Today's content

(i) Vertical traversals of tree

(top, bottom, diagonal) ↳ idea.

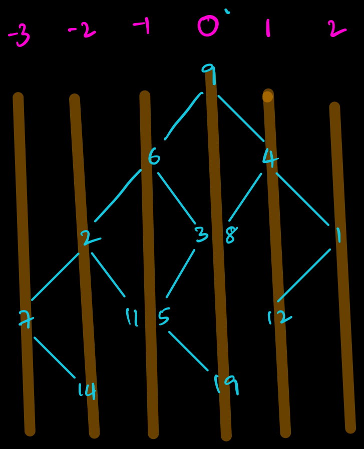
(ii) Diameter of a tree

(iii) Populate next pointer.

Problem solving session this sunday: (7 idea).

Q1:

Vertical level order traversal.



o/p:

-3 : 7

-2 : 2 14

-1 : 6 11 5

0 : 9 3 8 19

1 : 4 12

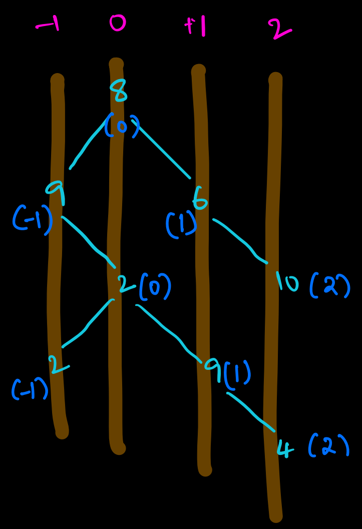
2 : 1

top view

order doesn't matter.

bottom view.

Because we've key & value pairs => try to use hashmap.



-1 : 9 2 → minL

0 : 8 2

1 : 6 9

2 : 10 4 → maxL

~~(0,8)~~ ~~(-1,9)~~ (1,6) (2,10)

↳ (0,(8,2)) ↳ (-1,(9,2))

To traverse level by level -> Queue

Queue <Pair>. queue

Key & value ?

↓ vertical level (int) ← hashmap.

↓ All nodes of a level. (List <Integer>)

class Node

    int val
    Node left
    Node right.

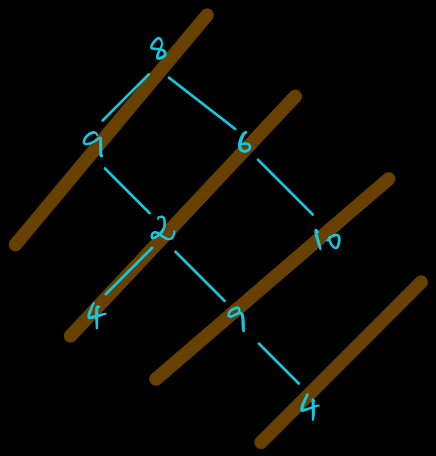
class Pair

    Node node
    int level.

Code:

```
void  vertical Level ( Node  root)
        HashMap < Int, List <Int>>  hm = { } //  hm = { }.

        Queue < Pair >  q ;

        q . enque( new Pair (root, 0))

        while ( q. size() >0)
                Pair  data =  q.pop()
                t   = data. node
                L   = data. level
                minL = min(minL, L)
                maxL = max (maxL, L)
                hm[l] .insert (t. val) //  hm[l] = t. val (update)

                if(t .left != null)
                        q. enqueue (new Pair (t .left, l-1)
                if (t. right != null)
                        q. enqueue (new Pair (t.right, l+1)

        i = minL ; i <= max L ; i++
                // At a particular vertical level i, to access all nodes => hm[i].
                // Simply print hm[i] {TO-DO} . // Vertical level .
                // print 1st ele in hm[i] // top view
                // print last ele in hm(i) // bottom view.
```

Diagonal traversal.



o/p:

8 9

6 2 4

10 9

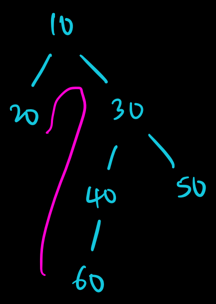
4

idea:

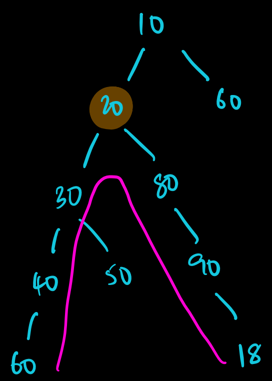
(i) keep on adding left to queue as long as its present

(ii) When left is not there, pop out from the queue and push right & all again keep on pushing as long as left is there.

Q3: Diameter of a binary tree. [longest path between any two leaf nodes, print count of nodes in the path].



o/p: 5

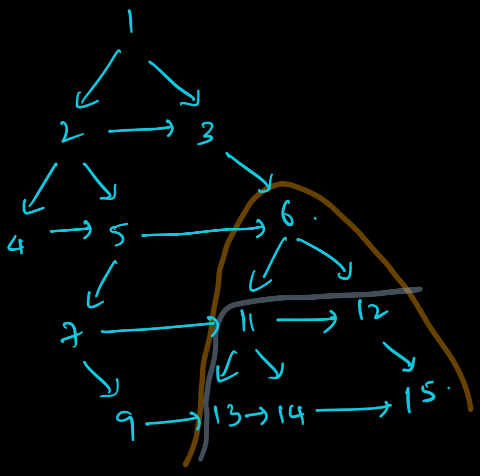


o/p: 7.

```
// result is a global variable.
int height (Node root)
        if(root == null)
            return 0
        lh = height (root.left)
        rh = height (root.right)
        dia = 1 + lh + rh
        res = max(res, dia)
        return max(lh, rh)+1

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

TC : O(N)
SC : O(h)

Q3: find the First node in next level.



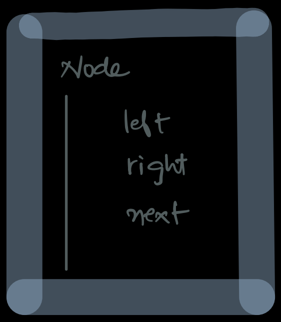
i/p : 2 ; o/p : 4

i/p : 3 ; o/p : 6

i/p : 7 ; o/p : 9

i/p : 11 ; o/p : 13

i/p : 12 ; o/p : 15



for 11 if both the children are null then

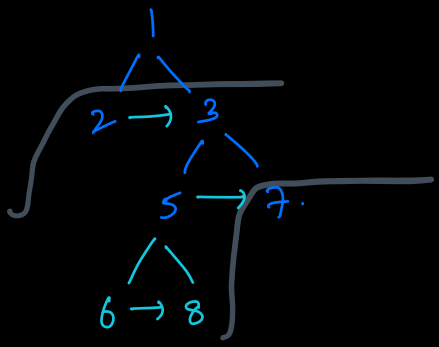
o/p = 15.

Steps:

i) If left is not null, then ans = left.

ii) else if right is not null, then ans = right.

iii) else ans = findNodeInNextLevel(node.next).

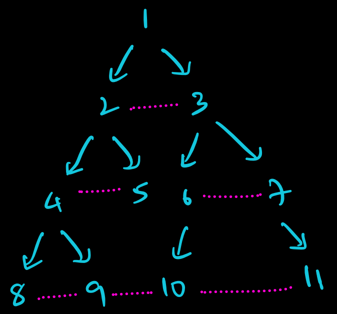


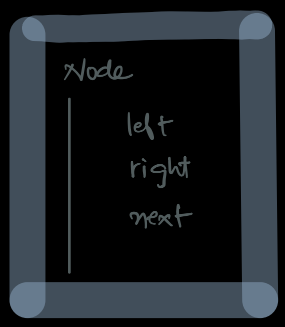
i/p : 2 , o/p : 5.

i/p : 7, o/p : null

```
Node findFirstNodeInNextLevel (Node node)
|
|       if(node==null)
|           return null
|       if(node.left!=null)
|           return node.left
|       if(node.right!=null)
|           return node.right
|       return findFirstNodeInNextLevel(node.next)
```

Q4: Connect nodes in same level.





Observations:

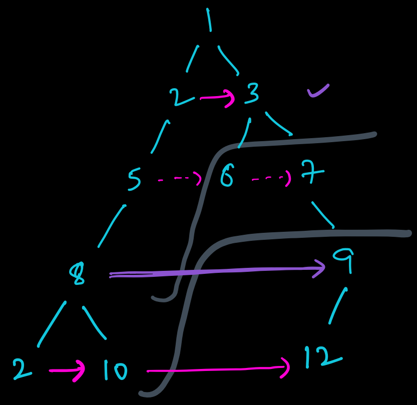
(i) If both left & right are present => node.left.next = right

(ii) If left is not null => node.left.next = first Node In Next level (node.next)

(iii) If right is not null => node.right.next = first Node In Next level (node.next)

Code:

```
Void ConnectNodesInSameLevel(Node root)
    while (root != null)
        Node node = root.
        while(node != null)
            if(node.left != null)
                if (node.right != null)
                    node.left.next = node.right
                else
                    node.left.next = first Node In Next level (node.next)

            if (node.right != null)
                node.right.next = first Node In Next level (node.next)

            node = node.next
        root = first Node In Next level (root)
```



TC: O(n).

SC: O(width of tree).