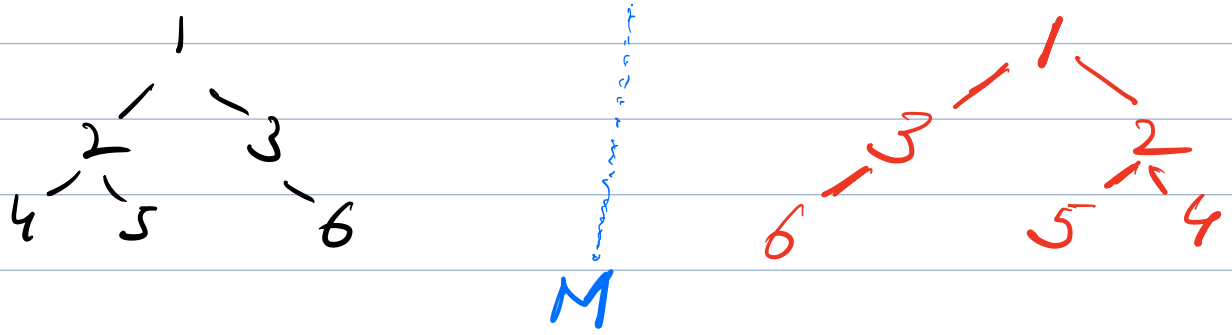
Google

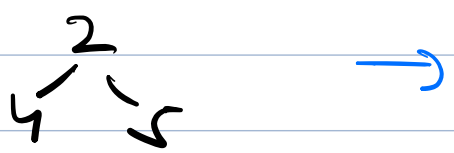
Homebrew

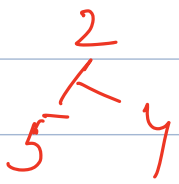
Q3 Invert a Binary Tree

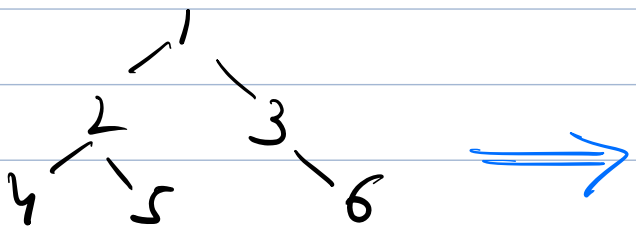
↳ convert to mirror image

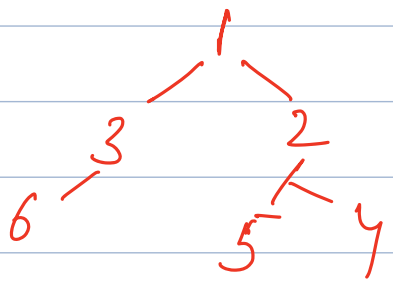


Invert









Obs: 1) Left → Right && Right → Left

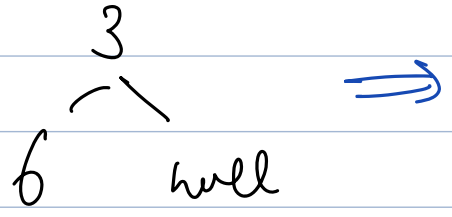
2) If we can invert left subtree (LST) & right subtree (RST)

You need to return a new tree

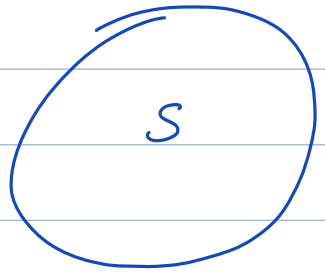
```
def invert (root):
    if ( root == null )
        return null

    x = invert(root.left)
    y = invert (root.right)

    root.right = x
    root.left = y

    return root
}
```

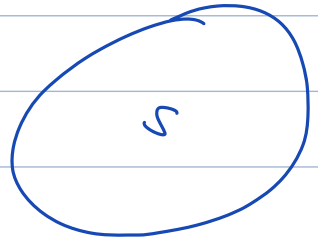
TC: O(N)
SC: O(height)

x = null
y = 6



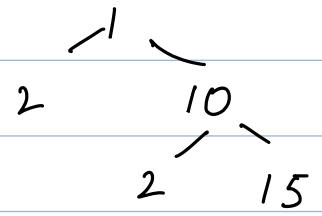
S + S = total_sum
2S = total_sum
S = total_sum/2





Q2 Equal tree Partition

Split into 2 subtrees of equal sums

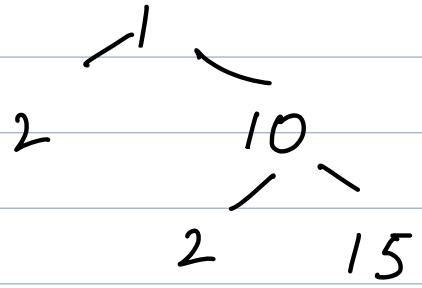


ans = true

Obs: Total sum has to be even

Obs 2: There has to be a subtree with

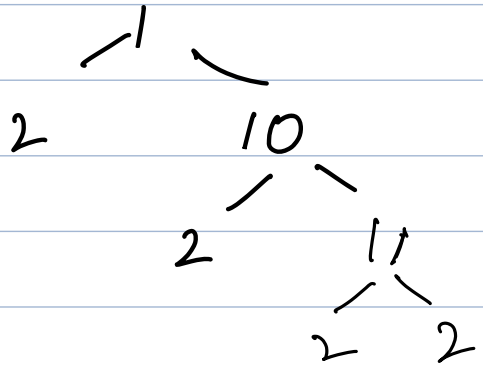
sum = tot_sum/2



tot_sum = 30

sum of subtree = 30/2 = 15

req_sum = 15



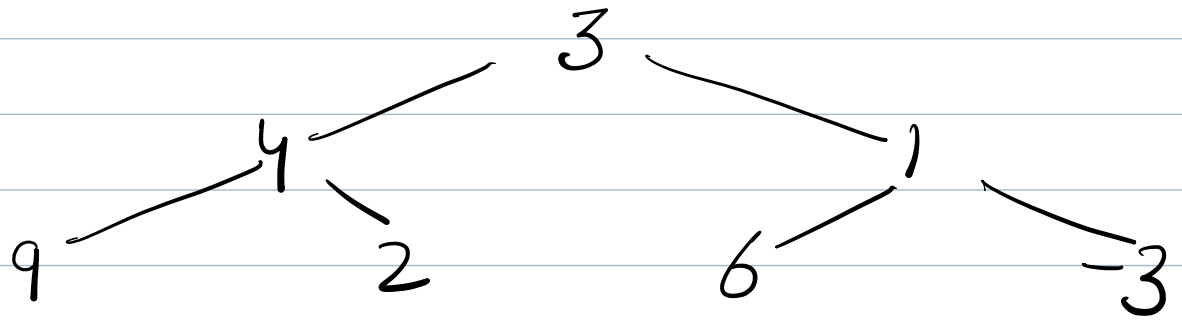
Code

```
bool has_sum ( Node node , int k ){
    if ( node == null )
        return false
    left sum = findsum ( root . left )
    right sum = find sum ( root . right )
    if ( left sum == k || right sum == k ||
        has_sum ( root . left , k ) ||
        has_sum ( root . right , k ) )
        return true
    else return false
}
```

```
bool partition ( Node root ) {
    tot_sum = find_sum (root)
    if ( tot_sum % 2 == 1 )
        return false
    req_sum = tot_sum / 2
    if ( has_sum (root, req_sum ) == true )
        return true
    else
        return false
}
```

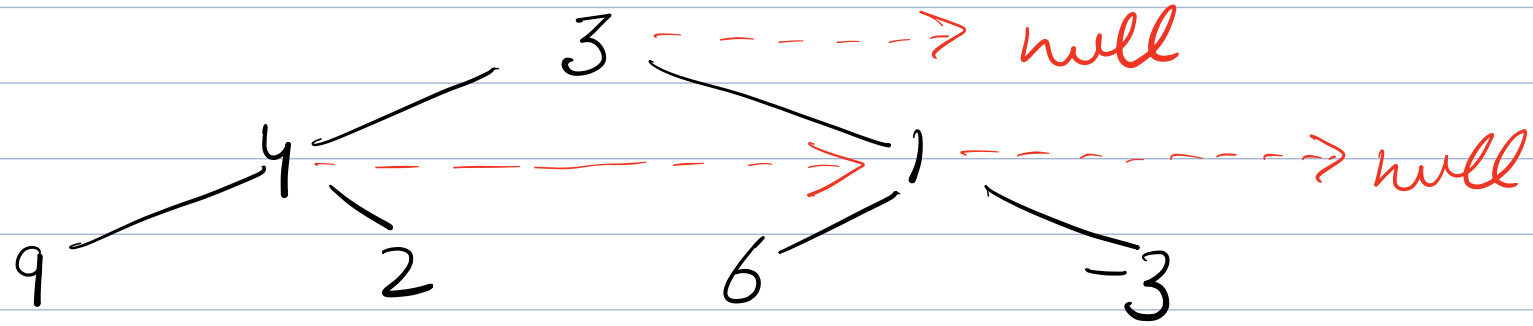
TC: O(N)

Q3 Populate right next pointers.



data -
left
right
next

Given is a **perfect tree**



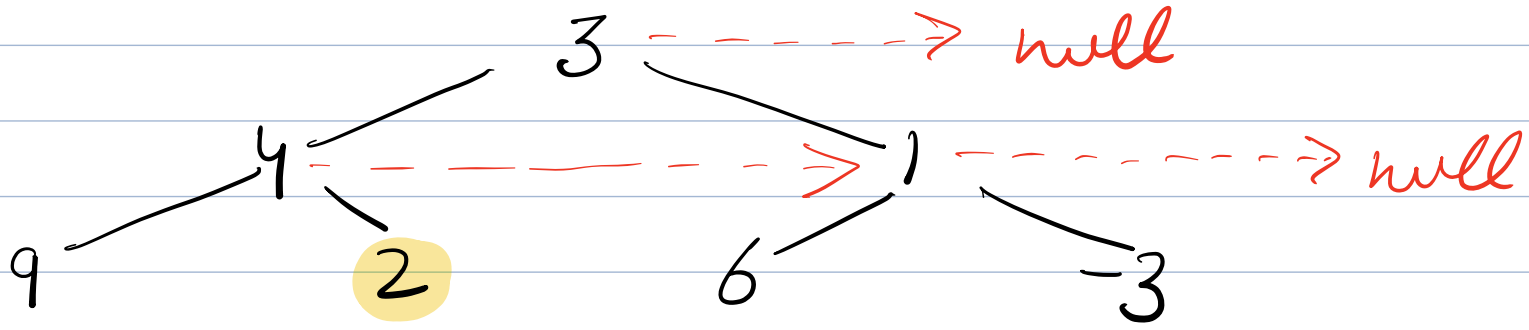
Idea: We need to do level order traversal

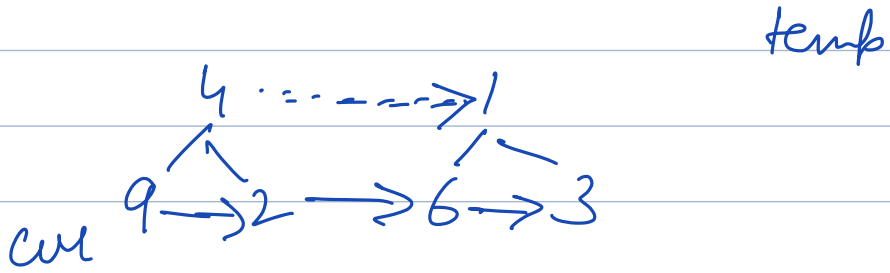
TC: O(n)

SC: O(n)

Challenge: O(1) SC

Idea :



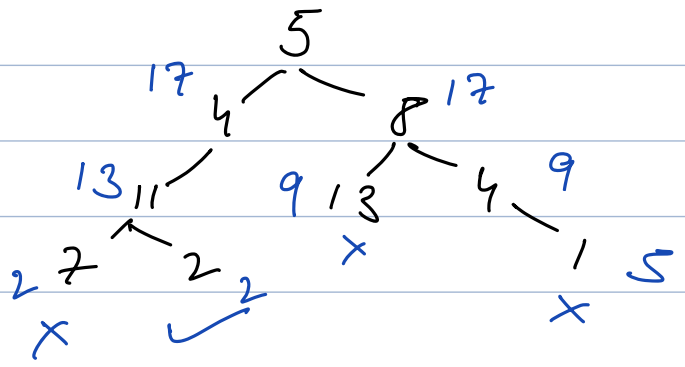


Code

```
while (cur != null && cur.left != null){
    temp = cur
    while (temp != null) {
        temp.left.next = temp.right
        if( temp.next != null)
            temp.right.next = temp.next.left
        temp = temp.next
    }
    cur = cur.left
}
```

TC: O(N)
SC: O(1)

Q Root to leaf path = k



k = 22

ans = true

Code

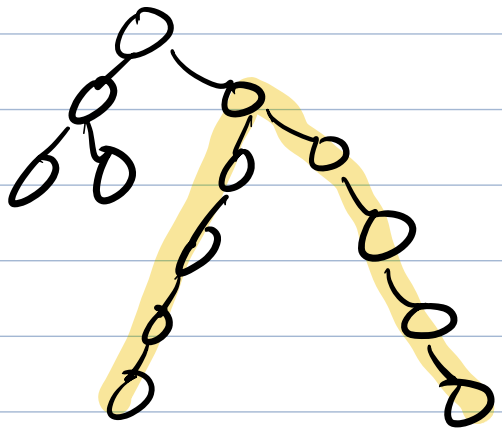
```
bool hasSum (Node node, int sum) {
    if (node == null)
        return false
    if (node.left == null && node.right == null) {
        if (sum == node.val)
            return true
        else
            return false
    }
    x = hasSum (root.left, sum - root.data)
    y = hasSum (root.right, sum - root.data)

    return (x || y)
}
```

OR operator

TC : O(N)

SC : O(height)

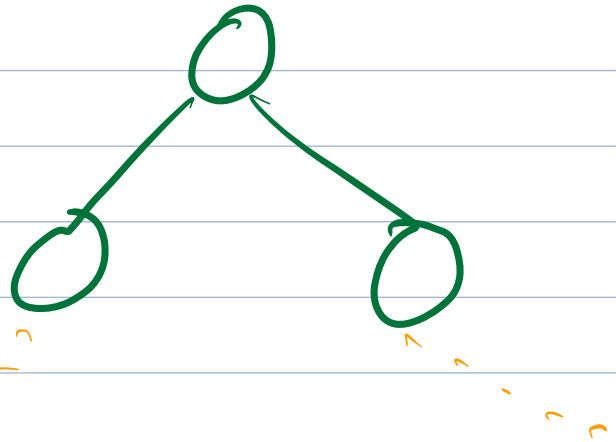
Q5 Find diameter (longest path) of the tree



ans = 8

Obs: Path will look like

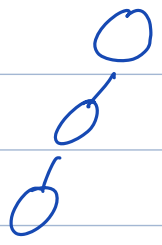
Lets find biggest distance starting at each node



ans(node): represent max distance from node to any leaf in the subtree

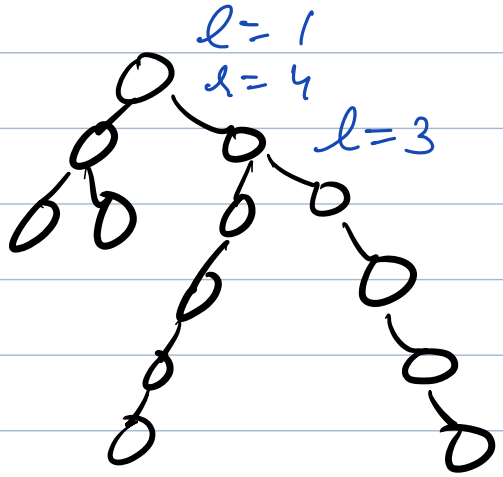
ans(root) = max (ans(left), ans(right)) + 1

dia = max (dia, ans(left) + ans(right) + 2)



Code

```
int diameter = 0                    // global var
int height (Node node) {
    if (node == null)
        return -1
    l = height (node.left)
    r = height (node.right)

    diameter = max (diameter, l+r+2)

    return max(l,r) + 1
}

int calc_dia (Node root) {
    diameter = 0
    height (root)
    return diameter
}
```



dia = ~~0~~ ~~7~~ 8

(done)