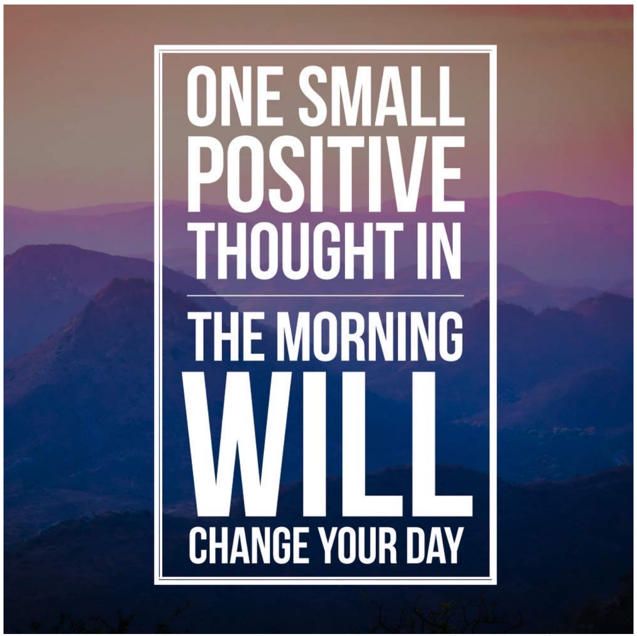
Lab session on

Binary Trees



Content

01. Equal tree partition

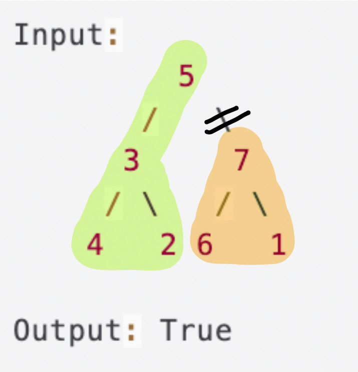
02. Path sum = k

03. Check height balanced

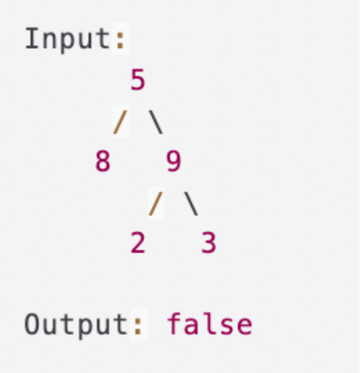
04. Construct Binary tree from inorder & postorder

Equal Tree Partition

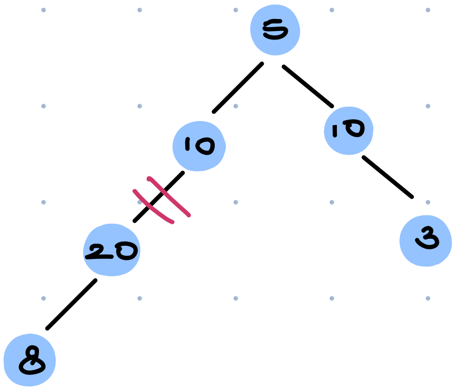
< *Question* > : Given the root of a binary tree, return true if the tree can be split into two non-empty subtrees with equal sums, or false otherwise.



```
Input:
       5
      /
     3     7
    / \   / \
   4   2 6   1

Output: True
```



```
Input:
     5
    / \
   8   9
      / \
     2   3

Output: false
```



Yes, it is possible to split tree into part with equal sum.

Find total sum and then find half of it while traversing.

* Observation

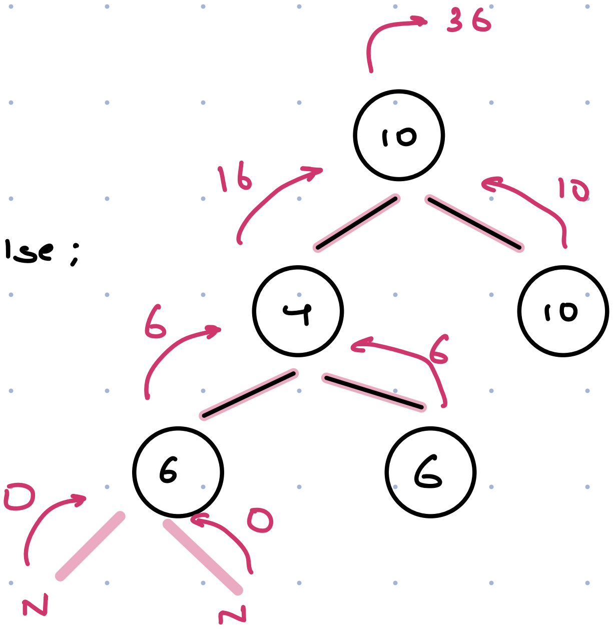
01. If sum of entire tree is odd → Not possible
↳ is even → check

02. Sum of subtree if that is equal to $\frac{tsum}{2}$ or not.

ans=false

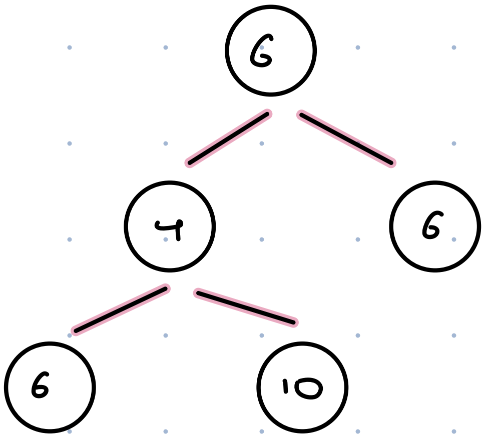
Totalsum=36

```
class Solution {

    boolean ans

    boolean solve ( root )
        ans= false;
        totalsum = sum(root)
        if ( totalsum % 2 ==1) return false;
        subsum ( root , totalsum);
        return ans;
    }

    int subsum ( root , totalsum)
        if (root == Null ) return 0;
        int lsum = sub sum (root.left , totalsum);
        int rsum = sub sum (root.right , totalsum );
        if ( lsum == totalsum/2 || rsum== totalsum/2 )
            ans= true;
        }
        return lsum + rsum + root.val;
    }
```



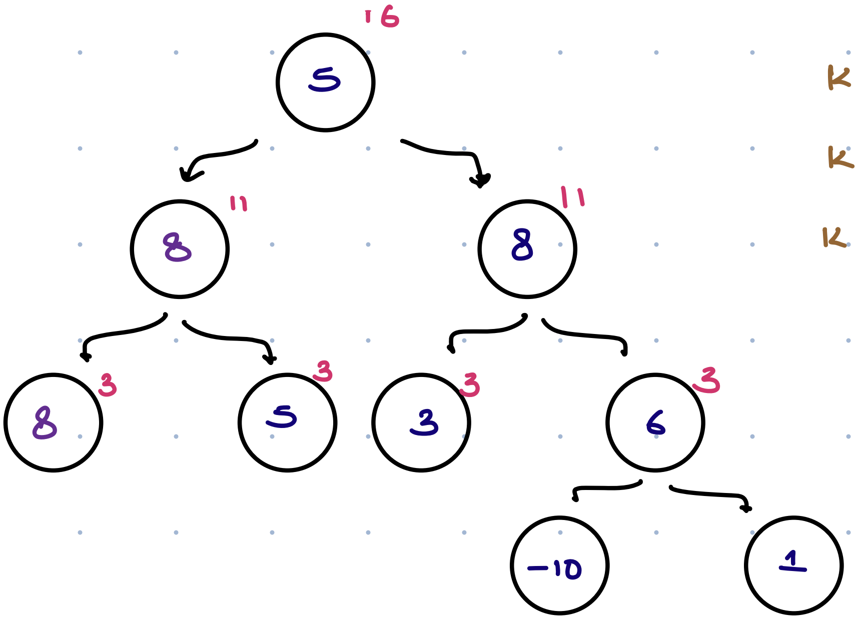
Travel & Change

```
    int sum (root )
        if (root == Null ) return 0;
        int lsum = sum (root.left );
        int rsum = sum (root.right);
        return lsum + rsum + root.val;
    }

}
```



Path sum

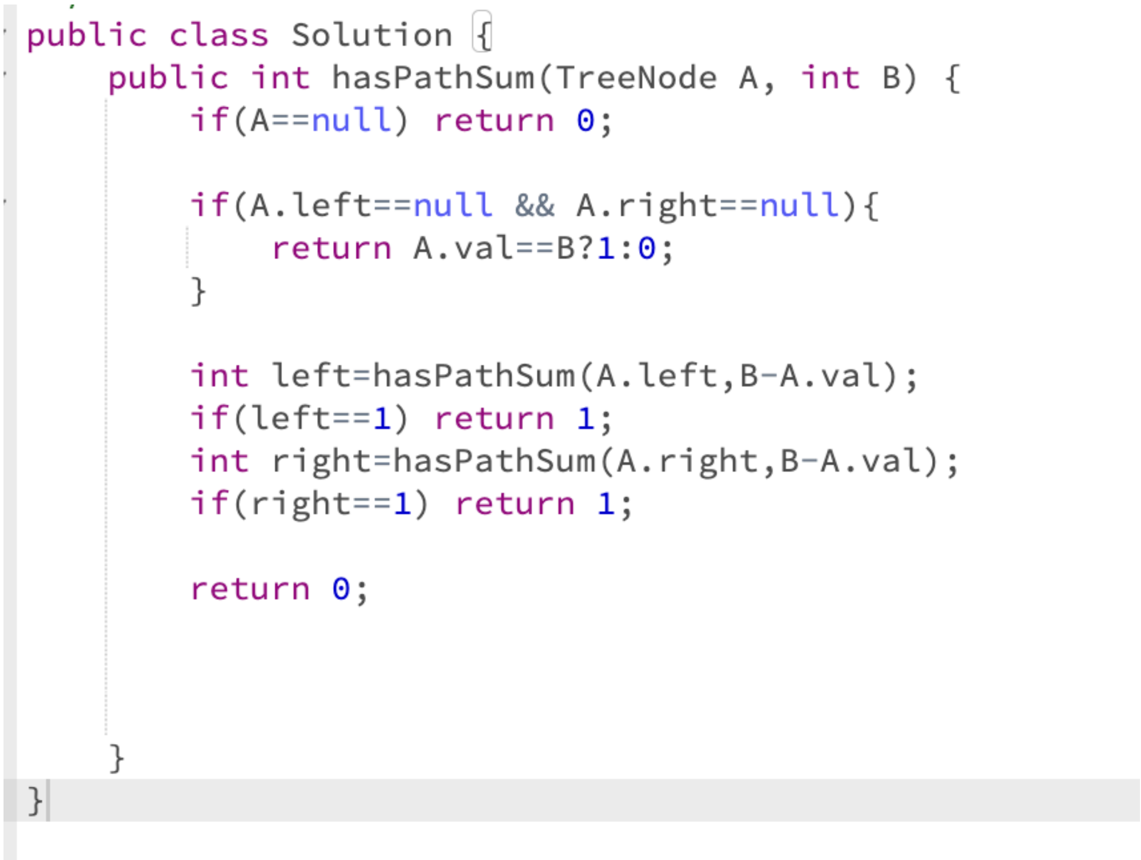
* Given a binary tree & an integer K. Determine if there exist a root to leaf path sum = K



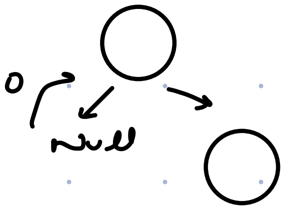
K = 16 → true

K = 10 → false

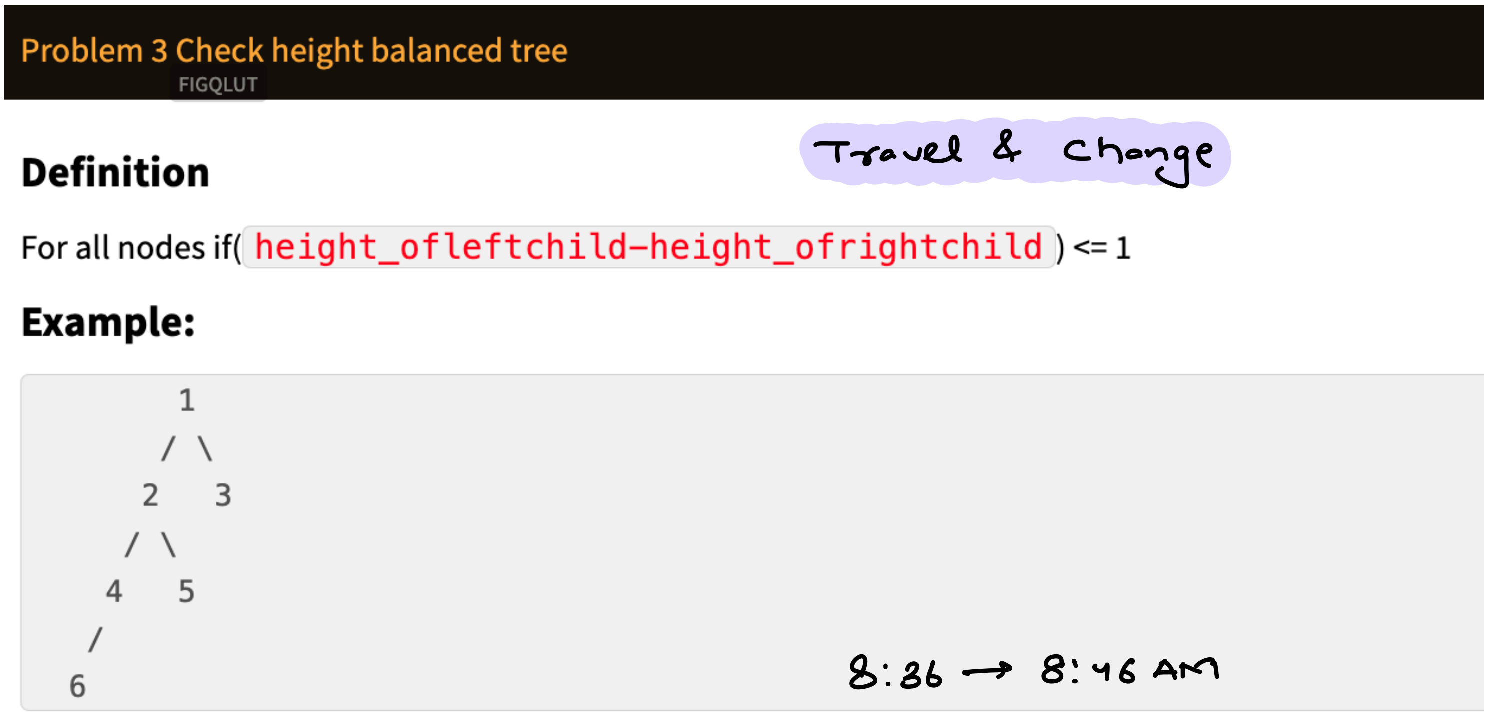
K = -2 → false



```
public class Solution {
    public int hasPathSum(TreeNode A, int B) {
        if(A==null) return 0;

        if(A.left==null && A.right==null){
            return A.val==B?1:0;
        }

        int left=hasPathSum(A.left,B-A.val);
        if(left==1) return 1;
        int right=hasPathSum(A.right,B-A.val);
        if(right==1) return 1;

        return 0;



    }
}
```



8:19 → 8:30 AM



Problem 3 Check height balanced tree

FIGQLUT

Travel & Change

Definition

For all nodes if(`height_ofleftchild-height_ofrightchild`) <= 1

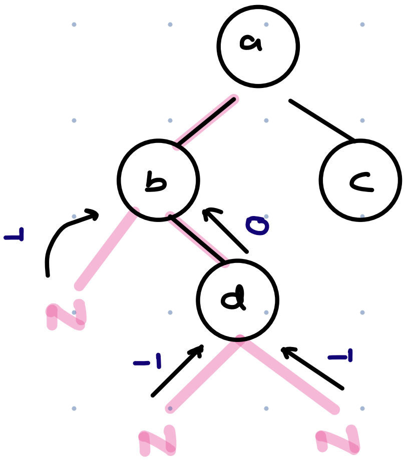
Example:

```
        1
       / \
      2   3
     / \
    4   5
   /
  6
```

8:36 → 8:46 AM

* Height of tree (in terms of Edges)

```
int height ( Node root )
    if (root == Null) return -1;
    int lh = height (root. left);
    int rh = height (root. right);

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



Ht in terms of edges . base case → return -1

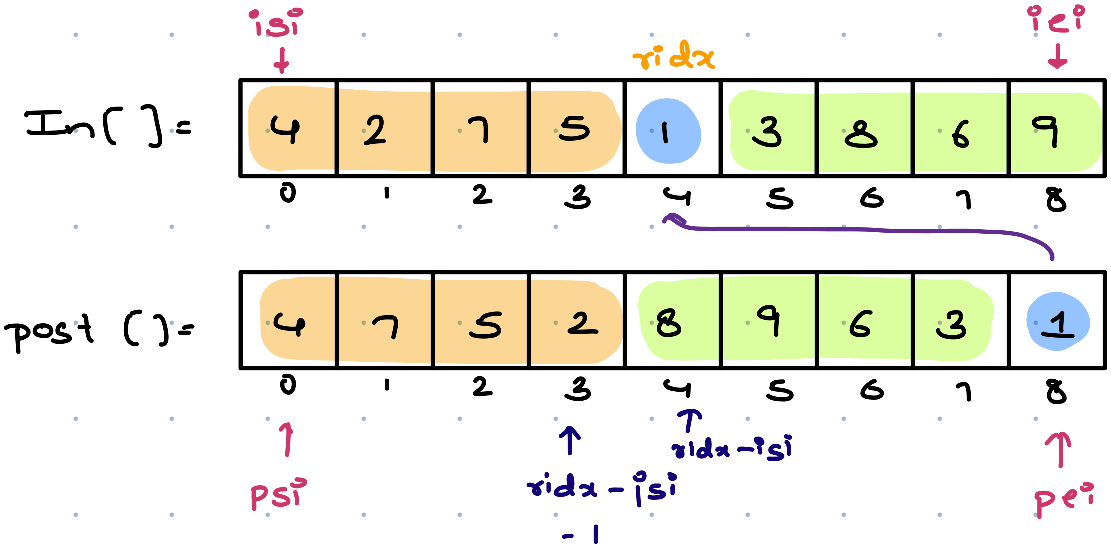
Ht in terms of nodes, base case → return 0

```
boolean isbal;
int checkbalance ( root )
{
      isbal = true;
      height (root);
      return isbal;
}

int height ( Node root )
{
    if (root == Null) return -1;
    int lh = height (root. left);
    int rh = height (root. right);
    if ( Math. abs (lh - rh) > 1 ) isbal = false;
    return max( lh, rh) + 1;
}
```

LNR LRN

* Construct binary tree from inorder & postorder.

Note → we will always have distinct values



In[ ]=	4	2	7	5	1	3	8	6	9
	0	1	2	3	4	5	6	7	8

post ( )=	4	7	5	2	8	9	6	3	1
	0	1	2	3	4	5	6	7	8

01. Find the root node = post[pei];

02. Find root node in inorder array = ridx

$$\text{In[ ]} = \begin{cases} \text{LST} = isi \quad ridx-1 \\ \text{RST} = ridx+1 \quad iei \end{cases}$$

$$\text{count} \begin{cases} a = isi \\ b = ridx-1 \end{cases}$$

count of ele = $b - a + 1$

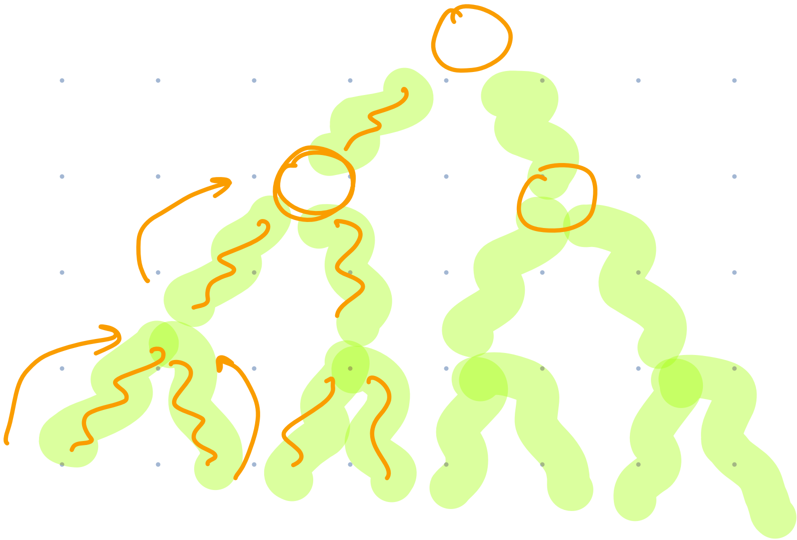
$= ridx - 1 - isi + 1$

$\Rightarrow ridx - isi$

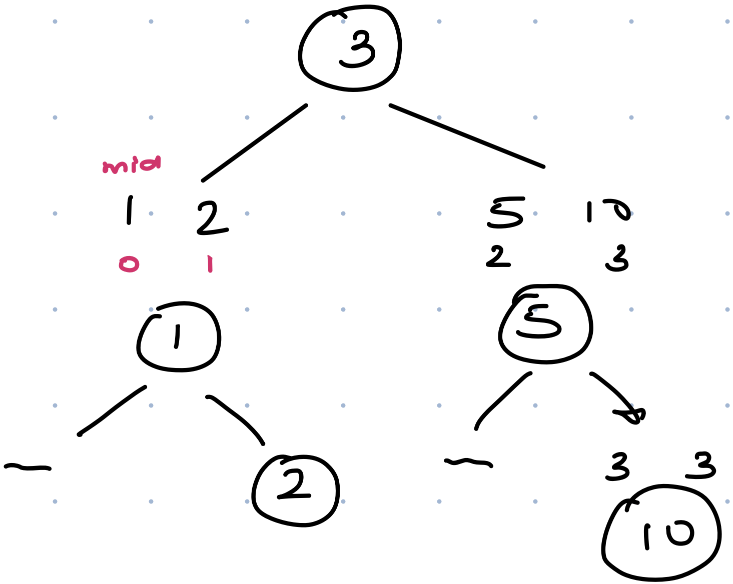
post( ): LST = $\{psi \ , \ psi + (ridx - isi) - 1\}$

RST = $\{psi + (ridx - isi) \ , \ pei - 1\}$

```
                                   isi   iei   psi   pei
                                    ↓     ↓     ↓     ↓
Node construct ( in[], post[],  0,  n-1 ,  0 ,  n-1 )

    if (isi > iei || psi > pei) return null;

    Node root = post[pei];

    int ridx = -1;

    for ( i=isi ; i≤ iei ; i++) {
        if ( In[i] = = root.val)        }
            ridx = i;                   }    HashMap
            break;                      }
        }
    }
root.left = construct (in, post, isi, ridx-1, psi, psi+(ridx-isi)-1);
root.right = construct (in, post, ridx+1, iei, psi+(ridx-isi), pei-1);
return root;
}
```



A( ) = {1 2 3 5 10}


3
mid
1 2
0 1
5 10
2 3
1
~
2
5
~
3 3
10