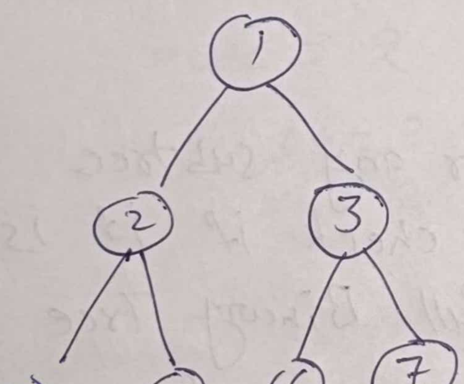
Count Nodes in Complete Binary Tree

Complete BT :- Every level, except the last level, is completely filled in Complete BT, & all nodes in last level are as left as possible.

ex



Brute force

```
inorder(node, &cnt)
{   if(node == NULL)
        return;

    cnt++;
    inorder(node->left, cnt);
    inorder(node->right, cnt);
}
```

$T.C. = O(N)$

$S.C. = O(H) = O(\log N)$

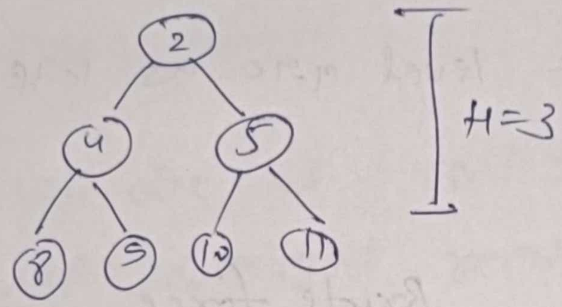
H = Height of BT & in complete BT the $H = \log N$

Imp :- To solve this problem in less then O(N)

we use the property of ~~complete~~ full BT.

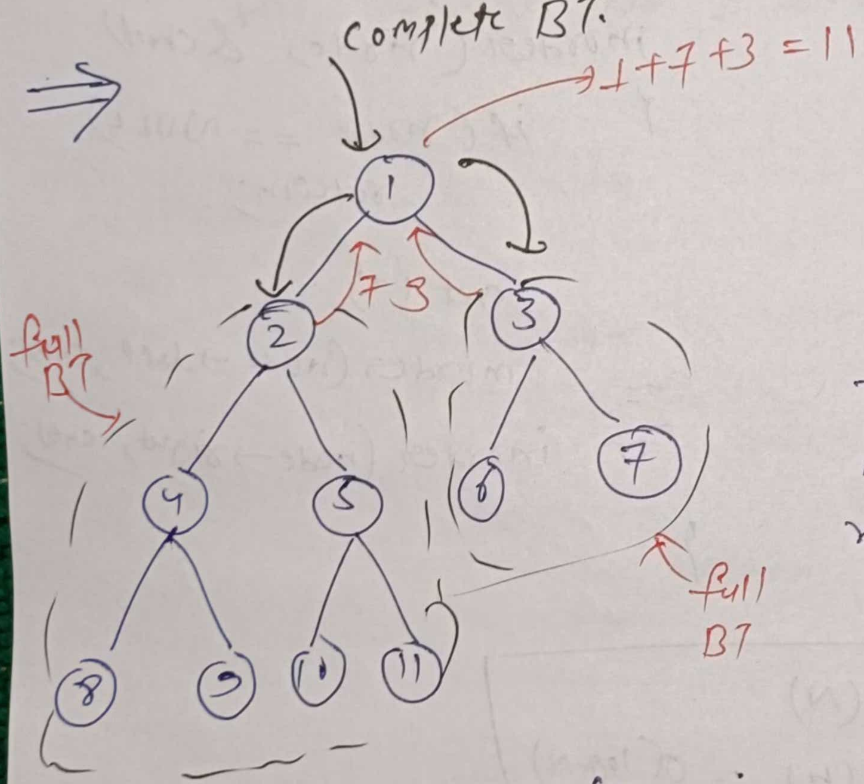
Number of nodes in complete BT = $2^H - 1$

ex:



nodes = $2^3 - 1$
= 7

⇒



for any subtree we check if it is a full Binary Tree then number of nodes in this sub tree is

nodes = $2^H - 1$

where H = height of subtree

How to check if subtree is full Binary tree or not

Compute lh ← left height
rh ← right "

if (lh == rh) ⟶ Then it is a full Binary tree

nodes = $2^{lh} - 1$

else

nodes = 1 + (left subtree nodes) + (right subtree nodes) recursive call

1st for node = 1

$lh = 4$

$rh = 3$

$lh \neq rh$

$$\text{nodes} = 1 + \underbrace{(7)}_{\text{nodes in left subtree}} + \underbrace{(3)}_{\text{nodes in right subtree}}$$

nodes in left subtree

node = 2

$lh = 3$

$rh = 3$

$lh == rh$

nodes $= 2^3 - 1 = 7$ → return

nodes in right subtree

node = 3

$lh = 2$

$rh = 2$

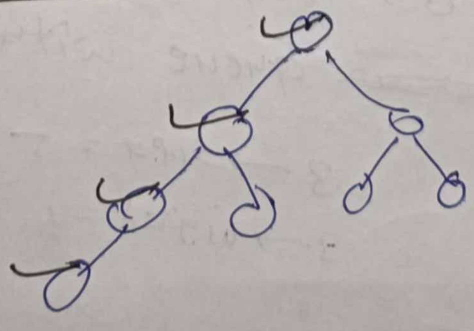
nodes $= 2^2 - 1 = 3$

node is B.T. $= 1 + 7 + 3 = 11$

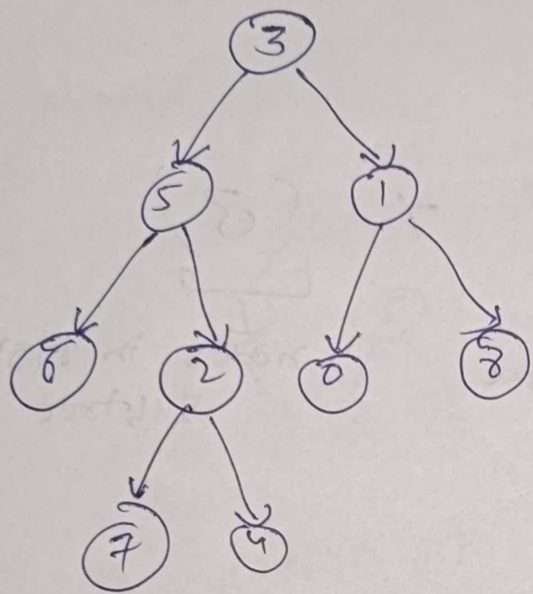
$$\boxed{\begin{array}{l} T.C. = O((\log N)^2) \\ S.C. = O(\log N) \end{array}}$$

→ $\underbrace{\log N}_{\text{for traversing}} * \underbrace{\log N}_{\text{for computing height for every node}}$

ex



Nodes at distance K



K=2, target=5

ans = {7, 4, 1}

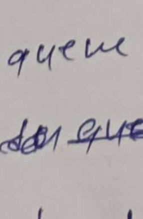
T.C. = O(N) + O(N)
S.C. = O(N) + O(N) + O(N)

Approch :- In order to solve this problem the first thing that comes in our mind that if we have target node like = 5 then the nodes at distance K=2 can be upward from target node & can be downward from target node.

But in BT we can't go upward direction so for that we have to first create a parent table for every node.

To create parent pointer we do BFS traversal for that queue DS required.

initially ~~dequeue~~ queue with root node



3 → left = 5
3 → right = 1

mark parent
5 → 3
1 → 3.

Stack: 1, ~~8~~

now

5 → left = 6

5 → right = 2

parent

5 → 3
1 → 3
6 → 5
2 → 5

Stack: 2, 6, ~~1~~

now

1 → left = 0

1 → right = 8

parent

5 → 3
1 → 3
6 → 5
2 → 5
6 → 1
8 → 1

Stack: 8, 0, 2, ~~6~~

now

6 → no left & right

Stack: 4, 7, 8, 0, ~~2~~

now

2 → left = 7

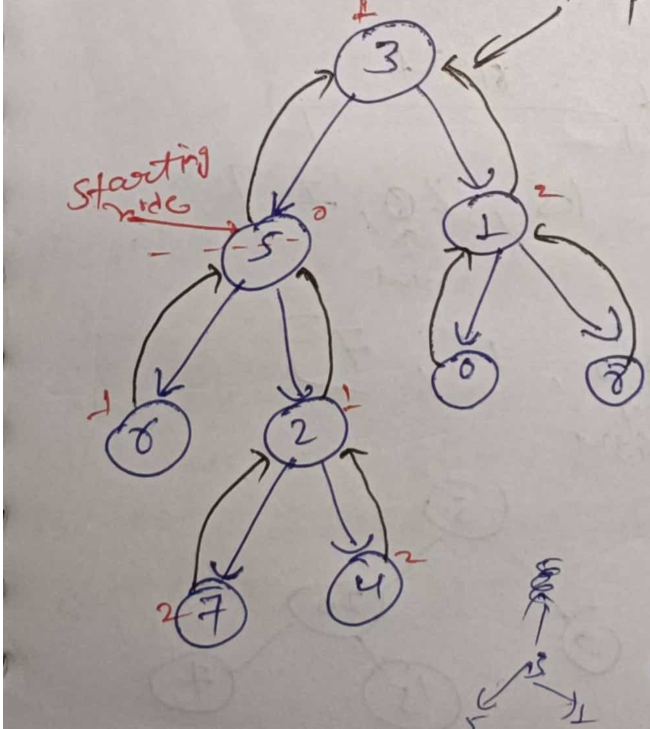
2 → right = 4

parent

5 → 3
1 → 3
6 → 5
2 → 5
0 → 1
8 → 1
7 → 2
4 → 2

So on.

Now we have



Now we use Bfs traversal bcz if we consider starting node as level 0 then at level = 2 = K then our ~~node~~ ans.

imp queue में डालते ही node को visited कर दो