Top View

Top View of Binary Tree

Difficulty: **Medium** Accuracy: **38.43%** Submissions: **310K+** Points: **4**

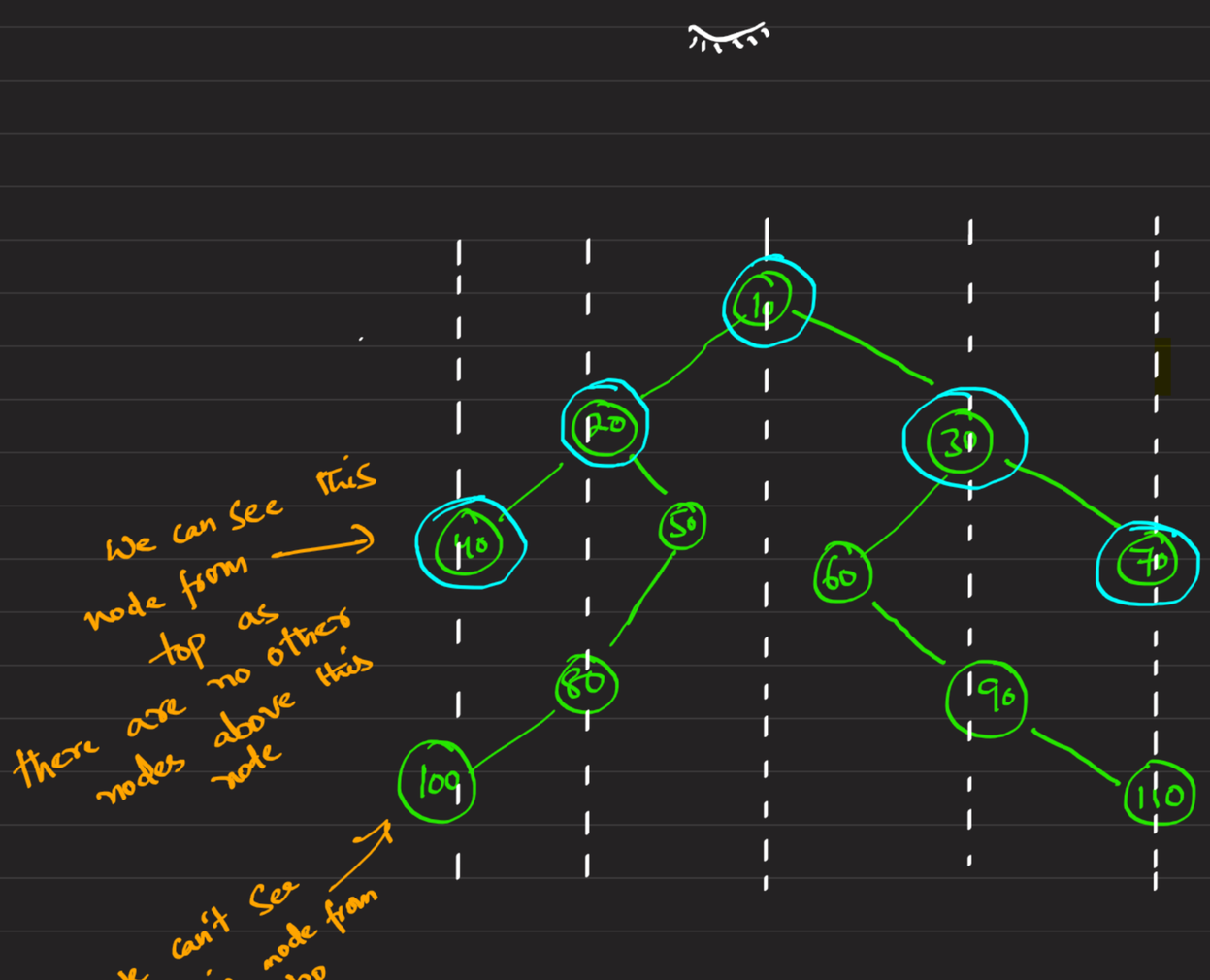
Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. For the given below tree

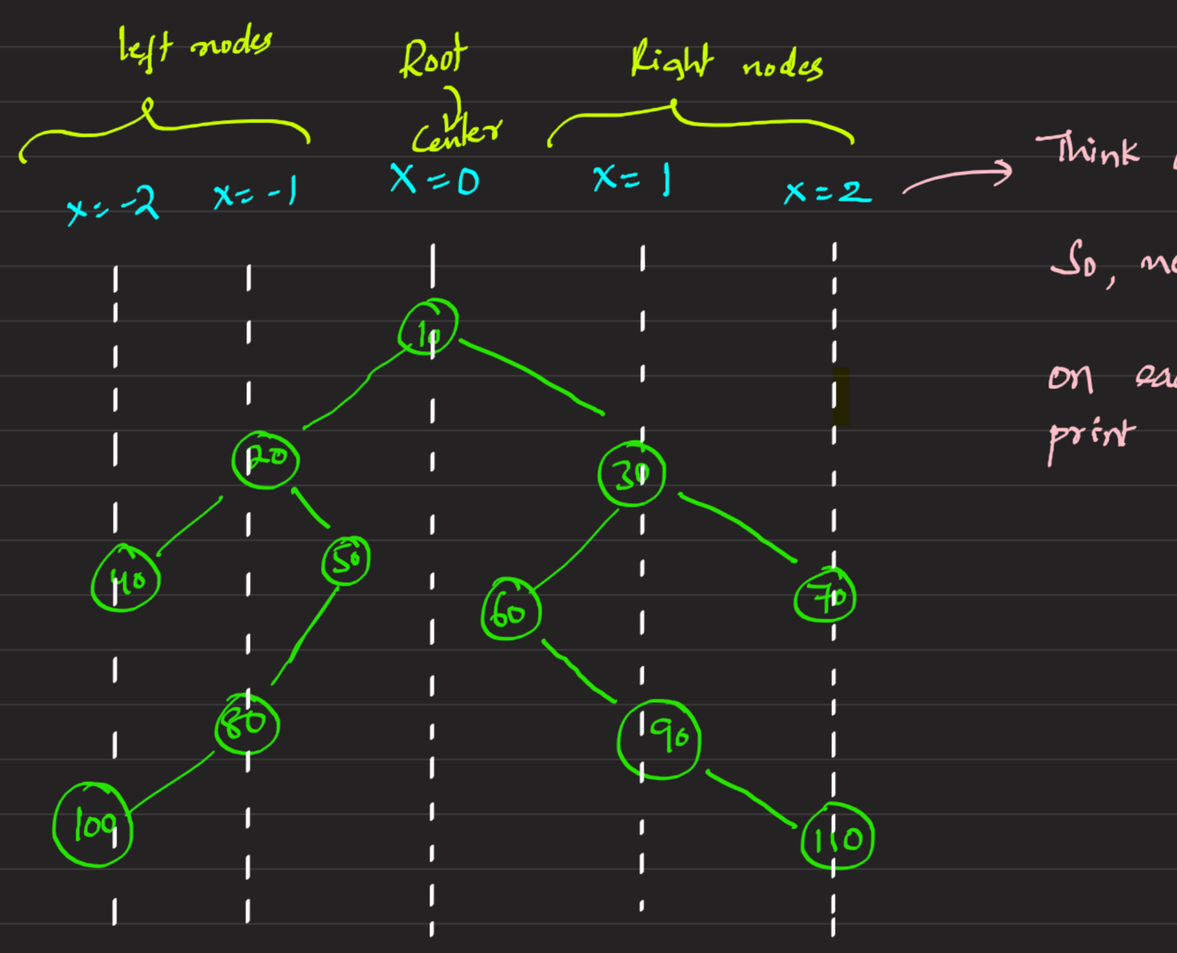
```
    1
  /   \
 2     3
/ \   / \
4  5 6  7
```

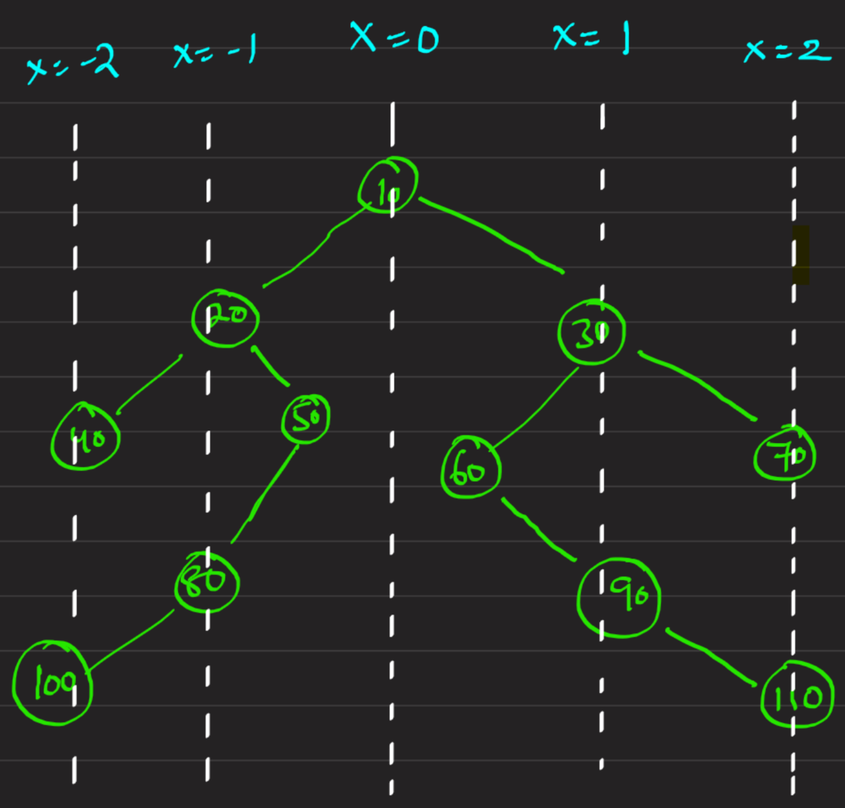
Top view will be: 4 2 1 3 7

Note: Return nodes from **leftmost** node to **rightmost** node. Also if 2 nodes are outside the shadow of the tree and are at same position then consider the left ones only(i.e. leftmost). For ex - **1 2 3 N 4 5 N 6 N 7 N 8 N 9 N N N N N** will give **8 2 1 3** as answer. Here 8 and 9 are on the same position but 9 will get shadowed.

* We have to print nodes that are visible from Top







let us Create a hashmap, and Store the node & x-axis

for example : (2, 30)

States that, on x=2, the node visible from top is 30

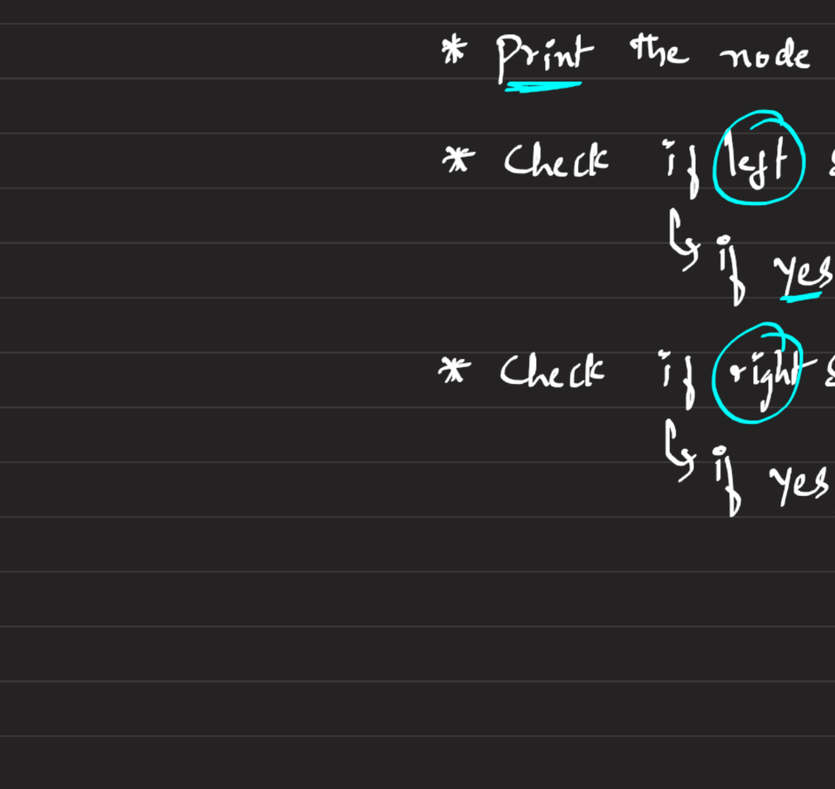
Hash Map

Initially, no nodes are added to HashMap

Steps to follow :

* Simple level Order, but Keep track of Vertical indexes,
  1. add root to the queue
  2. Iterate till queue is not Empty
     - 2.1 while iterating
       * Remove node from queue
       * Print the node Value
       * Check if left Subtree is there for root
         - if yes, add that to queue
       * Check if right Subtree is there for root
         - if yes, add that to queue

While adding, add node and its vertical index, if that Vertical index is not present in hashmap



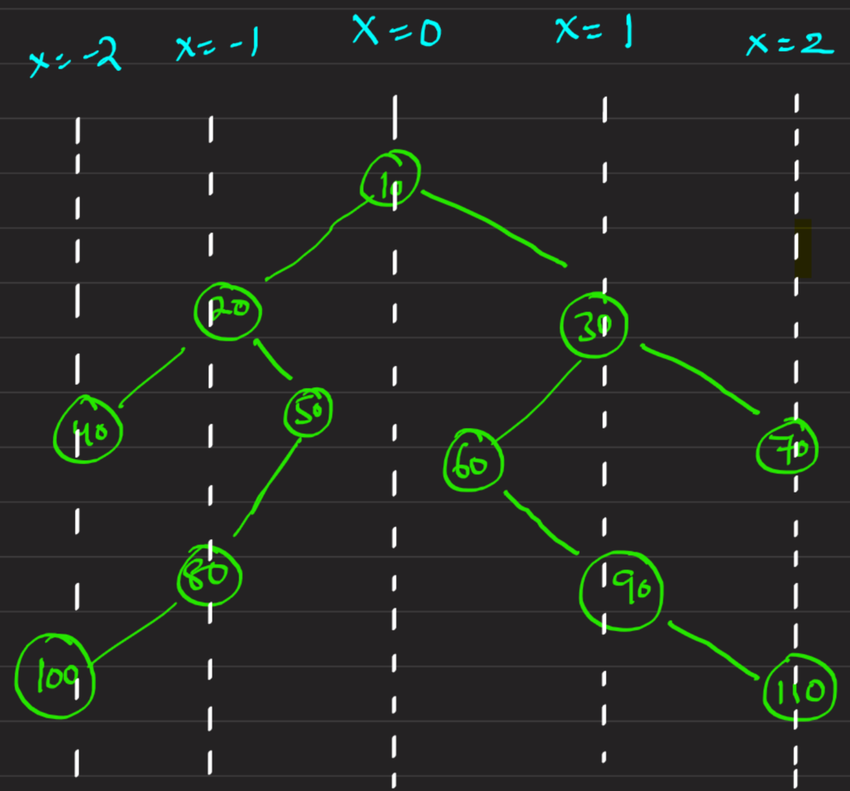
queue : Storing node with Vertical index

(10, 0)

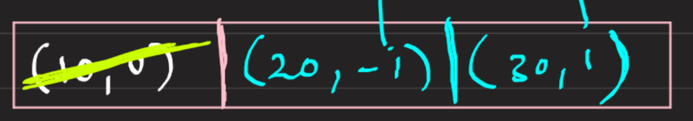
HashMap :

(0, 10)

after adding to queue, check whether 0 vertical index is there, if not add it to hashmap



queue :



↳ for left child, Subtract -1 from parent Vertex

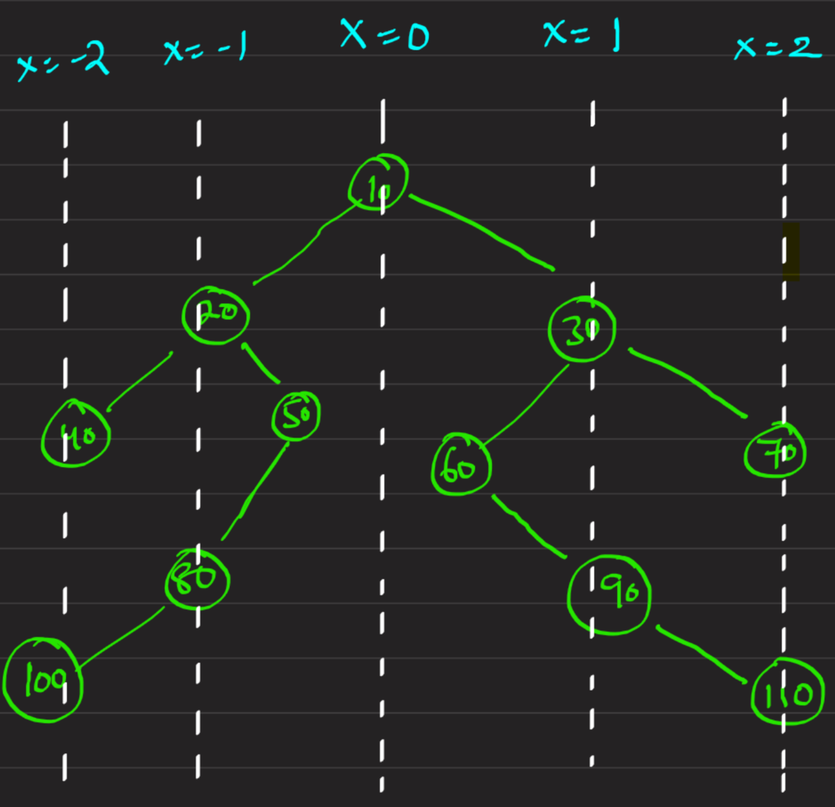
↳ for right child Add +1 from parent Vertex

→ there parent vertex is 0. So left = -1 right = 1

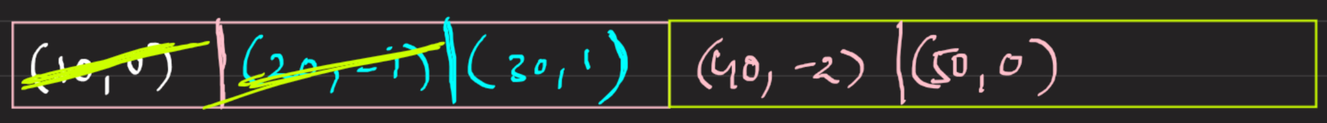
HashMap :

this 10 is val of node no need to store node, we are looking to just print

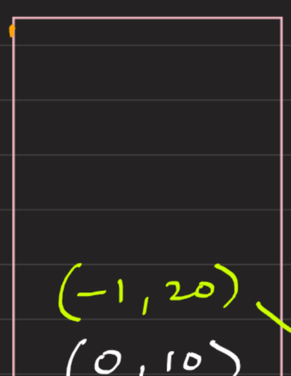
* Now add 10's children with their vertices



queue :



HashMap :

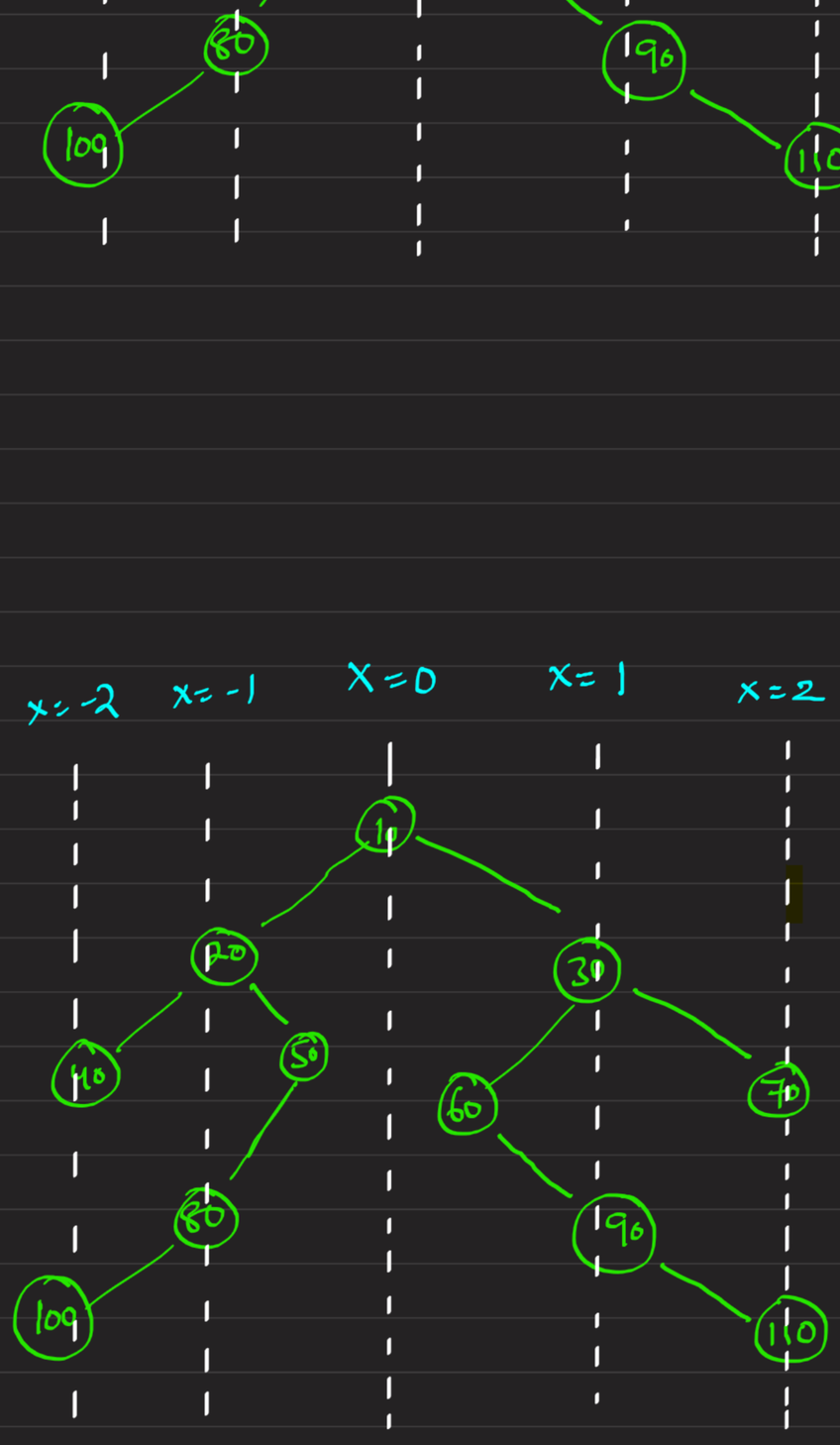


-1 is not there

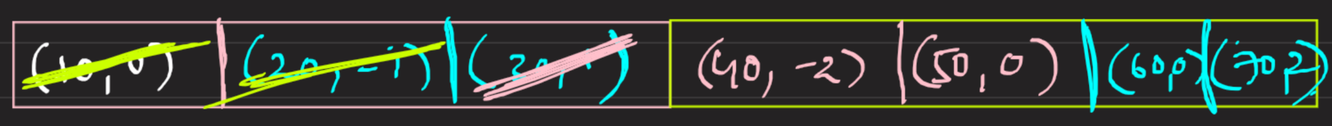
Add its child
⇓
AS parent Vertex = -1

left: Vertex = -1-1 = -2

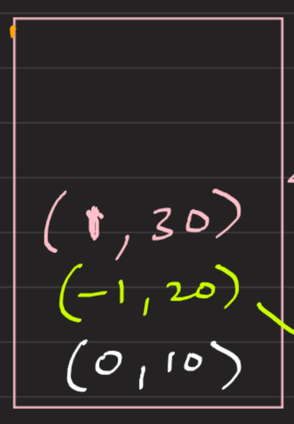
right: Vertex = -1+1 = 0



queue :



HashMap :

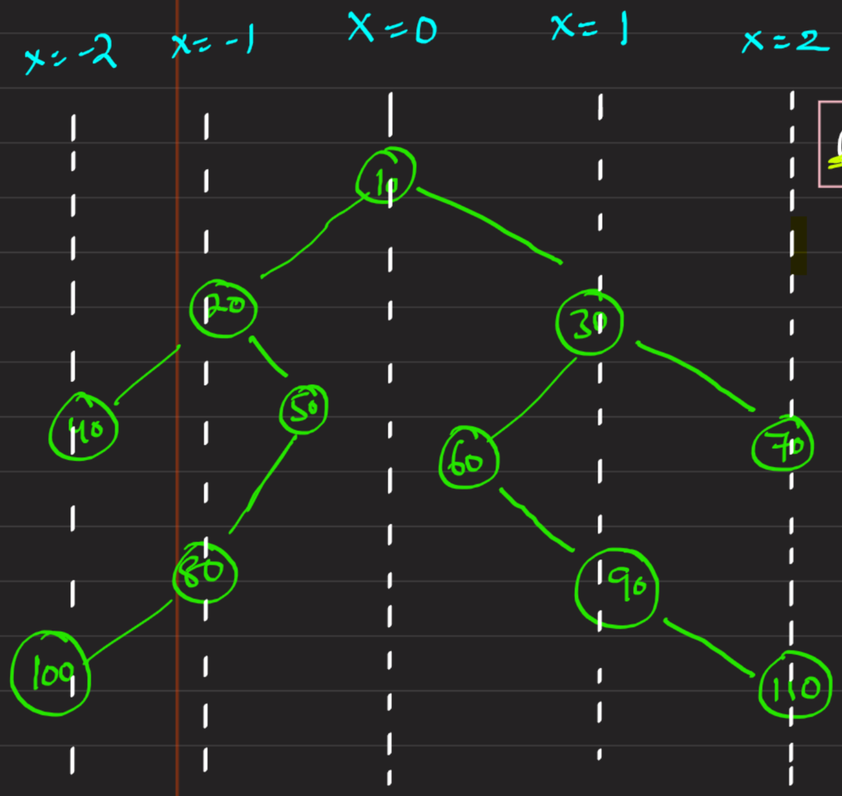


1 is not there

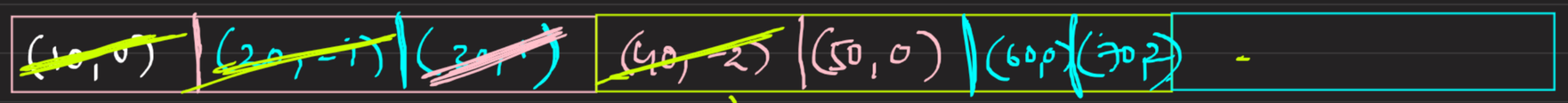
Add its child
⇓
AS parent Vertex = 1

left: Vertex = 1-1 = 0

right: Vertex = 1+1 = 2



queue :



HashMap :

(-2, 40)
(1, 30)
(-1, 20)
(0, 10)

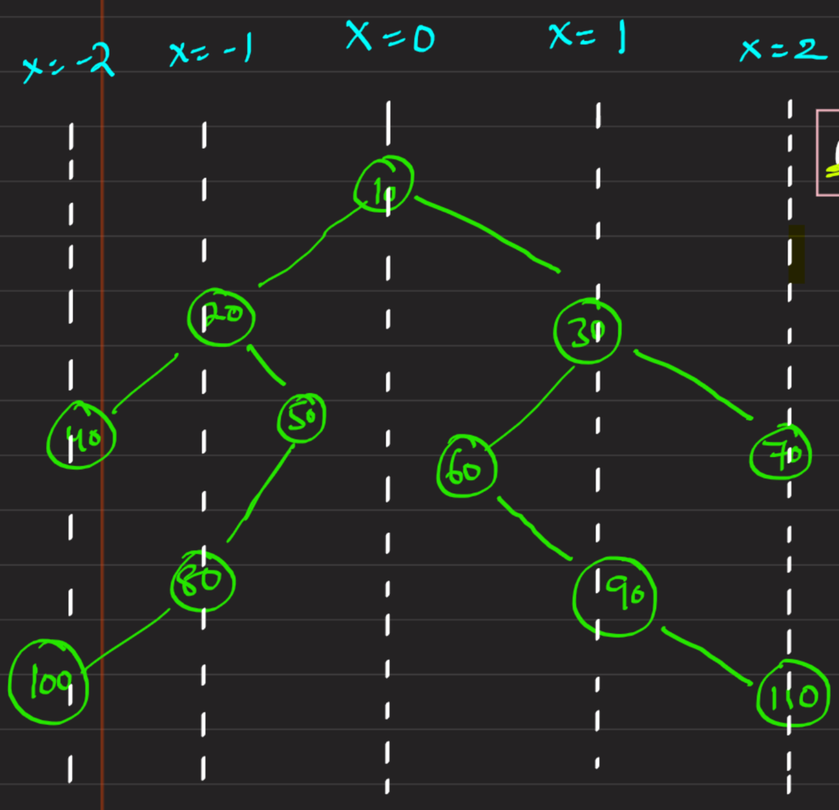
-2 is not there

Add its child
⇓
As parent Vertex = -2

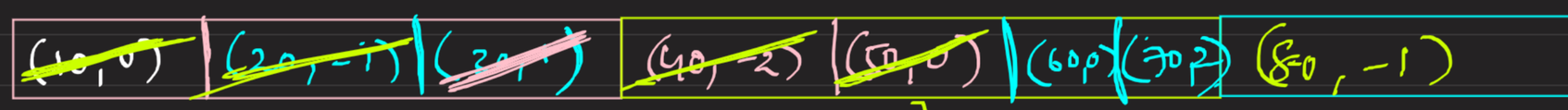
left → Vertex = -2-1 = -3

right → Vertex = -2+1 = -1

(no children for 40 → skip)



queue :



HashMap :

(-2, 40)
(1, 30)
(-1, 20)
(0, 10)

0 is there

So don't add to hashmap

with updated Vertices

* points to remember :

this is for pointing

since, we have some Vertex levels, like in our example:

$\{-2, -1, 0, 1, 2\}$

-2 is the minimum & 2 is the maximum

So, if you have (-2, 2) stored in some variables (min, max)

* then it is easy to iterate from -2 to 2 & get the nodes respective to that Vertex level which lies b/w {-2, 2}

* for that keep track of min & max of Vertex while pushing to hashMap itself.

Pseudo Code

```
class Pair{
    Node n;
    int vl;          // vertex level
    Pair(Node n, vl)
    { this.n = n;
      this.vl = vl;
    }
}
```

```
Void topView(Node root)
{
    Queue<Pair> q;
    HashMap<Integer, Node> map;

    q.add(new Pair(root, 0));          // store min, max = 0

    while (q.size() > 0)
    {
        Pair rem = q.remove();
        Node remNode = rem.n;          // node
        int remVl = rem.vl;            // vertex level

        if(map.containsKey(remVl) == false)   // if not there in map add it to map
        {
            map.put(remVl, remNode.val);
        }

        if(rem.left != null)
        {
            q.add(new Pair(remNode.left, remVl - 1));   // update min, max
        }

        if(rem.right != null)
        {
            q.add(new Pair(remNode.right, remVl + 1));  // update min, max
        }
    }
}
```

* check code, for detailed implementation