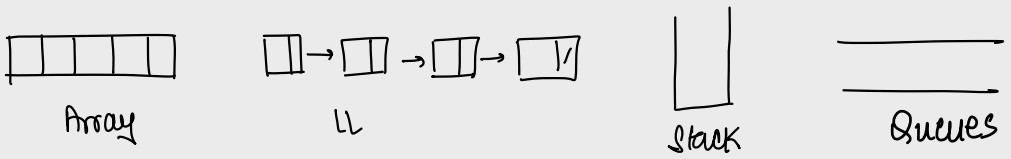
Content :

1) Trees Basics & Terminologies
2) Tree Traversals
3) size() / height() / fillDepth()
4) search(x)
5) Path from source to root.

⇒ Content

⇒ Pace

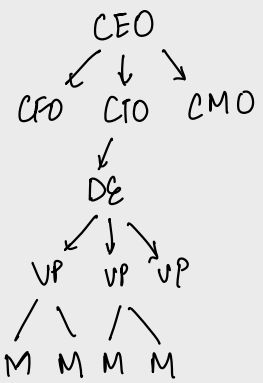
⇒ Time



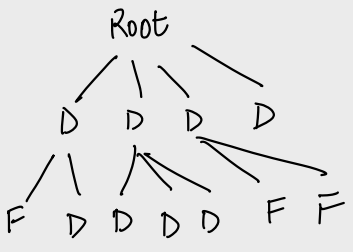
① Data is arranged linearly
② No relationship among ele is there

} LINEAR DS.

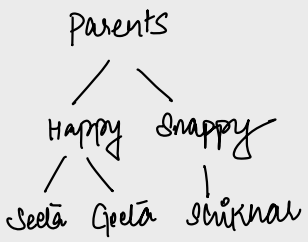
Office



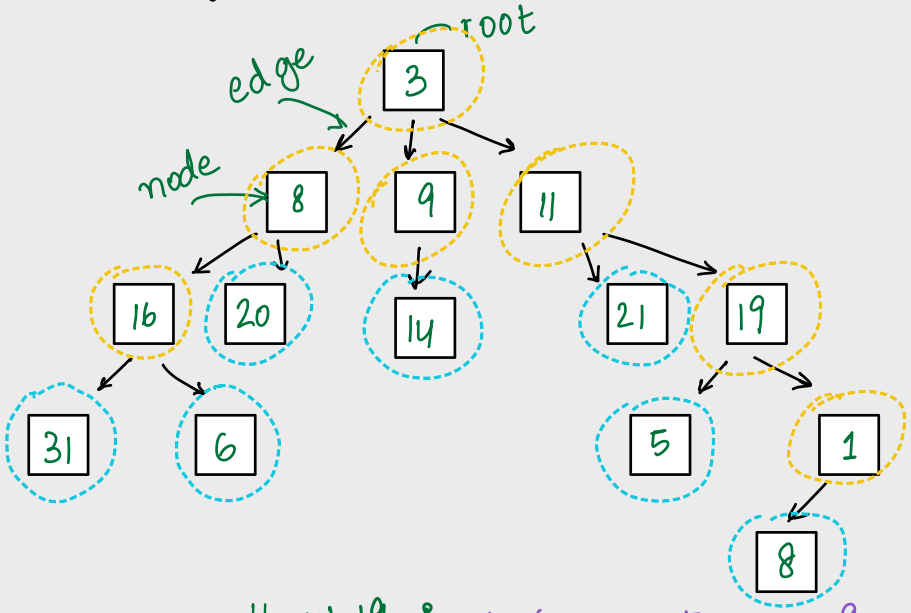
File System



Family Tree



Terminologies — Tree is a hierarchical data structure



11 wrt 19 : 11 is parent of 19

19 wrt 11 : 19 is a child to 11

21, 19, 5, 1, 8 wrt 11 : descendant of 11

11 wrt 21, 19, 5, 1, 8 : 11 is the ancestor to all

16 & 20 : siblings

16 & 14 : cousins

root node : that is without parents

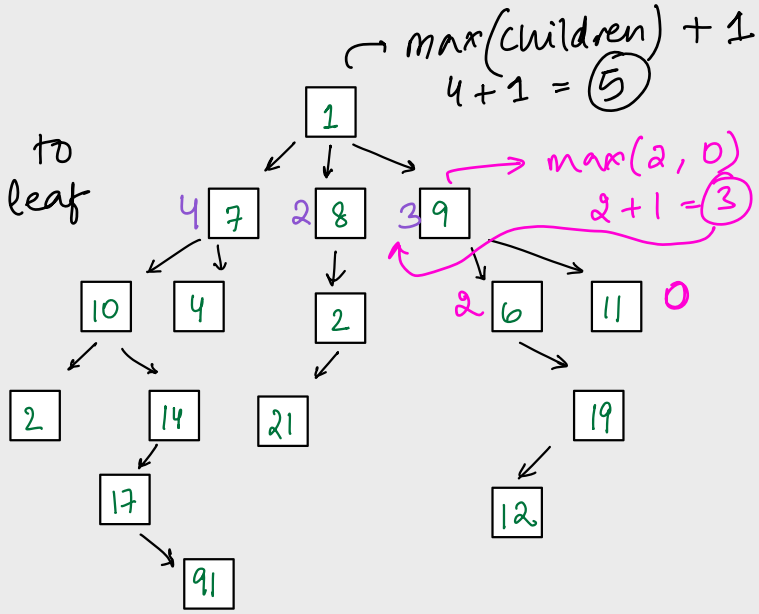
leaf nodes : nodes without children

non-leaf nodes : nodes having atleast 1 child.

Height of a node :

Distance from the node to the farthest reachable leaf

[Distance : No. of edges]



Height (9) 3

Height (10) 3

Height (7) 4

Height (91) 0

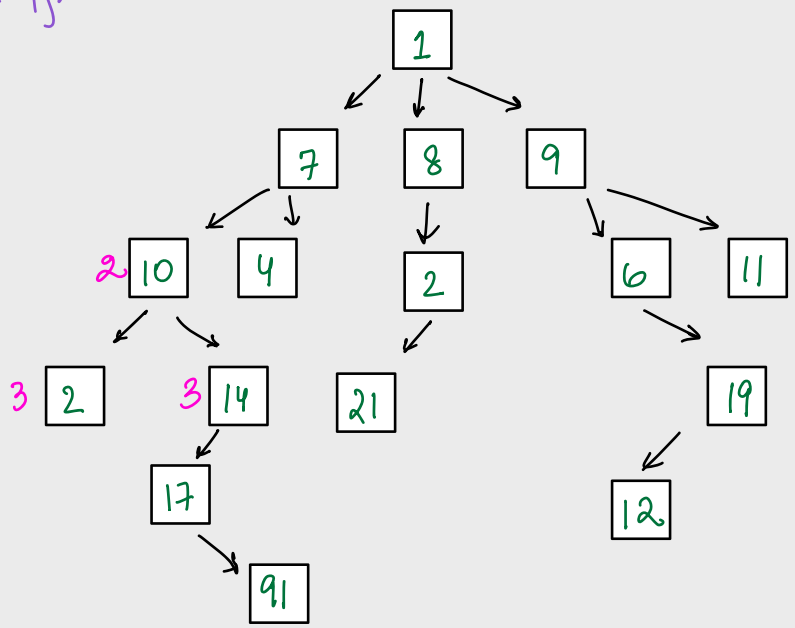
Height (1) 5

Height (node) = 1 + max(height of child nodes)

Height (Tree) : Height (root node)

Depth of a Node

Distance of a node from root node



Depth(8) : 1

Depth(10) : 2

Depth(21) : 3

Depth(19) : 3

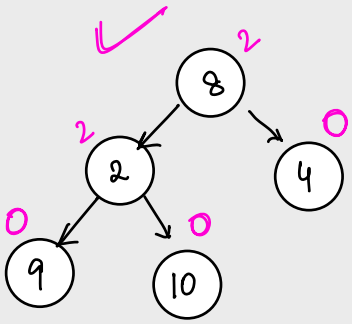
Depth(1) : 0

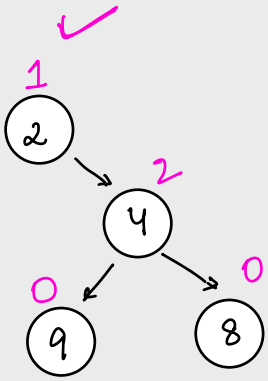
Observation : depth(Node) = d

↳ depth of its child nodes = d+1

depth(Root Node) = 0

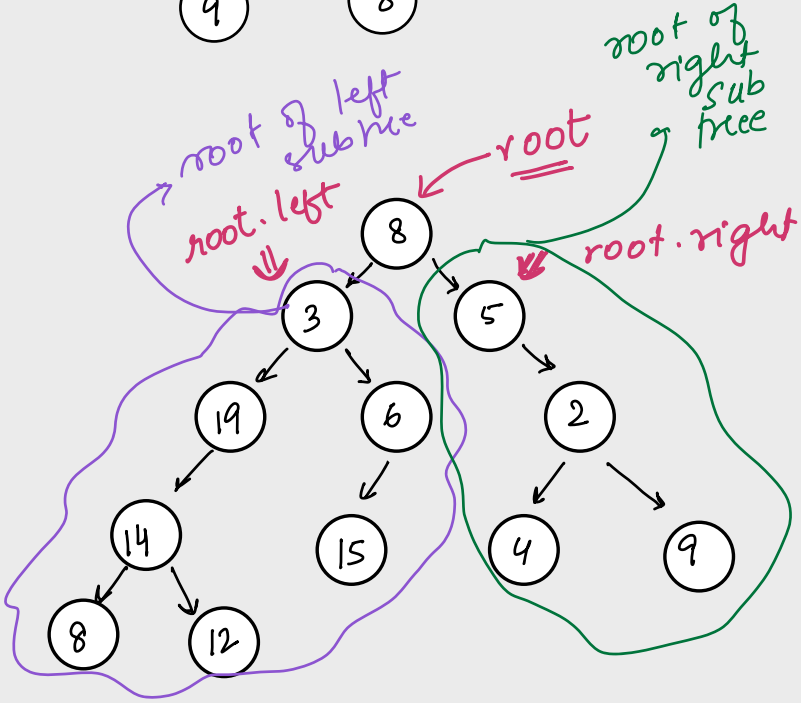
Binary Tree: For every node, no. of child nodes <= 2 (0, 1, 2)





Node structure

```
class Node {
    int data
    Node left
    Node right

    Node(x) {
        data = x
        left = NULL
        right = NULL
    }
}
```



Technique: Recursion

Recursion

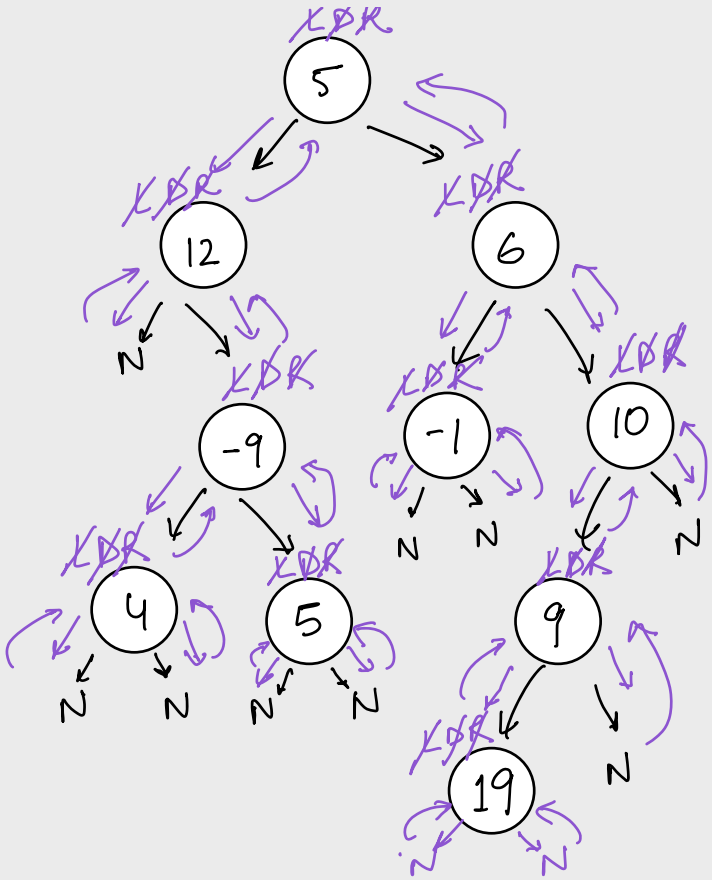
1. Assumption: decide what your function does & believe that it works.
2. Main logic: solving problem using recursive step
3. Base case: stopping condition.

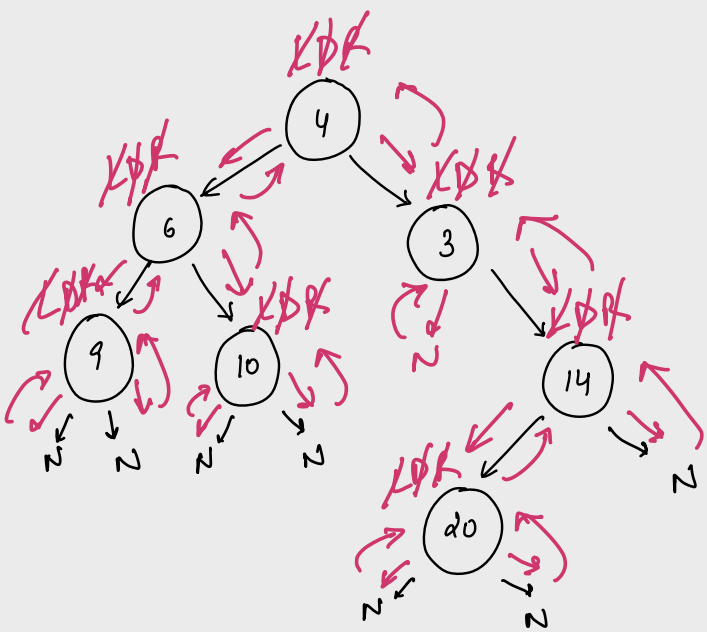
Tree Traversals

preorder	D L R
Inorder	L D R
Postorder	L R D

Inorder :

12 4 -9 5 5 -1 6 19 9 10



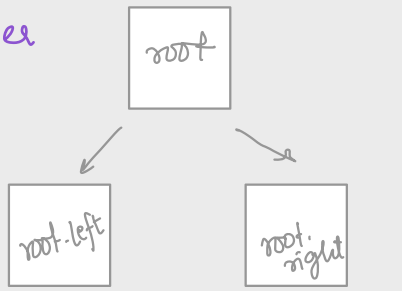


Inorder

9 6 10 4 3 20 14

Pseudocode

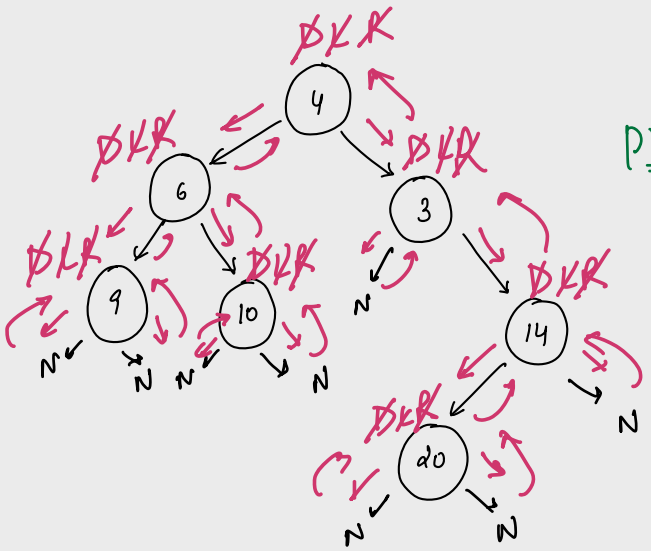
Assumption: print the tree in inorder manner.



inorder : L D R

```
void inorder(Node root) {
    1  if (root == NULL) return
    2  inorder(root.left)
    3  print(root.data)
    4  inorder(root.right)
}
```

inorder : 1 2 3 4
preorder : 1 3 2 4
postorder : 1 2 4 3



preorder : D L R

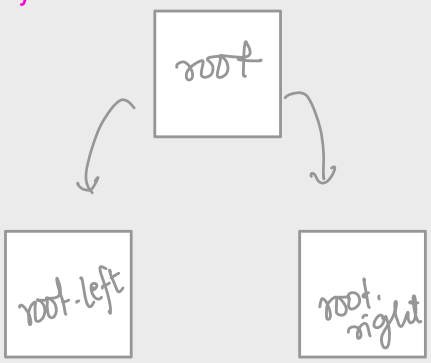
4 6 9 10 3 14 20

postorder : TODO / DIY

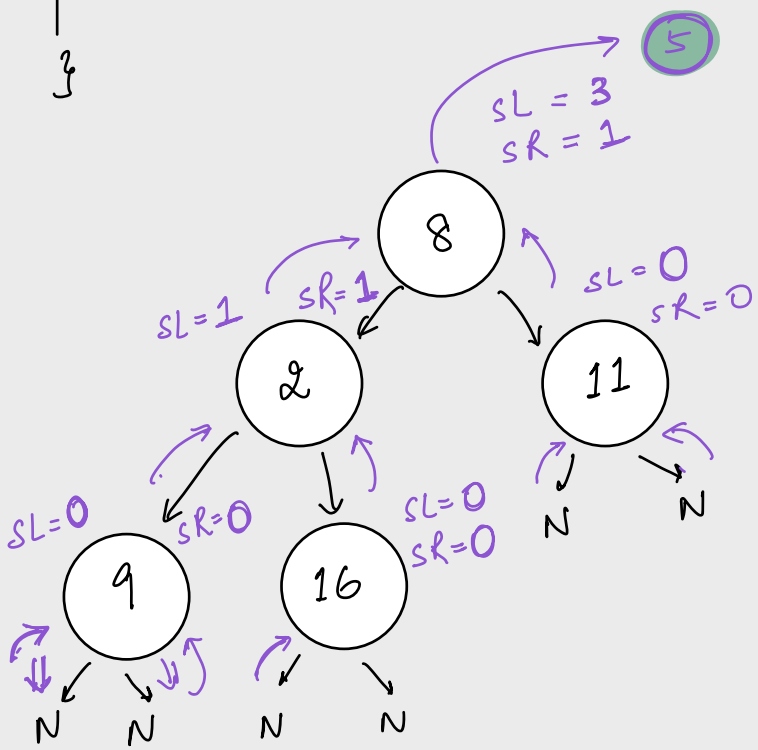
Do it yourself

Calculate size of the tree ⇒ find the no. of total nodes

Assumption: returns the total nodes of the tree.




```
int size(Node root){
    if(root == NULL) return 0
    int sL = size(root.left)
    int sR = size(root.right)
    return 1+sL+sR
}
```



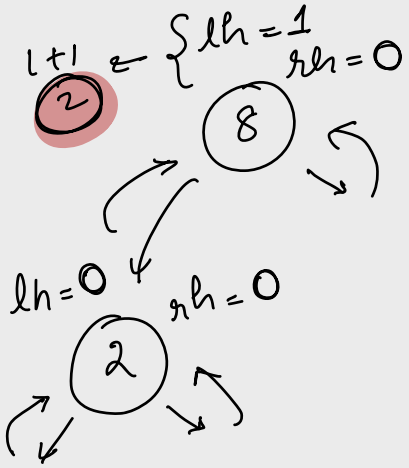
Time : 10:26

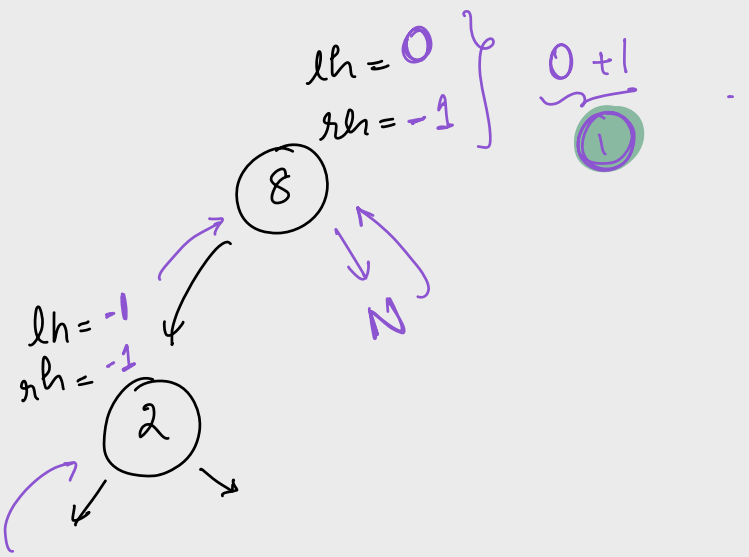
Calculate Height of the Tree

// height (node) = 1 + max(height (LST) , height (RST))

Assumption : Func returns the height of the tree, given the root node.

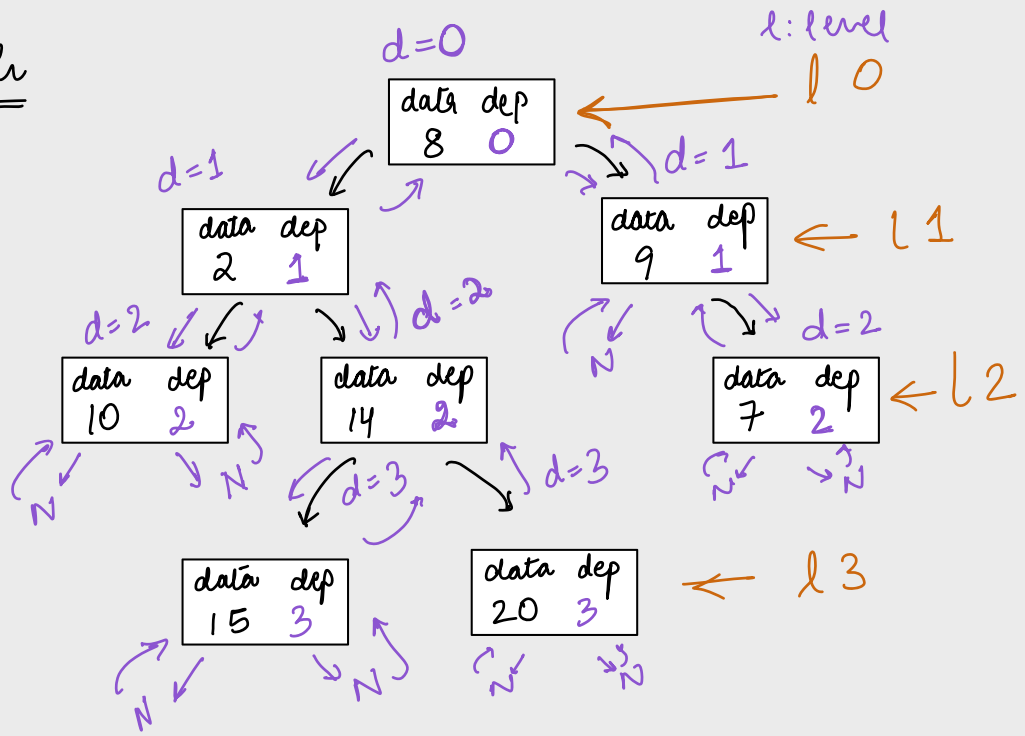
```
int height ( Node root){
    if (root == NULL) return 0;

    lh = height(root.left)
    rh = height (root.right)

    return  1 + max ( lh , rh)
}
```





Que. Fill Depth

```
class Node {
    Node left
    Node right
    int data
    int depth
}
```

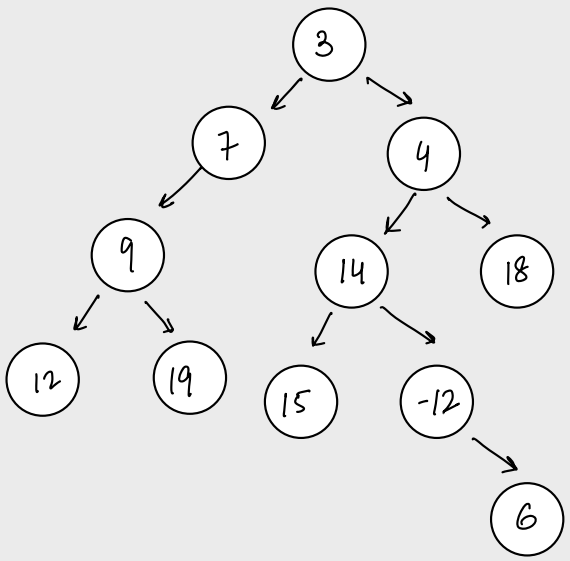



```
void fillDepth( Node root, int d){
    if (root == NULL)
        return

    root.depth = d
    fillDepth( root.left, d+1)
    fillDepth( root.right, d+1)
}
```

fillDepth (root, 0) → calling from main function

0 ⇒ bcz root.depth is 0

Que. Given BT containing all unique values, search if there exists a K in BT.



K = 6 ✓

K = 19 ✓

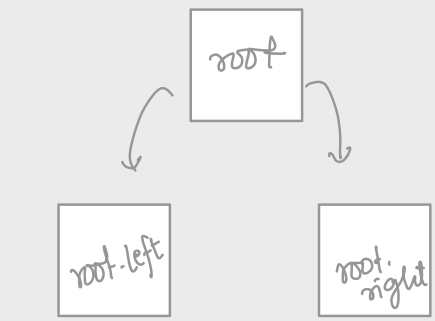
K = 25 ✗

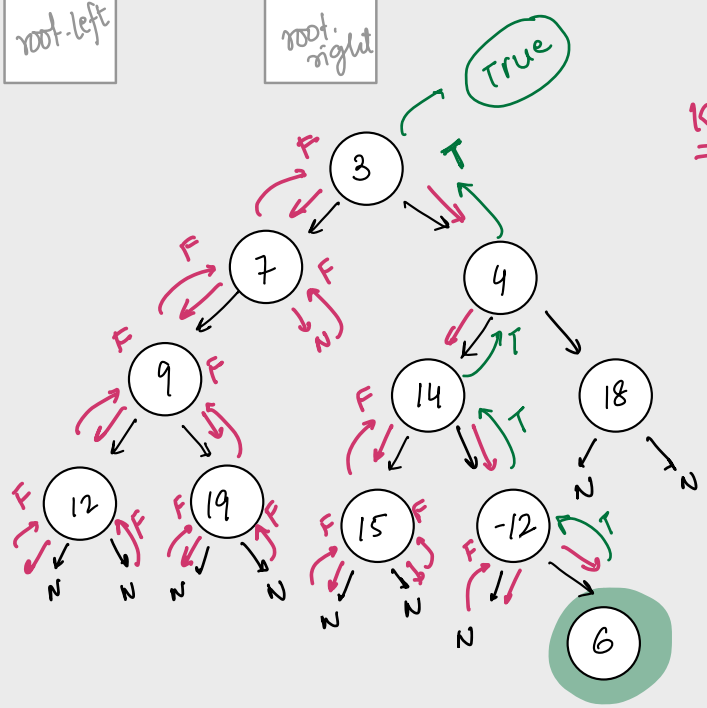
- 1) root.data == K
- 2) check in left subtree
- 3) check in right subtree

} True.

4) if K is not present in any ⇒ return false

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



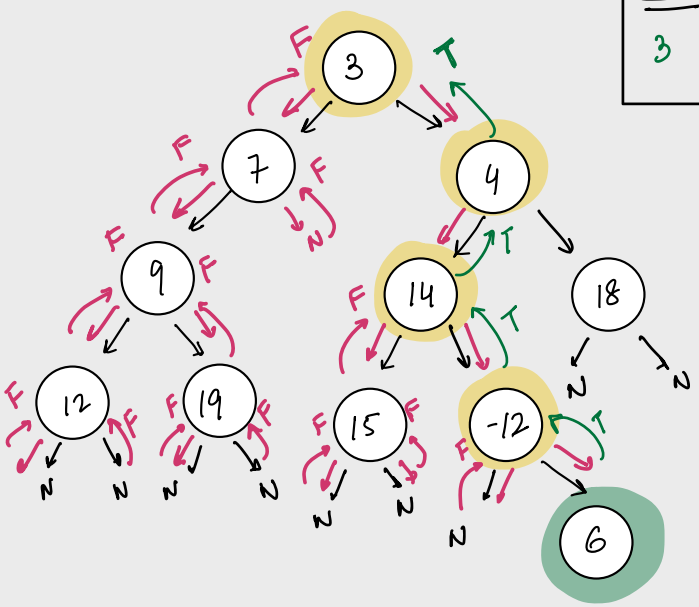


K = 6

Ques. Given a B.T, which contain all unique values.
get the path of node K
↳ assume it's present

Eg: K=6

Path
3 4 14 -12 6



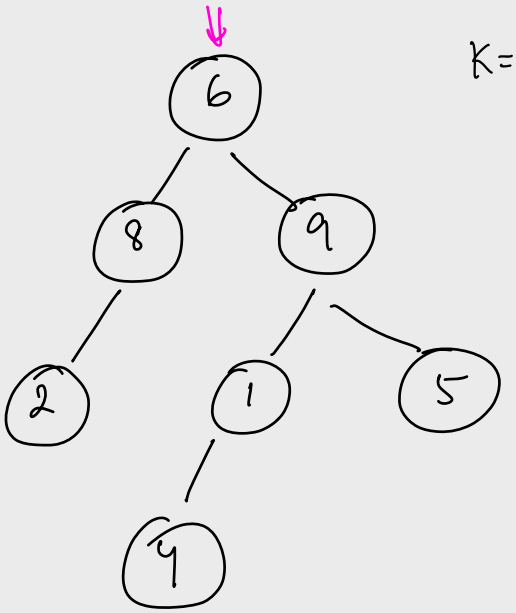
list ⇓ references } O/P.

6	-12	14	4	3

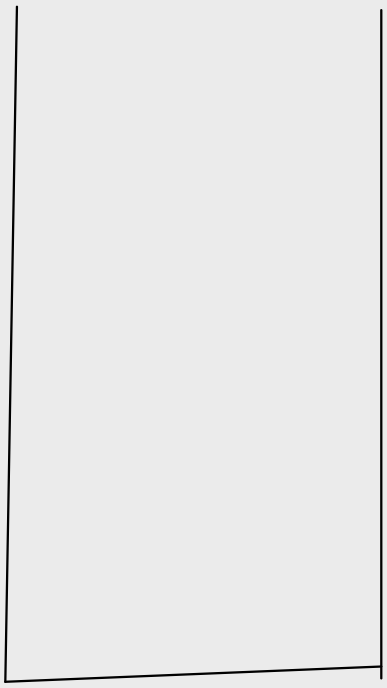
⇓ reverse

3	4	14	-12	6

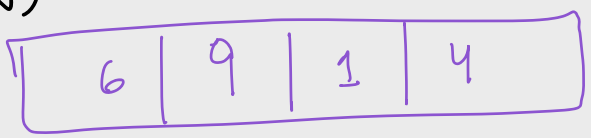
```
list<Node> l ;
bool check ( Node root , int K){
    if (root . NULL) return false
    if ( root.data == K) {
        l.add(root)
        return true
    }
    if (check(root.left, K) || check(root.right , K)) {
        l.add(root)
        return true
    }
    else return false
}
```



K = 4



O(N)



DONE