Todays Content

a) Search k in BT
b) LCA in BinaryTree
c) No:of Nodes at k from given Node
d) Fill right Pointer

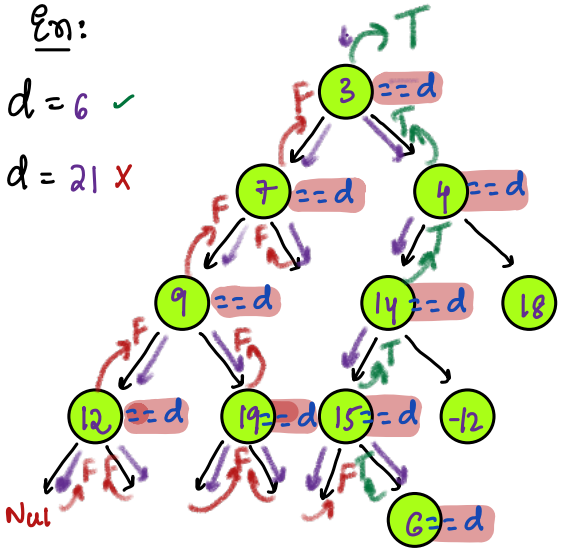
Q1: Given a B.T which contains all unique values.

↓ Search if there exists a d in Binary Tree

Eg:

d = 6 ✓

d = 21 X



Ass: Given Tree, Search for d return T/F

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

Path d: 6 ⇒ 3 4 14 15 6

Obs: If we store all nodes, which returns true, it gives Path between source & root node.

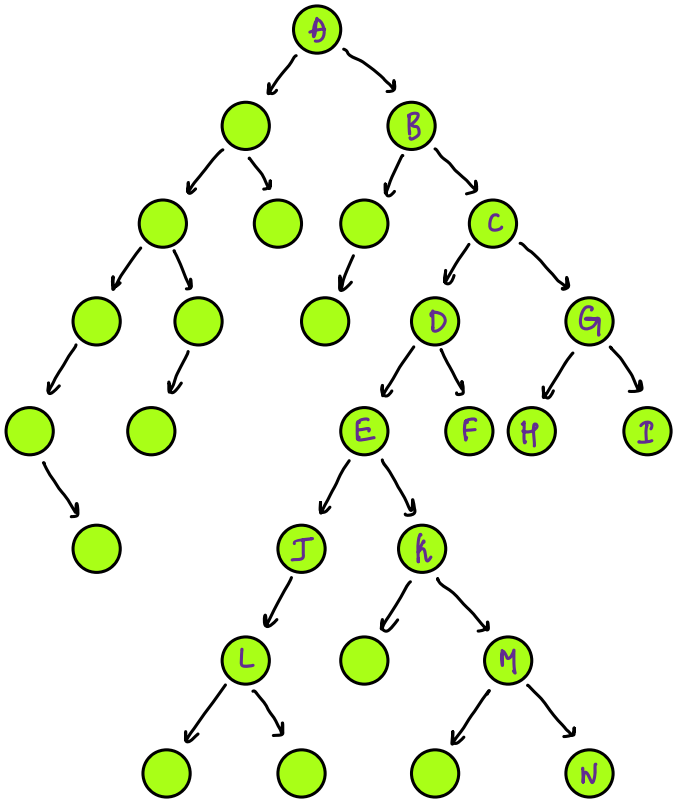
Note: According to above logic: 6 15 14 4 3 : Source node to root node

```
ArrayList<Node> al;
boolean check(Node root, int s){ TC: O(N) SC: O(H)
    if(root == null){return false}                    // Height of Tree
    if(root.data == s){al.add(root); return true}
    if(check(root.left,s) || check(root.right,s))
    {al.add(root); return true}
    else{return false}
}

ArrayList<Node> getPath(Node root, int s){
    al = new ArrayList<>(); // Only before function we reinitialize
    check(root,s);
    //reverse arraylist al
    return al;
}
```

LCA: Lowest Common Ancestor : lowest common ancestor is defined between 2 nodes p & q as the lowest level node in Tree, which is an ancestor to both p & q. Ancestor: itself/parents/grandparent.../rootnode

Note: We allow a node to be ancestor to itself.



Eg: LCA(K, L) : Node E is lowest Common Ancestor

K : A B C D E K

L : A B C D E J L

Eg: LCA(J, M):

J : A B C D E J

M : A B C D E K

Eg: LCA(D, C):

D : A B C D

C : A B C

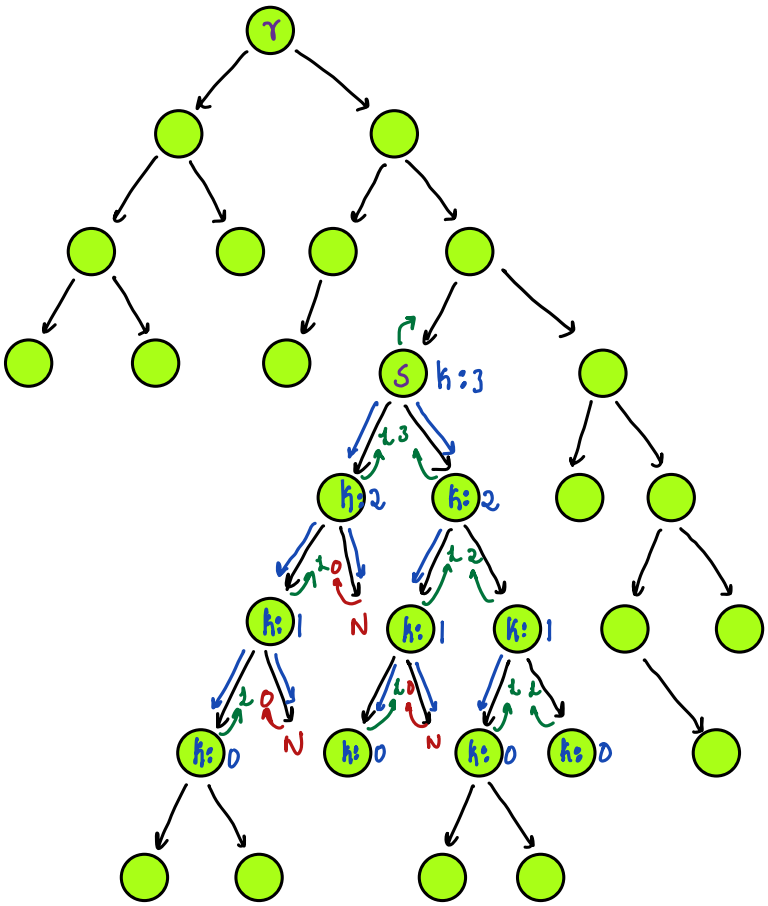
```
Node LCA(Node root, int p, int q){ TC: O(N)
    //Step1:
        1. get path of p from root
        2. get path of q from root
    //Step2:
        Iterate on path & get last common node.
}
```

Q2: Given a source node how many nodes are there distance exactly k.

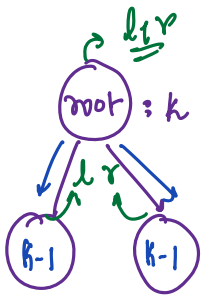
Note1: All Nodes be below source.

Note2: Distance between nodes calculate, no: of edges between them

Eg:



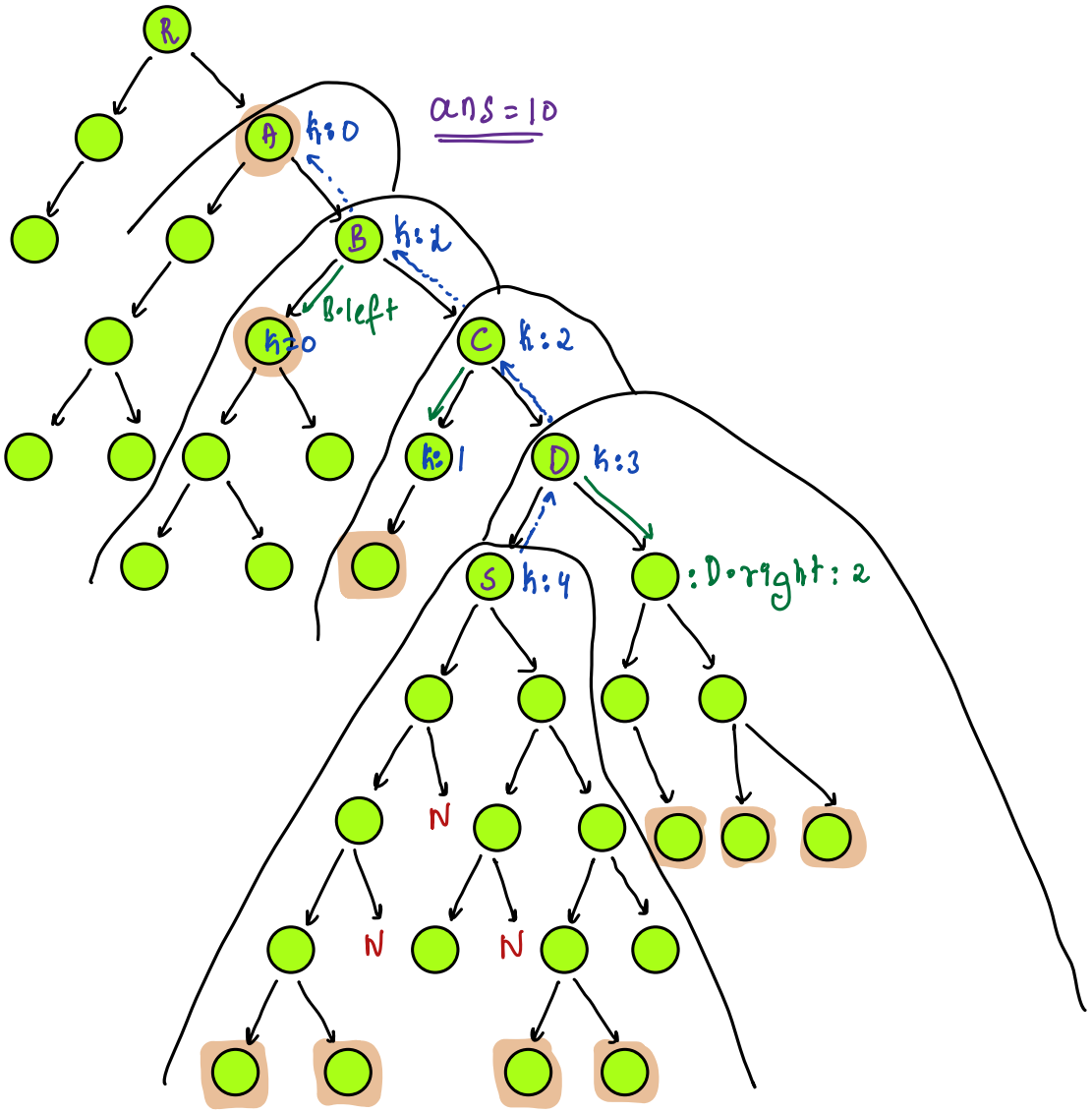

```
int below(Node s, int k){
    if(s == NULL){return 0}
    if(k == 0){return 1}
    int l = below(s.left, k-1)
    int r = below(s.right, k-1)
    return l+r
}
```



38 Calculate no: of nodes are at distance k from source.

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

Note2: Binary Tree contains only distinct values



Step1: Get path from source → root node

Note: In path for node at index i its child is at index i-1

	0	1	2	3	4	5
Steps:	S	D, k-1	C, k-2	B, k-3	A k-4	R
Path		S, k / D.right k-2	C.left k-3 / D, k-1	B.left k-4 / C, k-2	B, k-3 / if k-4 == 0: ans+=1 break;	
k =	below(S, k)					
below(S, k)	ans+=4	ans+=3	ans+=1	ans+=1		

	0	1	2	3	4
P:	S	D	C	B	A

```
int countNodes(Node r, int s, int d) {   TC: O(N)  SC: O(H)
    //Step1: Get path from source → root.
    ArrayList<Node> p = getPath(r, s); // Path from root → source
    p.reverse() // reverse it

    int ans = 0

    ans = ans + below(p.get(0), k);
    int N = p.length();                                   p[i-1]
    for(int i = 1; i < N; i++){
        int dist = d - i;
        if(dist == 0) { ans = ans + 1; break}
        // For ith index, child is at i-1th index
        if( p.get(i).left == p.get(i-1)){
            ans = ans + below(p.get(i).right, dist-1)
        }
        else{
            ans = ans + below(p.get(i).left, dist-1)
        }
    }

    return ans;
}
```

if (p[i].left == p[i-1]) {

p[i] → p[i-1]

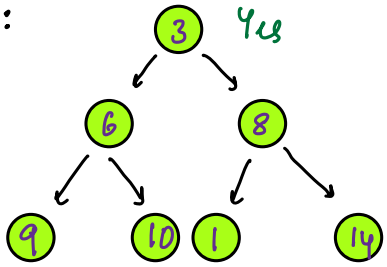
if p[i].right = p[i-1]

p[i] → p[i-1]

Perfect Binary Tree:

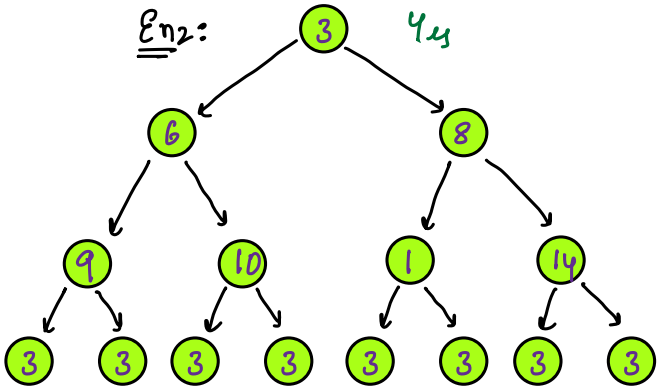
A binary tree in which all the leaf nodes are at same depth and all non leaf nodes have exactly 2 children

Eg1:



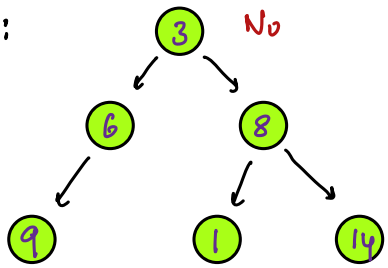
Yes

Eg2:



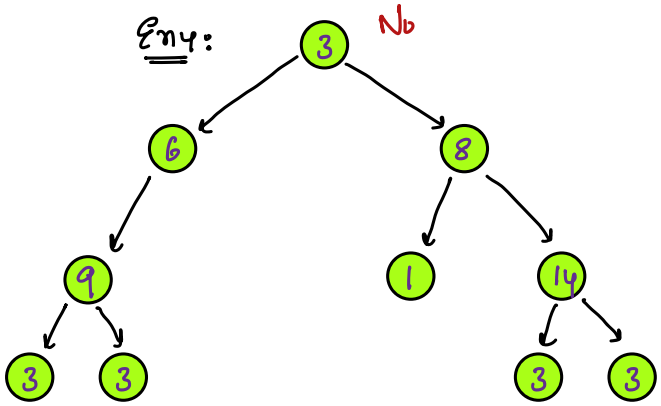
Yes

Eg3:



No

Eg4:



No