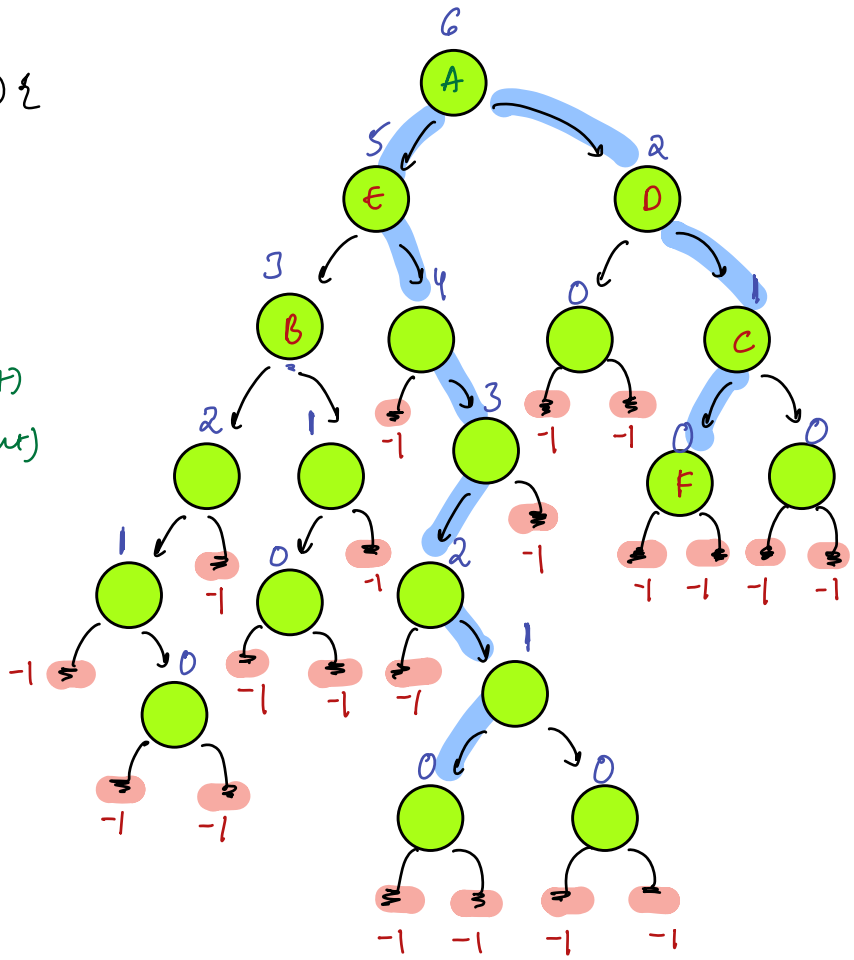
Today's Content :

- → Height
- → Diameter
- → Max Sum Path
- → Nodes at distance c

Height ( ) :

```
int Height ( Node root) {
    if (root == NULL) {
        return -1;
    }
    l = Height (root → left)
    r = Height (root → right)

    return max ( l, r) + 1
}
```



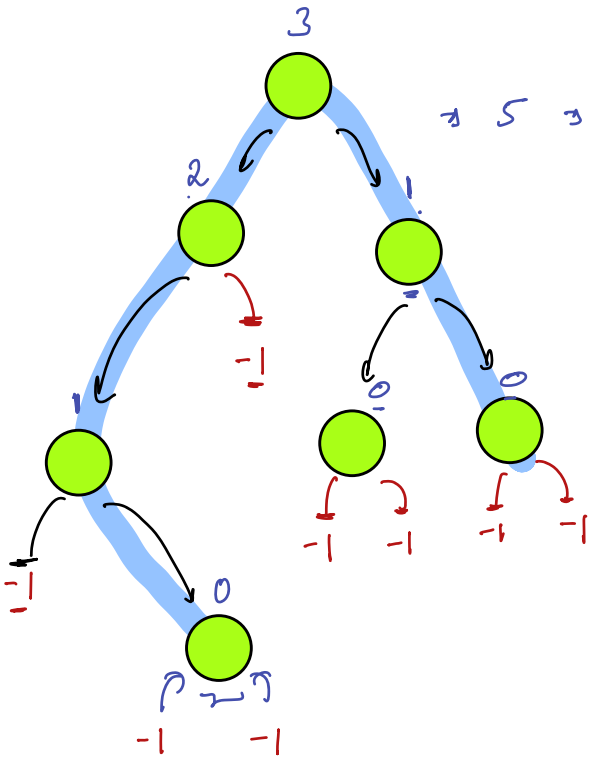
// length of path:

No: of Edges

Q1):

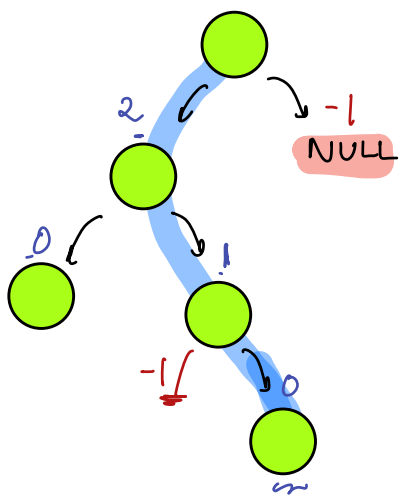
↳ len of max path going across root node : ?

= Height (LST) + Height (RST) + 2



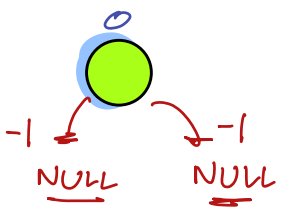
→ 5 → Height(LST) + Height(RST) + 2

→ 5



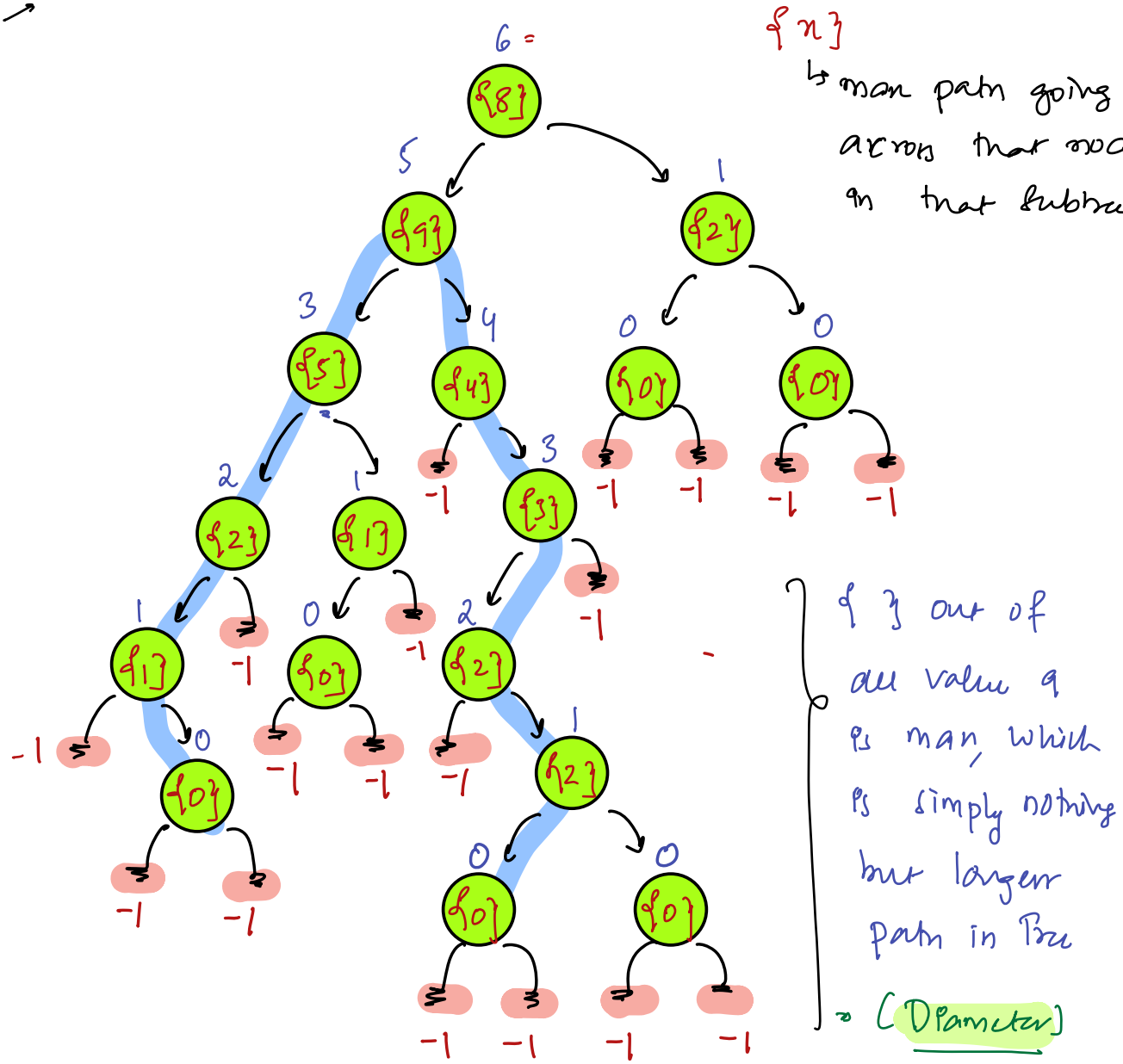
= Height(LST) + Height(RST) + 2

2 + -1 + 2 → 3



= Height(LST) + Height(RST) + 2 = 0

-1 -1


{n}
↳ max path going across that node in that subtree
{ } out of all value q is max, which is simply nothing but longer path in Tree
(Diameter)

// In Java :

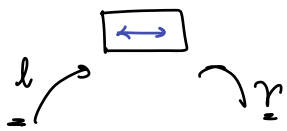
Pseudocode:

In Java

```
static int ans = -1;

int height(Node root) {
    if(root == NULL) {
        return -1;
    }
    l = height(root.left);
    r = height(root.right);
    // length of max path going across root, in that subtree
    ans = max(ans, l+r+2);
    return max(l, r) + 1;
}
```

l+r+2

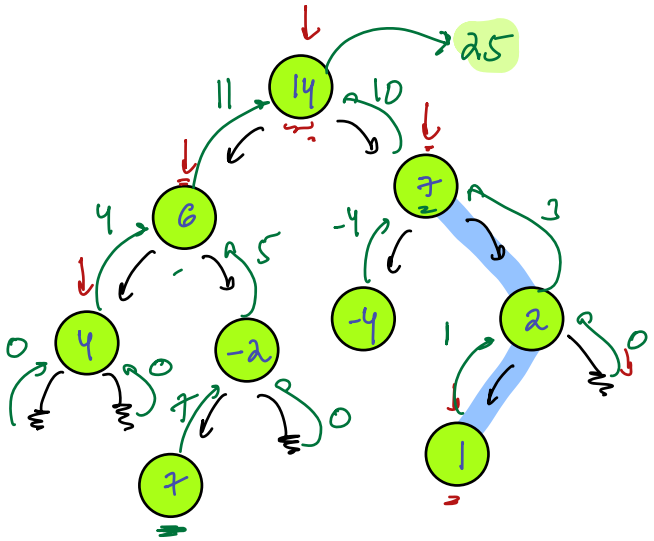


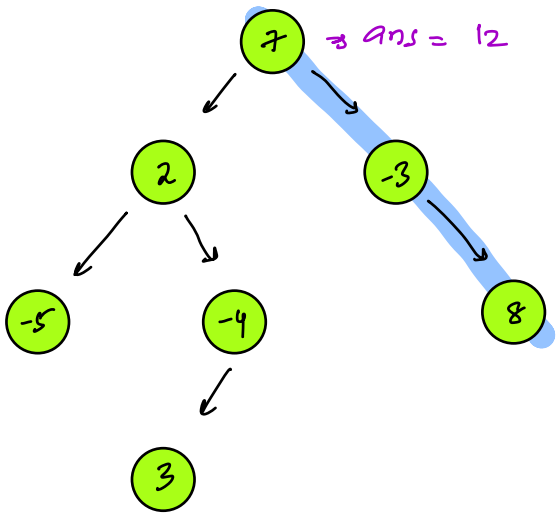
{To get diameter of Tree}

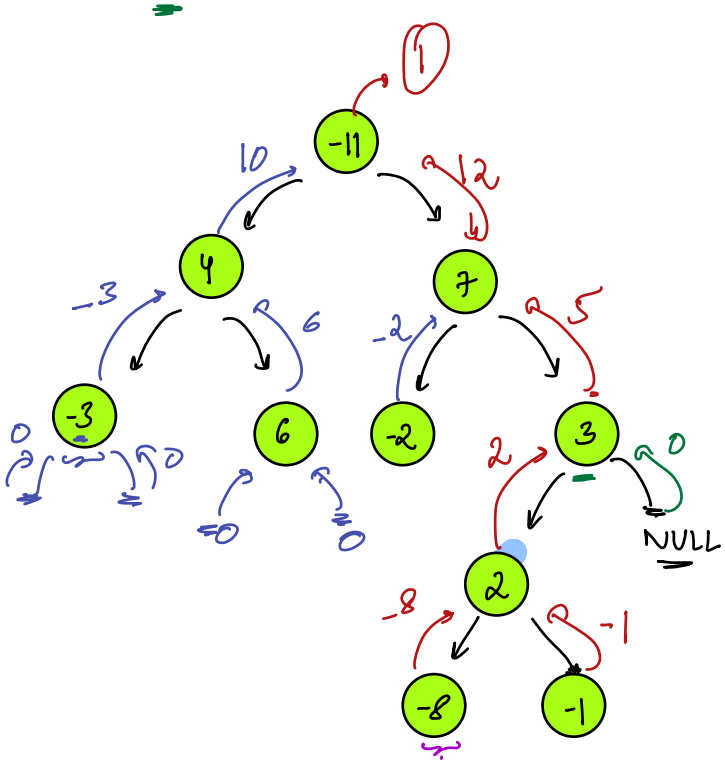
```
main() {
    // Tree construct given root
    // ans = -1 // reinitialize ans to avoid issue with Test cases
    height(root)
    return ans;
}
```

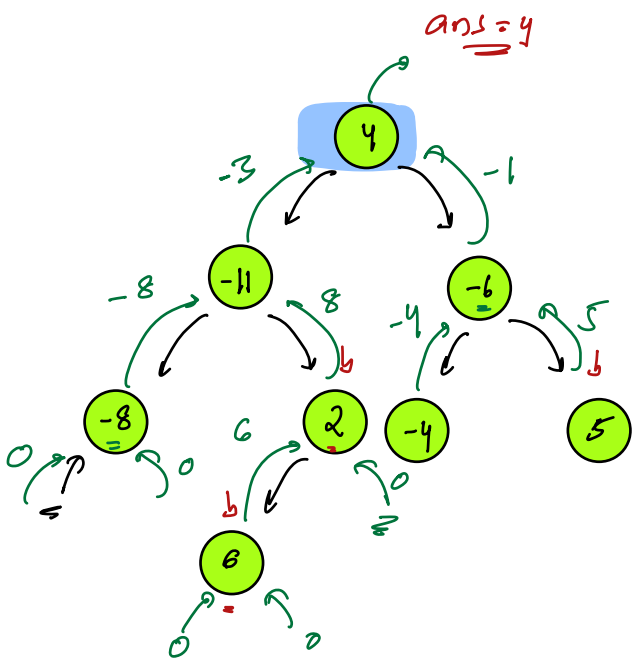
TC: O(N) SC: O(H)

28) Find max sum path starting from Root → { Include root }









Pseudocode:

Ass: Return max path sum, starting at root node

```
int pathSum(Node root) {        TC: O(N)   SC: O(H)
    if (root == NULL) { return 0 }
    int l = pathSum(root->left)   } max path sum in LST
    int r = pathSum(root->right)  } max path sum in RST
    return root.data + max(0, max(l,r))
}
return max(root.data, root.data + max(l,r))
```

Q: Max pathSum containing root node?

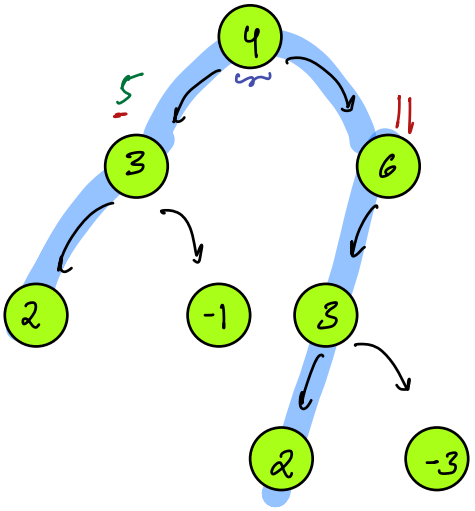
root.data + max(pathSum(lst), 0) + max(pathSum(RST), 0)

Q: Max pathSum ? { TODO } → path with max sum

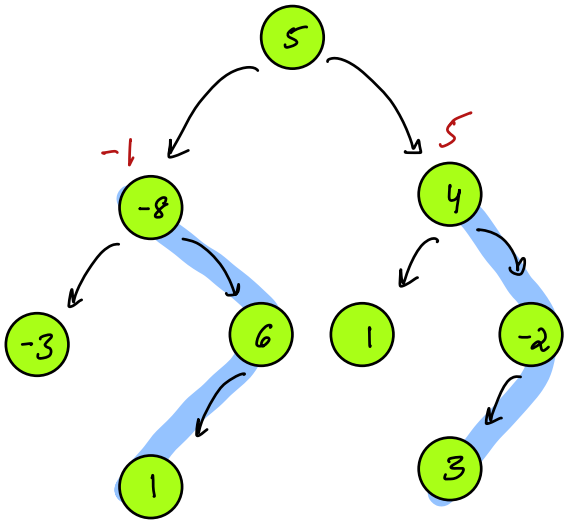
Idea :→ relate it do diameter

Eg1:

ans = 5 + 11 + 4 = 20

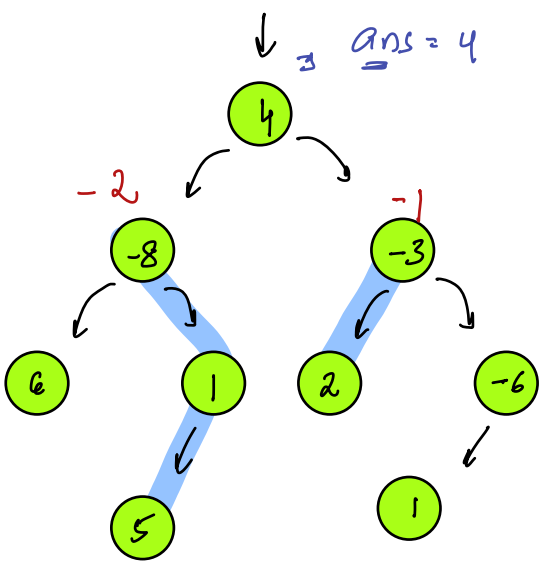


ans = 5 + 5



Eg2:

ans = 4



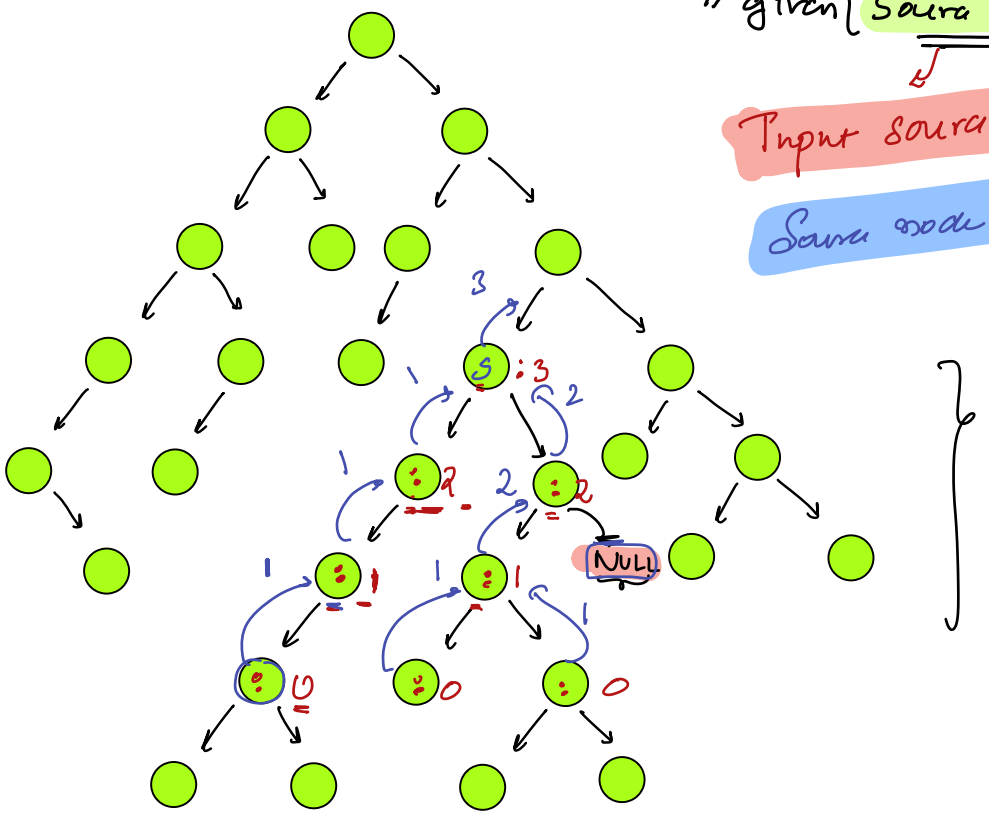
Pseudocode

Q4) Given a source node, how many nodes are there = k distance k, (All nodes should be below source)

// given { Source node } & k

Input source node

Source node address given



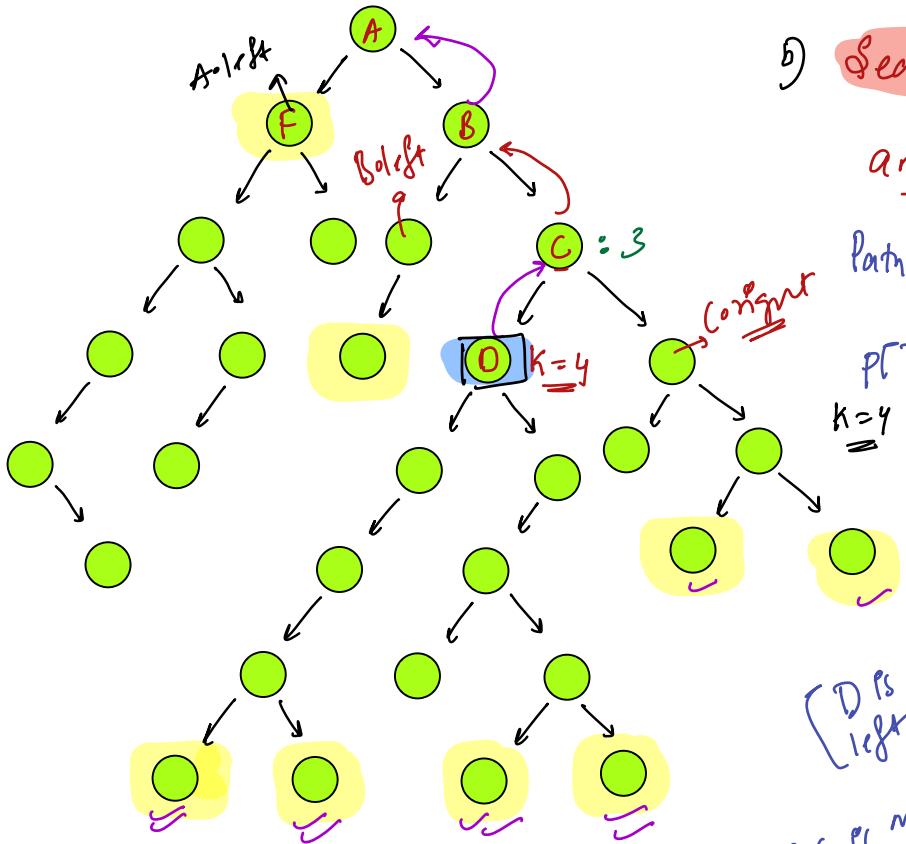
// Pseudocode: // number of nodes are at a distance k below S

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

Qs:) Calculate no: of nodes at a distance k from Source Node

a) Source Node data is given

b) Search for Source Node



ans = 8

Path from Source → Root

p[]:

0	1	2	3
D →	C	B	A

i

k = 4

Below k

(Constant, 2)

k - i - 1

[D is my left of C]

(B.left, 1)

k - i - 1

[C is my right of B]

(A.left, 0)

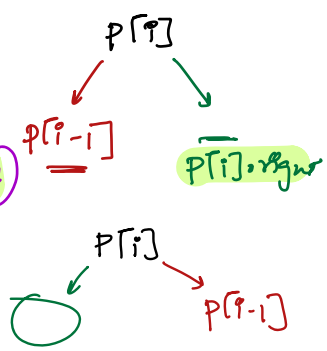
k - i - 1

[B is my left of A]

```
int findNodes(Node S, int k){
    if(S == NULL || k<0) {return 0}
    if(k==0) {return 1}
    int l = findNodes(S.left, k-1)
    int r = findNodes(S.right, k-1)
    return (l+r)
}
```

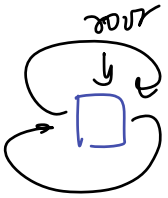
```
int findallNodes(Node root, int s){
    list<Node> p = path(root, s) // path from source → root
    // p[0] is your source node
    int ans = findNodes(p[0], k)
    int n = p.size();
    for(i=1; i<n; i++){ // p[i] is parent of p[i-1]
        if(p[i].left == p[i-1]){
            // search in right
            ans = ans + findNodes(p[i].right, k-i-1)
        }
        else{
            // search in left
            ans = ans + findNodes(p[i].left, k-i-1)
        }
    }
    return ans;
}
```

TC: O(N) + O(N)
≈
TC: O(N)



Tuesday:

Doubts in assignment



LST = COLL

RST = CON