Todays Content:

- → Path from root to given node
- → LCA in Binary Tree / LCA in a BST
- → No: of Nodes at k from given node
- → Check root to leaf node with sum = k { TODO }

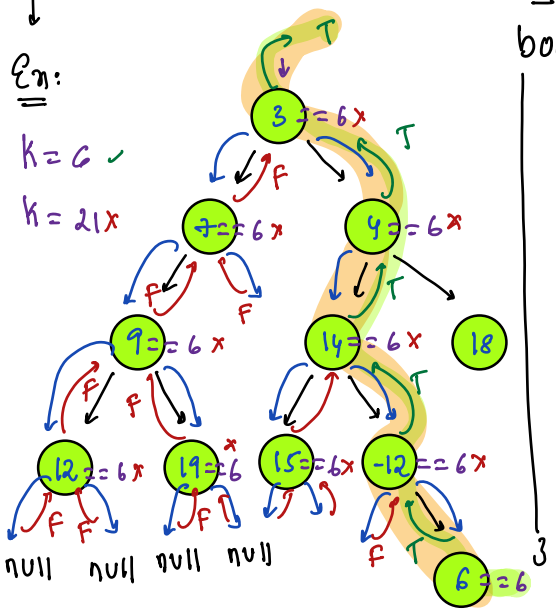
Q1: Given a B.T which contains all unique values, Search if there exists a k in B.T

Ass: Search for k in BT & return True/False

Ex:

k = 6 ✓

k = 21 ✗




```
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
}
```

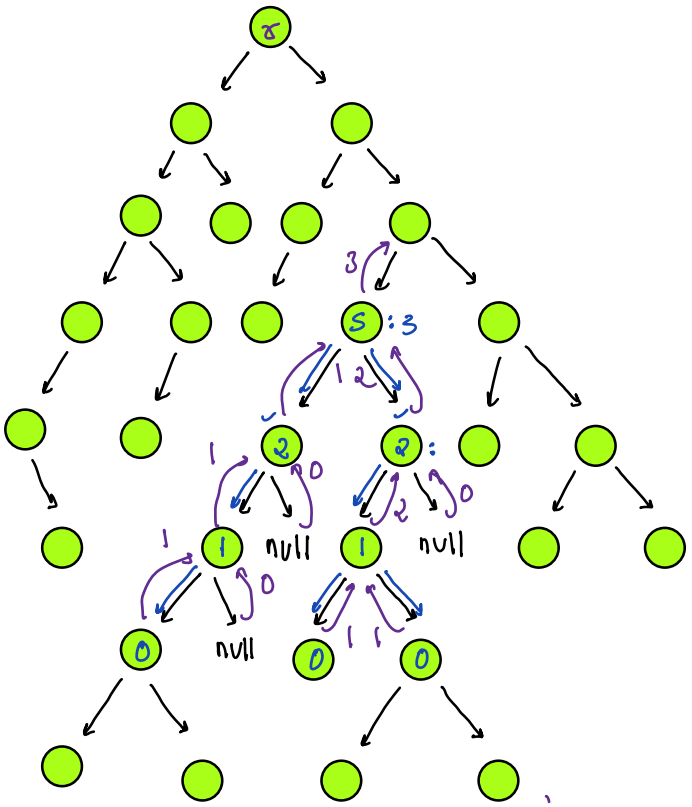
Obs: In above trace, if we store all nodes which are returning true, it is nothing but path from root node to source node

```
list<Node> path;
bool check(Node root, int k){ TC: O(N) SC: O(H)
    if(root == null){ return false }
    if(root.data == k){ path.insert(node); return true }
    if(check(root.left, k) || check(root.right, k)){
        path.insert(node);
        return true;
    }
    return false
}
```

For above ex: 6 -12 14 4 3

↳ Note: if its root node to soure we need to reverse path.

Q2: Given a sourcenode (address of node) how many nodes are there at distance k, (All nodes should be below source)

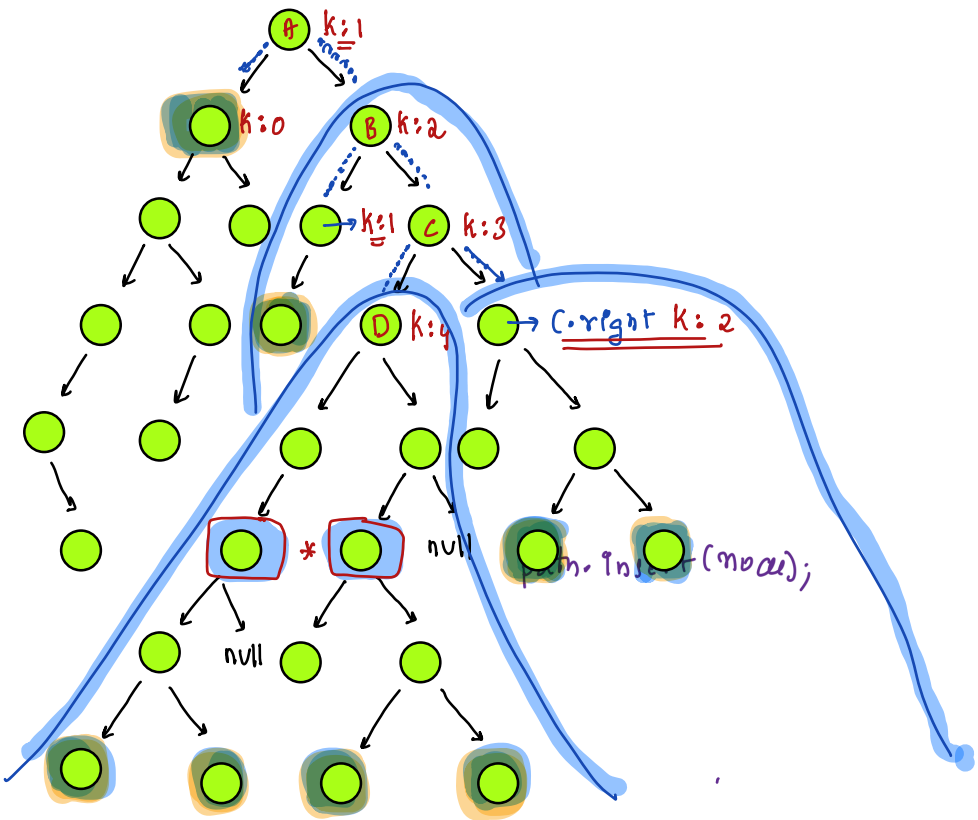



```
int countnodesbelow(Node s, int k){
    if(s==null || k<0){return 0}      // if k negative.
    if(k==0){return 1}
    return {countnodesbelow(s.left, k-1)
            countnodesbelow(s.right, k-1)}
}
```

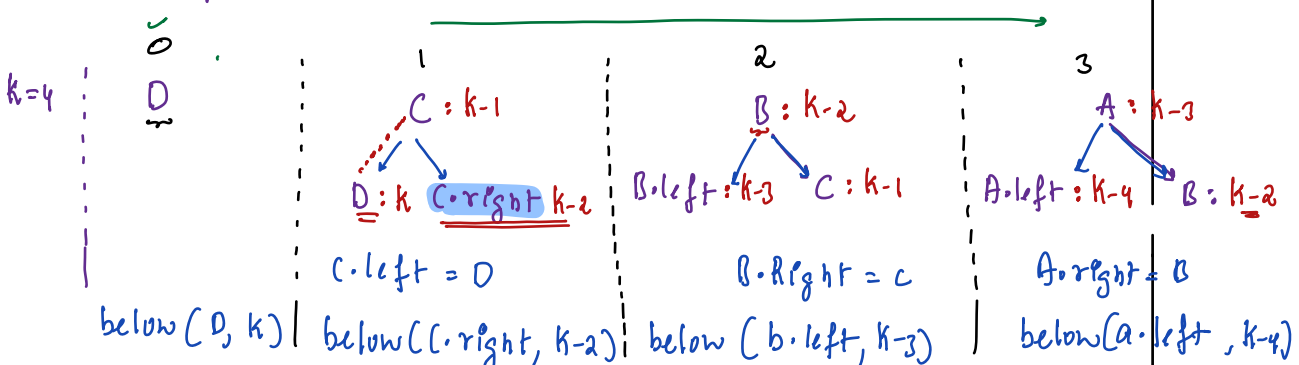
38) Calculate no: of nodes are at a distance k from Source

→ Note: Only Sorce node value is given, we need to search it first

→ Note2: Binary Tree contains only distinct values



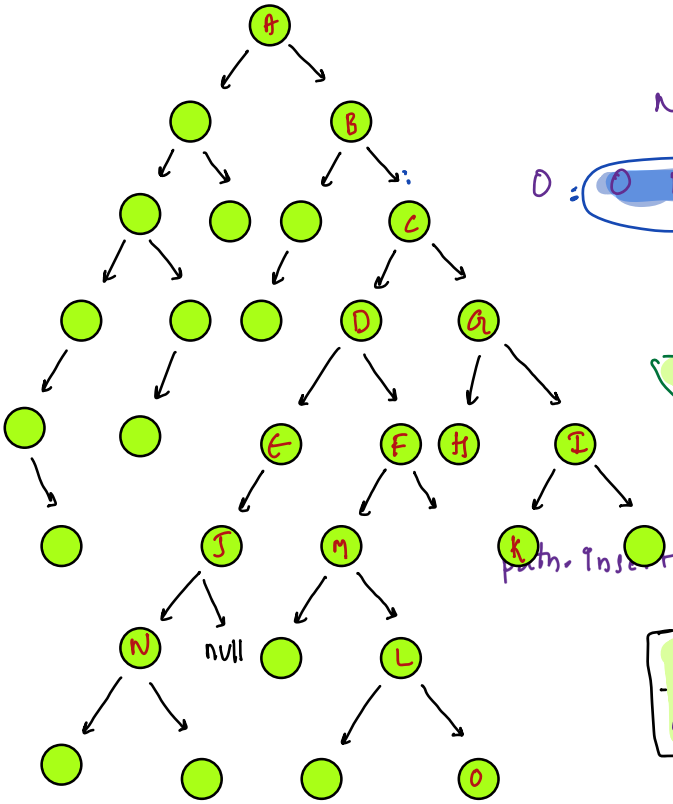
Idea: Get path from source node to root node.

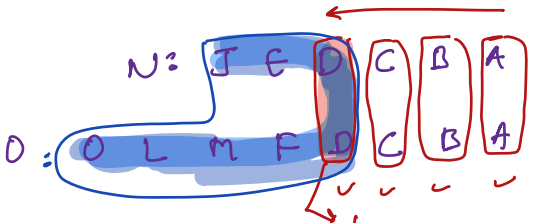


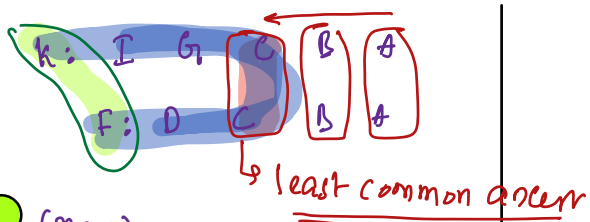
```
int countNodes(Node r, int s, int k){ TC: O(N)  SC: O(H + H) = O(H)
    List<Node> path; // get path from source value → root node
    int c = countNodesBelow(path[0], k)
    int n = path.length
    for(i=1; i<n; i++){
        // node we are at is path[i]
        // from path[i] how do we know, wether to search left or right?
        if(k-i-1 < 0) {break}
        if(path[i].right == path[i-1]){ // search in left
            c += countNodesBelow(path[i].left, k-i-1)
        }
        else{ // search in right
            c += countNodesBelow(path[i].right, k-i-1)
        }
    }
    return c;
}
```

(k-i-1 : it might become -ve)

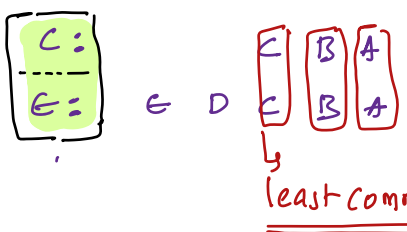
LCA: Least common ancestor







path.insert(node);



// Given S_1, S_2 & root r

- → get path from $S_1 \rightarrow r$
- → get path from $S_2 \rightarrow r$
- → { get least common ancestr by iterating Right to left