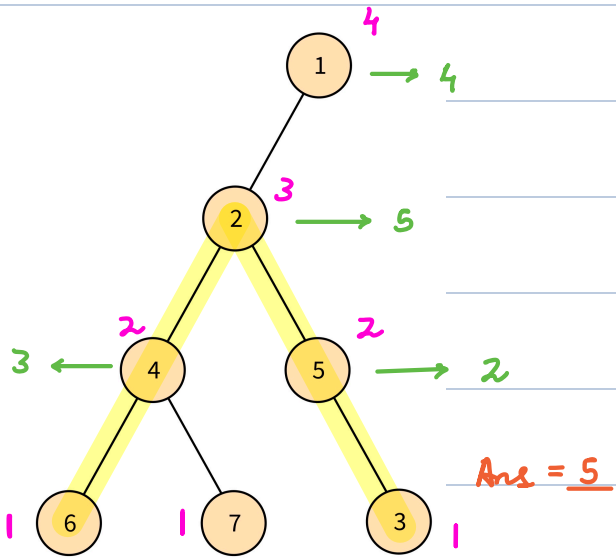
Diameter of Binary Tree

- **Definition of Diameter :**

The diameter of a binary tree is defined as the number of nodes along the longest path between any two leaf nodes in the tree. This path may or may not pass through the root.

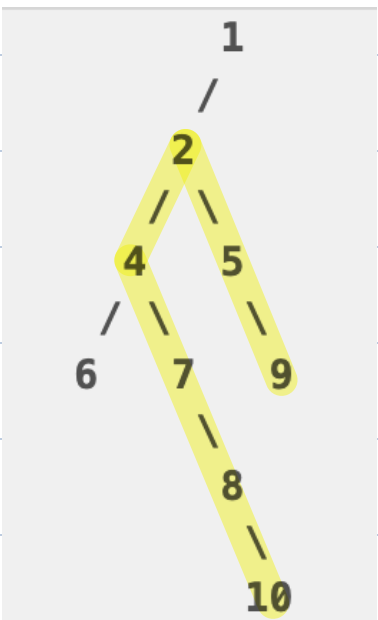


Ans = 4



Ans = 5

Hint → use height of tree. height(leaf) = 1



height(node) = max(left, right) + 1

diameter(node) = height(left) + height(right) + 1

Ans = 7