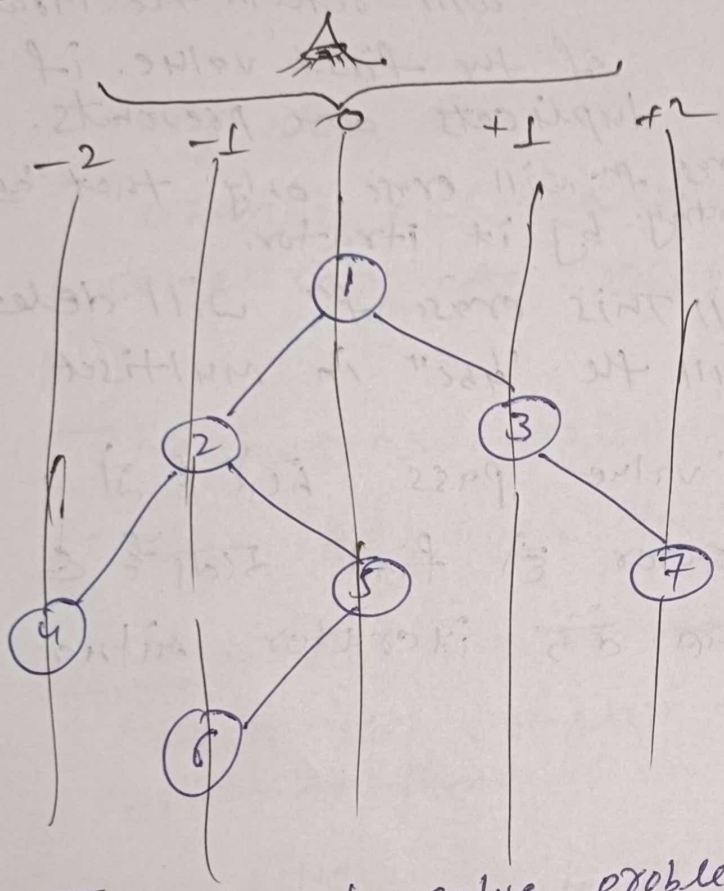
Top View of Binary Tree



Top View → 4 2 1 3 7

T.C. = O(N)
S.C. = O(N)

In order to solve problem on BT we have to traverse the tree for which we have traversal

- BFS (level order)
- DFS (inorder, preorder, postorder)

we use level order traversal here.

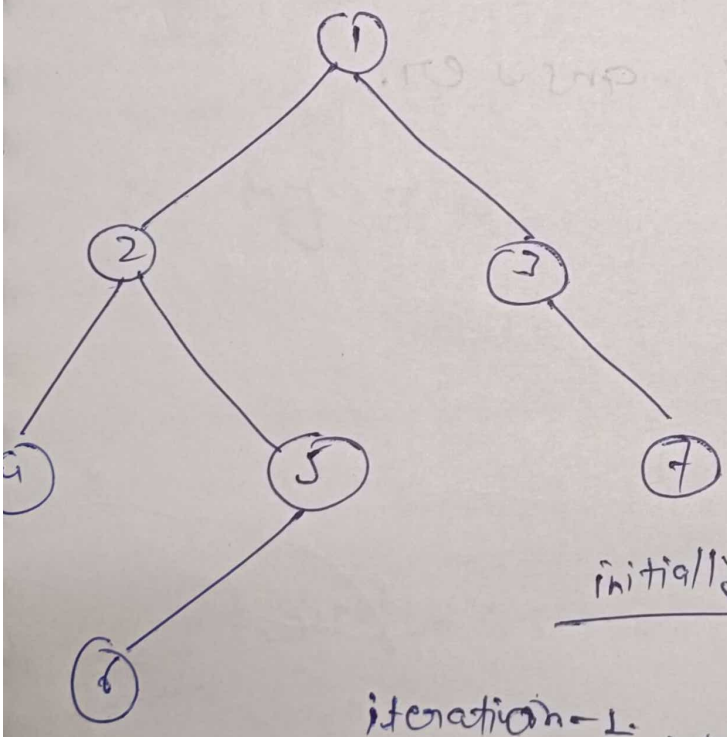
Why not recursive (DFS) traversal this will be discuss at end.

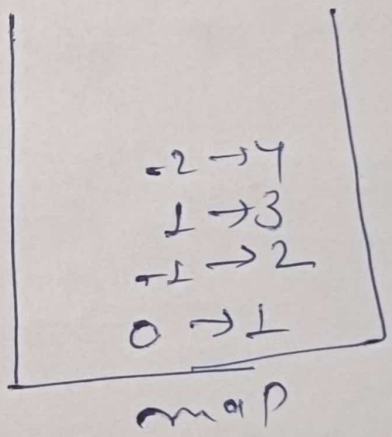
Intuition :- In vertical order traversal we use line concept where we assign every node to a coordinate here we are using similar kind of concept where which we use only line concept like That the first node in every line will be my top view.

Data structure

(i) queue <Node*, line>

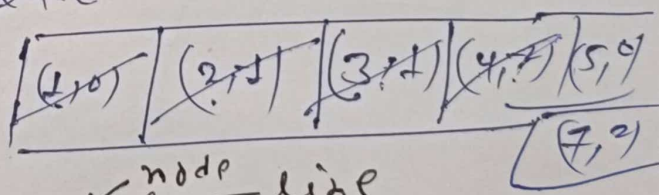
(ii) map < line, Node→val>





initially

queue



node line

iteration-1. node = (1,0)

if(map.find(0) == mp.end())

map.insert(0, 1)

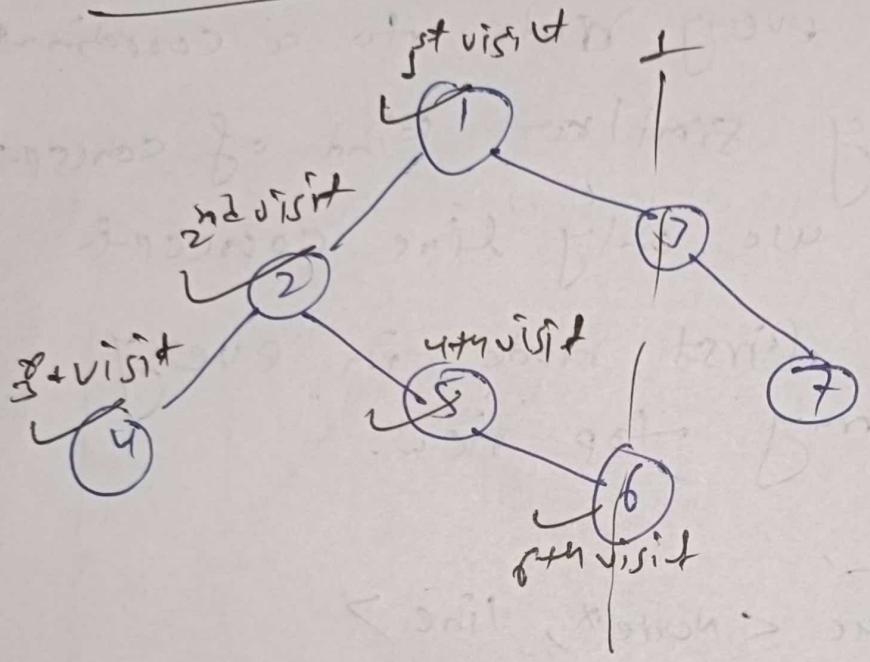
insert left & right child in que

2nd → node = (2,-1)

map.find(-1) == mp.end() →

3rd → node = (5,0) but map.find(0) != map.end()

why we not use recursive traversal.

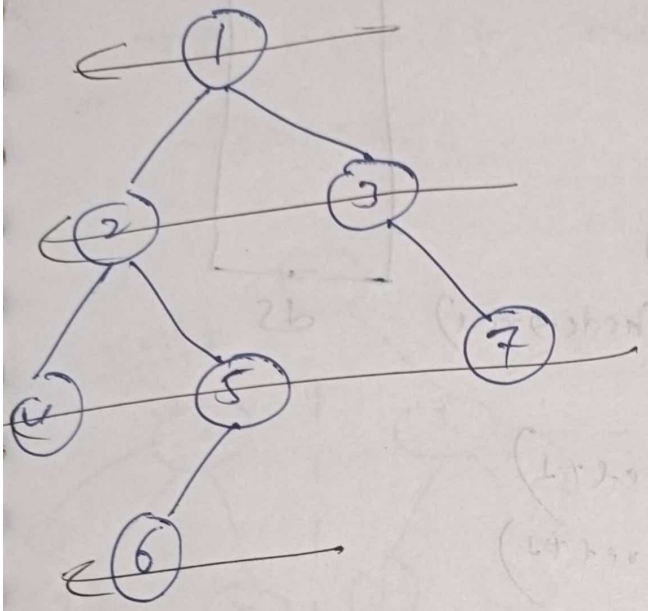


suppose you inorder traversal.

for like 1 we visit 6 before 3 that's this will give wrong answer.

Right / left View of Binary Tree

Right view = 1 3 7 6



The last node of every level is indeed in my right side view

If I reverse the logic then the first node of every level is ~~may~~ indeed in my right side view.

what kind of traversal we can use here

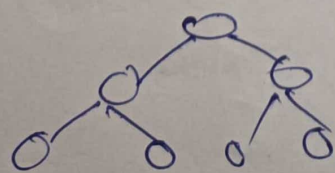
Recursive | Iterative (level order)

T.C. = O(N)
S.C. = O(H) ← worst case
↳ height

O(N)

why we are not going with level order traversal.

bc2. In the worst case T.C. = O(N)
S.C. = O(N)

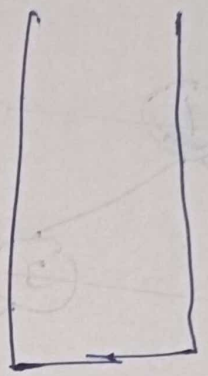


→ last level is filled so queue contain at max n/2 nodes.

f(node, level)

2

if (node == NULL)
    return;



if (level == ds.size)
    ds.push_back(node->val)

f(node->~~left~~ right, level+1)
f(node->left, level+1)

4

Reverse Preorder → Root left right

↳ Root right left

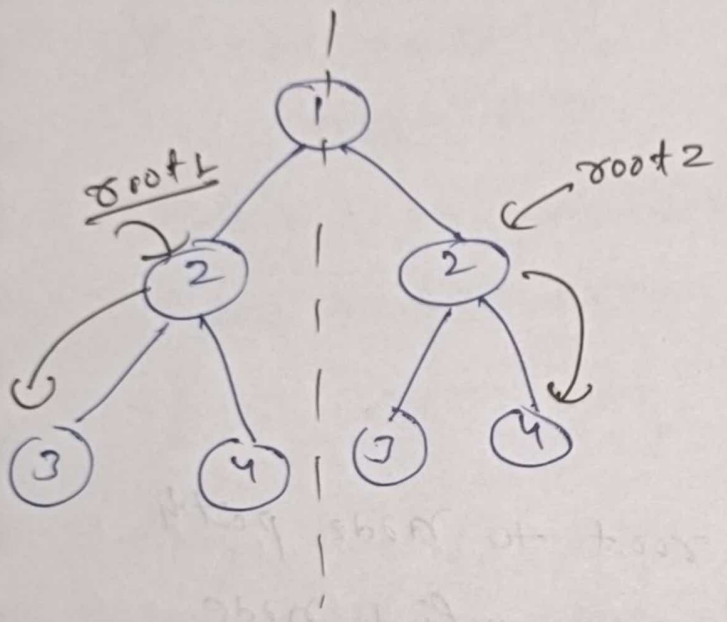
left side view

~~change the~~ apply only pree order

f(node->left, ——)
f(node->right, ——)

Symmetrical Binary Tree

If tree is mirror of itself then symmetric,




```
if (root1 != root2)
    return false

int x = f(root1->left, root2->right)
int y = f(root1->right, root2->left)

return (x & y);
```