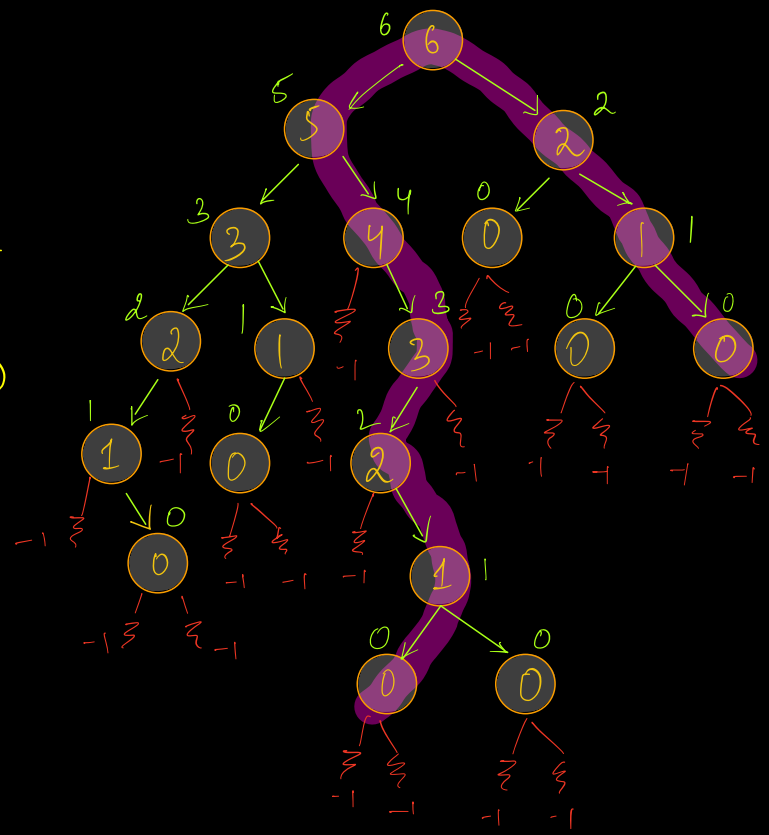
Today's Agenda :-

1) Diameter of a Binary Tree

2) Max Sum Path

3) Vertical Level Order Traversal
   - Top View
   - Bottom View

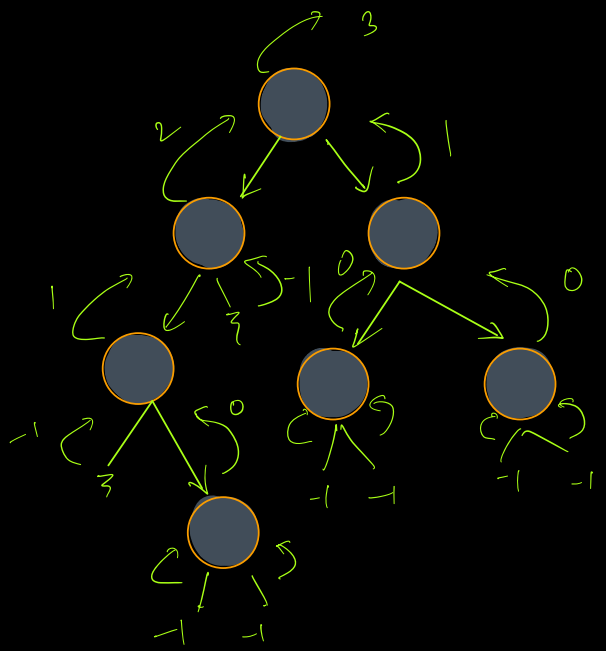
```
int height (Node root) {
    if(root == NULL) return -1;
    int l = height(root.left)
    int r= height( root.right)
    return max(l,r)+1 ;
}
```



Path - edges

Length of max path going across root Node = 9

$\Rightarrow$ Height (LST) + Height (RST) + 2

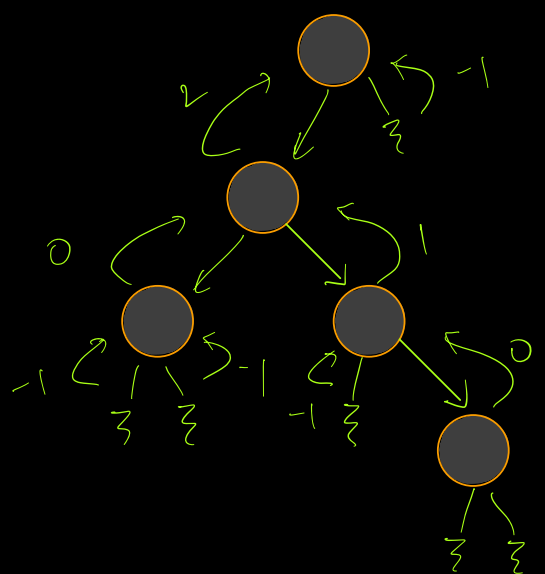


Longest path going across root Node :- 5

Height(Lst) + Height(RST) + 2

2 + 1 + 2

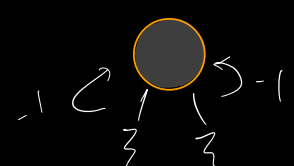
⇒ 5



Longest path going across root Node :- 3

Height(Lst) + Height(RST) + 2

2 + (-1) + 2 : 3

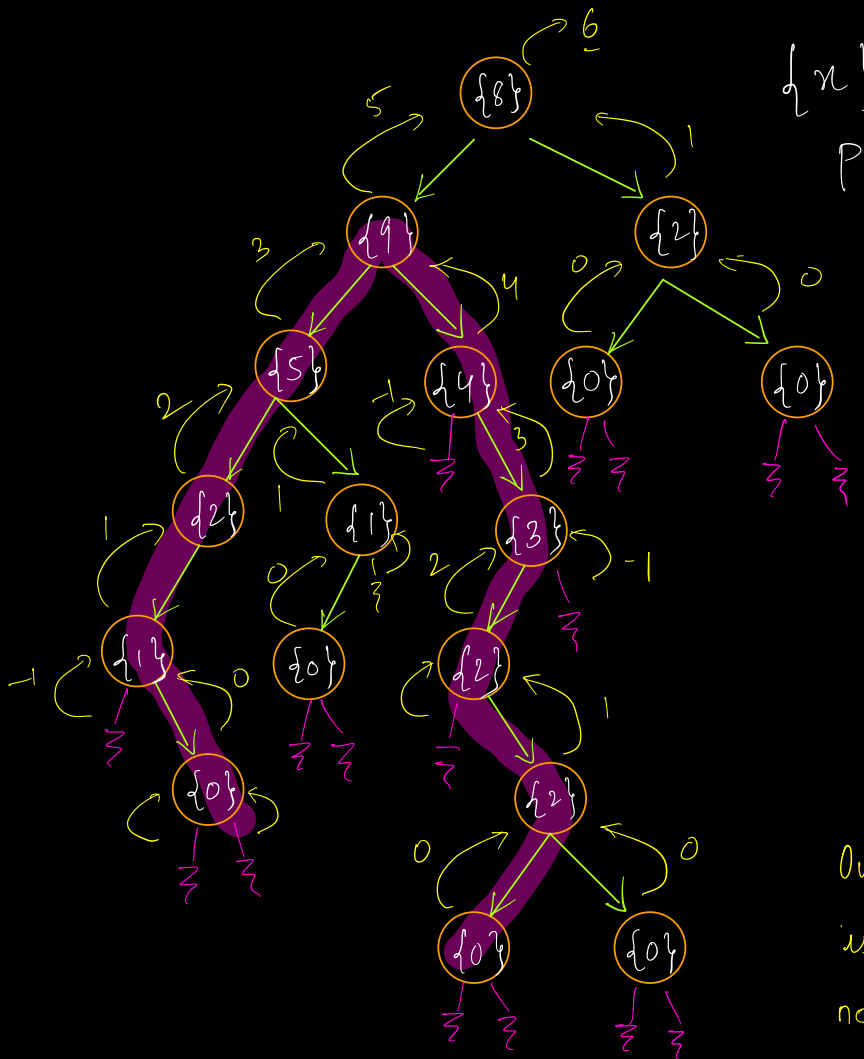


Longest path going across root Node :- 0

Height(Lst) + Height(RST) + 2

(-1) + (-1) + 2

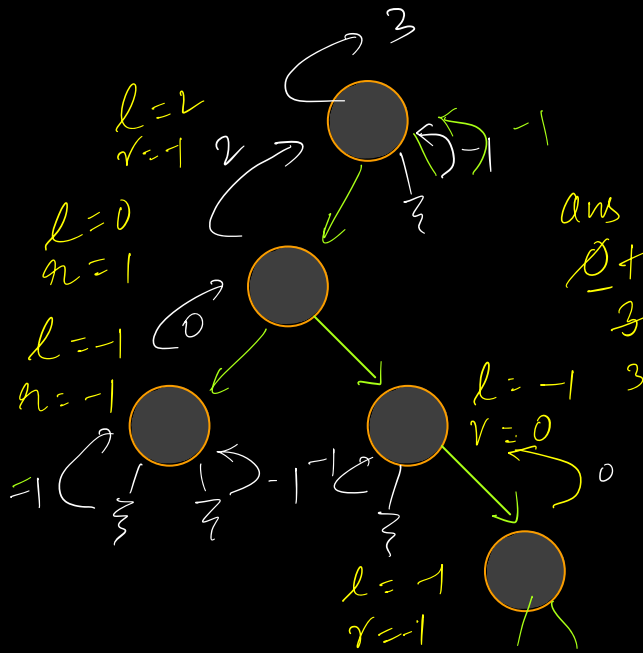
0



{x} - longest path passing across that node

Out of all values, 9 is max which is nothing but longest path in tree.

Diameter



Dry Run

Given tree, find Diameter.

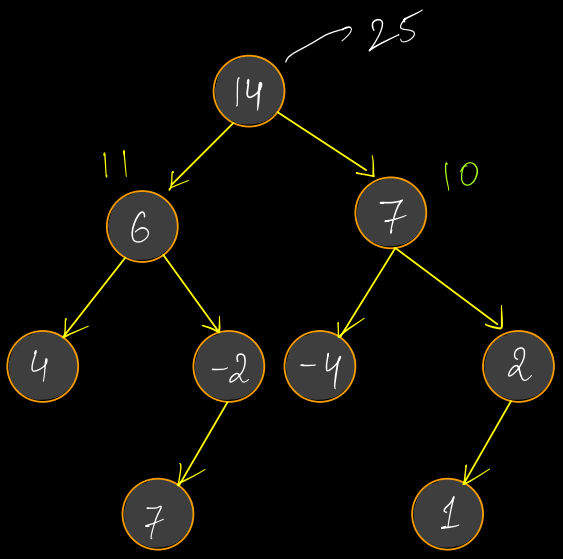
ans = —

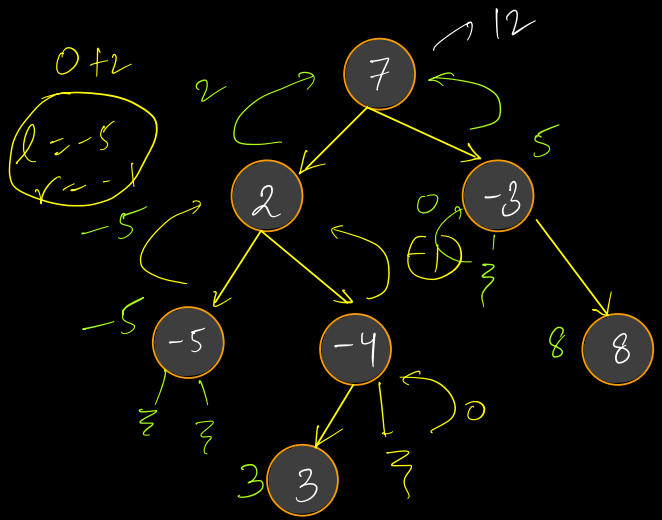
```
int height (Node root){
    if(root == NULL) return -1;
    int l = height(root.left)
    int r = height(root.right)
    // length of max path going across root, in this subtree
    ans = max(ans, l + r + 2);
    return max(l, r) + 1;
}
```

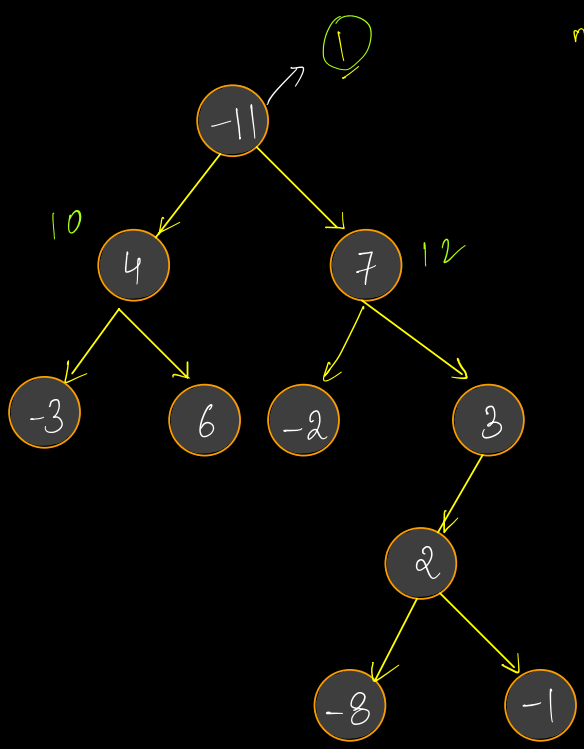
```
int diameter (Node root){
    height(root)
    return ans
}
```

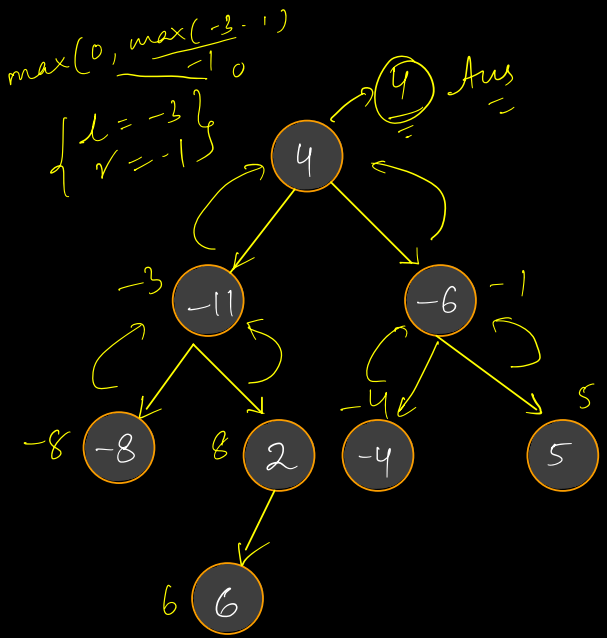
```
main() {
    ans = 0
    h = height(root) ✓
    return ans;
}
```

20) Find max sum path starting from root.









Pseudo Code

Ass: Return max sum path, starting at that root Node

```
int maxSumPath(Node root) {
    if(root == NULL) { return 0 }
    || root = Princess
    int l = maxSumPath(root.left)
    int r = maxSumPath(root.right)
    return root.data + max(0, max(l, r));
}
```

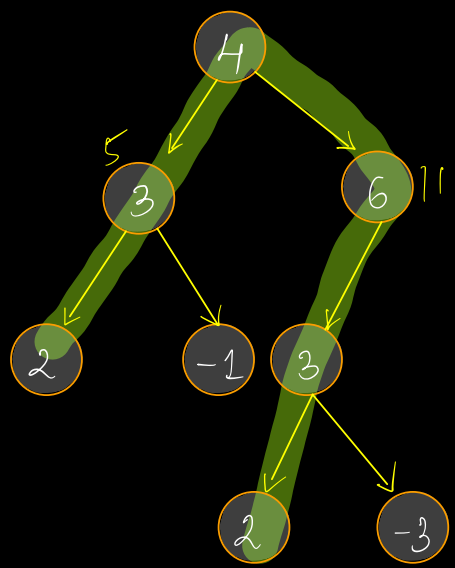
Q) Max Sum Path containing the root node? (pass through)

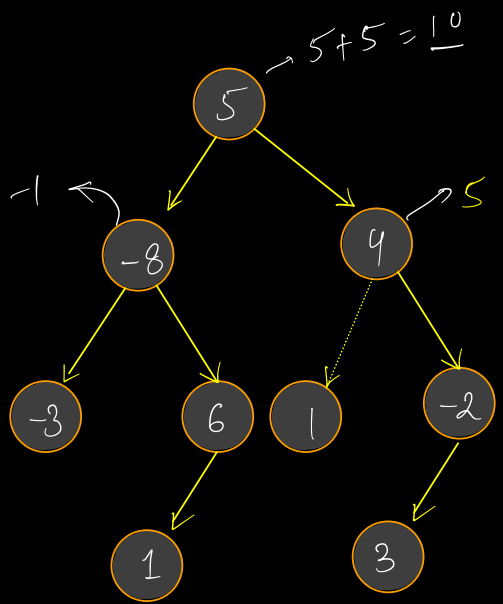
root.data + max(0, maxPathSum(LST)) + max(0, maxPathSum(RST))

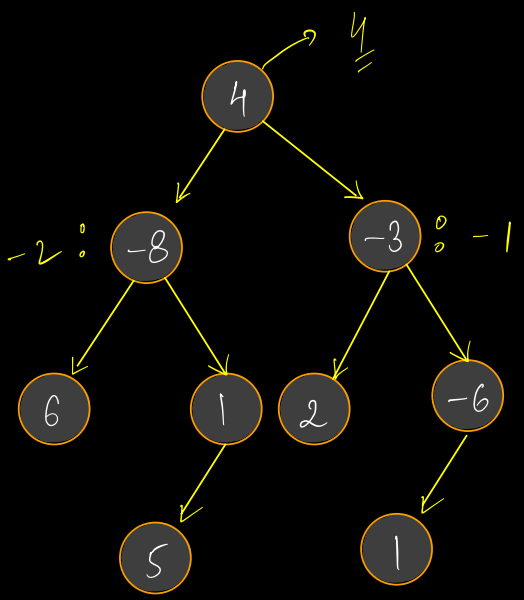
Q) Max Sum Path? (TODO)

Idea: To relate to diameter

Q) ans = 5 + 11 + 4 = 20

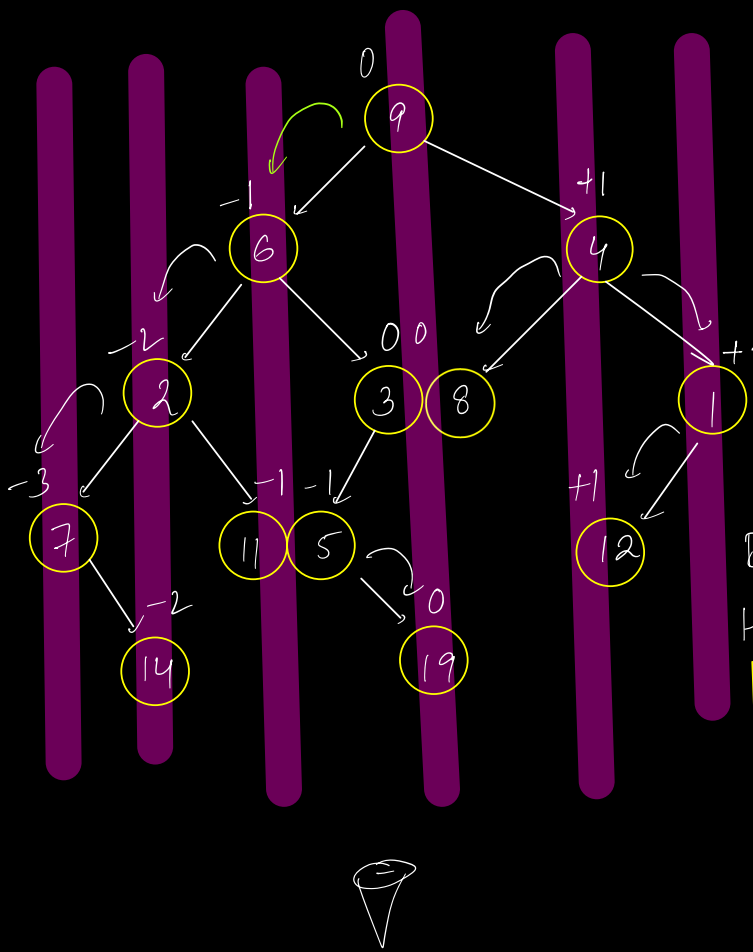






2 min break

Vertical Level Order Traversal :

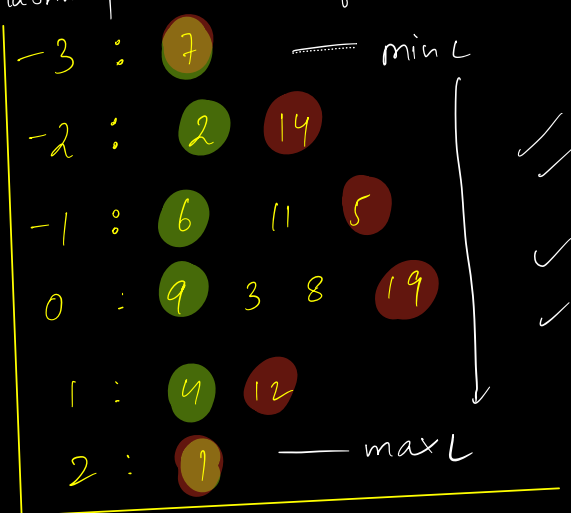


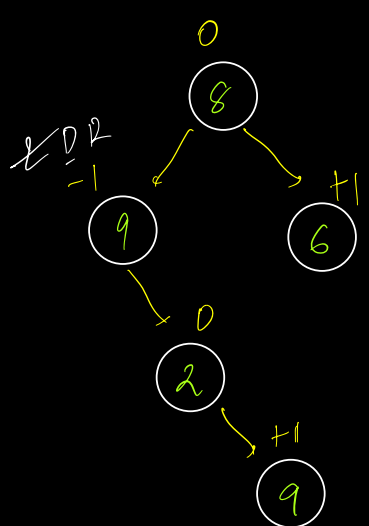
Expected o/p

7
2 14
6 11 5
9 3 8 19
4 12
1

Data Structure :

Hashmap < level, list of Nodes >





Expected HM

-1 : 9
0 : 8 2
1 : 6 9

Preorder D L R

-1 : 9
0 : 8 2
1 : 9 6

Inorder LDR

-1 : 9

0 : 2 8

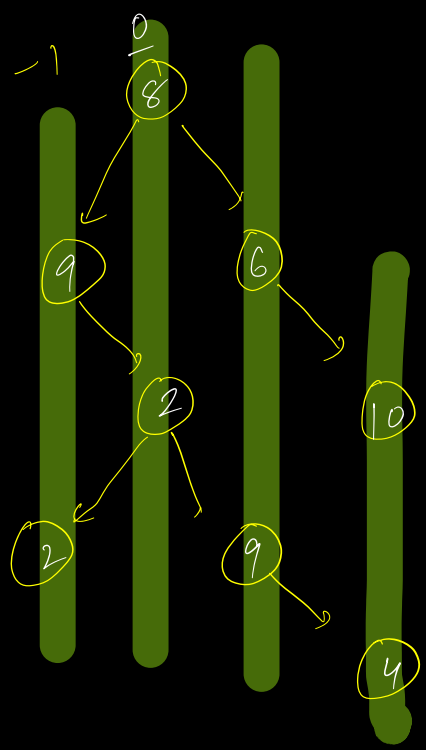
+1 : 9 6

Postorder LRD

-1 : 9

0 : 2 8

+1 : 9 6



Fill HM using level order:

HM

0 : 8 2

-1 : 9 2

+1 : 6 9

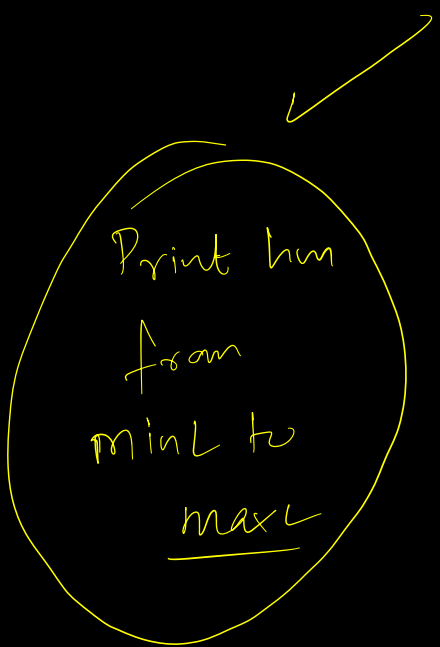
+2 : 10 4

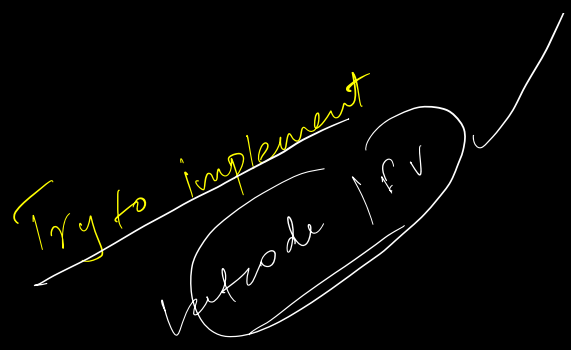
<8,0>	<9,-1>	<6,+1>	<2,0>	<10,2>

<2,-1>	<9,+1>	<4,2>

```
Hashmap < int, listofNodes > hm
Queue < {Node, int} > q
q.insert({root, 0})
while (q.size() > 0) {
    {Node, int} f = q.front()
    q.remove()
    Node t = f.first, int l = f.second

    //insert t at l
    hm[l].add(t)
    //Add left & right in queue
    if(t.left != NULL) {
        q.push({t.left, l-1})
    }
    if(t.right != NULL) {
        q.push({t.right, l+1})
    }
}
```




Try to implement
Method IRV