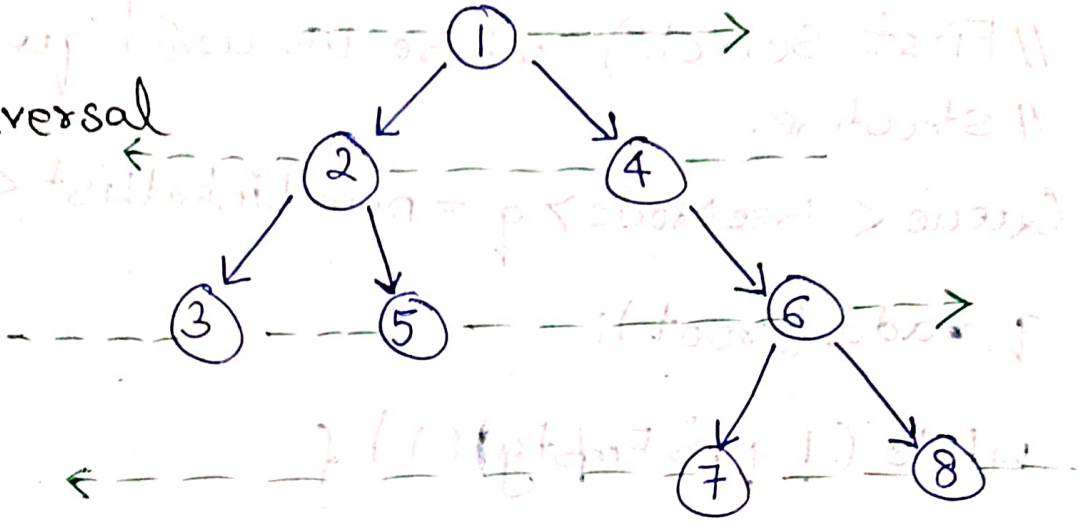
# Traversals in Tree

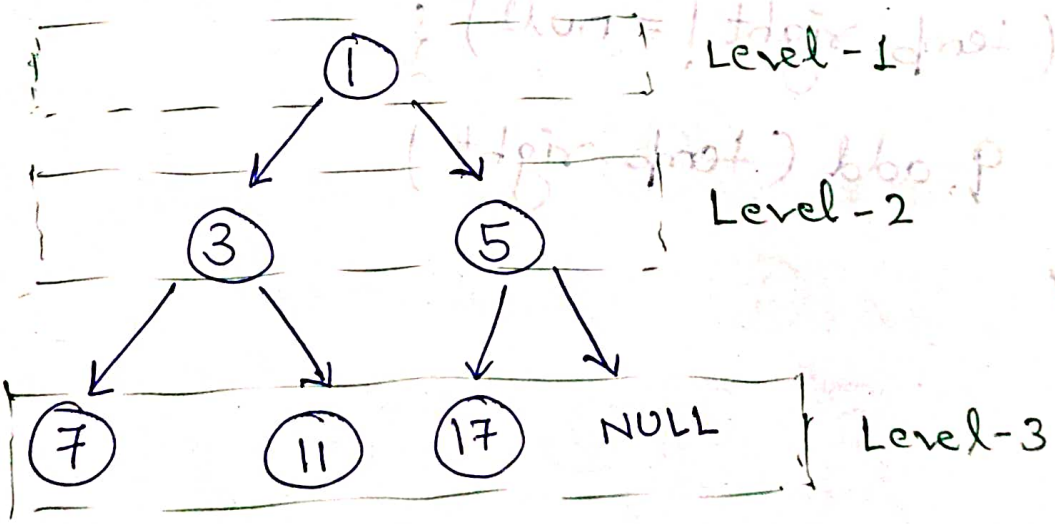
1. Zig-zag traversal
   or
   Spiral traversal




Soln: 1 4 2 3 5 6 8 7

Approach: Taking help of Level Order traversal. and a flag which tells us that in which direction we have to print i.e left to right or right to left.

2. Level Order Traversal




Soln:

1

3 5

7 11 17

```
void LOT (TreeNode root) {
// We'll do the traversal using BFS (Breadth
// First Search) where we use queue data
// structure.
Queue < TreeNode > q = new LinkedList<>();
q.add(root);
while (!q.isEmpty()) {
    TreeNode temp = q.peek();
    q.remove();
    System.out.print(temp.val + " ");
    if (temp.left != null) {
        q.add(temp.left)
    }
    if (temp.right != null) {
        q.add(temp.right)
    }
}
}
```

Soln: 1 3 5 7 11 17

But, we want to print the data level-wise.
i.e after each level printing should start from next line.

i.e
```
    1
  3   5
 7  11  17
```

For, that we need a separator.
Refer to the updated code.

3. Boundary Traversal

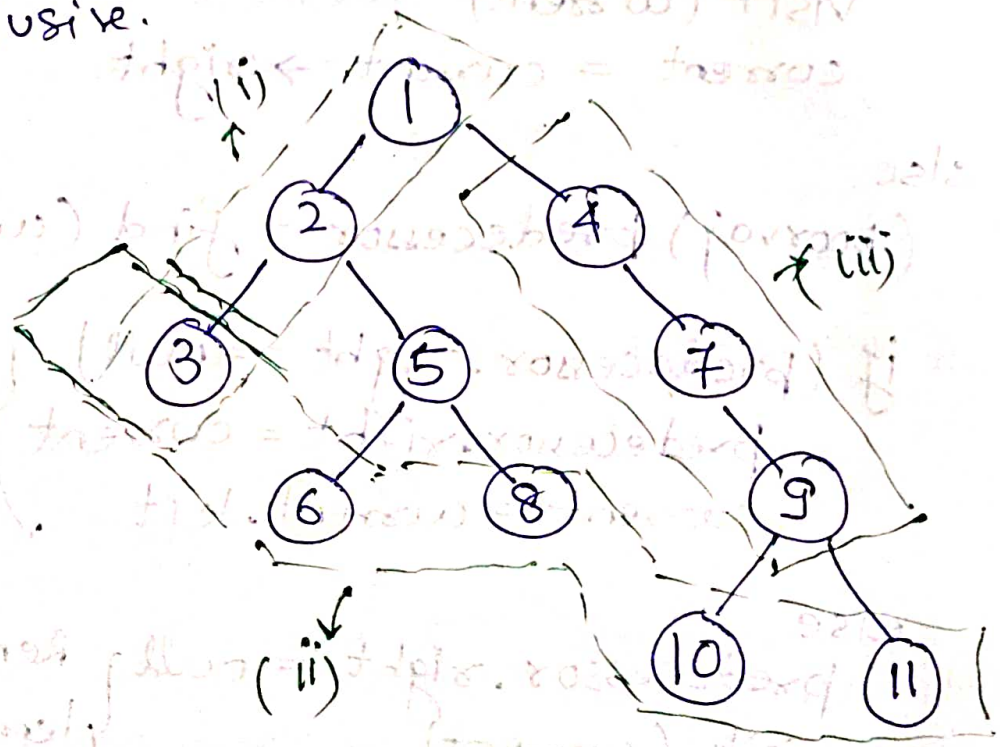
We'll divide this problem in three parts:

i) First print left part of the tree where leaf node is exclusive.

ii) Secondly, print the leaf nodes.

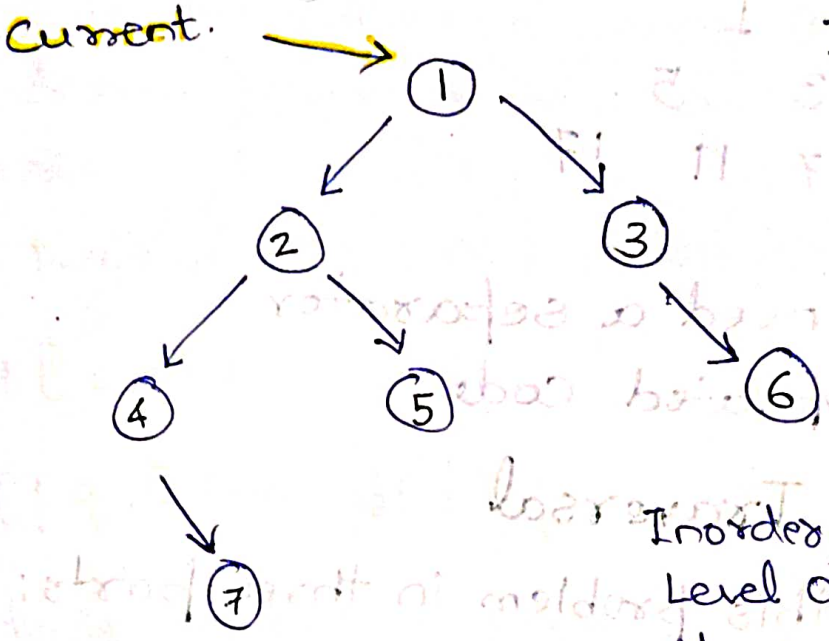
iii) Lastly, print the right part of the tree but from bottom to top where leaf node is exclusive.

Ex




Soln: 1 2 3 6 8 10 11 9 7 4

# 4. Morris Traversal




It traverse the tree in

TC → O(n)

SC → O(1)

Inorder, postorder, Level order, ... takes space but morris donot

Algorithm:

1) Current = root

2) While (current != null)

if left not exists

visit (current)

current = current → right

else

(poorvaj) predecessor = find (current)

if (predecessor.right == null)
predecessor.right = current
current = current.left

} Creating temporary links using this

else

predecessor.right = null
visit (current)
current = current.right

} Removing temporary links

Dry Run: ⟹ current pointer on the root i.e 1 initially.

if → false as left of 1 exists i.e 2

else.

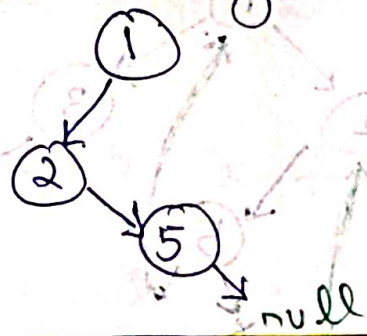
find predecessor of 1.

↓

ek left jao and then right jaate raho jab tak right ka right null naa ho jaaye.

i.e


then right till we donot get null

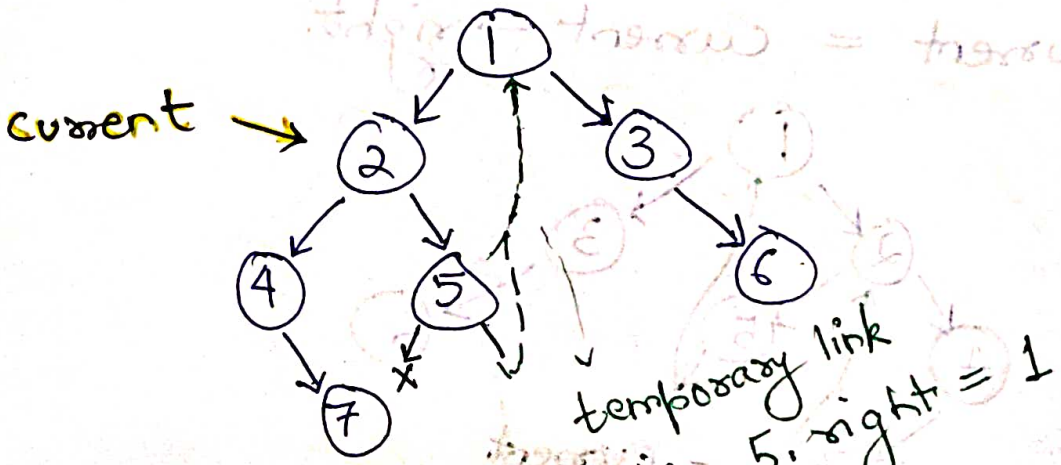



So, predecessor = 5

if 5.right == null [true]

so, 5.right = 1

and current will move to 2.

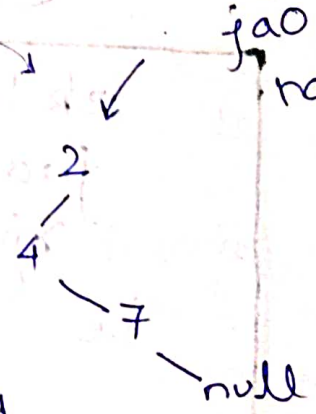


⟹ if cond^n → false as 2.left exists.

else.

predecessor of 2 → ek baar left main jaao and then right main jaate jao jab tak null naa mill jaaye.

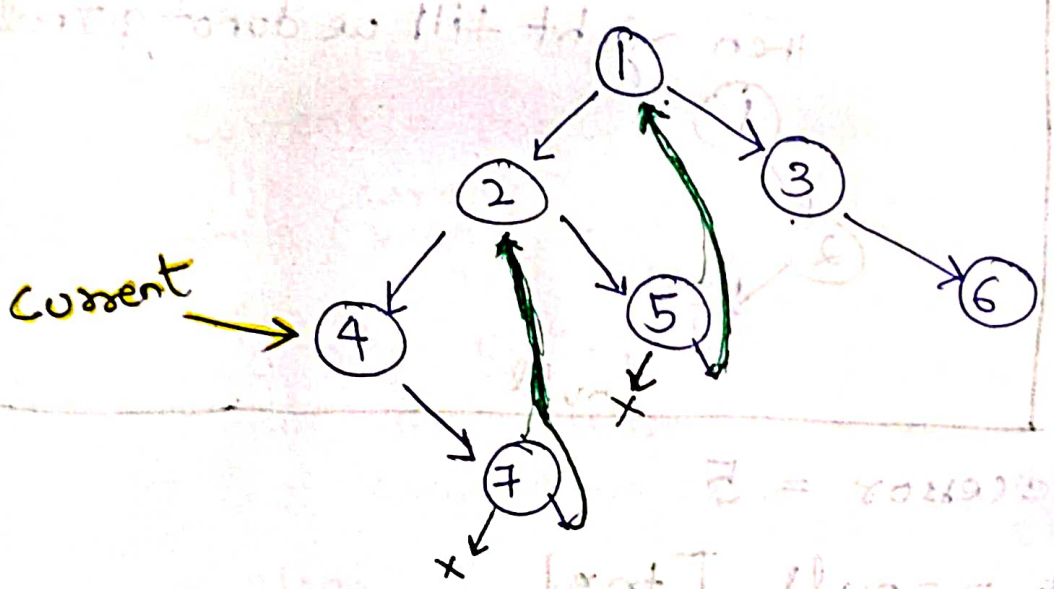
i.e predecessor = 7




if (7.right == null)

7.right = 2

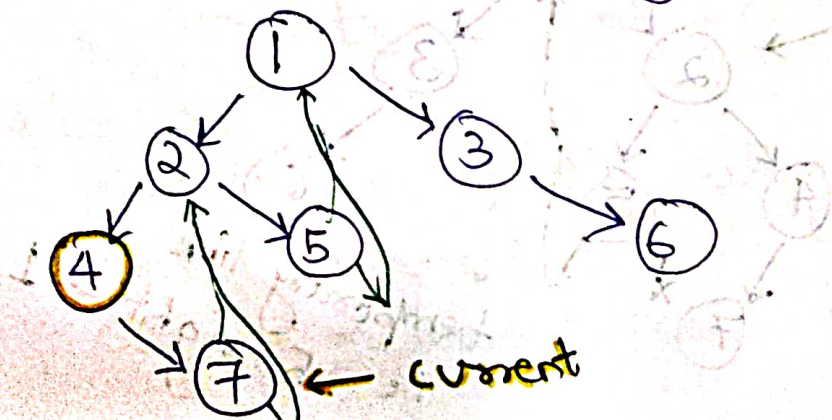
∴ current will move to 4




⟹ if cond^n → True.

visit (4) i.e. print (4)

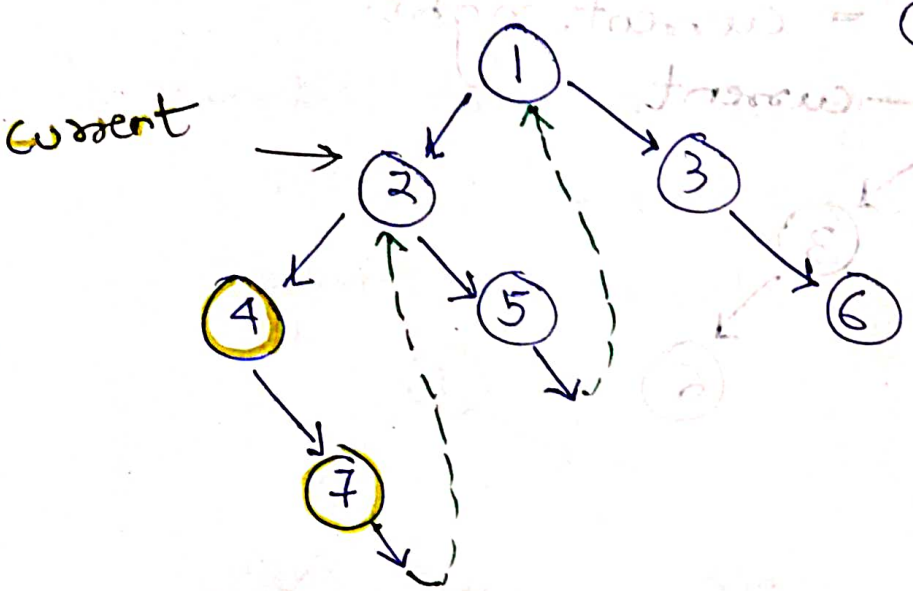
current = current → right.



⇒ if → True as 7.left == null.

So, visit (7) i.e print (7)

current = current.right.




⇒ if cond$^n$ → false

else.

predecessor of 2 is 7.

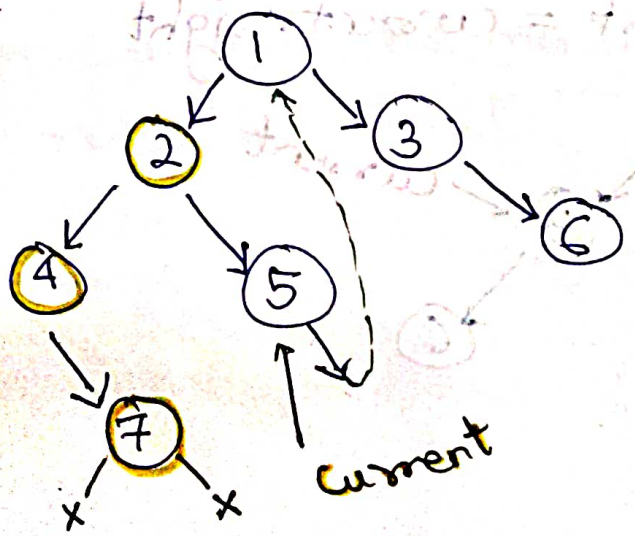
if (7.right == null) → false

else.

7.right == null

visit (2) i.e print (2)

current = current.right

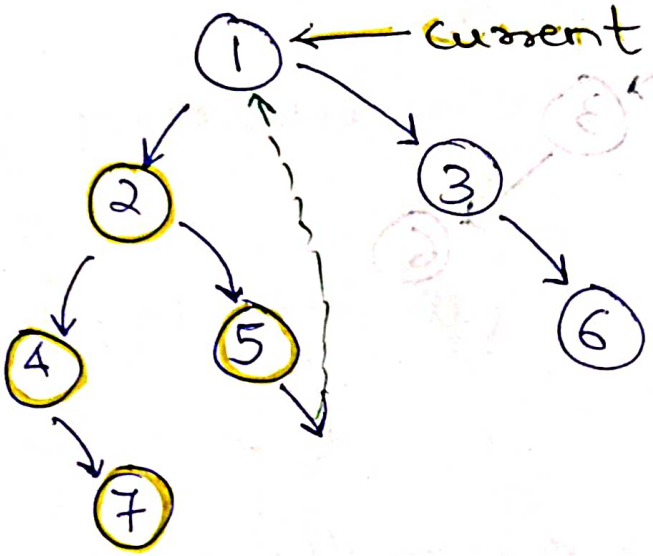


⇒ if condition true

so,

visit (5) i.e print (5)

current = current.right




⇒ if condit$^{n}$ ~~true~~ false

else

predecessor of 1 is 5

if (5.right == null) → false

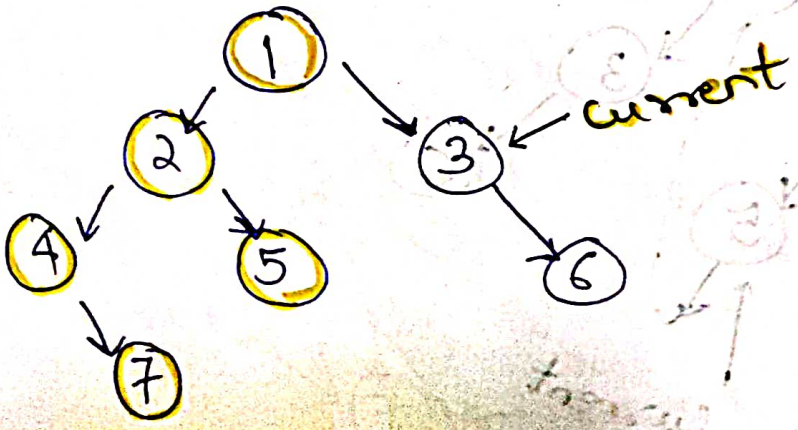
so,

else.

5.right = null

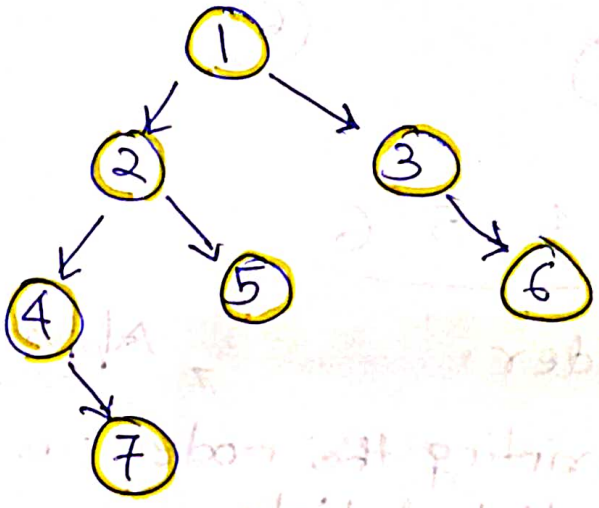
visit (1) i.e print (1).

Current = current.right



⇒ if condition true as 3.left == null
visit (3) i.e print (3)
current = current.right i.e at 6

⇒ if condit$^n$ true as 6.left == null
visit (6) i.e print (6)
current = current.right i.e null

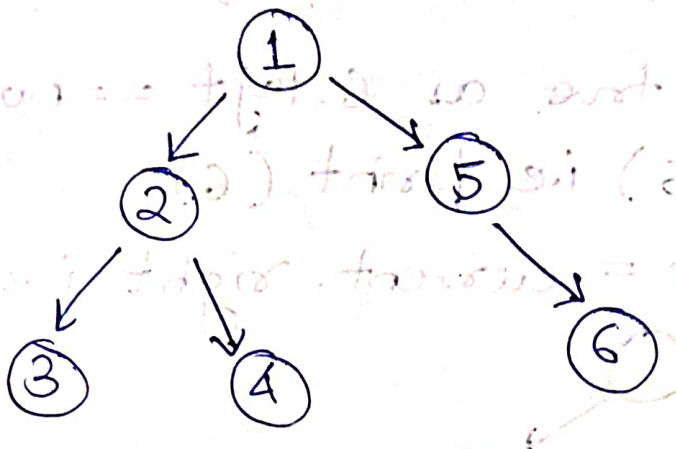



Soln 4 7 2 5 1 3 6

↑
Inorder Traversal output

if, we change the print statement then this algo can also be modified to print preorder traversal.

# Flatten a binary tree to Linked List




Soln 1 2 3 4 5 6

Preorder.

Approach - 1

So, in place of printing the node we have to just make a linked lists.

i.e
```
{Node nn = new Node (root.val)
 ---------
 ---------
 Preorder (root.left)
 Preorder (root.right)
```

But, we have to make the binary tree to Linked list without the usage of auxiliary data structure.

Expected TC. : O(n)

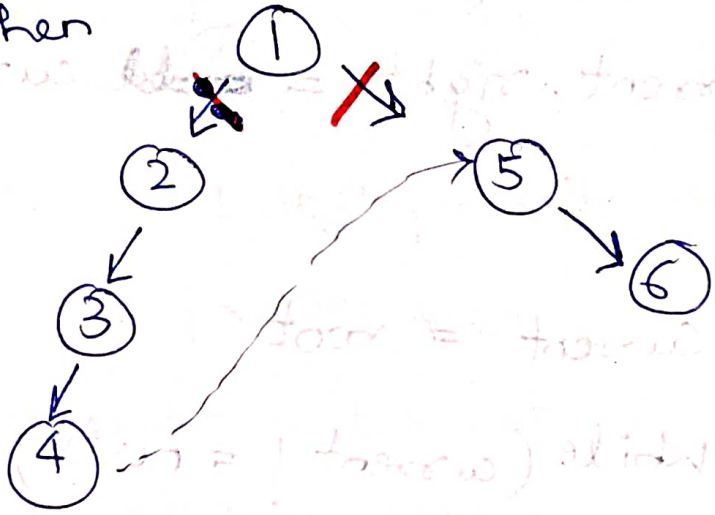
Expected SC : O(1)

i.e Approach - 1 is wrong.

Approach-2 : Use recursion.

i.e make right subtree as a linked list
& make left subtree as a linked list

then



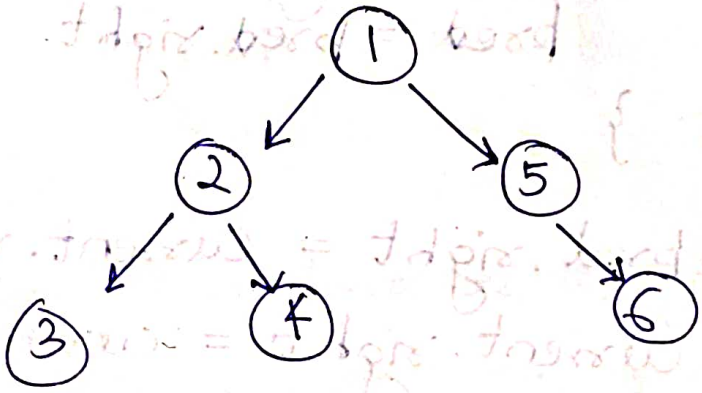

but
SC: O(n)

So,

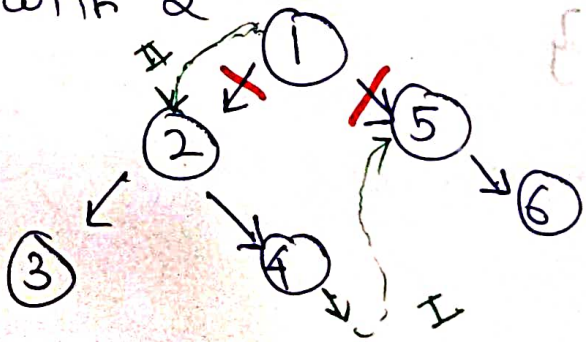
Approach : 3 : Morris Traversal.

as Morris traversal traverse the tree in O(1) space.

PreOrder :
N L R




link inorder predecessor of 1 i.e 4 to 5.
and (1) ke right & left pointer ko hatao
and link it with 2



i.e inorder predecessor nikaalo and.

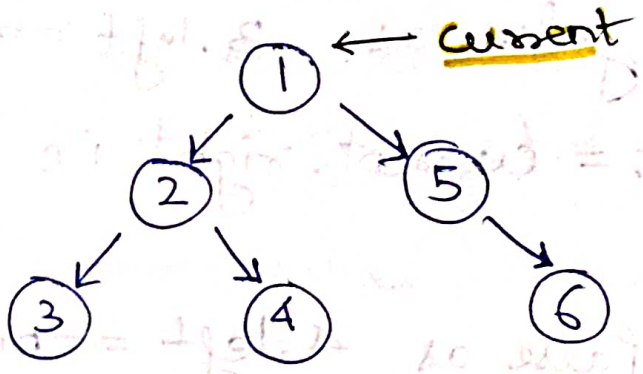
Step-I pred.right = current.right

Step-2 current.right = ~~null~~ current.left

Algo :-

```
current = root
While (current != null)
{
  if curr left exists.
  {
    Predecessor finding →
      prevd = curr.left
      While (pred.right != null) {
        pred = pred.right
      }
    pred.right = current.right
    current.right = current.left
  }
  current = current.right;
}
```

Dry Run:



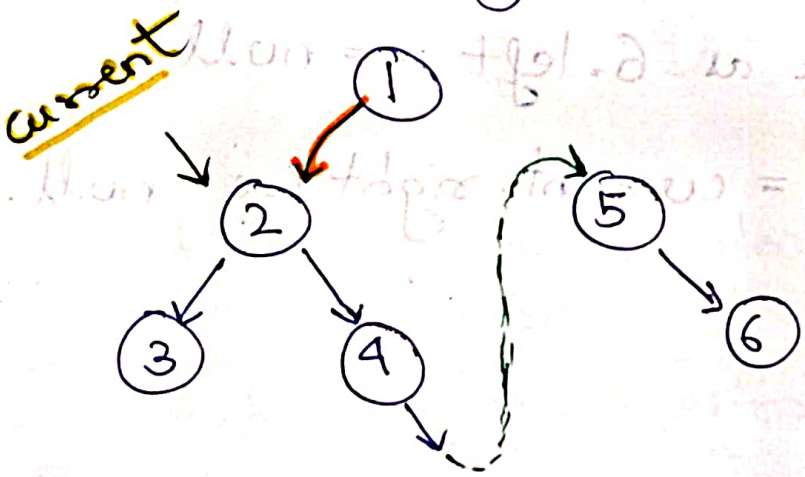

initially,
current = root.

⇒ if cond$^n$ → 1.left != null → true.

so, find predecessor of 1 i.e 4.

4.right = 1.right i.e 4 points to 5 now

1.right = 1.left



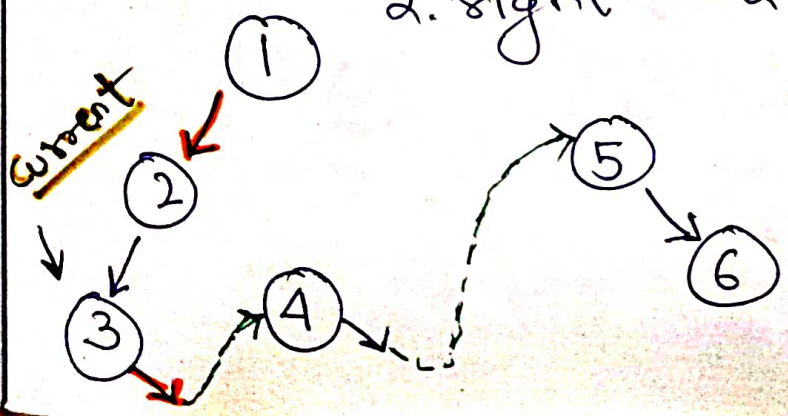

current = current.right

⇒ if cond$^n$ → true as 2.left != null

predecessor of 2 is 3

3.right = 2.right i.e 3 points to 4 now

2.right = 2.left




current = current.right

⇒ if cond^ → false as 3.left == null

So, current = current.right i.e 4

⇒ if cond^ → false as 4.left == null

So, current = current.right i.e 5

⇒ if cond^ → false as 5.left == null

So, current = current.right i.e 6

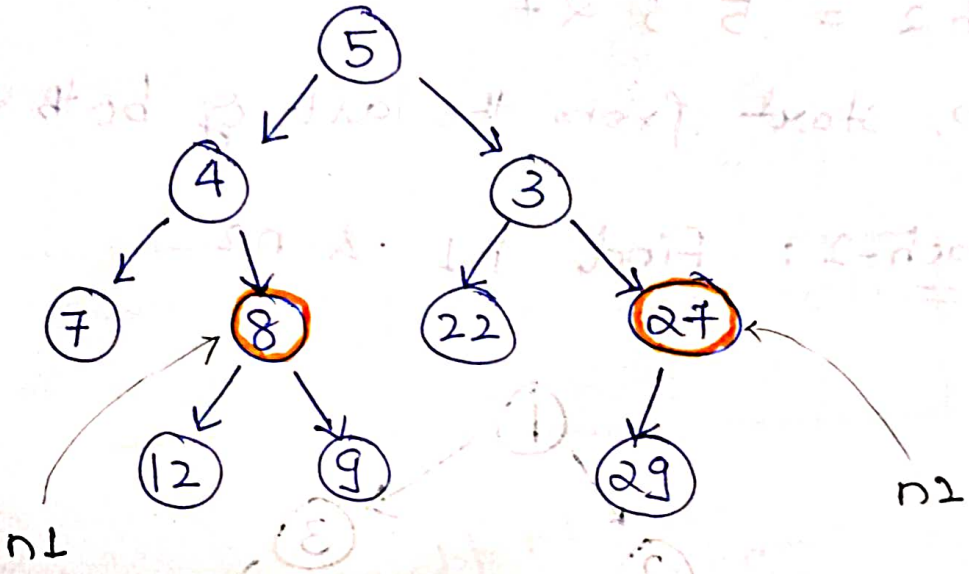
⇒ if cond^ → false as 6.left == null

So, current = current.right i.e null.

# Lowest Common Ancestor of a binary tree

Given two nodes values, n1 and n2. and we have to find their LCA.

Ex-1



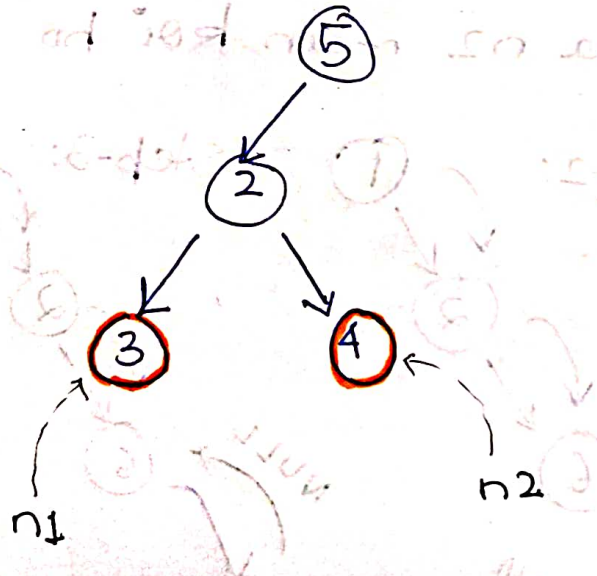

4, 5 are the ancestors of 8

whereas

3, 5 are the ancestors of 27.

but among these ancestors we need to take the lowest common ancestor of both n1 and n2.
i.e 5 here

Ex-2




2, 5 are the ancestors of 3

whereas

2, 5 are the ancestors of 4

but lowest common ancestor is 2.

Approach-1 : Find the path from root to n1 and root to n2.

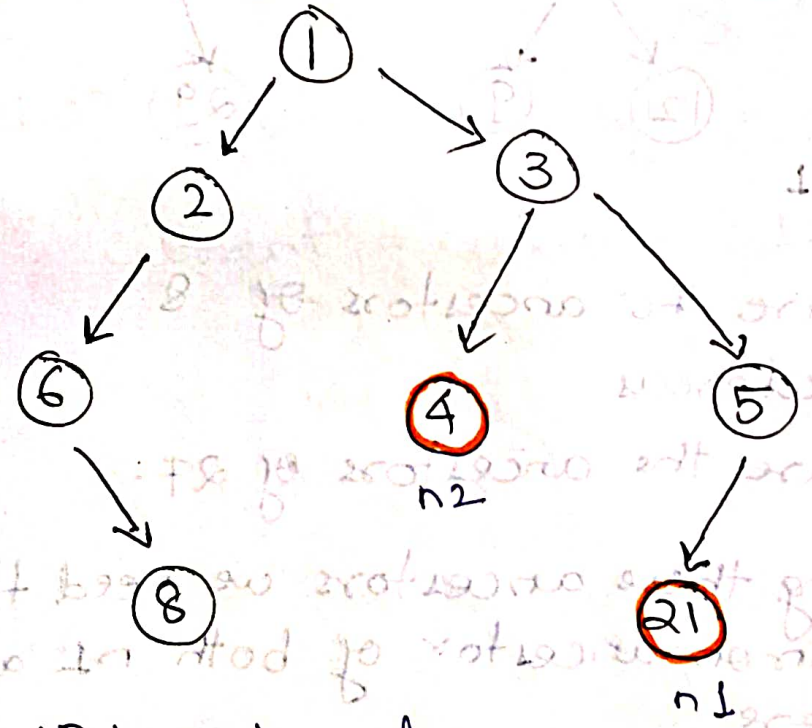
So, if we take example-1 as a reference, then.

Path 1 = 5 4 8

Path 2 = 5 3 27

Now, start from the last of both the paths.

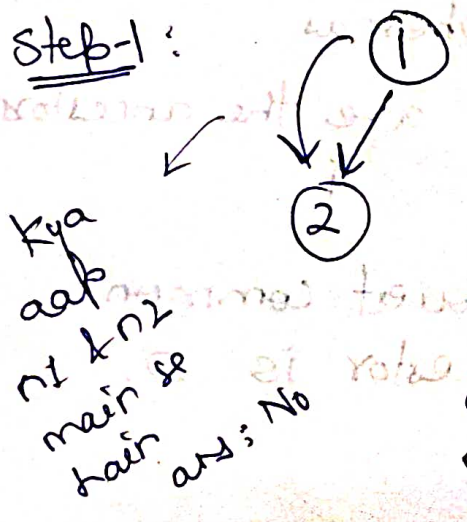
Approach-2 : Find n1 & n2.



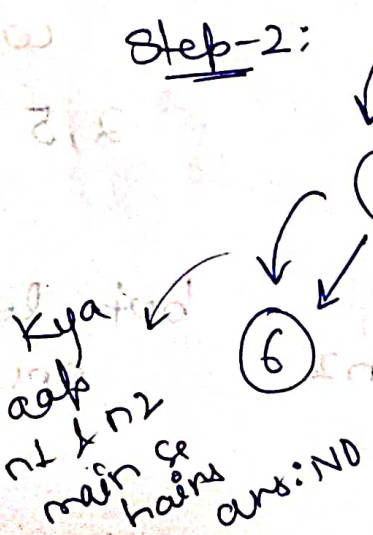

Ask quest^n to each node:

Kya aap n1 yaa n2 main koi ho.

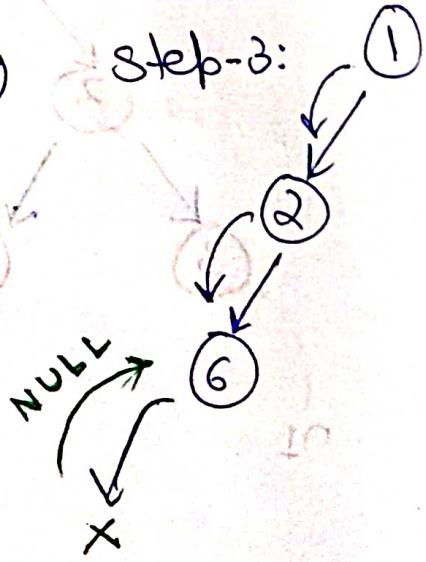
Step-1 :



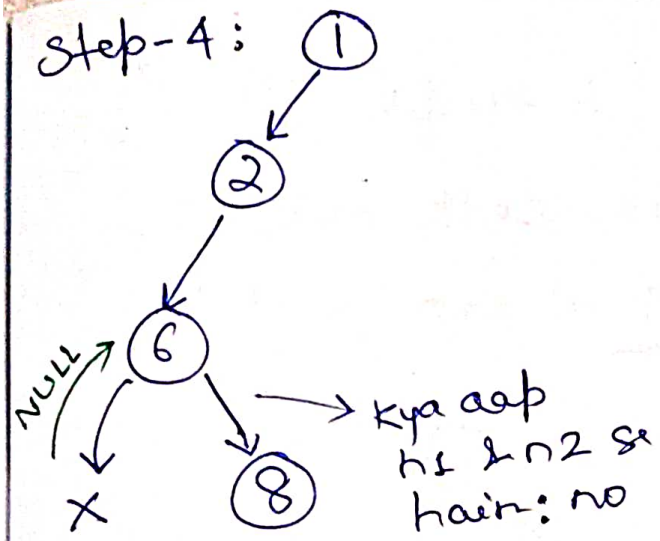

Step-2:



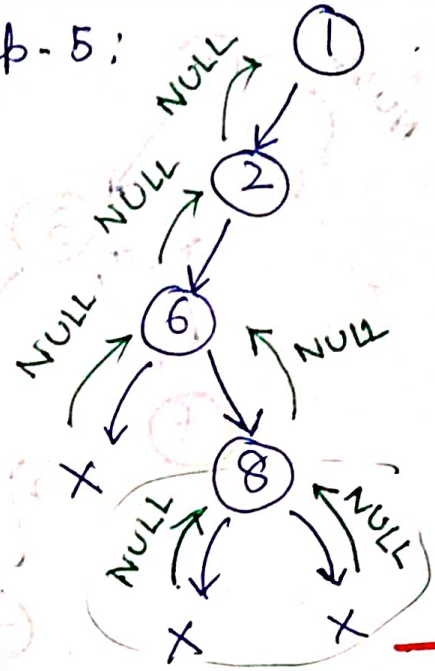

Step-3:



Step-4:

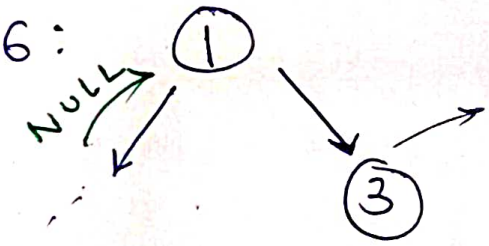



Step-5:

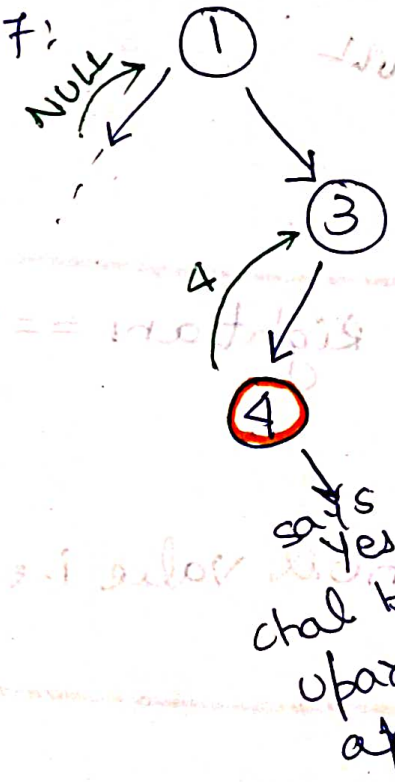



So, if Leftans & rightans are null then simply return null

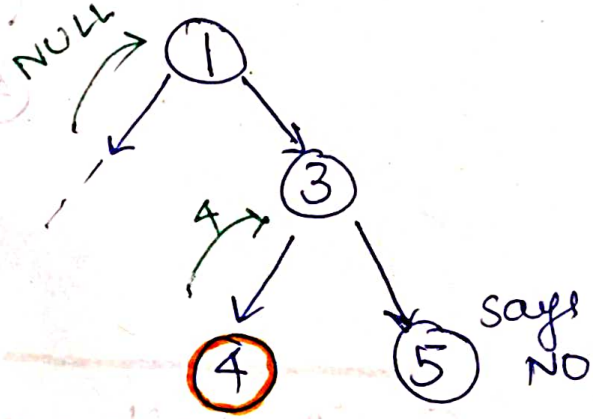
Step-6:



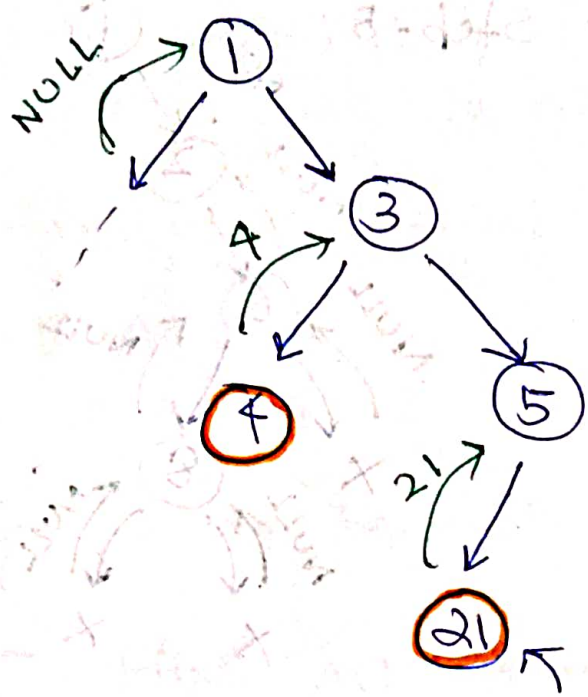

Step-7:




Step-8:

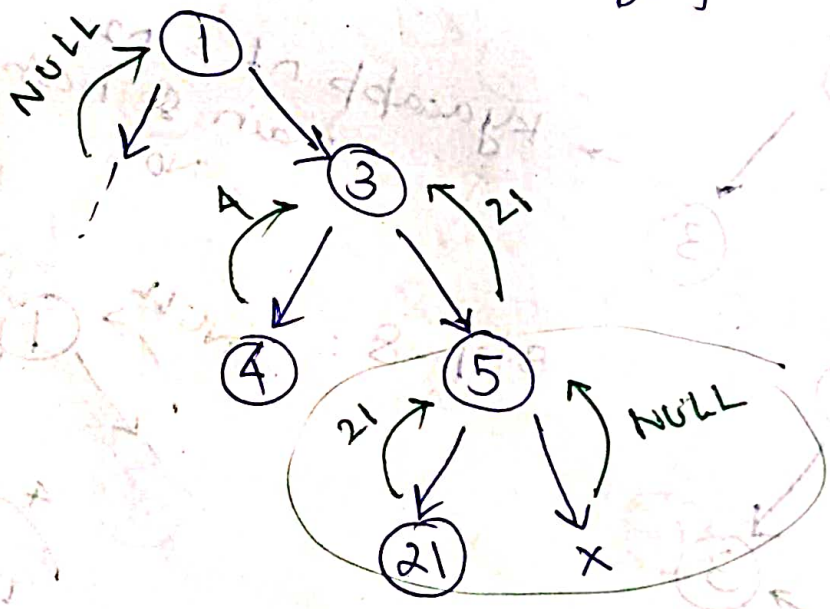


Step-9:




says yes: chal bhai apne aap ko bhej de.

Step-10:



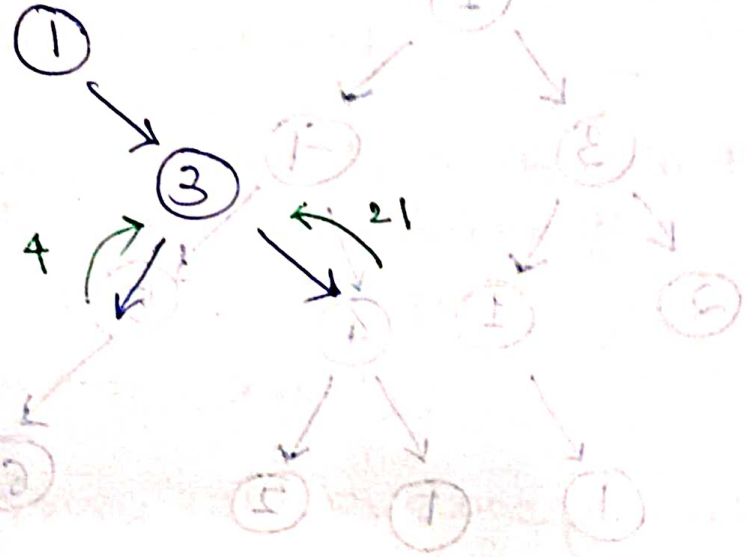

So, if Left ans != NULL & Right ans == NULL

or vice versa

then simply return non-null value i.e return 21 here

So, if leftans != null && rightans != null then that node will be our ans.

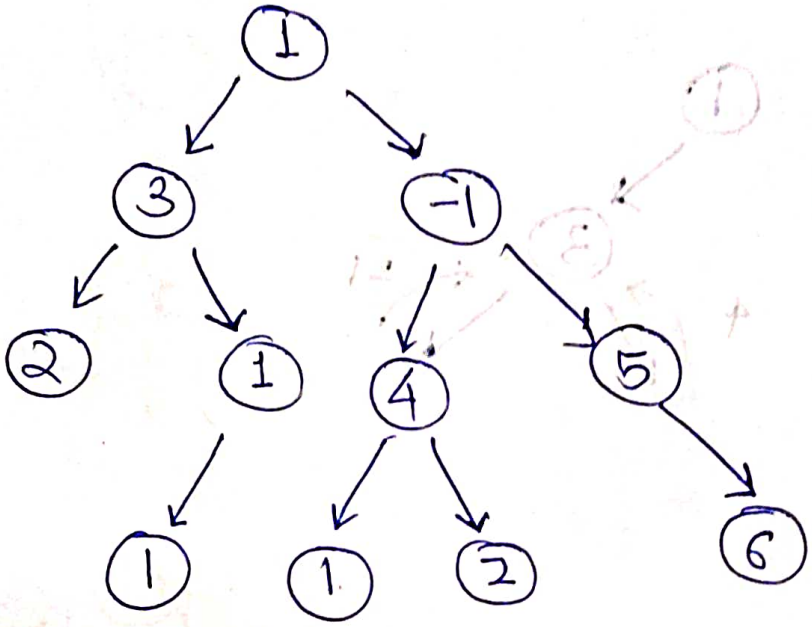
here ans = 3



# K Sum Path

Find the no. of paths in tree which have their sum equal to K.

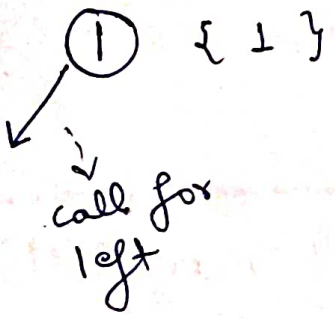
Ex



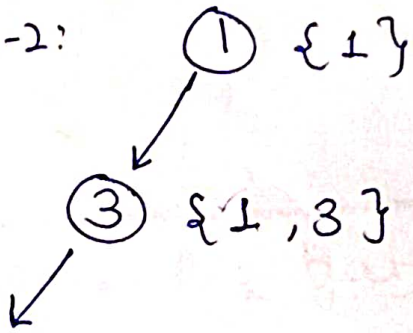

K = 5

Approach-1: Find all the paths. and find their sum respectively. and if their sum equal to k then increament count

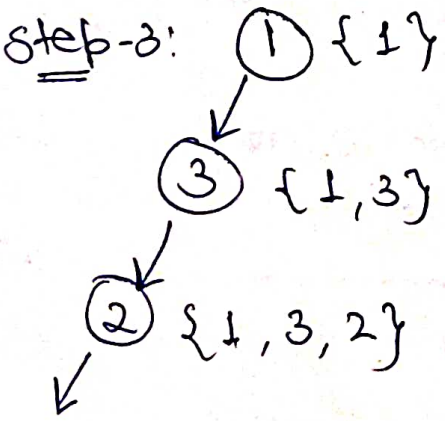
Step-1:



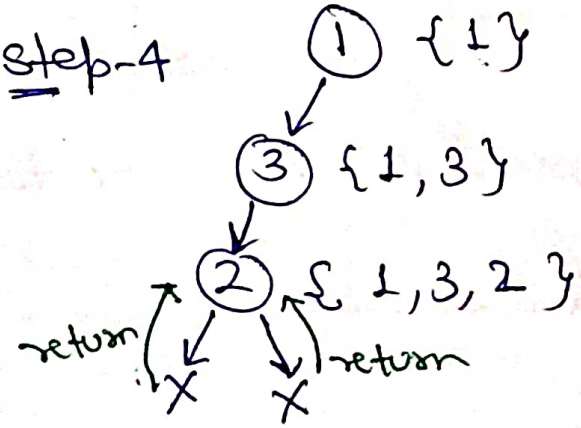

Step-2:




Step-3:




Step-4



size = 3
sum = 0

1st: i = 2, sum += 2 i.e sum = 2

2nd: i = 1, sum += 3 i.e sum = 5

i.e sum == 5

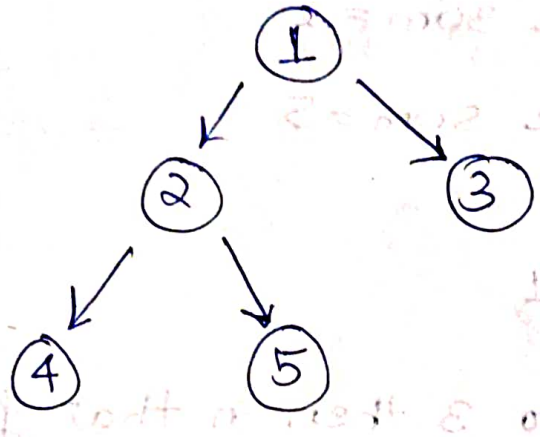
so, count += 1

Now when we go back to 3 then in that function frame path donot contains 2.
i.e backtrack the changes.

```
path.remove (size-1);
```

So, same process will be done for all the paths.

# Kth Ancestor of a Node

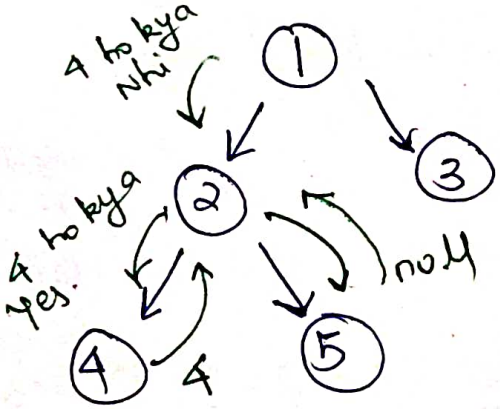



k=2
and node = 4.

Ancestors of 4 are : 2 & 1
but 2nd ancestor of 4 is 1.

Approach: Find the path from root node to n
-1
i.e 1 2 4
now, take element at the 2nd index from end.

Approach-2: Task-1 : find the node

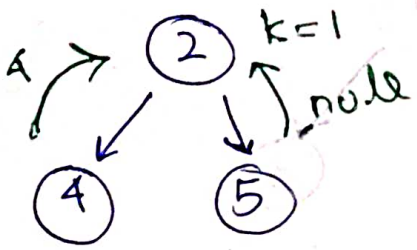



Step-1 Left ans != null & right ans == null
then k--

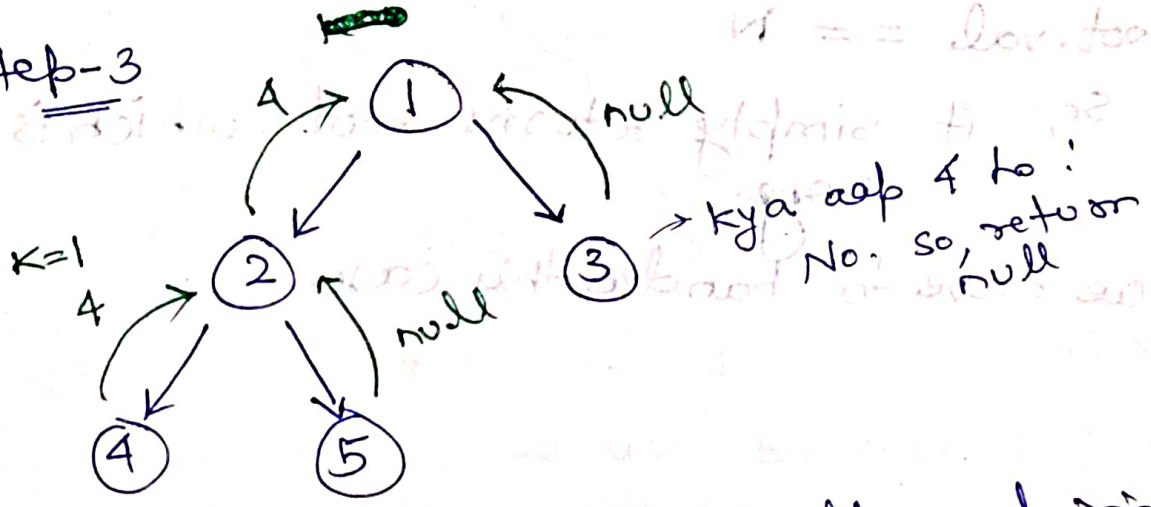
and check if k == 0 or not

```
if k==0 {
    return root;
}
```

Step-2 as, at 2., k!=0 so, simply return 4.




Step-3




as leftans = 4 i.e not null and rightans = null
ie k--=1 i.e k==0 now.

check: if(k==0)
return root → i.e 1

Logic:

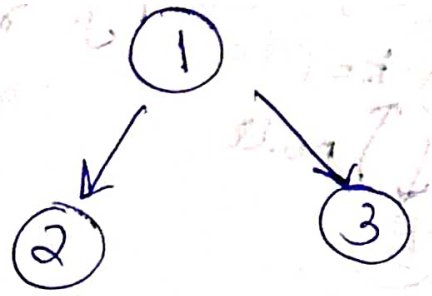
→ if ( leftans != null & rightans == null)
or
vice versa

{
k --= 1 ;

→ if ( k != 0)
return (!null value.)

Ex

Imp




K = 5

N = 1

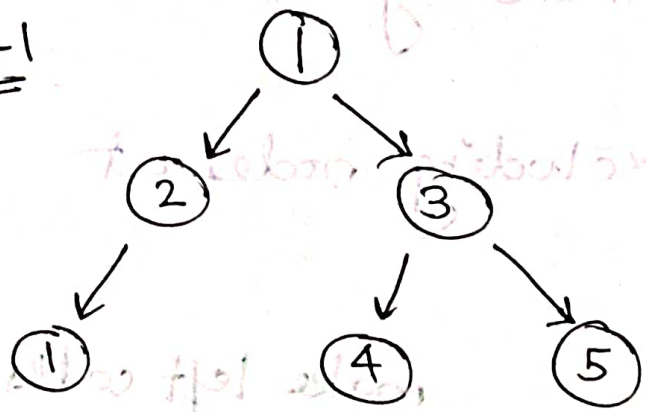
at. root.val == N

So, it simply returns root. which is wrong.

So, we have to handle this case.

# Maximum Sum of non-adjacent nodes

Ex-1




So, if we included 1 : in the maximum sum then we can ~~do~~ use ~~subtree~~ 2 & ~~subtree~~ 3

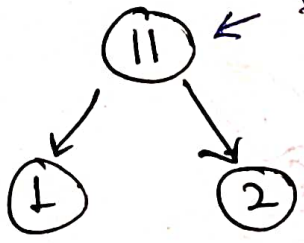
i.e sum1 = 1 + 1 + 4 + 5 ⇒ 11

if we had taken 2 then we cannot take 1 & 1 but can take 3. as 4 & 5 are also connected to 3 i.e we cannot take them.

So, sum2 = 2 + 3 ⇒ 5

ans = max(sum1, sum2) i.e 11

Ex-2




Take 11 :
ans = 11

Take 1 :
sum = 3
but ans = 11 (remains same)

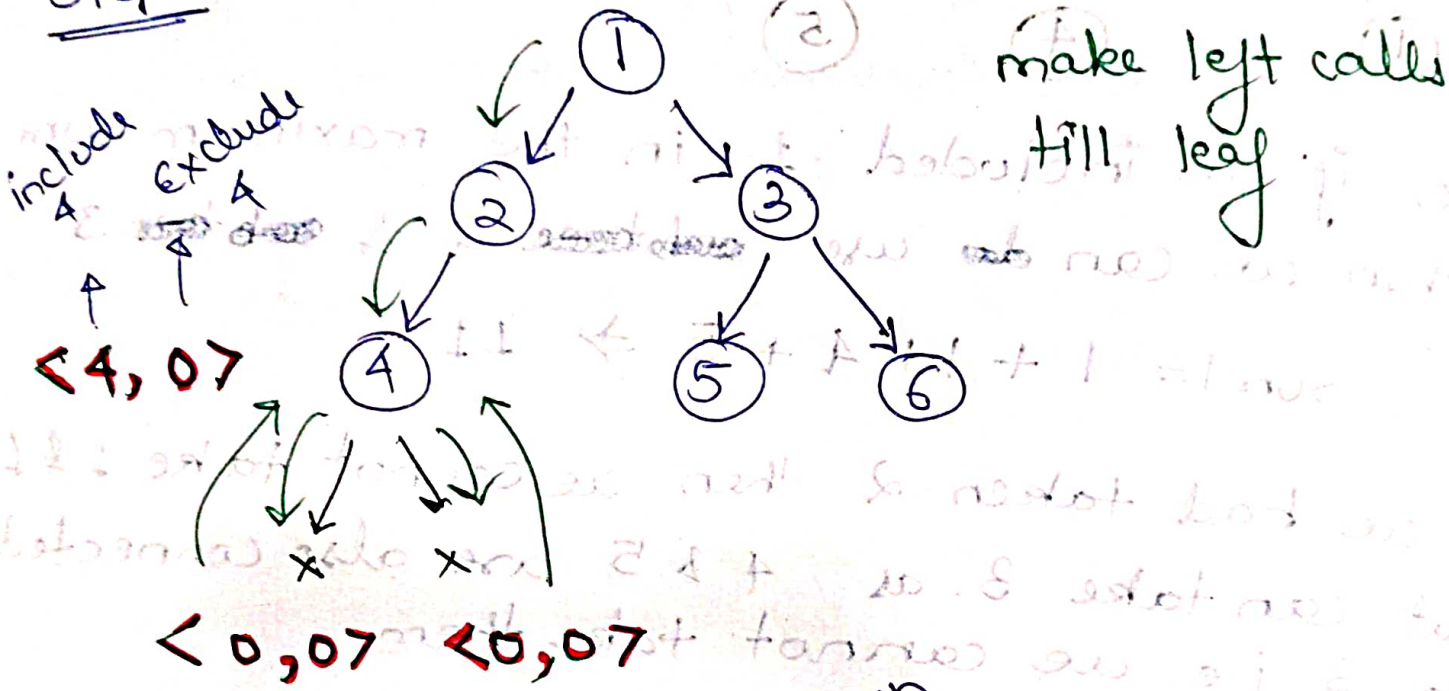
Take 2 :
sum = 3
So, ans = 11

Approach: < a , b >

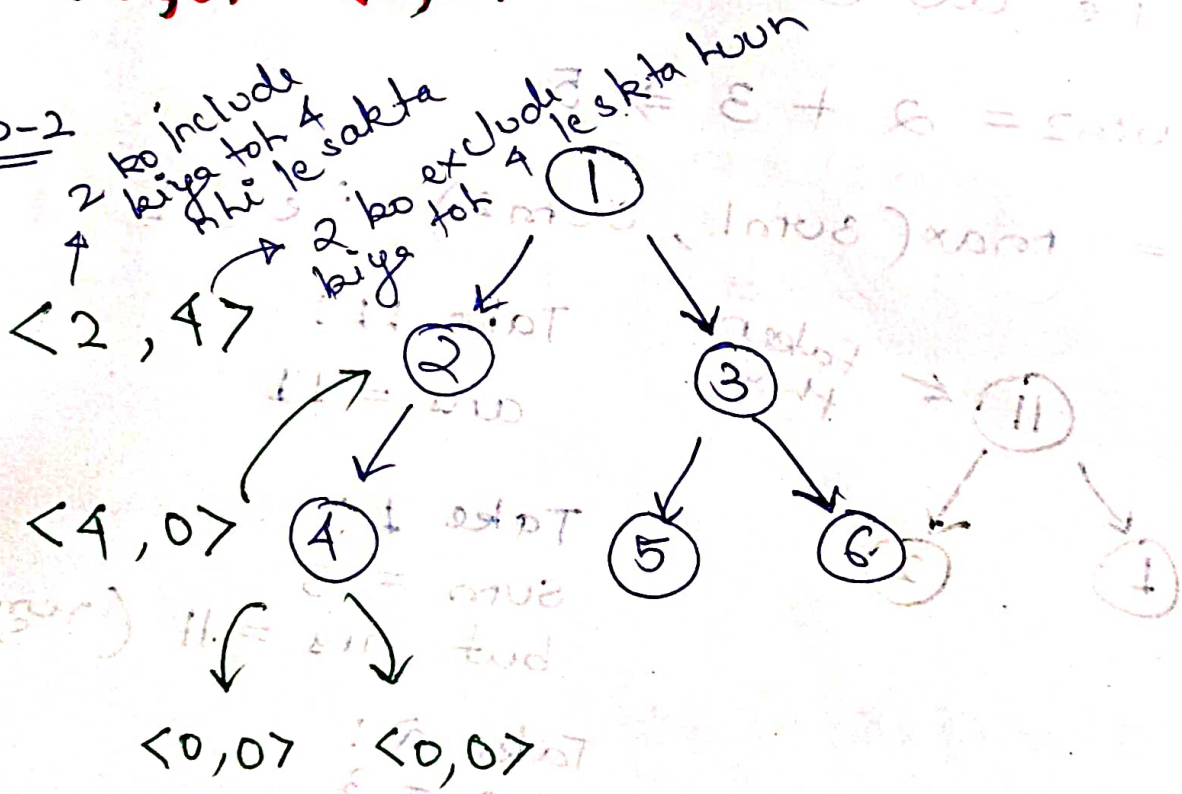
a : maxSum by including nodes at current level

b : maxSum by excluding nodes at current level.

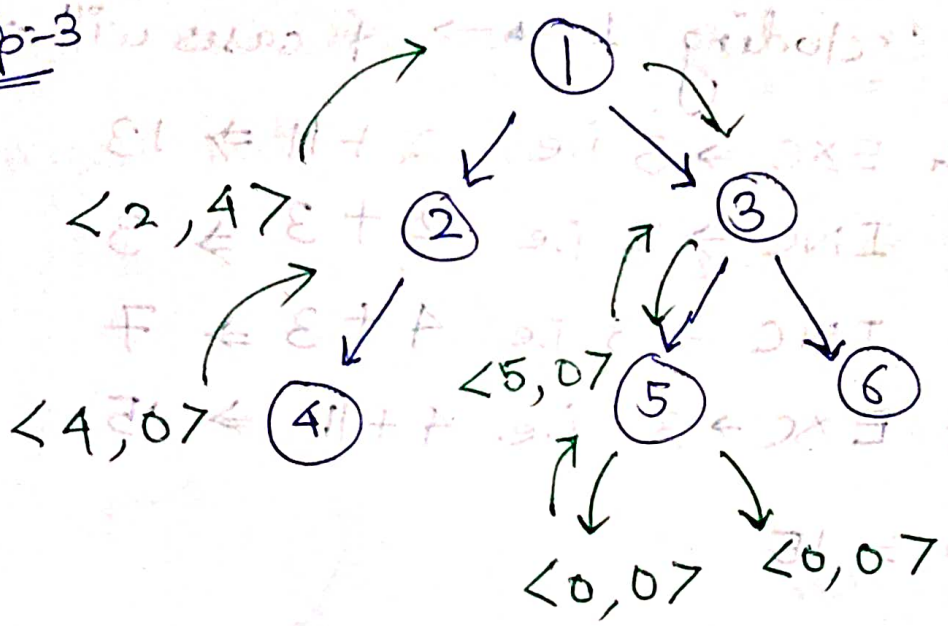
Step-1



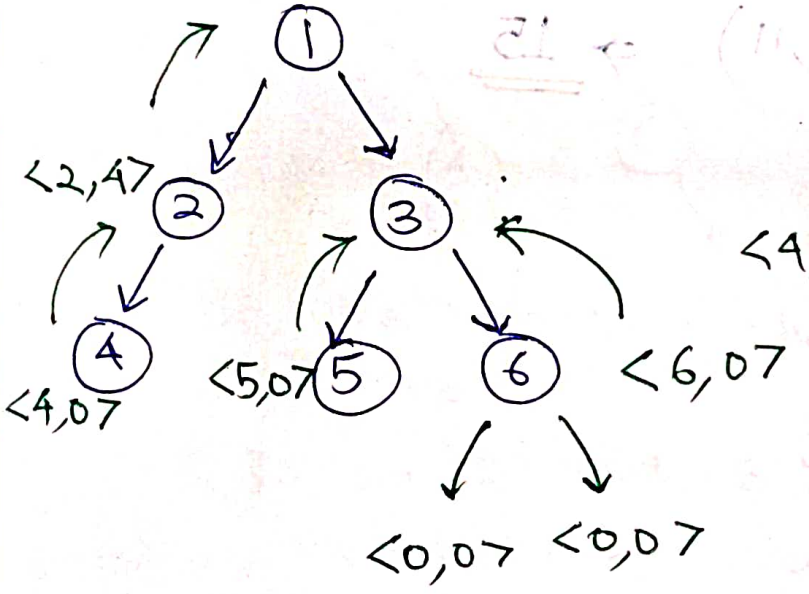

Step-2



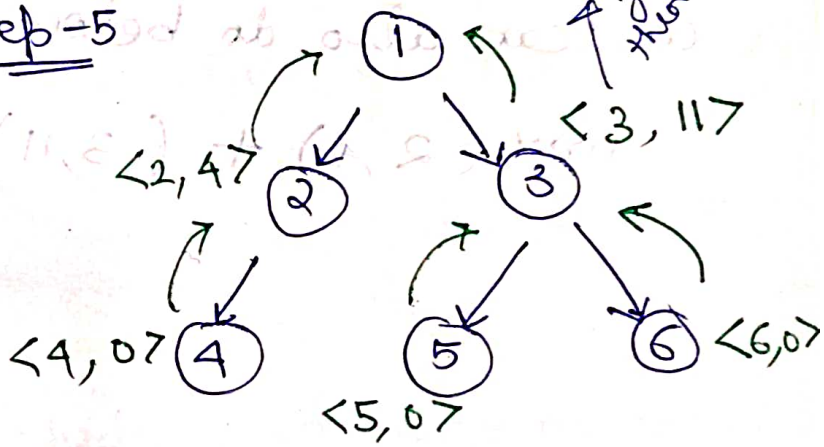
Step-3



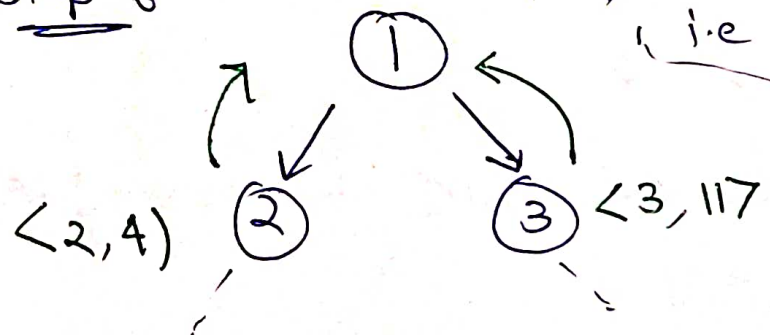

Step-4




Step-5




Step-6




ans = 16

<1, 4+11, max(2,4) + max(3,11)>
i.e <16, 15>

1 ko include kiya toh 2 & 3 ko exclude karna hoga i.e their excluded data 4 & 11.

1 ko exclude kiya toh we have 4 choices i.e
INC→2 & EXC→3 i.e 2+11 ⇒ 13
INC→2 & INC→3 i.e 5
EXC→2 & EXC→3 i.e 15
EXC→2 & INC→3 i.e 7

Incase of excluding 1 → 4 cases will arise

INC → 2 & EXC → 3 i.e 2 + 11 ⇒ 13

INC → 2 & INC → 3 i.e 2 + 3 ⇒ 5

EXC → 2 & INC → 3 i.e 4 + 3 ⇒ 7

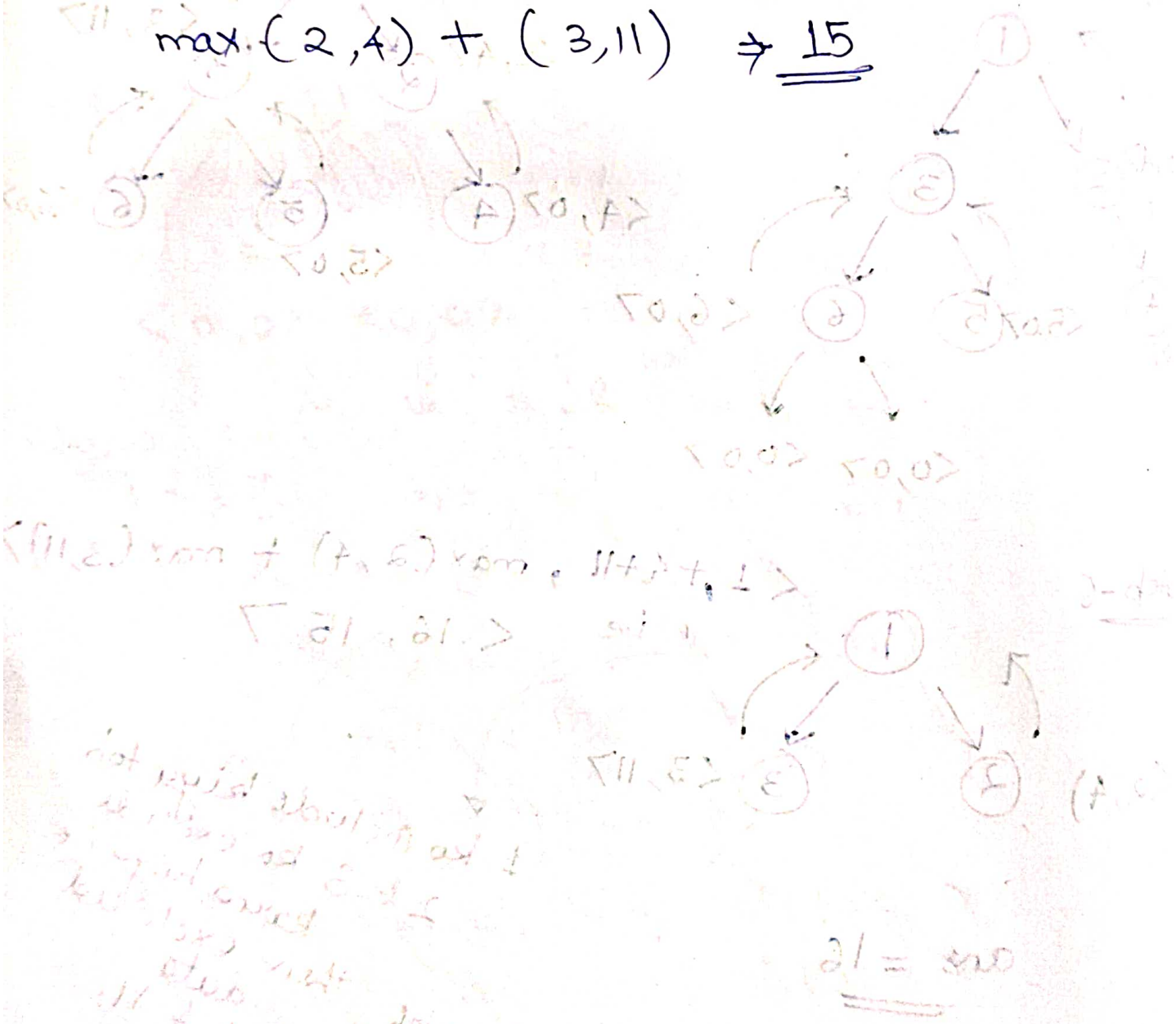
EXC → 2 & EXC → 3 i.e. 4 + 11 ⇒ 15

So, max = 15

or

we can also do below thing.

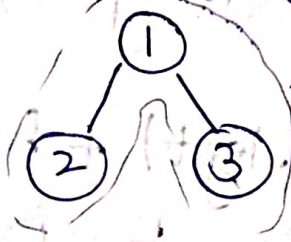
max.(2,4) + (3,11) ⇒ 15



# Diameter of a binary Tree

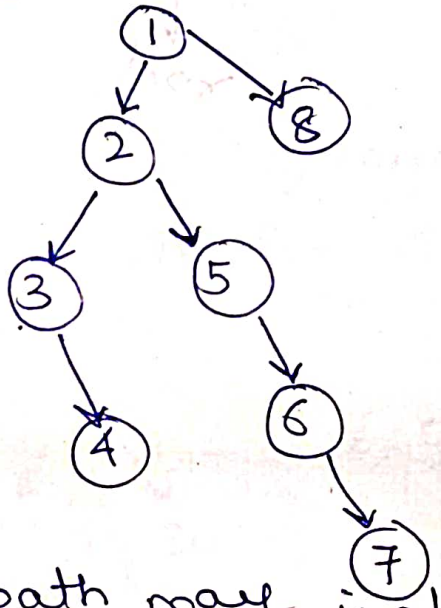
Longest path b/w any 2 nodes.

Ex-1




diameter = 3

Ex-2




Path-1 : 1 2 3 4

Path-2 : 4 3 2 1 8

Path-3 : 4 3 2 5 6 7

diameter = 6

So, path may include root node or may not.

In Ex-2 : Path-3 i.e diameter do not includes root node.

Longest path b/w any 2 end nodes

Leaf or root node.

Approach :- Diameter can be find in three ways

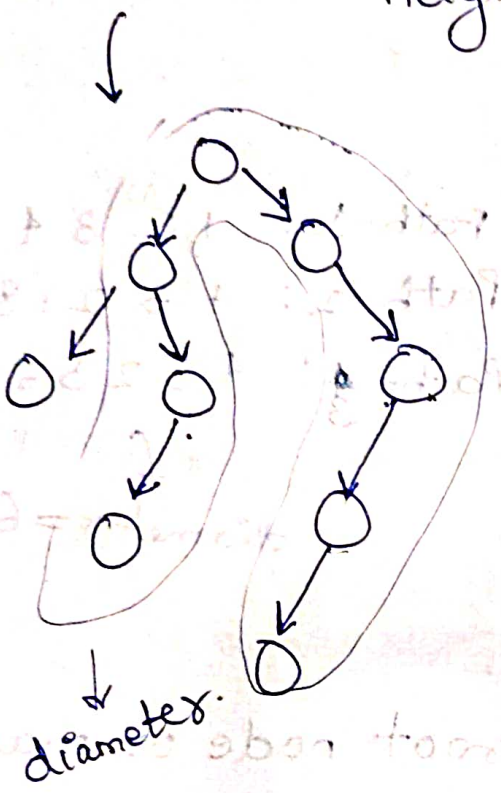
- ans in Left subtree
- ans in right subtree
- ans made by combining both left and right subtree.

Ex-2

option-1 = diameter(root.left)

option-2 = diameter(root.right)

option-3 = height(root.left) + height(root.right) + (1)

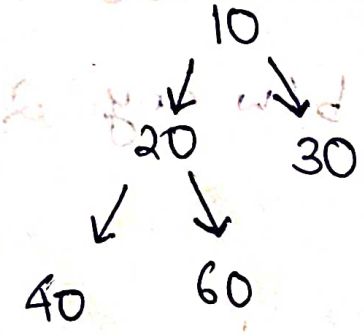



root

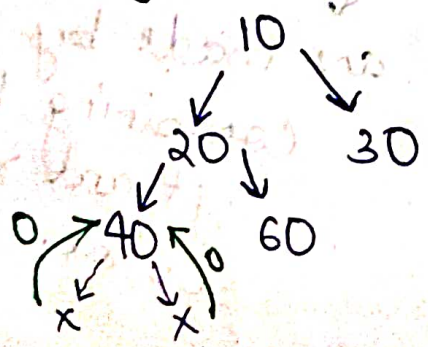
ans = max of op-1, op-2 & op-3.

Dry Run:

Ex




go to left till root != null.




i.e opt1 = 0

opt2 = 0

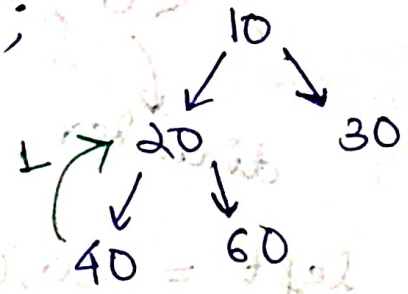
Now calculate opt3.

opt3 = 0 + 0 + 1 ⇒ 1

i.e diameter of a single node is 1.

as max of opt1, opt2, opt3 is opt3

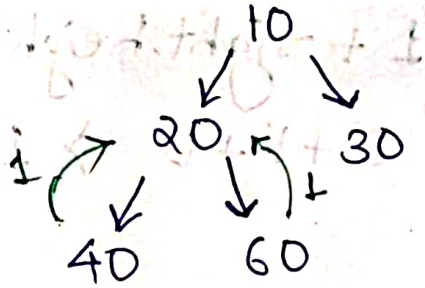
return 1;




Step-2 In the f^n frame of 20, opt1 = 1

now call for right i.e 60.

and similar to 40, 60 will also return 1.




So, opt^2 = 1.

opt3 = 1 + 1 + 1 ⇒ 3

1 → Left subtree height

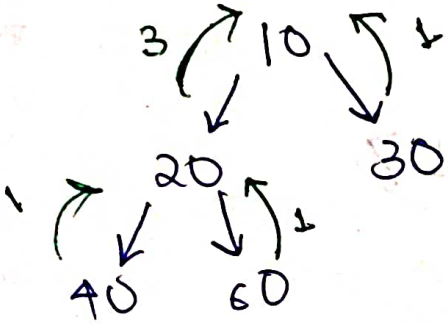
1 → right subtree height

Since, opt3 is the max.

So, ans = 3 till now

return 3

Step-1




for 10:

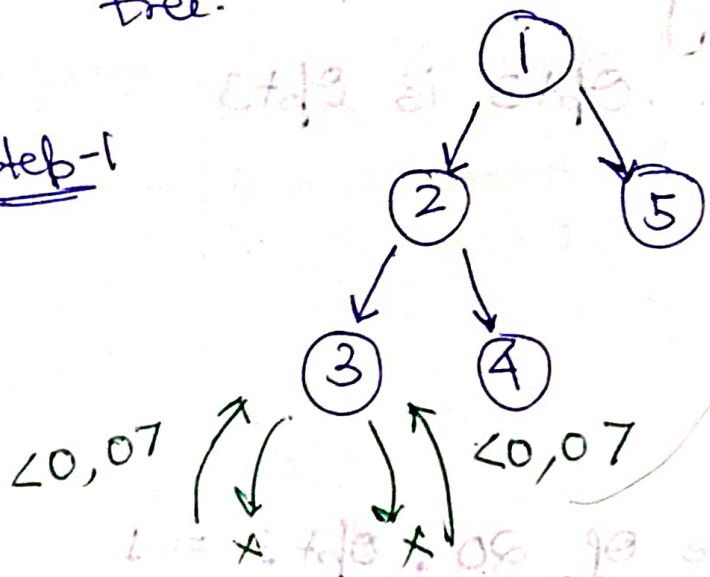
opt_1 = 3

opt_2 = 1

opt_3 = 2 + 1 + 1 ⇒ 4

So, ans = 4

# Optimised code for finding diameter of a binary tree.




Step-1

Pair.

<0, 0>

diameter height.

left = <0,0>
right = <0,0>

So, Opt1 = left.diameter = 0

Opt2 = right.diameter = 0

Opt3 = left.height + 1 + right.height

i.e Opt3 = 0 + 1 + 0 → 1

So, dia = 1

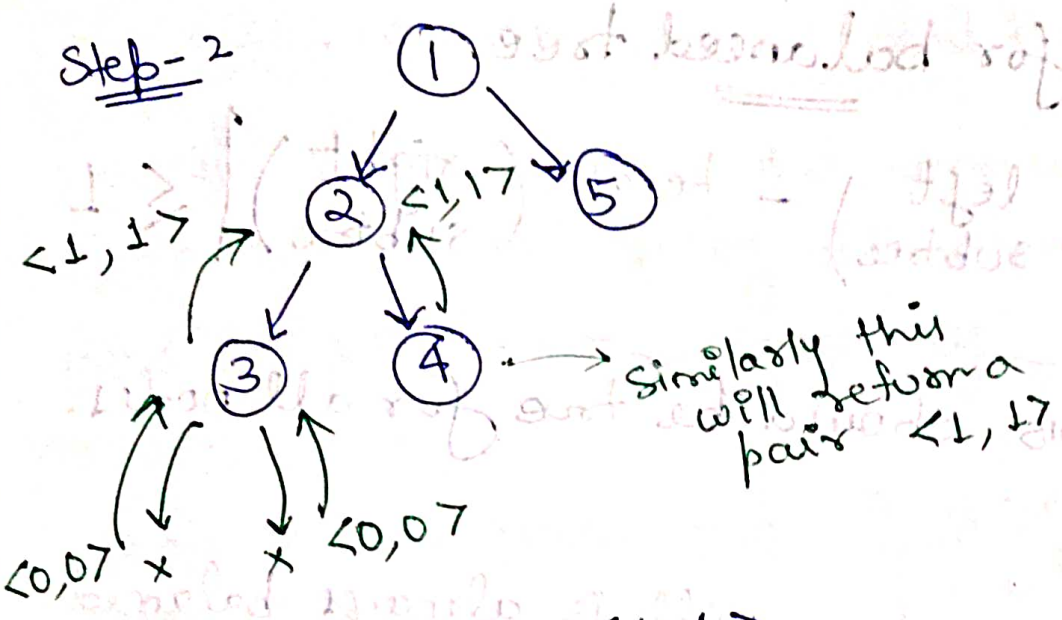
high = Math.max(left.height, right.height)

+ 1

Now, make a pair.

```
Pair ans = new Pair()
ans.diameter = dia
ans.height = high
return ans  i.e <1, 1>
```

Step-2




for 2: left = <1,1>

right = <1,1>

opt1 = left.diameter ⇒ 1

opt2 = right.diameter ⇒ 1

opt 3 = left.height + 1 + right.height

⇒ 1 + 1 + 1 ⇒ 3

max of opt1, opt2 & opt3 is opt3

dia = 3

high = Math.max(left.height, right.height) + 1.

i.e high = 1 + 1 i.e 2

so, ans = < 3, 27

return ans.

# Check for balanced tree

if : $\left| \text{height}(\text{left subtree}) - \text{height}(\text{right subtree}) \right| \leq 1$

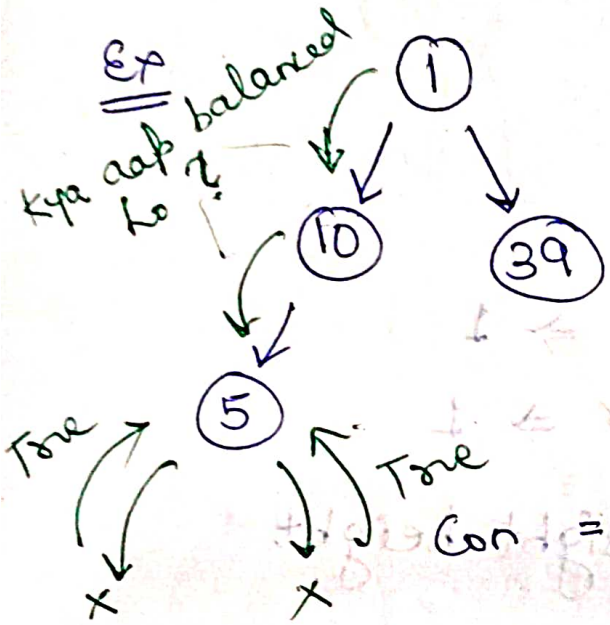
Math.abs

This should be true for all nodes.

Ex

balanced

Kya aap ko ?




null is always balanced

so, leftans = True

rightans = True

con = $|h(l) - h(r)| \leq 1$

$|0 - 0| \leq 1 \rightarrow$ True

```
if ( leftans == True && rightans == True &&
                                con == True )
{
    return true
} else {
    return false
}
```

But, this codes traverse each node twice.

ie $O(n^2)$

# Optimised check for balanced tree

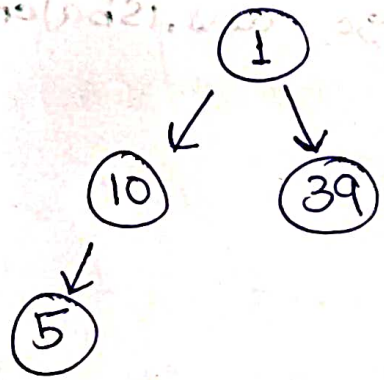
We will optimise it in the same way as we had optimised in the case of diameter.

Here we will make a pair of : boolean & int

```
class Pair {
    boolean isbalanced = true;
    int height = 0;
}
```

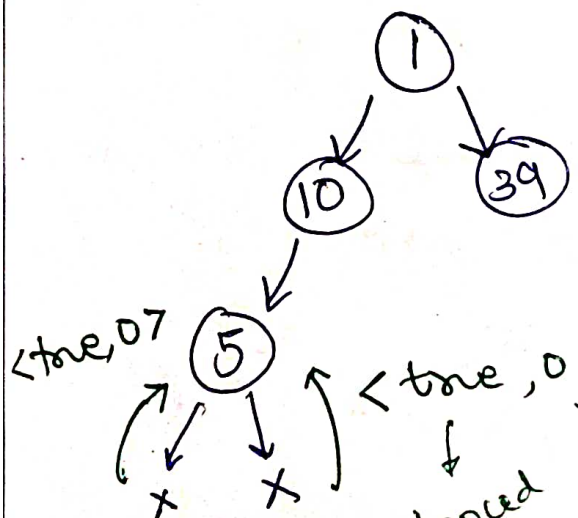
} ▷ Initially when root == null

Ex▷




```
Pair left = check(root.left)
Pair right = check(root.right
```

Step-1 1 bolega 10 se bhaj tu balance hain then, 10 bolega ruk abhi bataata. Similarly 10 asks to 5




<true, 07

<true, 07

isbalanced height

for 5:

leftans = left.isbalanced
i.e true

rightans = right.isbalanced
i.e true

leftheight = left.height = 0
ans

rightheight = rightans.height
= 0

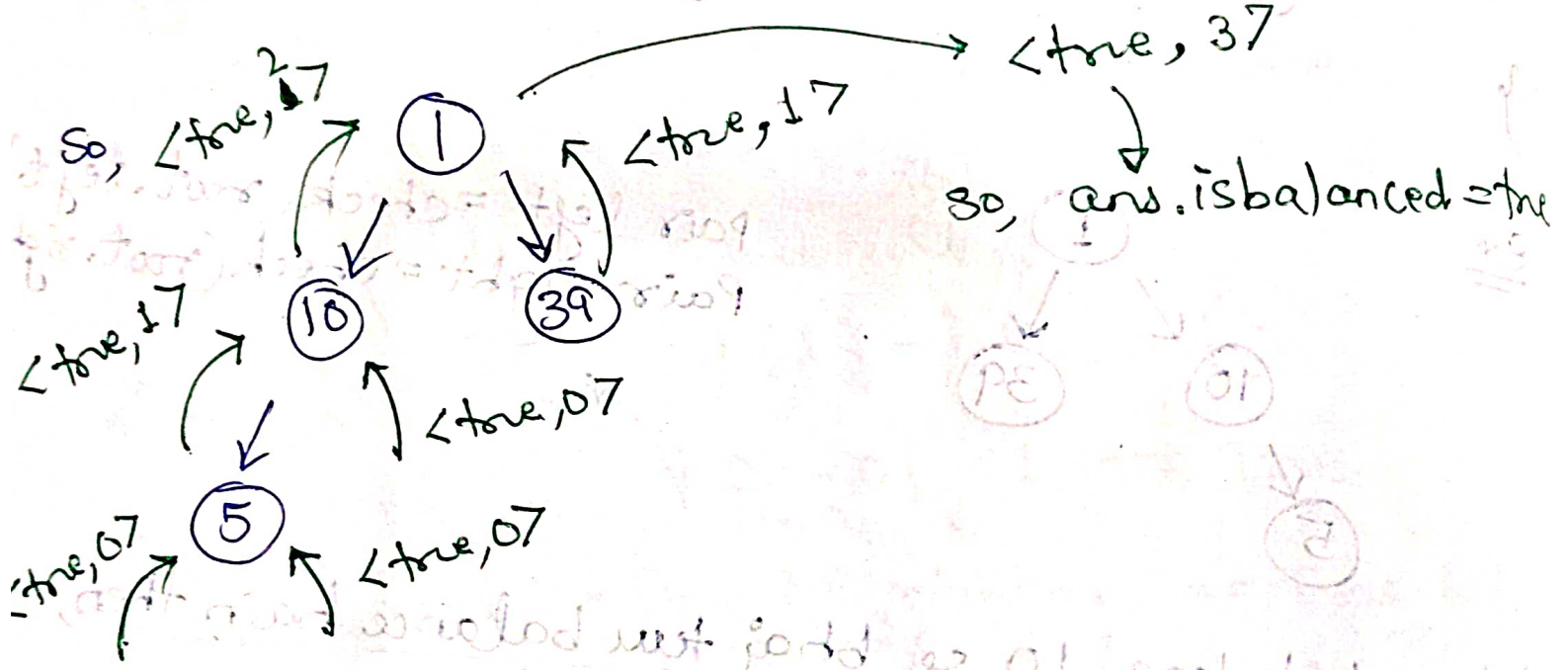
diff = (leftheight − rightheight) ≤ 1
i.e true

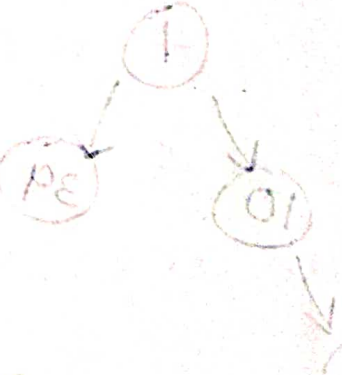
if ( leftans && rightans && diff )
{ ans.isbalanced = true;
} make a pair
Pair ans = new Pair
};

high = Math.max( leftheight, rightheight ) + 1.

so, ans.height = high

return ans → <true, 1>



# Determine two trees are identical or not


Approach-1: if root1.val == root2.val

Case-1

{ then bacha kaam recursion par chodd de
&
if root1.val != root2.val
then return false.

movement in tree1 should be same in case of tree2 i.e if a we are calling root1.left then in tree2 also we are calling root2.left.

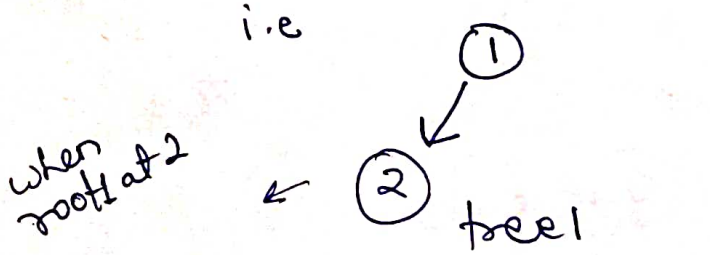
Case-2 if (root1 == null && root2 != null)

or

vice versa

then return false

i.e




So, return false.

So, if both the case gives true i.e.

root1.val == root2.val → case-1

return 2

&

case-2 :

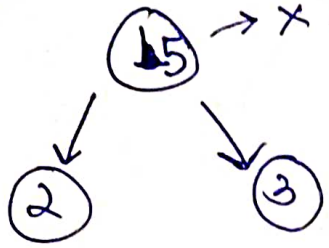
then return true.

else

return false.

# Sum Tree

for every node X other than leaf node has a value equal to the sum of left subtree & right subtree.

i.e 


Note: Empty tree is also a sum tree.

X.value = 2 + 3 ⇒ 5

So, given tree is a sum tree.

Approach: left part ko check karke aao
right part ko check karke aao.
then current node ko check kar lo.

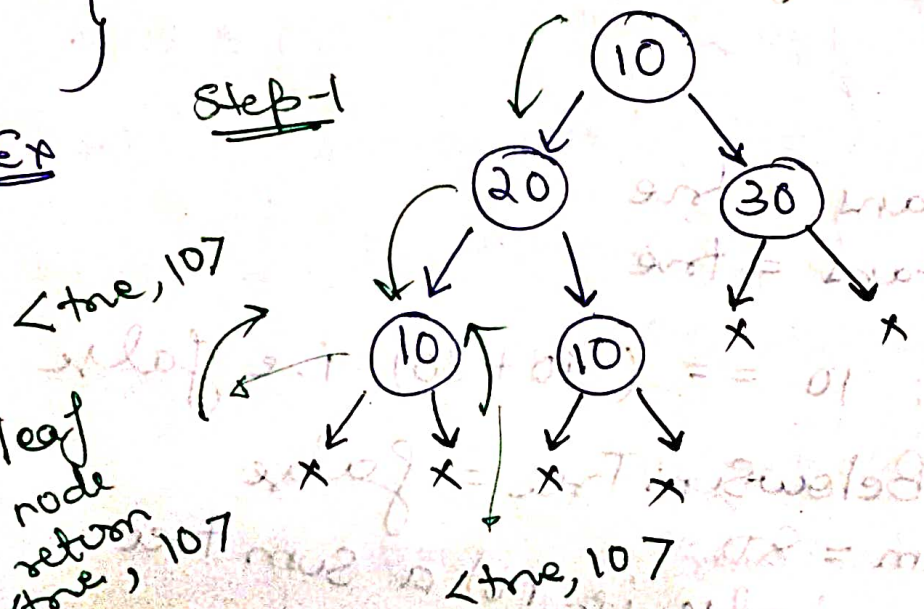
So, we will make a pair.

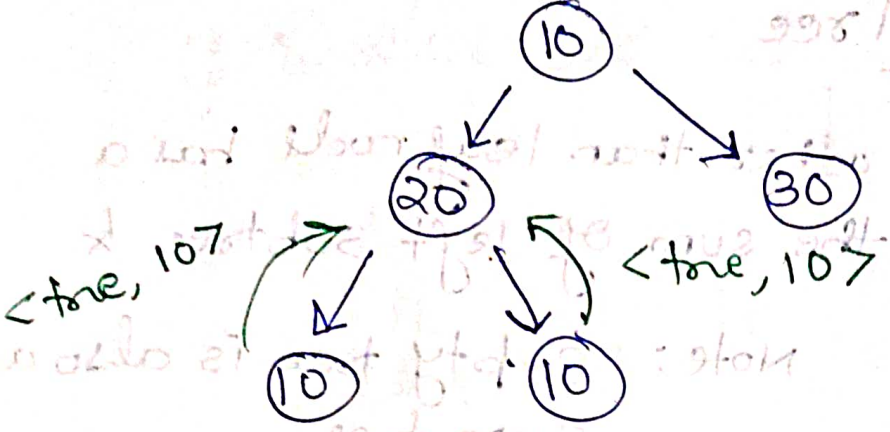
```
Class Pair {
    boolean isBelowSumTree = true;
    int sum = 0;
}
```

} when root == null

Ex

Step-1






for 20: leftans = true

rightans = true.

Condn : 20 == (10 + 10) i.e true

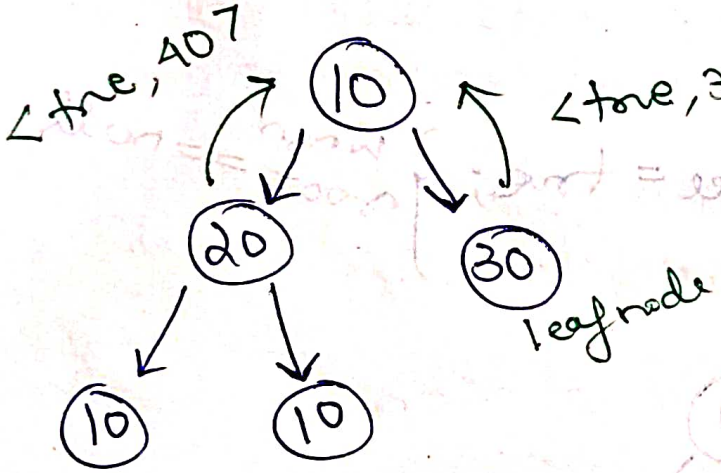
So, ans.isBelowSumTree = true

ans.sum = root.val + leftSum + rightSum

20 10 10

i.e 40

return ans; i.e <true, 40>




for 10: leftans = true

rightans = true

condn : 10 == (40 + 30) i.e false

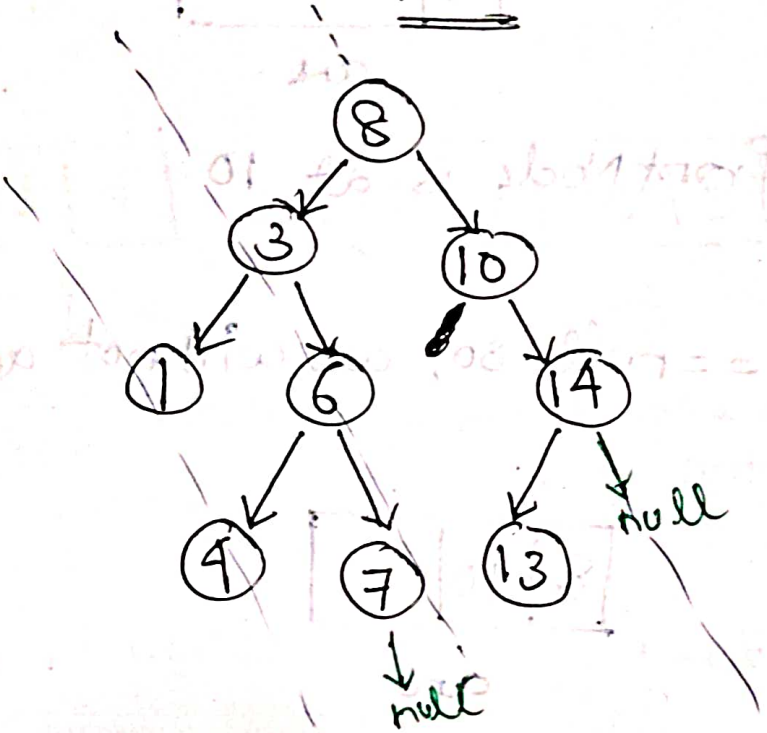
i.e ans.isBelowSumTree = false

ans.sum = 80.

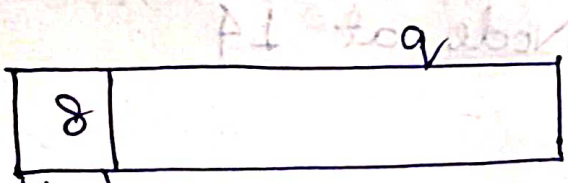
return ans, → false i.e not a sum tree

# Diagonal Traversal of Binary Tree

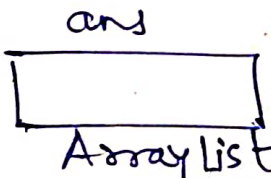
Ex




Step-1

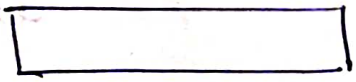



Queue - 4




While(!q.isEmpty)

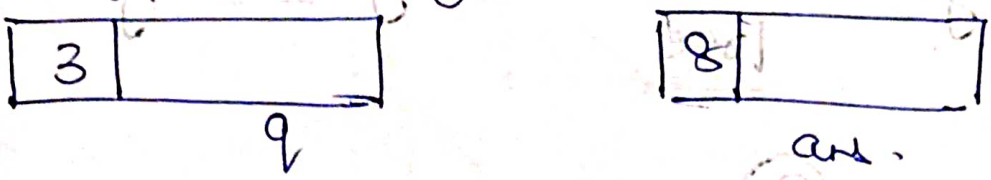
frontNode = 8 [q.peek()]

q.remove i.e 

Now go to 8 right till we do not get null and also check if a node has it's left child not null then store it in the Queue.

```
While ( frontNode != null)
    if ( frontNode.left != null)
        q.add (frontNode.left);
    ans.add (frontNode.val)
    frontNode = frontNode.right.
```

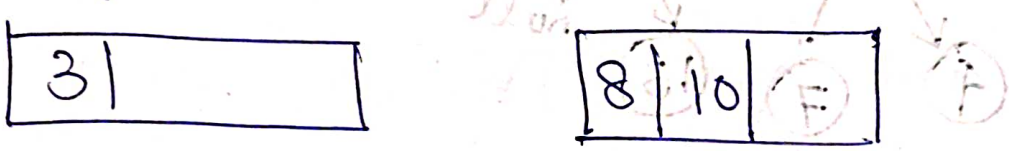
→ So, after entering second while loop:




q ans.

and frontNode is at 10

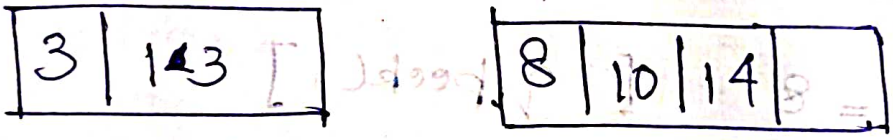
→ Since 10.left == null, so, we will not add it's left child.




q ans

↓ frontNode at 14

→




q ans

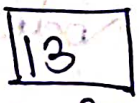
↓ frontNode == null

i.e go out of while loop

Step-2

frontNode = 3 (q.peek)

q.remove() i.e

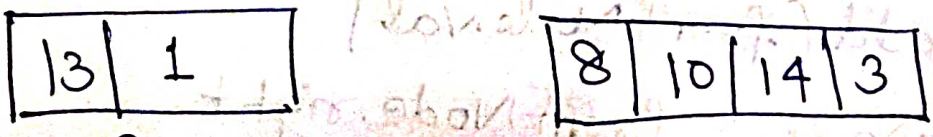



q

again while loop runs

inner

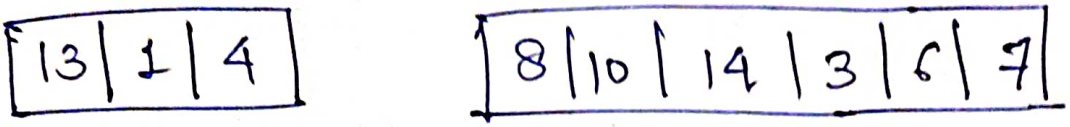
→




q

→ 

q ans

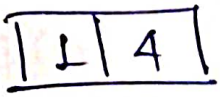
→ 

q ans

Now, frontNode == null
i.e go out of while loop

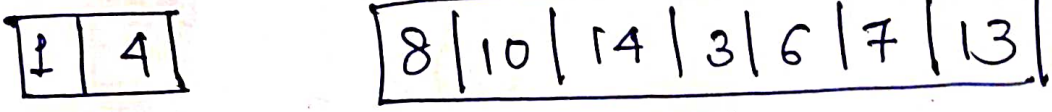
Step-3

frontNode = 13 (q.peek)

q.remove() i.e 

q

go to inner loop.

→ 

q ans

& frontNode == null

Similarly in Step-4 & Step-5 we'll get

ans ; 8 10 14 3 6 7 13 1 4

Now: Refer the Code.
↑
to