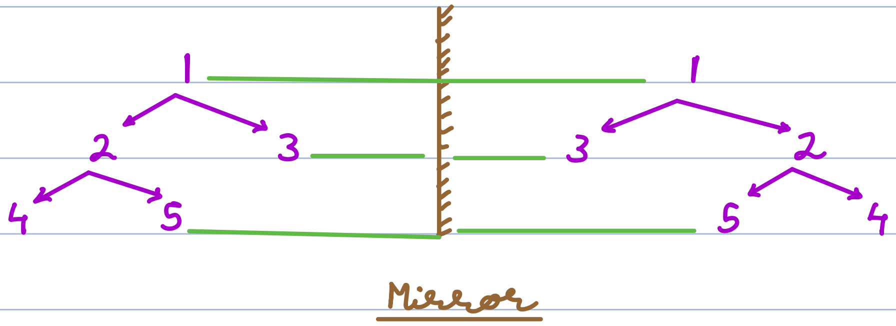
Q → Invert the given binary tree.

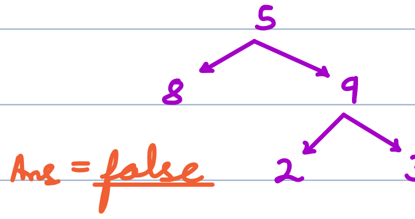


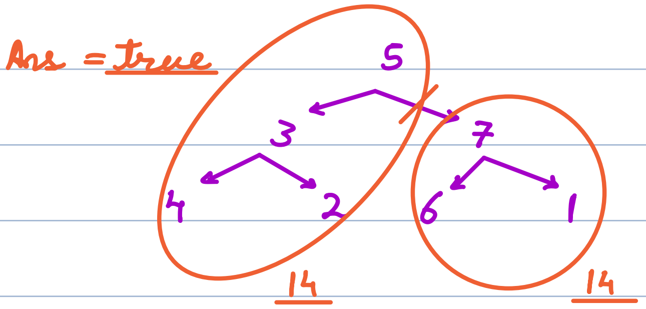
```
void invert (root) {
    if (root == null) return null
    t = root.left
    root.left = root.right
    root.right = t              N
    invert (root.left)          L
    invert (root.right)         R
}
```

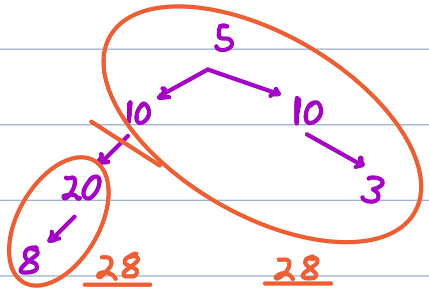
TC = O(N)

SC = O(H)

Q → Given a binary tree, check if it is possible to split it into 2 subtrees with equal sum of nodes.







1) Find total sum of nodes (S).

2) Check if there exist a subtree with sum = S/2.

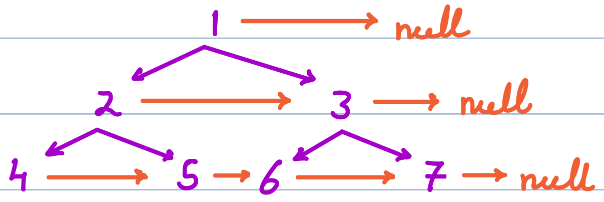
```
S = sum(root)
ans = false
```

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

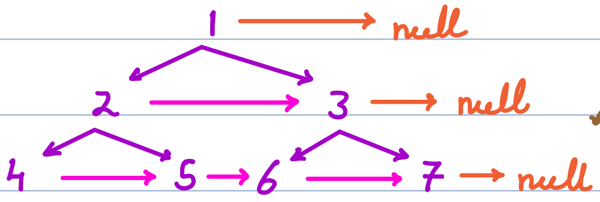
TC = O(N) SC = O(H)

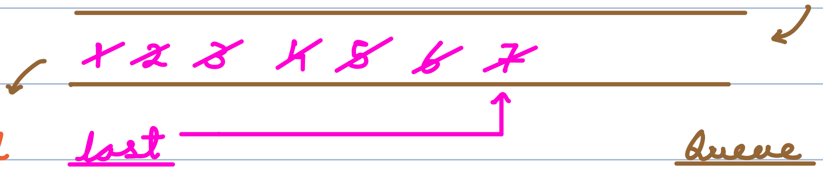
```
int check(root) {
    if(root == null)
        return 0
    total = root.data +
            check(root.left) +
            check(root.right)
    if(total*2 == S)
        ans = true
    return total
}
```

Q → Given a perfect binary tree (all levels complete), update the tree s.t each node is connected to next node in the same level from left to right.



Level order traversal





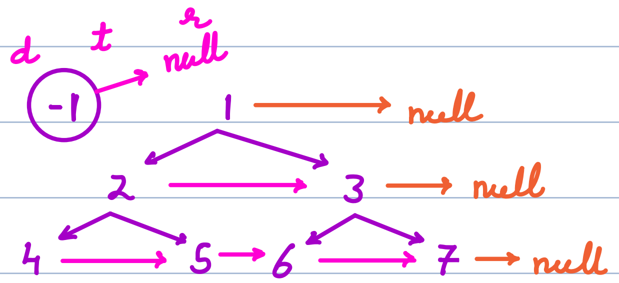
```
if ( root == null)  return
// Queue → q
q. enqueue (root)
last = root
while ( ! q.isEmpty ()) {
    x = q. dequeue ()
    if ( x . left != null)  q. enqueue (x. left)
    if ( x. right != null)  q. enqueue (x. right)
    if ( x != last) {
        x. next = q. front ()
    } else {
        x. next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

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

O(1) ?

Initially next ∀ nodes point to null.

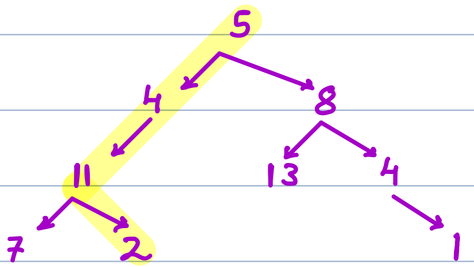
```
if ( root == null)  return
d = new Node (-1)
t = d
while ( root != null) {
    if (root.left != null) {
        t. next = root. left
        t = t. next
```



TC = O(N)   SC = O(1)

```
    }
    if (root.right != null) {
        t.next = root.right
        t = t.next
    }
    root = root.next
    if (root == null) {
        root = d.next
        d.next = null
        t = d
    }
}
```

Q → Given a binary tree & an integer K, check if there exist a root to leaf path sum = K.



K = 22 → Ans = true

K = 13 → Ans = false

K = 20 → Ans = false

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

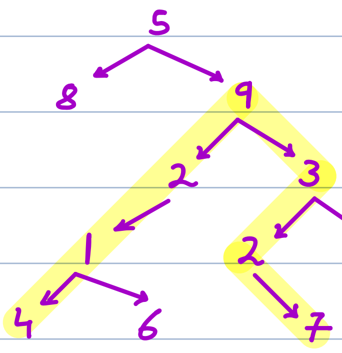
}

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

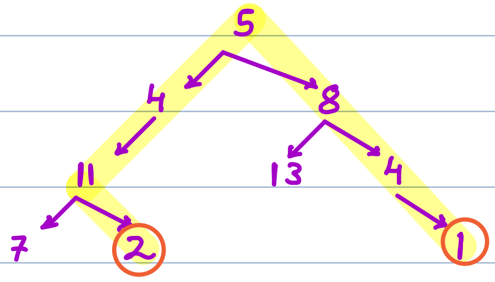
Q → Given a binary tree, find the length of longest path b/w any two nodes in the tree.

→ # edges

Diameter

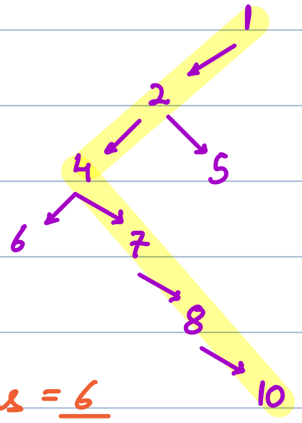


Ans = 7



Ans = 7

diameter (x) = height (x.left) + height (x.right) + 2



Ans = 6

```
diameter = 0
int height (root) {
    if (root == null)
        return -1
    L = height (root.left)
    R = height (root.right)
    diameter = max (diameter,
                    L + R + 2)
    return max (L, R) + 1
}
```

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