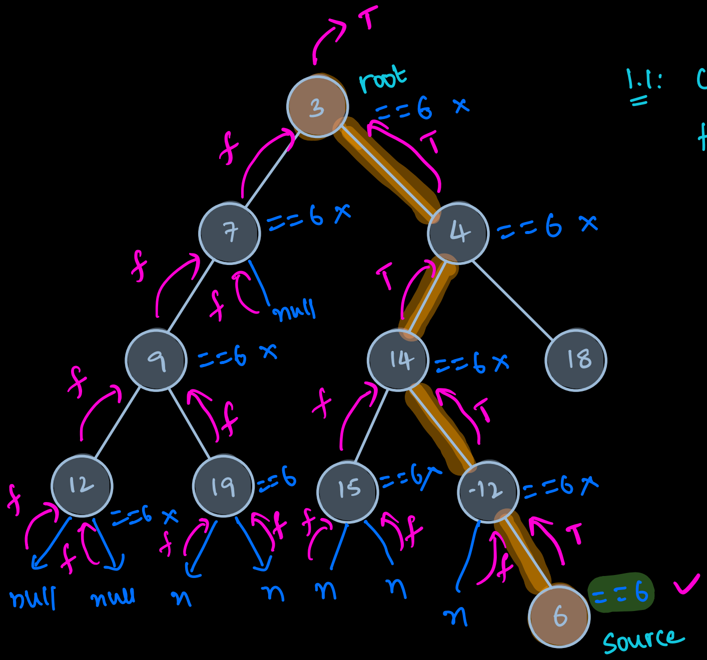
Today's content

(i) Path from root to a given node.

(ii) Nodes at distance k from given node.

(iii) LCA in binary tree

Q1. Given a binary tree with distinct values, find the path from root to source node.



1.1: Check if a given node is present in the tree or not.

```
bool check( Node root, int k )
    if(root==null) {return false}
    if(root.data==k) {return true}
    if(check(root.left,k) ||
            check(root.right, k))
        return true
    return false
```

Observation: If we store all the nodes that returned true, we will get path.

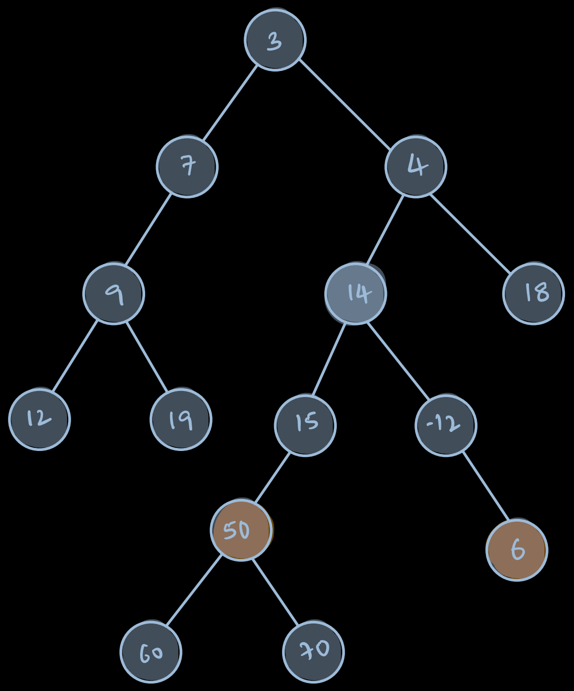
```
List<Node> path;
bool check( Node root, int k )
    if(root==null) {return false}
    if(root.data==k) { path.add(root) ;  return true }
    if(check(root.left,k) ||
            check(root.right, k))
        path.add(root)
        return true
    return false
```

path : [6, -12, 14, 4, 3]

TC : O(n)
SC : O(H)

// reverse path if required.

Q2: LCA , least common ancestor .



LCA(14,70) =

LCA(50,6) = 14.

LCA(60,70) = 50

LCA(19,18) = 3.

path from root to 50 → [3,4,14,15,50]

path from root to 6 → [3,4,14,-12,6]

[3,4,14,15,50] → ans.

[3,4,14,-12,6]

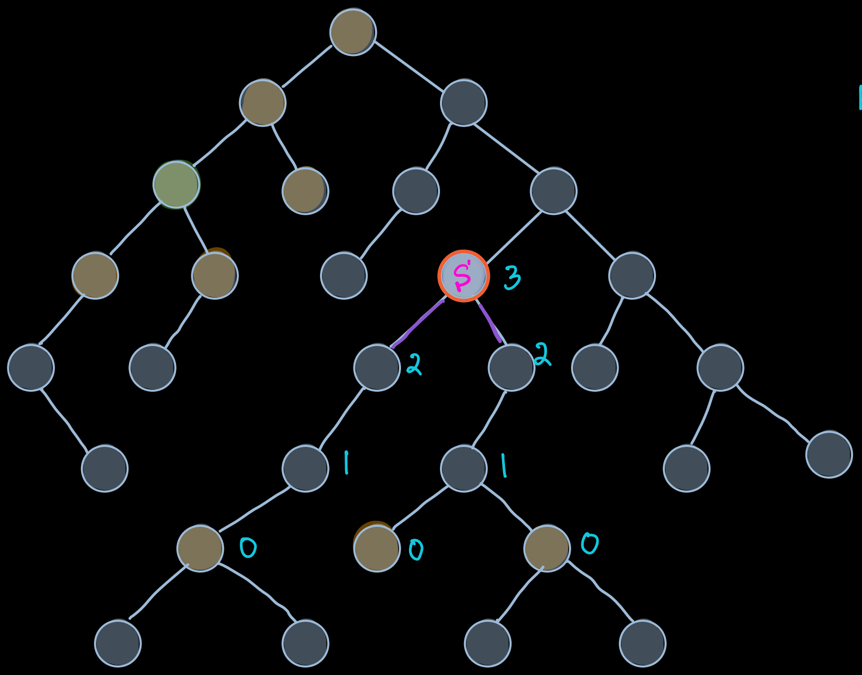
LCA(12,18)

path from root to 12 → [3,7,9,12]

path from root to 18 → [3,4,18]

38: Given a source node, find the no. nodes at distance 'k' from source node.

(All the nodes should be below source node).



k = 3 , ans = 3 .

idea: find source and apply level order traversal.

idea 2:

distance between source and node = k.

distance between source children and node = (k-1)

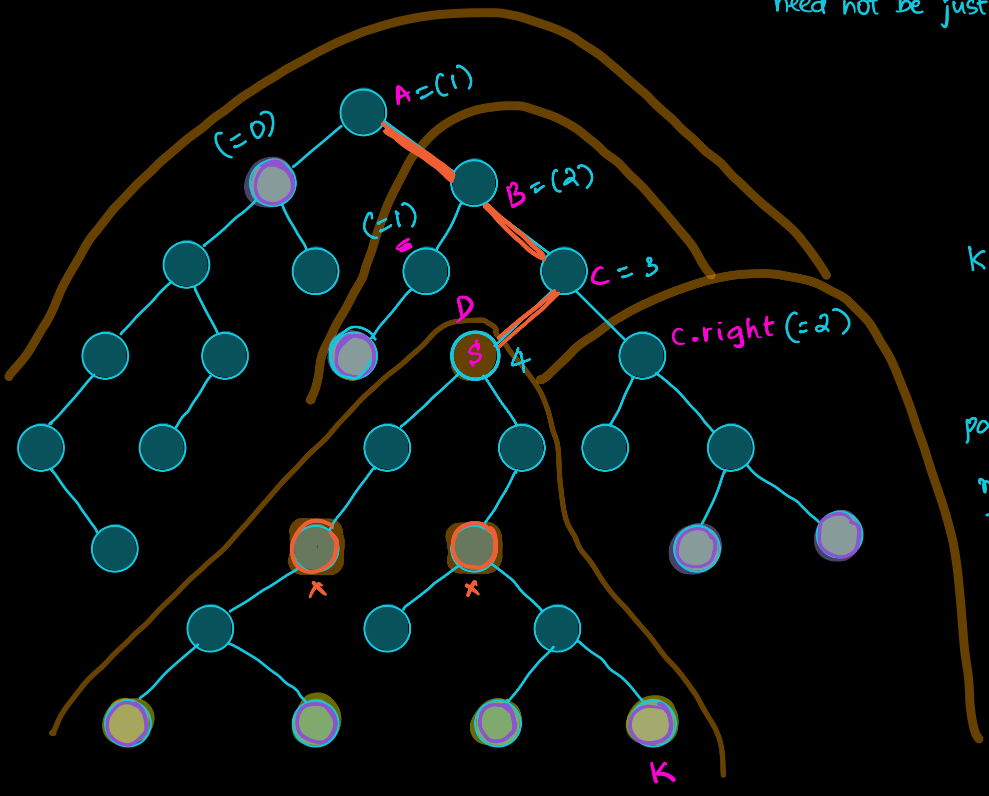
```
int   countNodesbelow( Node  s, int  k)

      if(s== null  || k<0)
          return  0.
      if(k==0)
          return  1

      return     countNodesbelow(s.left, k-1)
                 + countNodesbelow(s.right, k-1)
```

49. Given a source node, find the no. of nodes at distance 'k' from source node.

↳ need not be just below.



K=4, ans=8.

path from source to root

	0	1	2	3
node :	[D	C	B	A]

path[1].left = path[0]
C.left D.

path[2].left = path[1]
B. left C

i) get the path from root to source → path (array of nodes).

ii) Iterate on path array.

	path[0]	path[1]	path[2]	path[3]
K=4,	D below(D,K)	C D c.right path[1].left = path[0] C D. => below(C,right, K-2)	B O C path[2].left = path[1] B. C => below(B.left, K-3)	A O B path[3].left = path[2] A B => below(A.left, K-4)

$$\left.\begin{matrix} 1 \to K-2 \\ 2 \to K-3 \\ 3 \to K-4 \end{matrix}\right\} \Rightarrow i \to K-i-1$$

```
int countNodesAtkDistance( Node r, Node s, int k)
    List<Node> path ; // To-Do.
    int c = countNodesBelow( path[0] , k)
    int n = path.length();
    for (i=1; i< n; i++)
        // We're at the node path[i].
        // from path(i) we need to know whether to go left or right?
        if(k-i-1 <0)    // happens when path length is more than k.
            break
        if(path[i].left = = path(i-1))
            // I'm coming from left, look on right
            c = c+ countNodesBelow (path[i].right, k-i-1)
        else
            // I'm coming from right, look on left.
            c = c+ countNodesBelow (path[i].left , k-i-1)
    return c
```

TC: O(n) (∵ We're visiting every node only once)

SC: O(H)

If k==1 , how do we get ans=3.