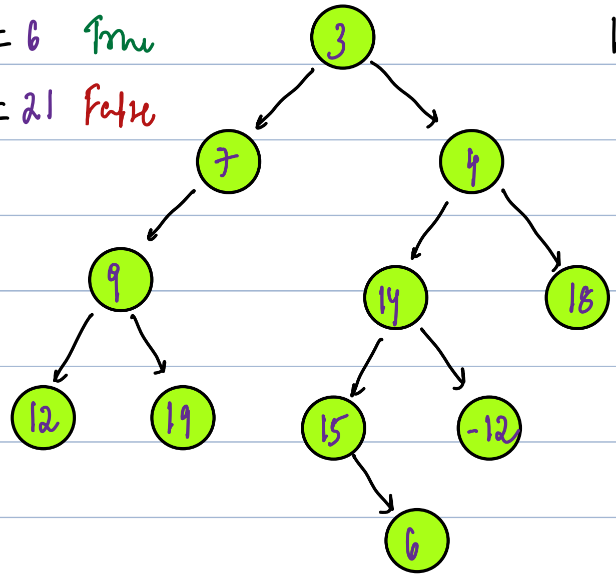
Todays Content

1. Search Node
2. Path from root to leaf
3. Nodes at k distance level

Q Given a B.T which contains all unique value, search if there exists a k in Binary Tree.

k = 6 True

k = 21 False



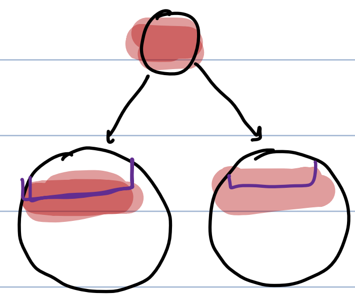
Idea1: Apply any tree traversal & search if k exist & return true/false

TC: O(N)

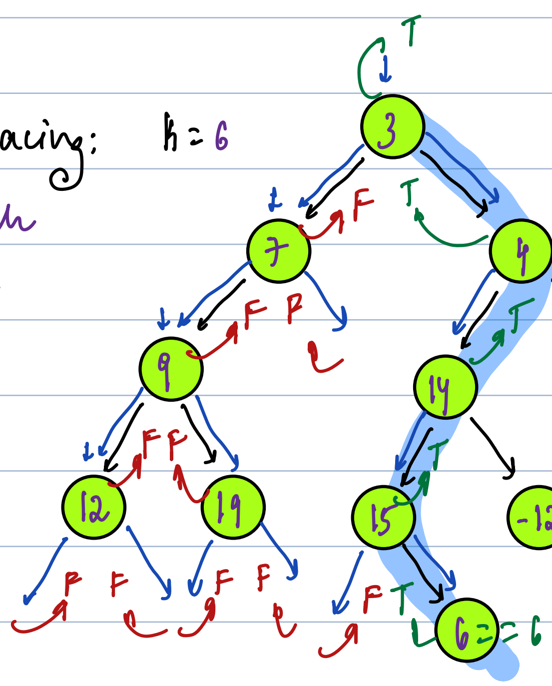
#obs: Given root of BT, search if k exists in that BT & return True/False.

```
bool check(Node *root, int k) {
1. if(root==nullptr) { return false; }
2. if (root->data==k) { return true }
3. if( check(root->left, k) || check(root->right, k)) {
4. } return true
5. return false;
}
```

root



Tracing: k = 6



obs: If we store all nodes, which are returning true, it forms path from node to root.

28 Path from node k to root:

p # Pass by reference

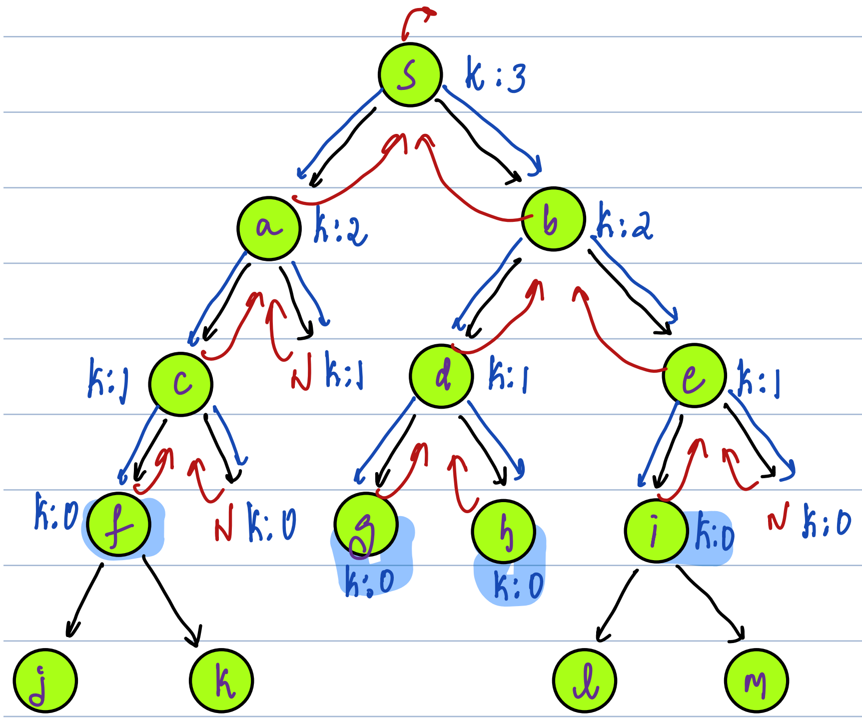
```
bool path(Node *root, int k, vector<Node*> &p){
    if(root==nullptr){return false;}
    if(root->data==k){
        p.push_back(root);
        return true
    }

    if(path(root->left, k, p) || path(root->right, k, p)){
        p.push_back(root);
        return true
    }
    return false;
}
```

```
vector<Node*> solve(Node *root, int k){
    vector<Node*> p;
    path(root, k, p); #once function call is done, path from node to root is
                       stored in p;
    return p;
}
```

Q Given root node & d, store all nodes at d distance

↳ Based on edges




```
void below(Node *root, int d, vector<Node*> &all ){
    if(root == nullptr) {return;}
    if ( d==0){
        all.push_back(root);
    }   return;
    below(root->left, d-1, all);
    below(root->right, d-1, all);
}
```

```
vector<Node*>  solve(Node *root, int d){
    vector<Node*> all;
    below(root, d, all); # Once call is done, all nodes at d distance
    return all;              is stored in all
}
```

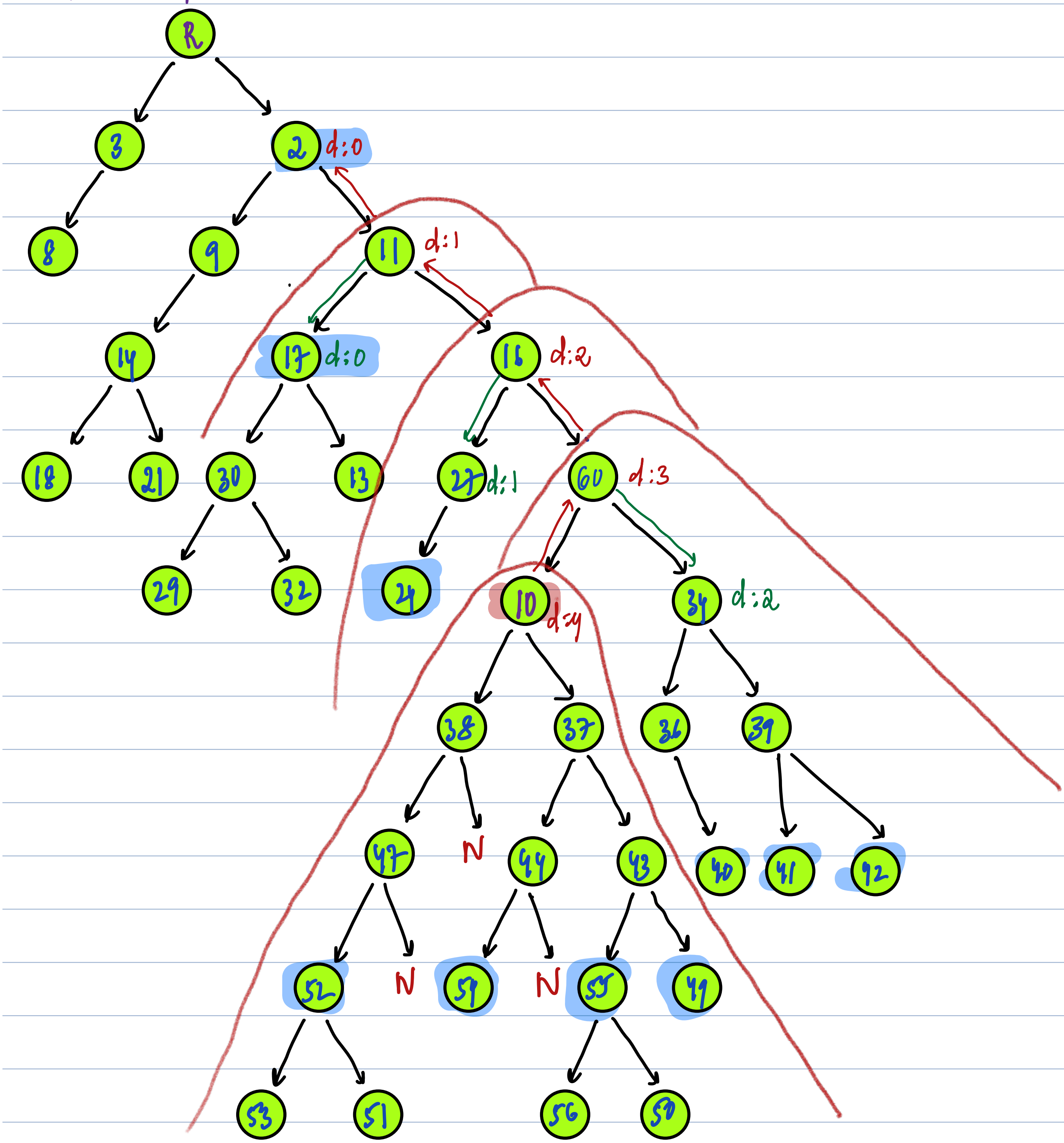
30. Given root node, value k, distance d:

Store all nodes at d distance from node with value k

Note1: Binary Tree contains only distance value.

Note2: Distance between 2 nodes is calculated based on edges.

Ex: k=10 d=4



Approach:

1. Search k in BT & get path from k to root node & store in p.

	0	1	2	3	4	5
p:	10	60	16	11	2	R

2. Apply below function from node p[0] with d distance

below(p[0], d);

3.

	0		1		2		3		4	5
p:	10		60		16		11		2	R
d:	4	d--	3	d--	2	d--	1	d--	0	

p[1].left == p[0]
b(p[1].right, d-1)

p[2].right == p[1]
b(p[2].left, d-1)

p[3].right == p[2]
b(p[3].left, d-1)

d == 0:
Add node
break;

```
vector<Node*> allNodes(Node* root, int k, int d){   TC: O(N)
    vector<Node*> p;
    Path(root, k, p); # Once function done, path filled in p.

    vector<Node*> allN;
    below(p[0], d, allN);
    d--;

    for(int i=1; i<p.size(); i++){
        if( d==0){ allN.push_back(p[i]); break;}
        if( p[i].left == p[i-1]){
            below(p[i].right, d-1, allN);
        }
        else{
            below(p[i].left, d-1, allN);
        }
        d--;
    }

    return allN;
}
```

Using Recursion: For Reading

Search : Recursion

: Not Finding : -1;

: Finding :

Search node using Recursion;

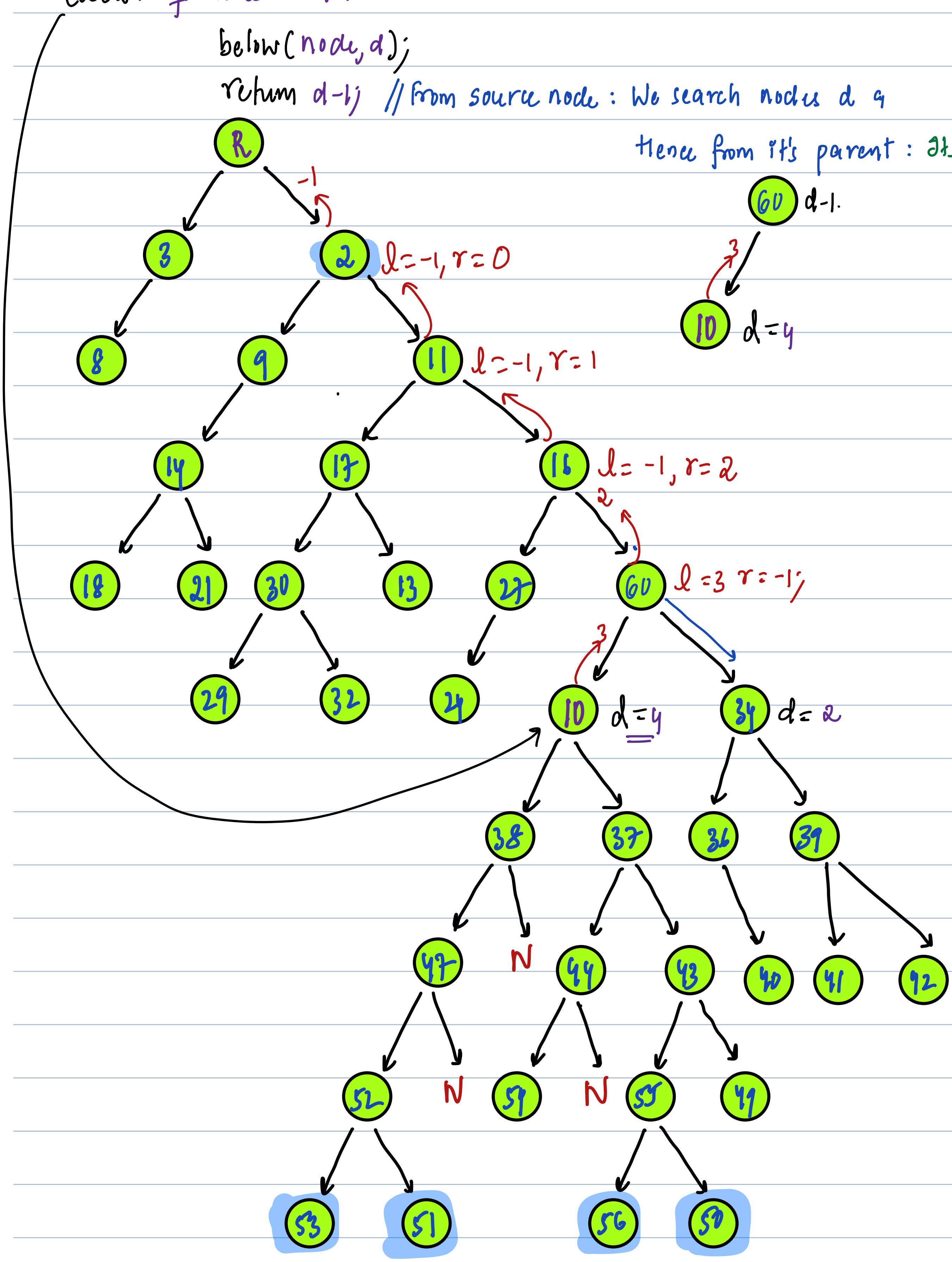
Case 1: If node present : return -1;

Case 2: If node == dest

below(node, d);

return d-1; // From source node : We search nodes d 4

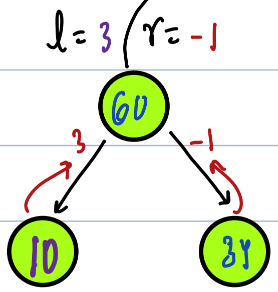
Hence from it's parent : It's d-1



Case 3:

From node 60 $l=3$, $r=-1$:, a. From 60 we need l dist & goto right & search $l-1$ & return $l-1$

return $l-1 : 2$

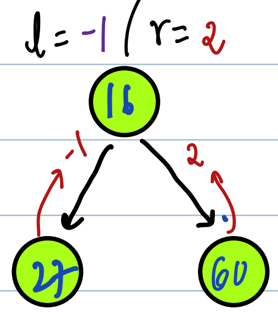


From 60: If it's $l:3$, In that case from it's parent node it's $l-1$.

Case 4:

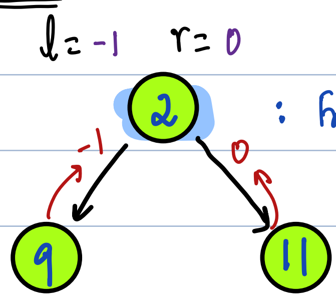
return $r-1 : 1$

a. From 16 we need r dist & goto left & search $r-1$ & return $r-1$.



From 16: If it's $r:2$, In that case from it's parent node it's $r-1$.

Case: 5



: For a node if $(l==0 \| r==0)$ That means from that node, we need 0 distance, directly add that node In ans.

```java
class Solution {
    ArrayList<Integer> ans;
    void nodesBelow(TreeNode root,int K){
        if(root==null || K<0){
            return;
        }
        if(K==0){
            ans.add(root.val);
            return;
        }
        nodesBelow(root.left,K-1);
        nodesBelow(root.right,K-1);
    }
    int nodesAtDistance(TreeNode root, TreeNode target, int k){
        if(root==null){
            return -1;
        }
        if(target == root){
            nodesBelow(root,k);
            return k-1;
        }
        int dl = nodesAtDistance(root.left,target,k);
        int dr = nodesAtDistance(root.right,target,k);
        if(dl==-1 & dr==-1){
            return -1;
        }
        if(dl==0 || dr==0){
            ans.add(root.val);
            return -1;
        }
        if(dl!=-1){
            nodesBelow(root.right,dl-1);
            return dl-1;
        }
        nodesBelow(root.left,dr-1);
        return dr-1;
    }
    public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
        ans = new ArrayList<Integer>();
        nodesAtDistance(root,target,k);
        return ans;
    }
}
```