Tree

Lec-1

1. Tree construction
   - → display()
   - → size()
   - → height()
   - → find().
2. Node to root path.
3. 236 // lowest common ancestor.
4. Root to node path.
5. Burn Binary tree starting from target node.
6. K down.
7. K distance far Node.

Lec-2

1. Diameter of Binary tree.
2. 112 // Path sum.
3. 113 // Path Sum-II
4. Max path Sum b/w two leaves of b. tree (gfg).
5. 124 // B. Tree Max path Sum. (Using 4th qn).

Lec-3

1. BFS traversal
   - → M-1
   - → M-2
   - → M-3.
2. Left view.
3. Right view.
4.

④ Vertical traversal
- → top view.
- → Bottom view.
- → Vertical sum.
- → Vertical sum using ll.

⑤ 987 // Vertical order traversal of Binary tree

lect-4

① Diagonal view.

② Boundary traversal.
- → geeks.
- → pep.

BST.

① Construction.
- → Max()
- → Min()
- → find().

② 98 // valid BST.

③ 99 // Recover BST.

④ 255 // Verify preorder traversal of BST.

⑤

lect-5

① add data.

② Remove data.

③ all solution pair.

④ 285 // Inorder Successor in BST.

⑤ 510 // Inorder Successor in BST-II.

⑥ All solution pair.

⑦ BST iterator // 173

LCA in BST // 235

Lect-6

1. AVL → construction
2. Balance a Binary search tree

Lect-7

Construction of B. Tree Based on

1. Const. b.t from pre and ino
2. " " post and in
5. " " pre and post
4. 968 // B. tree canvas
5. Const B.ST From pre-order
6. Const " post order
7. Level Order ~~traversal~~

Lect-8

1. B.T to dll
2. B.T to C. dll
8. Morris traversal inorder { traverse on t: O(n) & S: O(1)
4. Pre order morris traversal { pre t: O(n) S: O(logn)
5. Traversal using iterator pre, ~~in and~~ post and in
6. Zig-Zag traversal (H.W)
7. kth smallest element in BST
8. 653 // Two Sum-IV i/p in BST.
9. Check if pre, ino and post o of same tree
10. Find tree diameter using pre and in without making tree

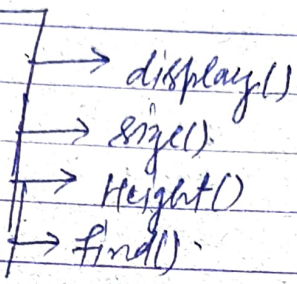
(11) Print all k sum path in b.t.

Not Imp

Lect-9.

~~generic tree~~ / asked in graph not in tree

(1) Construct



(2) Root to node path.

(3) Find Leaf

(4) Min no. of switches.

(5) Check tree foldable / not

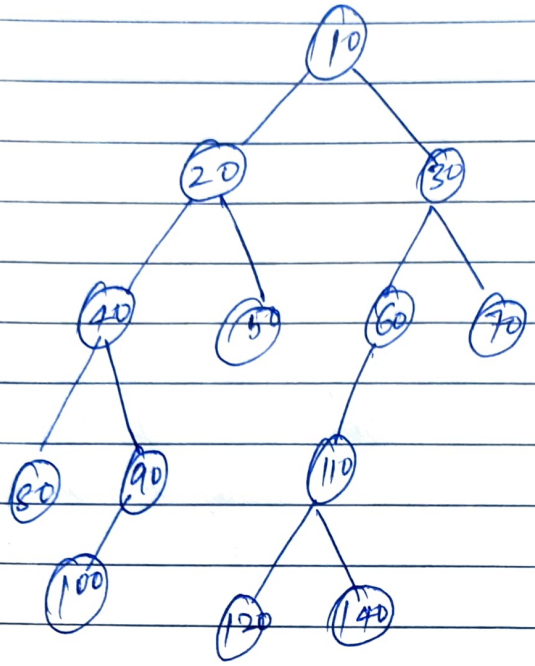
(6) Check mirror image.

(7) Linearize the tree.

	total	done	Rem	H.∞.
L-1	7		1 | 1	3 x
L-2	5		2 ✓	2 x
L-3	5		1 1	x x
L-4	6		x 2	1
L-5	8		2 x	1
L-6	2		1 1	x x
L-7	7		✓ 2	x 2
le-8	11		1 1	3
leet-9	7			
	58			

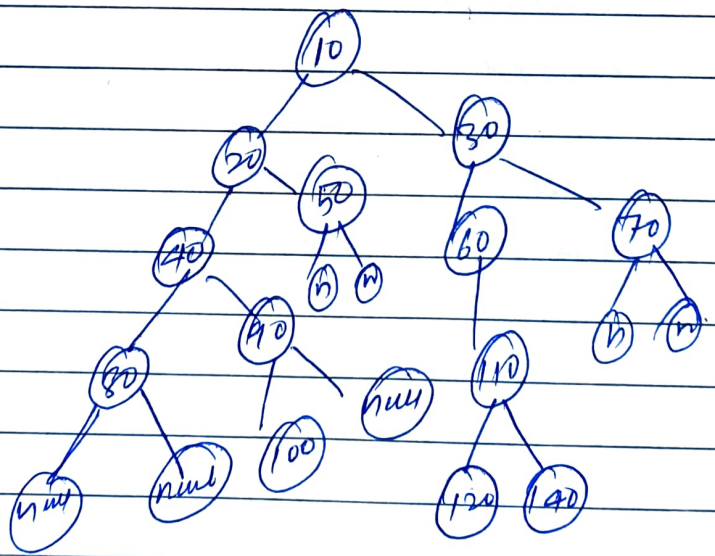
APCO
Date :
Page :

[Tree construction]



To make tree we use post-order construction means node बनाते के साथ साथ time connect nodes.

[10, 20, 40, 80, -1, -1, 90, 100, -1, -1, -1, 50, -1, -1, 30
60, 110, 120, -1, -1, 140, -1, -1, -1, 70, -1, -1]



APCO
Date :
Page :

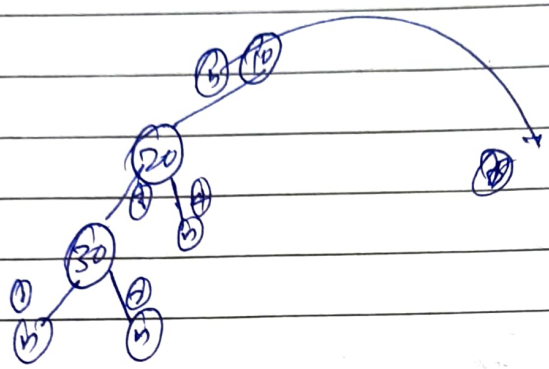
Basic fn

① display, size(), height(), find(),

Note :-

Tree construction takes place in post order

Eg First node creates then linking b/w nodes done.



no. of linking of nodes to another node.

// Codes

This code follows the serializing technique as it is created from array.

```
Public static class Node {
        int data = 0;
        Node left = null;
        Node right = null;

class   Node (int data) {
            this.data = data;
        }
}
```

APCO
Date :
Page :

Tree Construction in using Node class

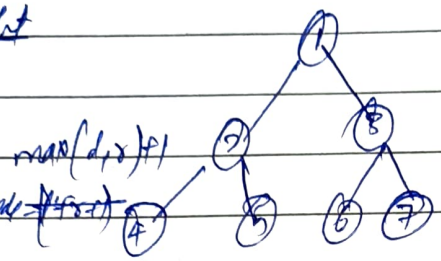
```
static int idx = 0;
(1) Public static Node constructTree(int[] arr) {
        if(idx >= arr.length || arr[idx] == -1) {
            idx++;
            return null;
        }

        Node node = new Node(arr[idx++]);
        node.left = constructTree(arr);
        node.right = constructTree(arr);

        return node; // because contain add of whole tree.
    }
// Display function.
(2) Public static void display(Node node) {
        if(node == null) return;

        StringBuilder sb = new StringBuilder();
        sb.append((node.left == null ? "." : node.left.data + " "));
        sb.append("<- " + node.data + "->");
        sb.append((node.right == null ? "." : node.right.data + " "));
        System.out.println(sb);
        display(node.left);
        display(node.right);

    }
```

APCO
Date :
Page :

③ Public static int size(Node node){

return node == null ? 0 : size(node.left) + size(node.right) + 1);

④ // height



In terms of ~~edges~~ nodes

if in term of edges
= max(l,r) - 1

Public static int height(Node node){
return node == null ? -1 : Math.max(height(node.left), height(node.right)) + 1 ;

⑤ Public static boolean find(Node node, int data){
if(node == null) return false;
if(node.data == data) return true;

return find(node.left, data) || find(node.right, data);

APCO
Date :
Page :

Question.

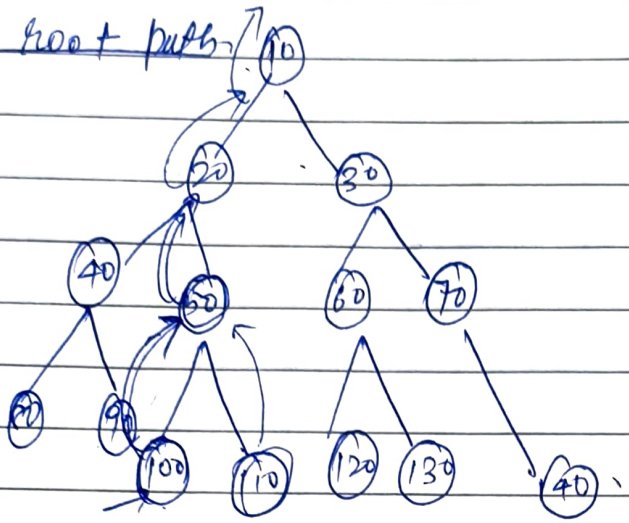
1. Node to root path.
2. 236 || lowest common ancestor.
3. Root to node path.
4. Burn the binary tree starting from the target node.
5. K down.
6. K ~~far~~ no dist. for node. / K nearest neighbour.

* Nearest neighbour VV Imp

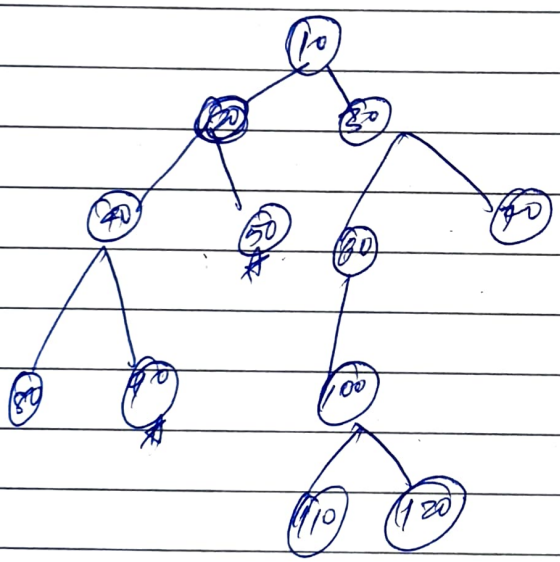
10 → 6.

APCO
Date :
Page :

① Node to root path.



AL in arguement is same as al static.



90	40	20	10	—

50	20	10

APCO
Date :
Page :

① Node To Root Path :

```
Public static boolean nodeToRootPath(Node node, int data,
    ArrayList<Node> ans) {
    if(node == null) return false;

    if(node.data == data){
        ans.add(node);
        return true;
    }
boolean res = nodeToRootPath(node.left, data, ans) ||
        nodeToRootPath(node.right, data, ans);
if(res) {
    ans.add(node);
}
return res;
}
```

②

```
Public static boolean rootToNodePath(Node node, int data,
    ArrayList<Node> ans) {
if(node == null) return false;
if(node.data == data){
    ans.add(node);
    return true;
}
ans.add(node);
boolean res = nodeToRootPath(node.left, data, ans) ||
        nodeToRootPath(node.right, data, ans);
if(!res) {
    ans.remove(ans.size()-1);
}
```

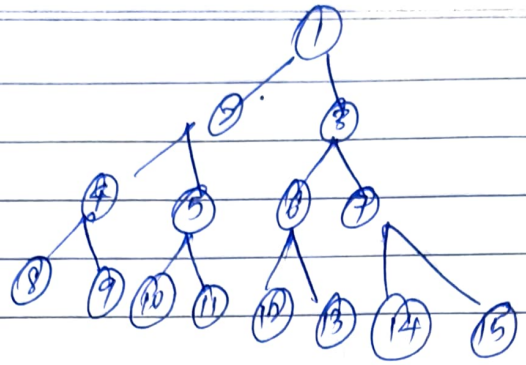
APCO
Date :
Page :

M-2. Node to Root Path:

```
Public static Arraylist<Node> nodeToRootPath(Node
    node, int data){.
    if(node == null) return null;.

    if(node.data == data){.
        Arraylist<Node> base = new Arraylist<>();.
        base.add(node);.
        return base;.
    }.

    Arraylist<Node> left = fn(node.left, data);.
    if(left != null){.
        left.add(node);.
        return left;.
    }.

    Arraylist Node<Node> right = fn(node.right, data);.
    if(right != null){.
        right.add(node);.
        return right;.
    }.

    return null;.
}.
```

APCO
Date :
Page :

(5) K down:



(4 down)

4 down:

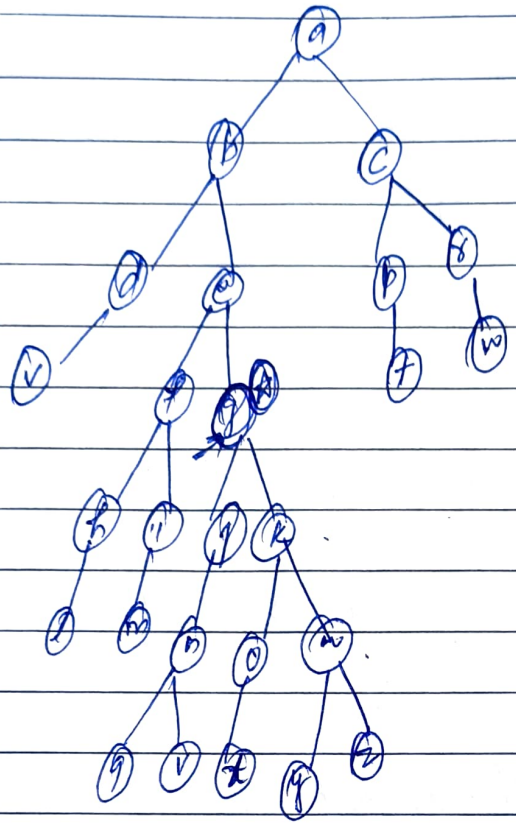
↳ (8, 9, 10, 11, 12, 13, 14, 15)

//Code

```
Public static void K-down(Node node, Node, block, int k,
    ArrayList<Integer> ans){
    if(node == null || node == block || k<0) return;

    if(k==0){
        ans.add(node.data);
        return;
    }
    Kdown(node.left, block, k-1, ans);
    Kdown(node.right, block, k-1, ans);

}
```

APCO
Date :
Page :

6. Qu. Print all k dist nodes.



Al of path

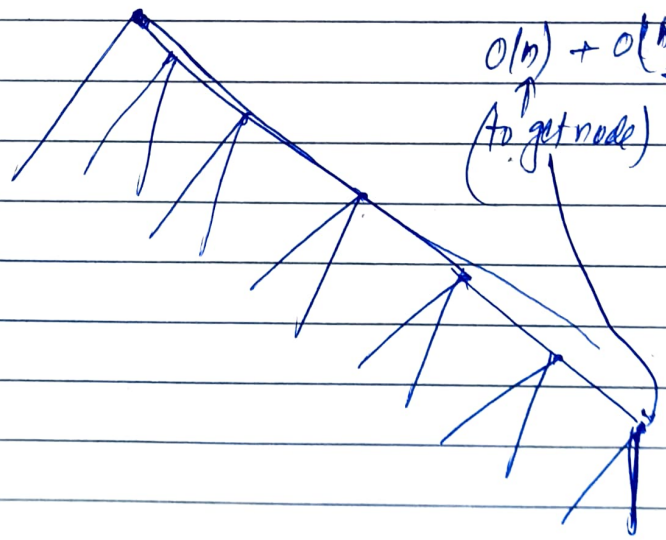
k	g	e	b	a

① Call k down from ~~with~~ each node with blocking of previous node so that they can't call again.

Al of node to root path.

g	e	b	a

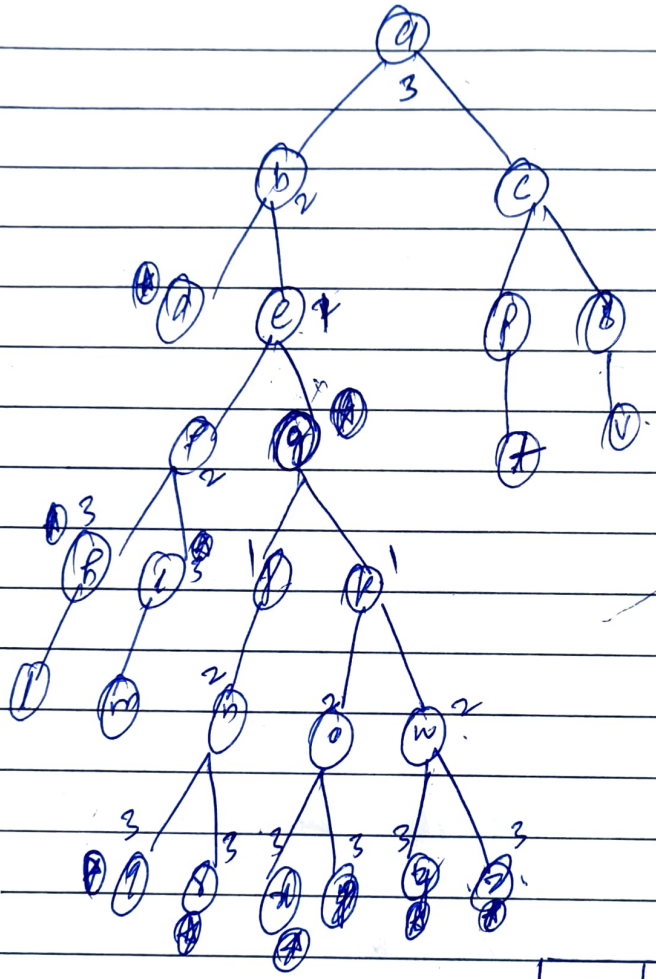
special case :-



APCO
Date :
Page :

T: $O(n)$.

S: $O(n)$. → S: $O(n)$. + Recursive space.



g	c	b	a
0	1	2	3

Tree | LL | DP basic
graph dfs, BFS.
Count island, Ro Hen oranges.

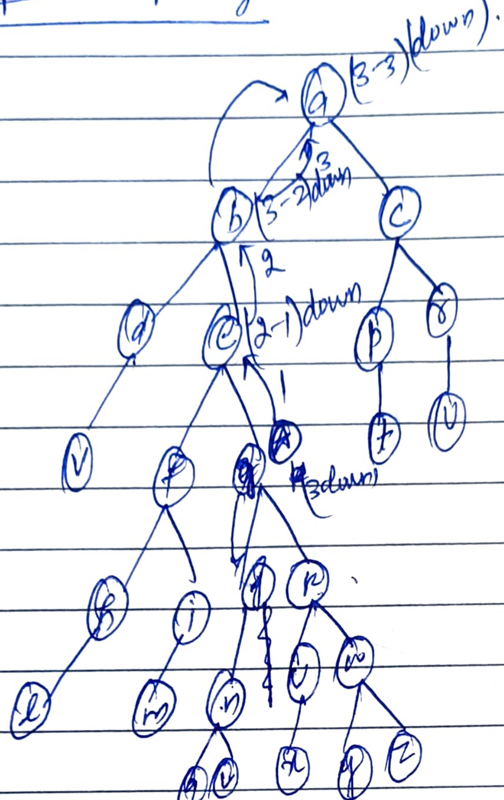
1. O/P based] 60
2. DS/OOPs
3. DBMS → standard
4. OS →

DBMS query based →

→ f2f
t, LL, dp
→ .

M-2

O(1) space optimized.



I/P k O/P का space not countable.

Algorithm :- ① after finding the node just call k down from there.

② then return (val+1) and call k down (k-val-1) with pass then blocking node of the previous node.

T: O(n + n) = O(2n) = O(n).
(first n: to find the node data; second n: to print k down)

S: O(1) as no extra space used.

APCO
Date :
Page :

M-1

```
Public static void kFar(Node node, int data, int k){
    ArrayList<Node> list = new ArrayList<>();
    nodeToRootPath(node, data, list);
    Node prev = null;

    for(int i=0; i<list.size; i++){
        kdown(list.get(i), prev, k-i, ans);
        prev = list.get(i);

    }

}
```

M-2

```
Public static int kFar2(Node node, int data, int k,
        ArrayList<Integer> ans){
    if(node == null) return -1;

    if(node.data == data){
        kdown(node, null, k, ans);
        return 1;
    }
    int ld = kFar2(node.left, data, k, ans);
    if(ld != -1){
        kdown(node, node.left, k-ld, ans);
        return ld+1;
    }
    int rd = kFar2(node.right, data, k, ans);
    if(rd != -1){
        kdown(node, node.right, k-ld, ans);
        return ld+1;
    }
    return -1;
}
```

APCO
Date :
Page :

4. Burn Binary Tree. Starting from the target node.

APCO
Date :
Page :

L-2

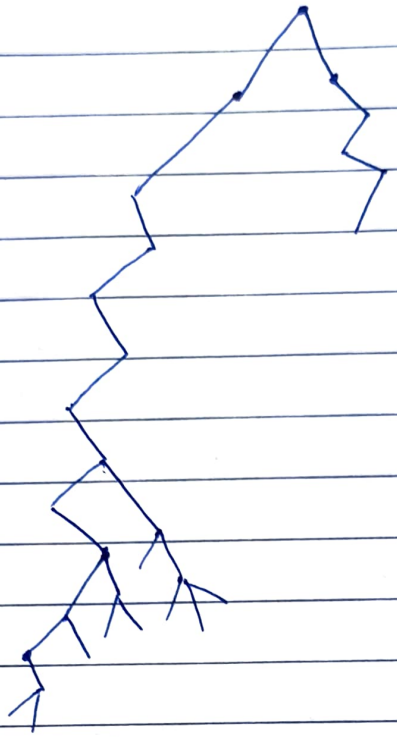
[Sept, 20, 2020]

Tree - lecture-2.

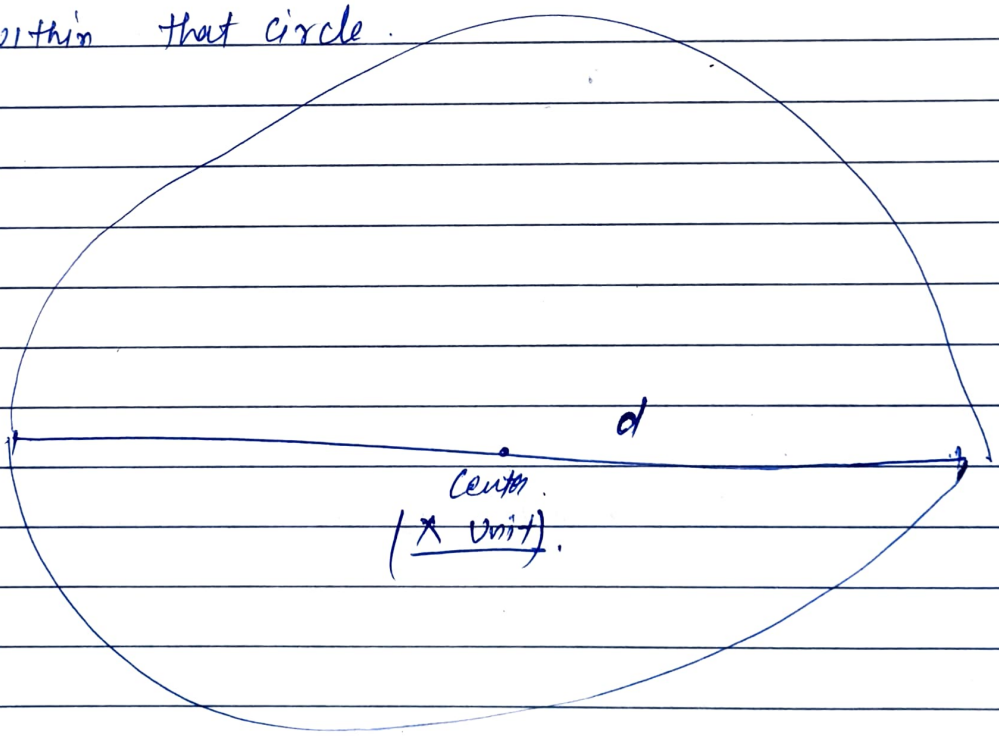
Agenda:

1. diameter of binary tree.
2. 543 - // diameter of BT.
3. 112
4. 113.
5. Max path sum b/w two leaves of b tree.
6. 124

Q4

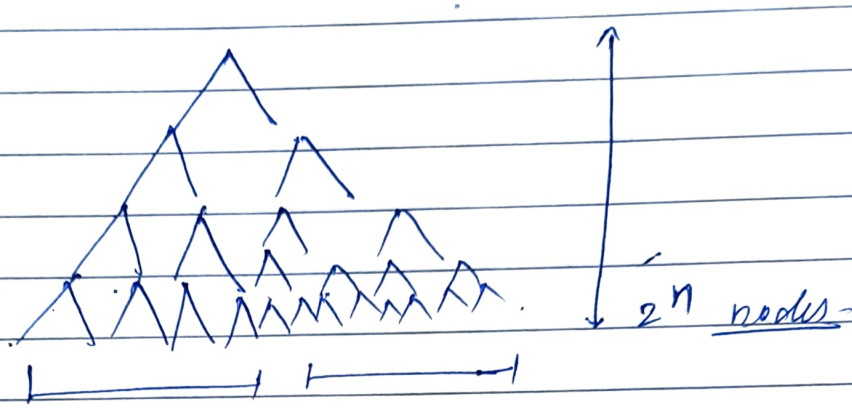


diameter :- If we stretch the farthest node then and draw circle then all the nodes contain within that circle.



uses eg If we said there are dog roaming around in boundry then we can say there is tree application.

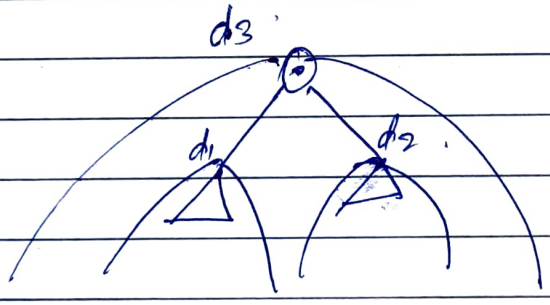
1-2 if we have CBT.



total diameter $= \frac{2^n}{2} = (2^{n-1})$.

total ways to combine dia $= ({}^{2^{n-1}}C_2)$.

☆ dameter need not to pass through root.



$d_3 = ?$

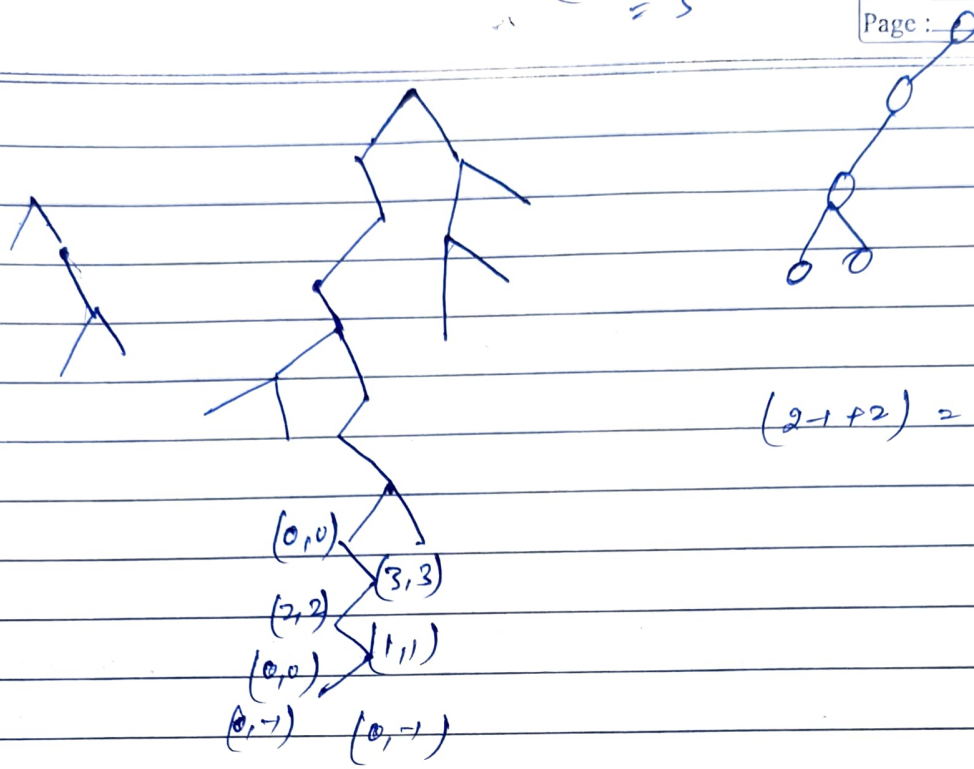
$= \max\left(d_1, d_2, (lh + rh + 2)\right)$;

T: $O(n^2)$

S:

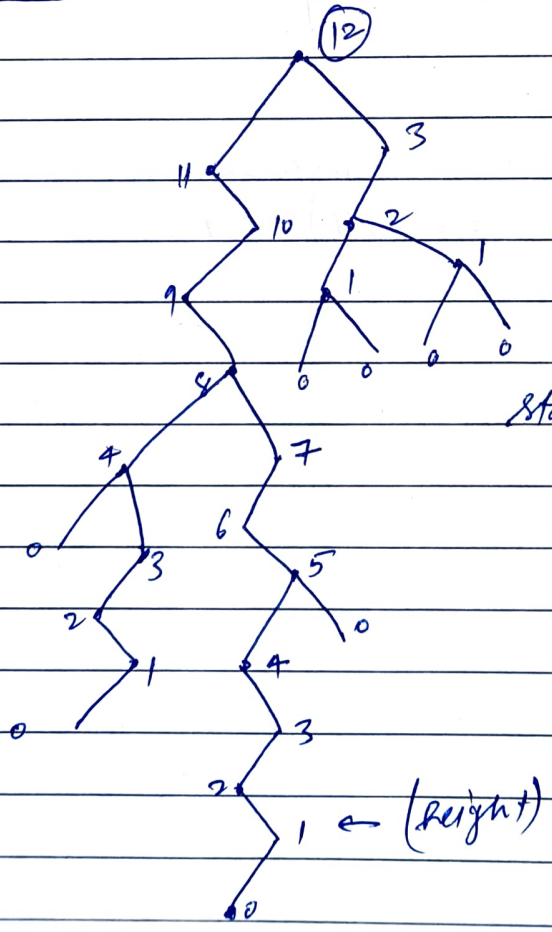
(2-1+2) = 3

APCO
Date :
Page :



(2-1+2) = (3)

M-3 dry Run.



← (Height)

static diam = 0 1 2 3 4 5 6 7 13

Since Recursion are taking space in stack so.

S: O(n).

APCO
Date :
Page :

M-1

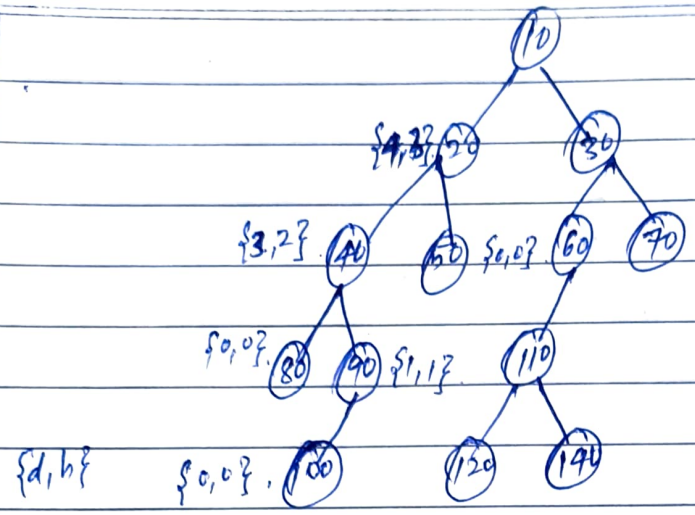
```
Public static int diameter_01(Node node){
    if(node == null) return 0;

    int ld = diameter_01(node.left);
    int rd = diameter_01(node.right);

    int lh = height(node.left);
    int rh = height(node.right);

    return Math.max(Math.max(ld,rd), lh+rh+2);
}
```

S: O(n)
T: O(n^2). (∵ every node visited twice for d, h).

M-2

```
Public static int[] diameter_02(Node node){
    if(node == null) return new int[]{0,-1}; // {d,h}

    int[] lres = diameter_02(node.left);
    int[] rres = diameter_02(node.right);

    int dia = Math.max(Math.max(lres[0],rres[0]), lres[1]+
                rres[1]+2);

    int hei = Math.max(lres[1],rres[1])+1;
    return new int[]{dia, hei};
}
```

eg



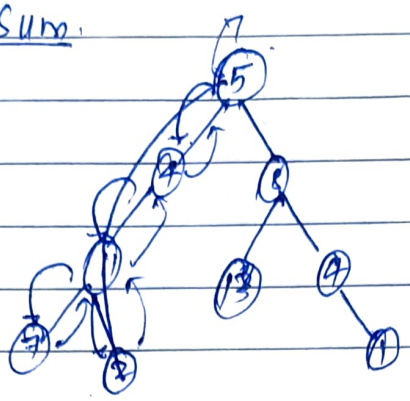
{d,h}

M-3 :

```
// static int diaAns = 0;          → here we can either use diaAns / ans[0]
public static int diameter_03(Node node, int[] diaAns){
    if(node == null) return -1;

    int lh = diameter_03(node.left, diaAns);
    int rh = diameter_03(node.right, diaAns);

    diaAns[0] = Math.max(diaAns[0], lh+rh+2);
    return Math.max(lh, rh)+1;

}
```

APCO
Date :
Page :

Qu 112 // Path Sum.



sum = 22

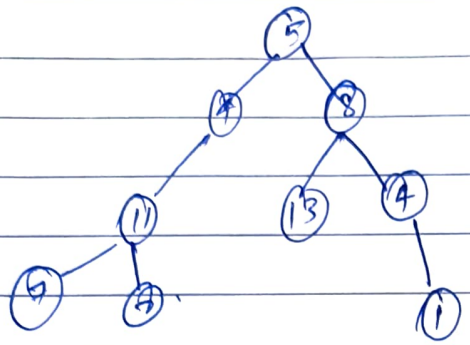
o/p = 5, 4, 11, 2

// code

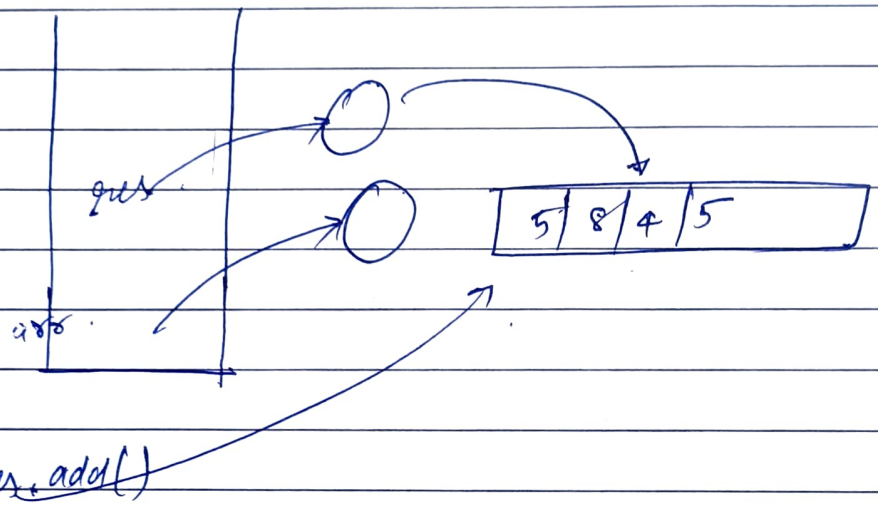
```
Public boolean hasPathSum( TreeNode root, int sum ){
    if( root == null ) return false;

    if( root.left == null && root.right == null &&
        sum - root.val == 0 ) return true;

    return hasPathSum( root.left, sum - root.val) ||
           hasPathSum( root.right, sum - root.val);
}
```

APCO
Date :
Page :

Qu 118 -



C-2



C-1 when make base -

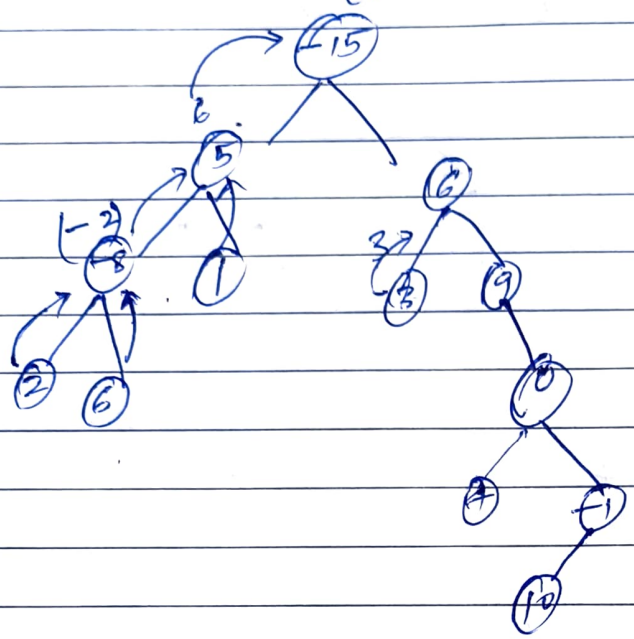
base = | 5 | 8 | 4 | 5 |

make copy of ~~ans~~ arr.

and add that to res.

APCO
Date :
Page :

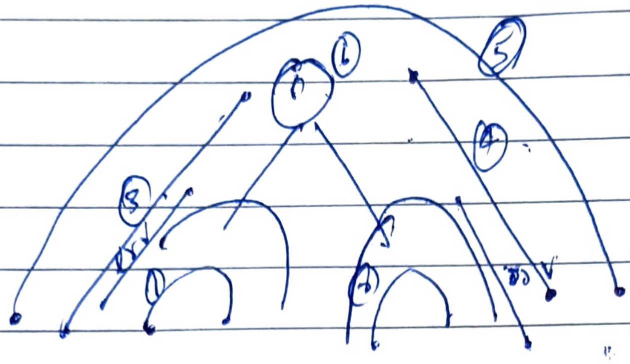
Q-1 Leaf to leaf max sum. (geeksforgeeks)



// code :

```
static int maxLL = -(int)1e8;
Public static int leafToLeaf(Node node){
    if(node == null) return (int)1e8;
    if(node.left == null && node.right == null)
        return node.data;

    int ntoLL = leafToLeaf(node.left);
    int ntoRL = leafToLeaf(node.right);

    if(node.left != null && node.right != null){
        maxLL = Math.max(maxLL, ntoLL +
                ntoRL + node.data);
    }

    return Math.max(ntoLL, ntoRL) + node.data;
}
```

124.



lrv + a = 3 ·

lrv + a + rrv = (5)

rrv + a = 4

max(max(lrv, rrv) + n-data , n-d);

107

- LV
- RV
- VO.
- BV
- TV.

APCO
Date :
Page :

Lec-4

(23-09-2020)

(Tree).

7488515461

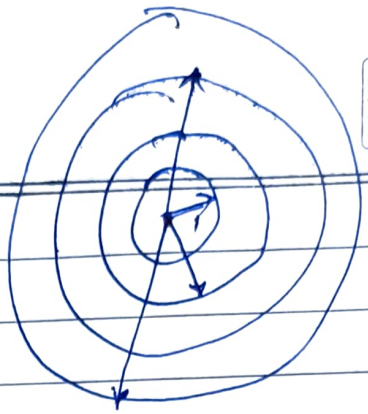
APCO
Date :
Page :

(Tree lecture - 8).
(21/Sept/2020)

Agenda :-

BFS - traversal

1. M-1, M-2, M-3.
2. ~~Side~~ Left view,
3. Right view.
4. Vertical traversal.
5. Top view.
6. Bottom view.
7. Top sum - ^
8. Bottom sum ^
9. Vertical sum.
10. gfg. using 1. (Vertical sum).
11. leetcode of top view. (~~1192~~). (987).
12. ~~Bottom~~ view -
13. ~~Top view~~.



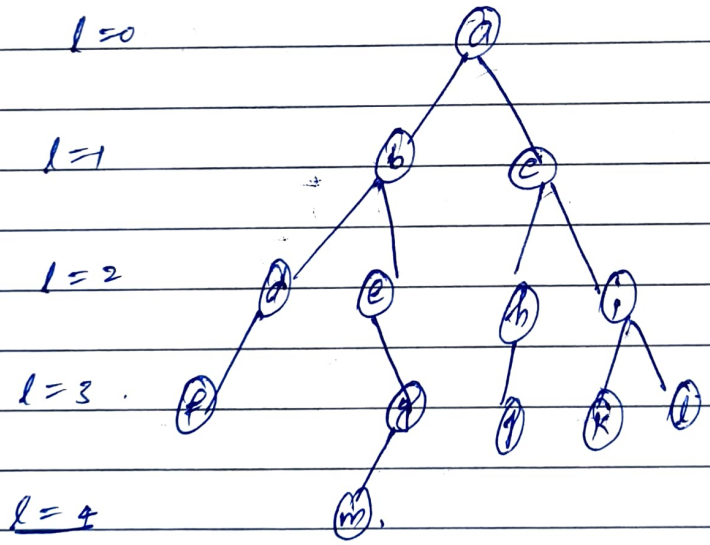
APCO
Date :
Page :

Algo of bfs:-

r = 1
r = 2
r = 3
.
.
.
r = n

This kind of movement is called bfs traversal.

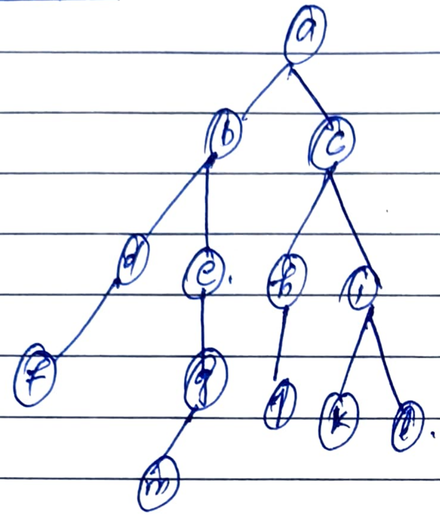
1. maintain queue.
2. insert root → queue.
3. while (q.size != 0) {
   3.1) remove Front().
   3.2) insert all child of remove element in q.



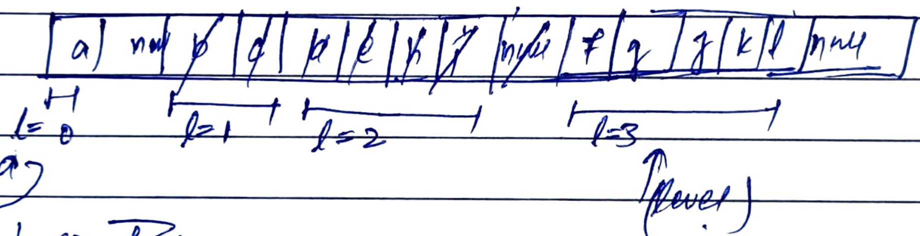
q	b	c

* Levelwise print is very important

② ~~Levelwise~~ Null BFS-



delimiter = null



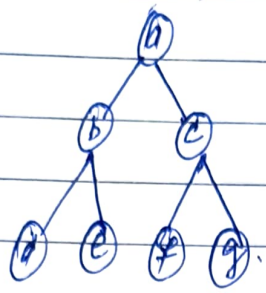
0 a
1 ↳ b c
2 ↳ d e h i
3 ↳ f g j k l.

T: O(n), space = O(n).

③ Very Imp while के अंदर while लगाना के

APCO
Date :
Page :

For view type question :-



b	c	d	e	f

0 → a .

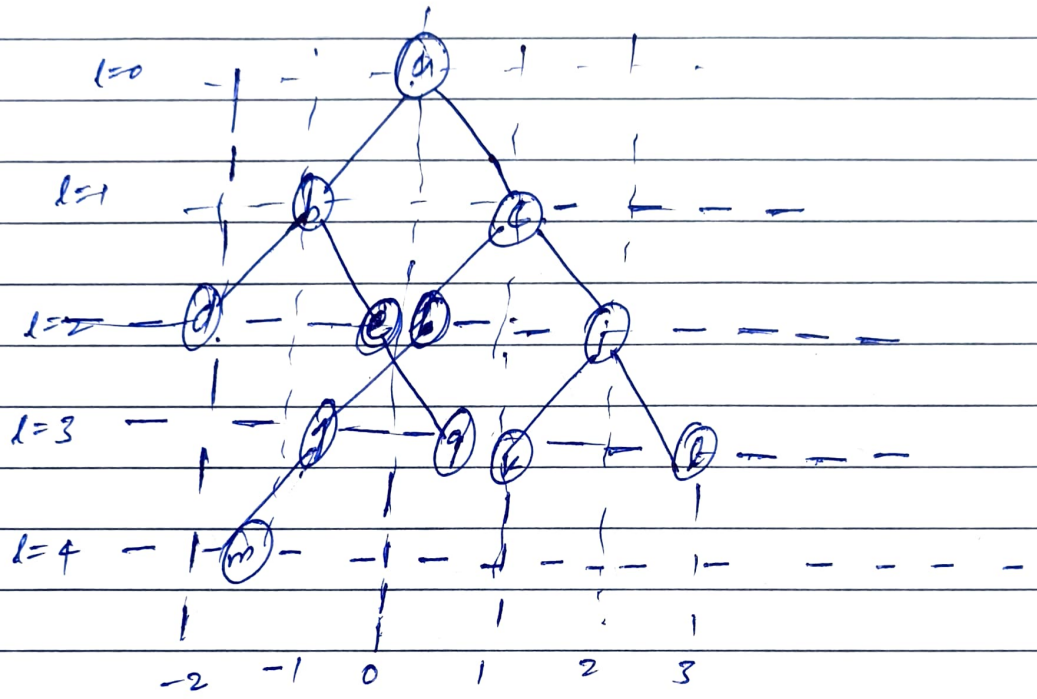
1 → b c

2 → d, e f

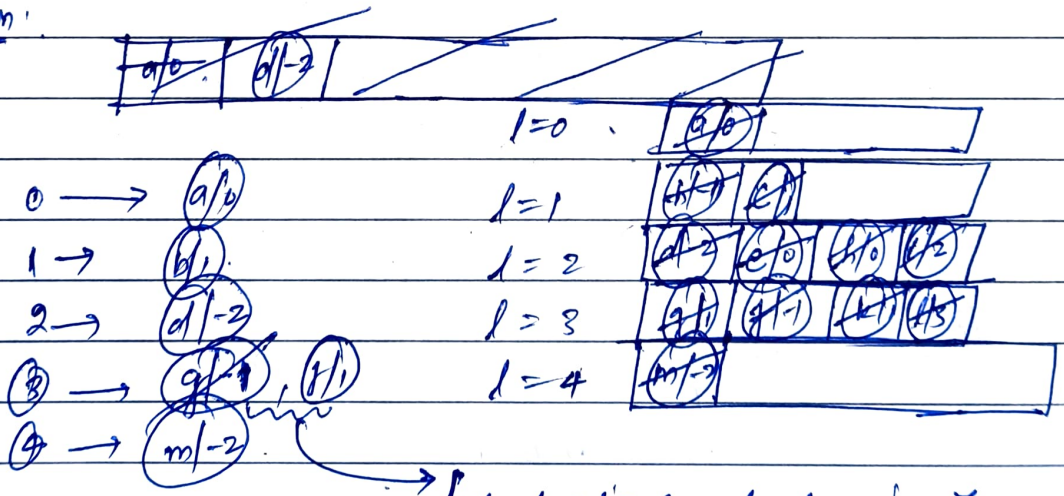
APCO
Date :
Page :

View type question :-

→ height and level both have to maintain apply level order.



dry Run :

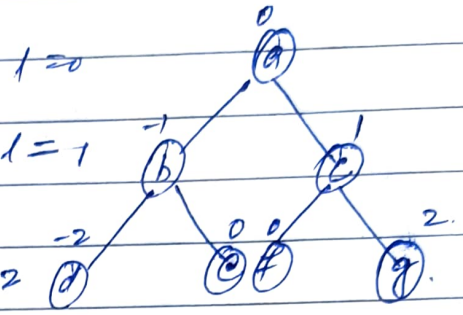


→ [at particular level min of x index gives the left view].

APCO
Date :
Page :

M-2 ~~Left view Recursive~~.

On // width :-



Uses of width :

;) It is very Imp as it is used for coordinate of x-axis. to pair node ;

// Code.

```
ps void width(Node node, int level, int[] maxMin){
    if(node == null) return;
    Math.min(
    maxMin[0] = Math.max(MaxMin[0], level);
    maxMin[1] = Math.min(MaxMin[1], level);

    width(node.left, level-1, maxMin);
    width(node.right, level+1, maxMin);

}
```

// Pair class.

```
Public static class Pair{
    Node node = null;
    int val = 0;
    Pair(Node node, int val){
        this.node = node;
        this.val = val;
    }
}
```

APCO
Date :
Page :

Left view : (Using concept of geometry).

How / Algo :-

1. make queue, Arraylist to store ans and level to keep track of level
2. Run while loop till size(q) != 0
   - 2.1 Run inner while till que ka size
   - 2.2 Remove element from q. add into que child of r.e.
   - 2.3 add ans int list.

// code :-

```
Public static List<Integer> left View (Node node){
    LL<Pair> que = new LL<>();
    que.addLast(new Pair(node, 0));
    int level = 0;
    AL<Pair> ans = new AL<>();
    while(q.size()!=0){
        int size = q.size();
        while (size-->0){
            Pair vtx = q.removeFirst();
            if (level == ans.size()) ans.add(vtx
            if (vtx.node.left != null) {
                q.addLast(new Pair(vtx.node.left
                    vtx.val-1);
            if (vtx.node.right != null) {
                q.addLast(new Pair(vtx.node.rig
                    vtx.val+1);
        }
        sysout();
        level++;
    }
}
```

APCO
Date :
Page :

(24) left view (without geometry).

```
Ps List<Integer> leftView Interview(Node node){
    LL<Node> que = new LL<>();
    que.addLast(node);

    AL<Integer> ans = new ArrayList<>();
    while(que.size()!=0){
        int size = que.size();
        ans.add(que.getFirst.data);
        while(size-->0){
            Node vtx = que.removeFirst();
            if(vtx.left != null)
                que.addLast(vtx.left);
            if(vtx.riget != null){
                que.addLast(vtx.riget);
            }
        }
    }
    return ans;
}
```

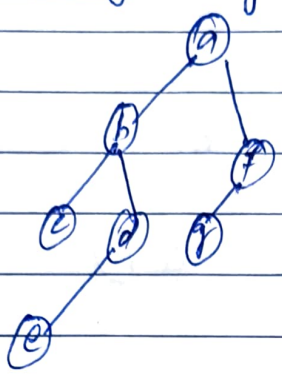
Flow / algo :-

1. take que and add node into it.
2. after removal of first element insert it child into q and add. remove elem into ans.

APCO
Date :
Page :

Right view without geometry.

dry Run.



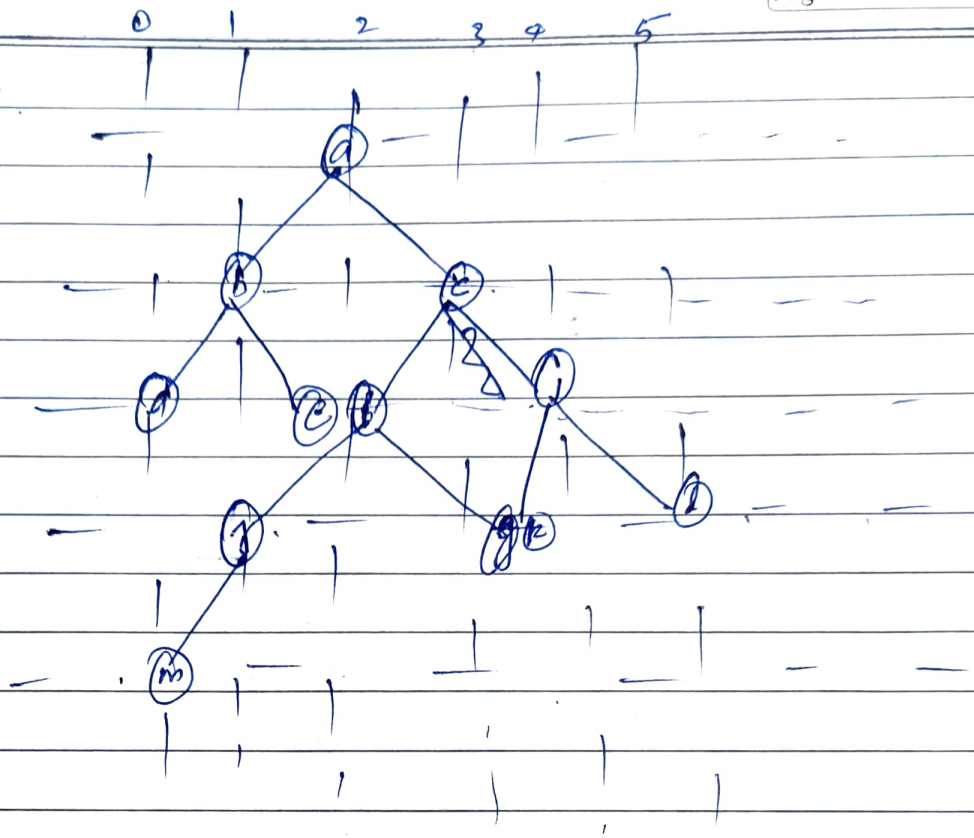
(a, f, g, e).

// code.

```
Public static List<Integer> rightView(Node node){
    LL<Node> que = new LL<>();
    que.addLast(node);
    AL<Integer> ans = new AL<>();
    while(q.size != 0){
        int size = q.size();
        int prev = -1;
        while(size-- > 0){
            Node vtx = que.removeFirst();
            if(vtx.left != null)
                que.addLast(vtx.left);

            if(vtx.right) != null){
                que.addlast(vtx.right);

            prev = vtx.data;
        }
        ans.add(prev);
    }
    return ans;
}
```

APCO
Date :
Page :

top view :



0 → d, m.

1 → b, f.

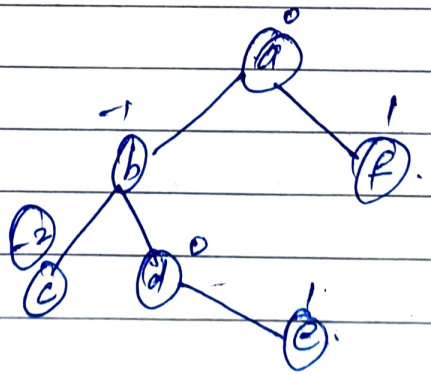
2 → a, e, h.

3 → c, g, k.

4 → i

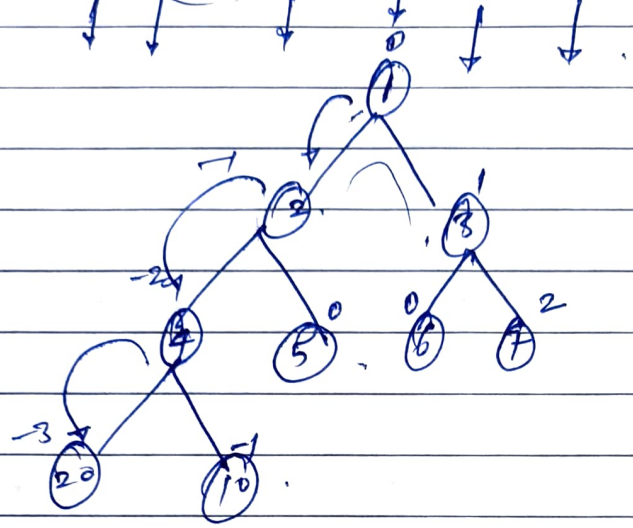
5 → l.

c-2



APCO
Date :
Page :

1. Top view.
2. Bottom view.
3. Vertical sum. (M-2) using M, M



0 → 1 -3 → 20.

+ → . -2 → 4.

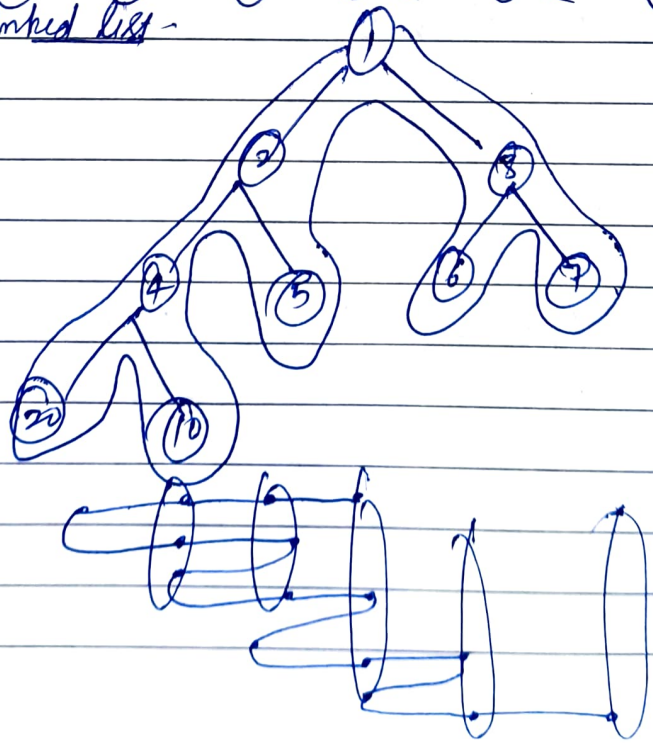
-1 → 0 2 + 10.

0 → 4..

1

2

M-3 using linked list.



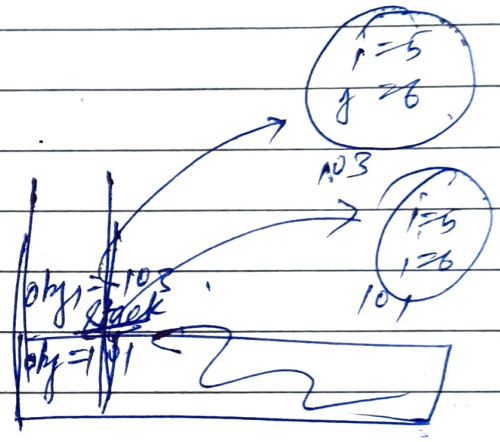
APCO
Date :
Page :

Three ways of Copying object

1. deep Copy.
2. shallow Copy.
3. cloning.

eg: deep Copy -

```
class A{
    int i;
    int j;
}
A obj = new A();
obj.i = 5
obj.j = 6;

A obj1 = new A();
obj1.i = obj.i
obj2.j = obj.j
```



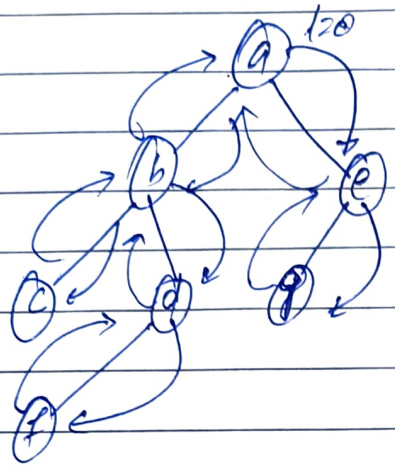
it's not a good way. because have to copy each element individually.

(A) cloning.

```
A obj = new A();

A obj1 = obj.clone(); and it fires.
                      Copy constructor.
```

APCO
Date :
Page :

Q. Right Side view using Recursion.



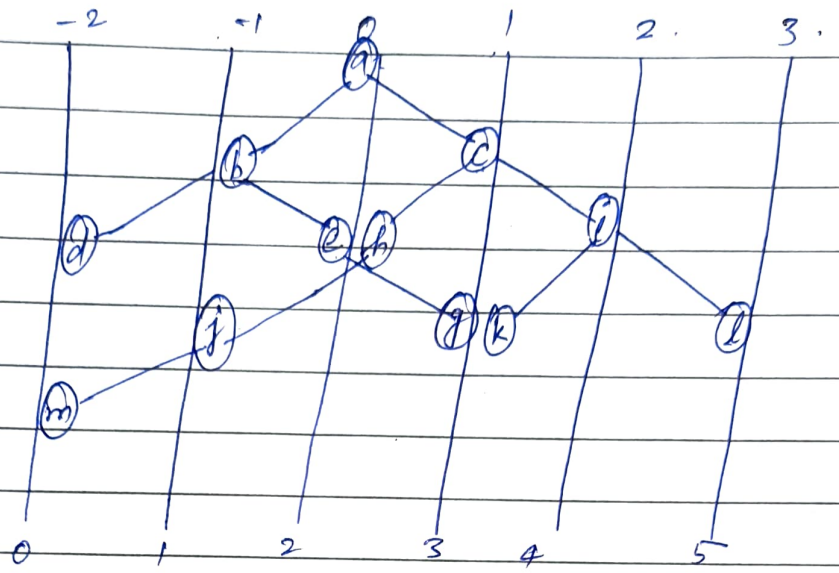
a	e	g	f	

// Code -

```
void RightSideView(TreeNode* node, int level, vector<int>& ans){
    if(node == nullptr)
        return;
    if(level == ans.size()) ans.push_back(node->val);

    RightSideView(node->right, level+1, ans);
    RightSideView(node->left, level+1, ans);
}
```

APCO
Date :
Page :

Que Vertical view / virtical order traversal.



0 → d, m.
1 → b, f
2 → a, e, h.
3 → c, g, k.
4 → i
5 → l.

bottom view with allignment

Consider left data in case of overlap.

0 → m
1 → f
2 → e
3 → g
4 → i
5 → l.

Take traversal of last element

Top view.

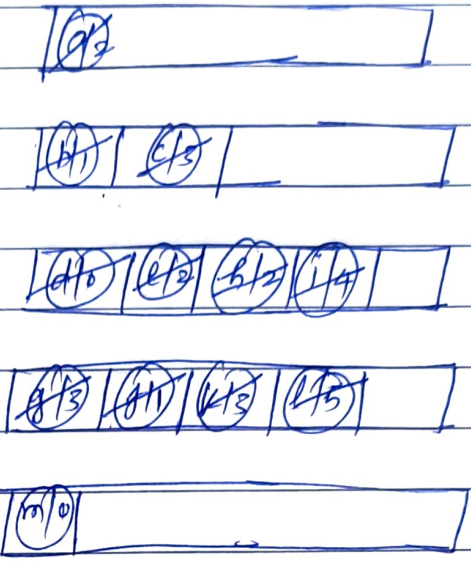
0 → d
1 → b.
2 → a
3 → c
4 → i
5 → l.

Take traversal of first element.

APCO
Date :
Page :

dry Run of Vertical order traversal.

0 → d, m.
1 → b, j
2 → a, g, e, h.
3 → c, g, k
4 → i,
5 → l.

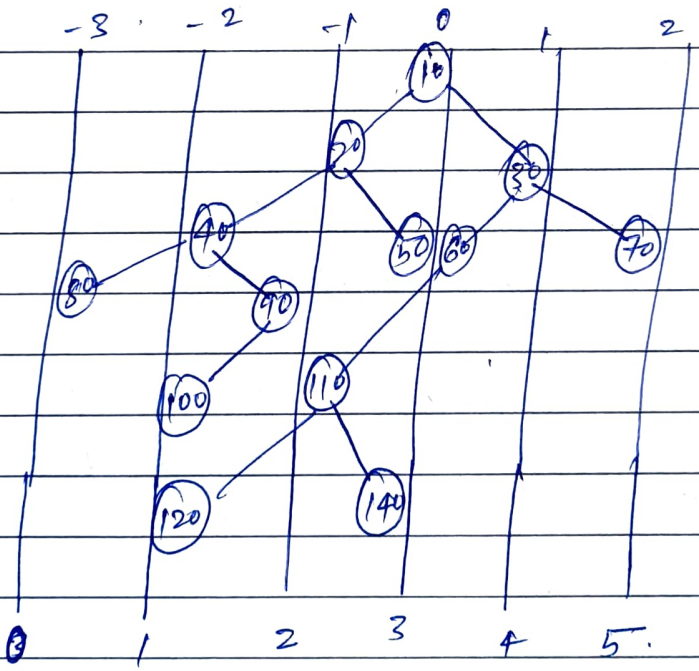


// Code -

```
PS: void VerticalOrderTraversal(Node node){
    int maxMin = new int[2];          MaxMin[1] //min
    width(node, 0, maxMin);           [0] //max

    int n = maxMin[0] - MaxMin[1] + 1;
    AL<Integer> ans[] = new AL[n];
    for(int i=0; i<n; i++) ans[i] = new AL<>()

    LinkedList<Pair> que = new LL;
    que.addLast(new Pair(node, -maxMin[1]));
    while(que.size()!=0){
        int size = que.size();
        while(size-->0){
            Pair vtx = que.removeFirst();
            ans[vtx.val].add(ans.node.data);
        if(vtx.node.left!=null) que.addLast(
            new Pair(vtx.node.left, vtx.val-1));
```

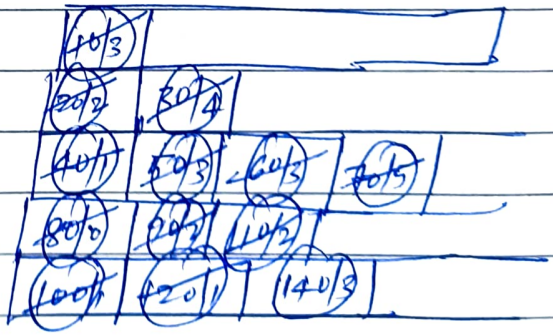
```
if (vtx.node.right != null) que.addLast(new Pair(
        vtx.node.right, vtx.val+1));
}. // inner while.
} // outer while.
} // fn close.
```

Qu. // vertical Sum →



dry Run:

0 → 80
1 → 40 +100 +120
2 → 20+90 +110 +
3 → 10 +50+ 60 +140
4 → 30
5 → 70



APCO
Date :
Page :

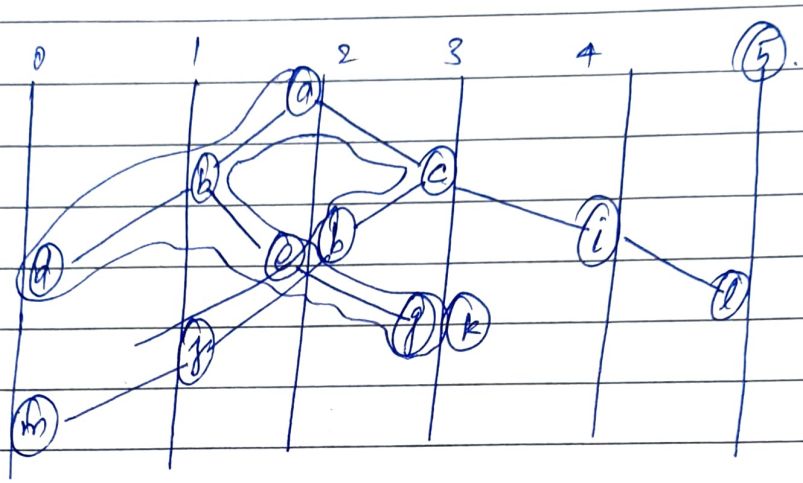
// Code.

```
Public static void verticalSum (Node node){
    int[] maxMin = new int[2];
    width( node, 0, maxMin);

    int n = maxMin[0] - maxMin[1]+1;
    int ans[] = new int[n];
    LL<Pair> que = new LL;
    que.addLast( new Pair(node, -maxMin[1]));

while (que.size() != 0) {
    int size = que.size();
      while (size-- >0){
        Pair vtx = que.removeFirst();
(*)     ans[vtx.val] += ans.add node.data;
        if(vtx.node.left != null) que.addLast(new
            pair(vtx.node.left, vtx.val-1));
        if( vtx.node.right != null) que.addLast(
            new Pair(vtx.node.right, vtx.val+1));
    } // inner while.

} // outer while.

} // fn.
```

APCO
Date :
Page :

M-3 Top view using hm and Rec.

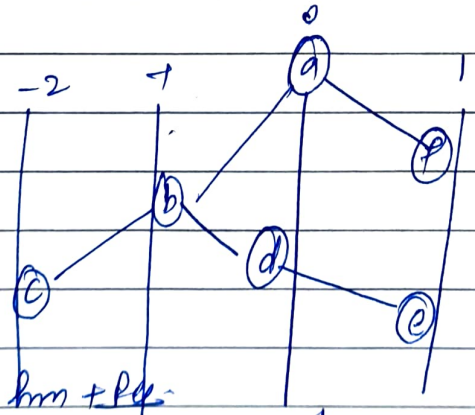
It fail some cases just for knowledge purpose only.



-2 → d, m.
-1 → b, j
0 → a, e, b.
1 → g, c, k ✗
2 → i
3 → l.

for top view 'c' should come before 'g'. so it fails.

T-2



(for top view)

using hm + Rec.

-2 → c
-1 → b.
0 → a, d
1 → e, f ✗
c, b, a, e ✗.

-2 → c
-1 → b.
0 → a, d
1 → f, e.

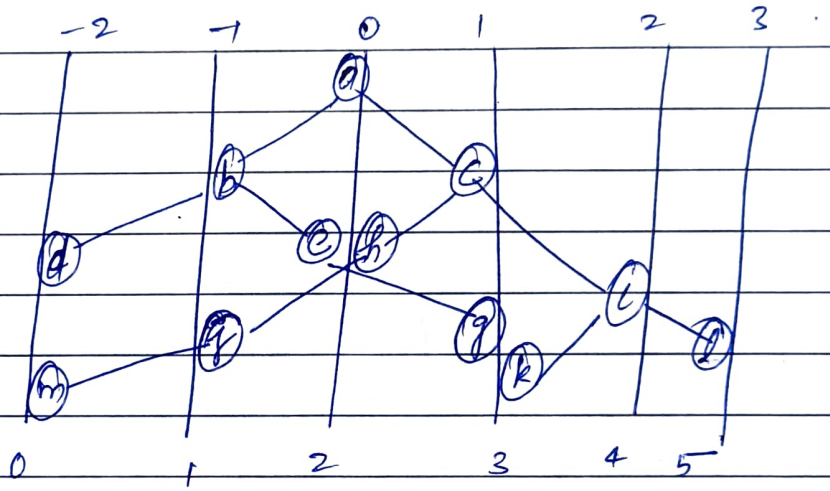
c, b, a, f

APCO
Date :
Page :

★ It can be corrected using priority queue with coordinate but time complexity get worse. So, former is better solution.

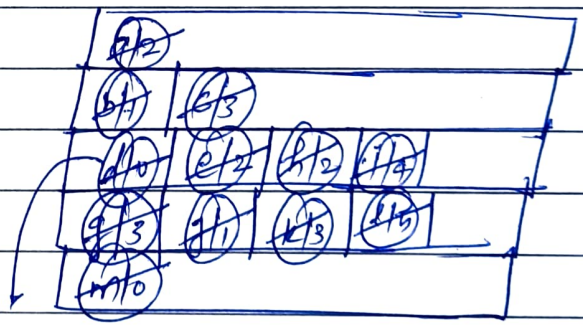
T: O(n)
S: O(n)

Qu Top view using level order.



dry Run-

0 → a
1 → b,
2 → e
3 → c
4 → j
5 → l.



becaz '0' index filled

Note: - fill arr index only if Vtx.val = null. ★

APCO
Date :
Page :

// Code:

```
Public static void topView(Node node){
    int[] maxMin = new int[2];
    width(node, 0, maxMin);
    int n = maxMin[0] - maxMin[1] + 1;
    Integer[] ans = new Integer[n];

    LL<pair> que = new LL;
    que.addLast(new pair(node, -maxMin[1]));

    while(que.size() != 0){
        int size = que.size();
        while(size-- > 0){
            pair vtx = que.removeFirst();
            if(ans[vtx.val] == null)
                ans[vtx.val] = ans.node.val;

            if(vtx.node.left != null){
                que.addLast(new pair(vtx.node.left, vtx.val-1));
            if(vtx.node.right != null){
                que.addLast(new pair(vtx.node.right, vtx.val));
            }

        } // inner while.

    } // outer while.

} // fn.
```

APCO
Date :
Page :

Q4 Bottom view.

Everything same just change. following:

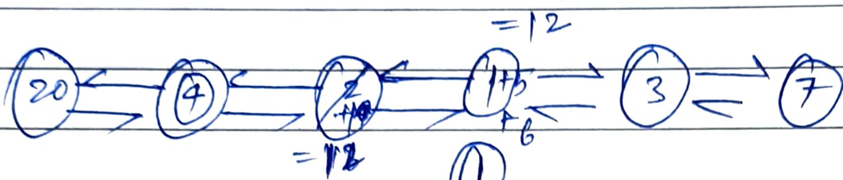
```
ans[vtx.val] = ans.node.data;
```

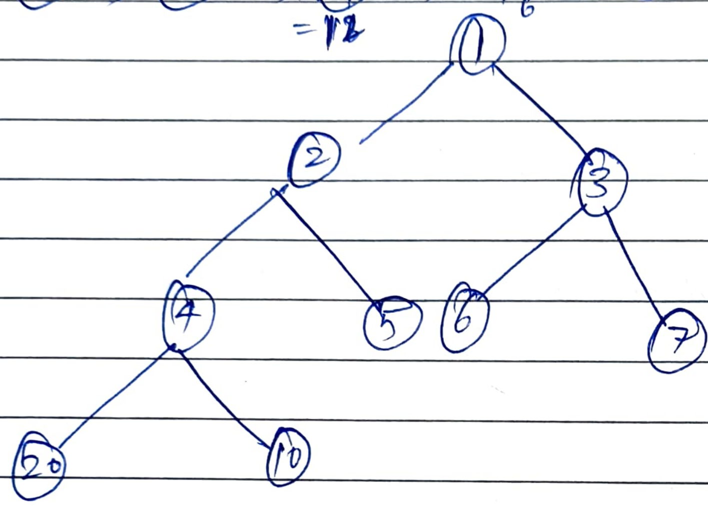
did this because this remain changing for that index till last and

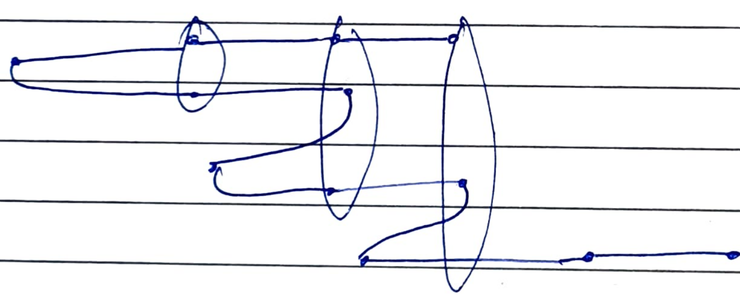
→ last data would be our answer for that index.

→ Same for all other index.

Q4. Vertical sum using LL (range) not go using this in interview but very useful and learning qn :-







APCO
Date :
Page :

O(n) based solution.
n → total vertical line or x-coordinates.

```
static void verticalSumDLL(TNode root){

    LLNode llnode = new LLNode(0);
    verticalSumDLLUtil(root, llnode);

    while(llnode.prev != null){
        llnode = llnode.prev;
    }

    while(llnode != null){
        System.out.println(llnode.data + " ");
        llnode = llnode.next;
    }
}
```

~~Constant~~

```
static void verticalSumDLLUtil(TNode tnode, LLNode llnode){

    llnode.data = llnode.data + tnode.data;
    if(tnode.left != null){
        if(llnode.prev == null){
            llnode.prev = new LLNode(0);
            llnode.prev.next = llnode;
        }
        fn(tnode.left, llnode.prev);
    }
```

```
if(tnode.right != null){
    if(llnode.next == null){
        llnode.next = new LLNode(v);
        llnode.next.prev = llnode;
    }
    fn(tnode.right, llnode.next);
}
```

APCO
Date :
Page :

// 987

APCO
Date :
Page :

(lecture-4).

(Sept 23, 2020)

(Agenda):

① diagonal view.
② Boundry traversal.
   → geeks वाला
   → bcp वाला

① BST start
   → Construction.
   → Max.
   → Min
   → find.

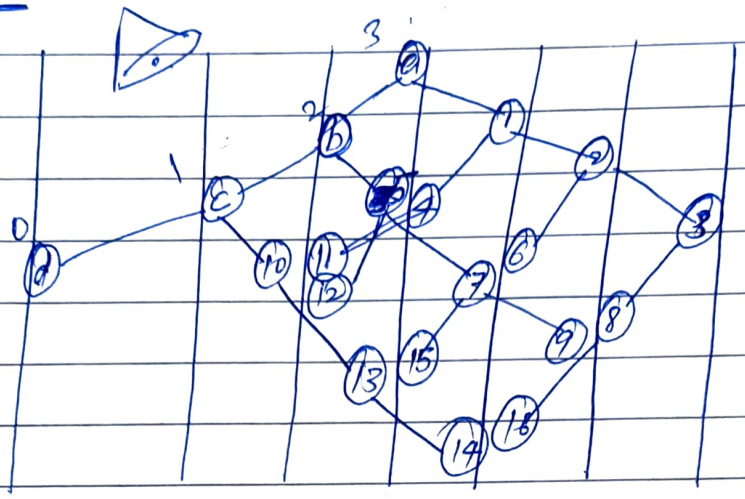
⑦ 98// valid BST. (M-1).
M-2

⑧ 99 recover BST.

⑨ verify preorder traversal in BST.

APCO
Date :
Page :

Q4 Diagonal view.



(pair).

M-1 approach: make a queue and put first (element/index) whenever remove element then if it is left child then put by reducing '1' index. and if it is right child then put as it is

day run :-

right child

left

Ans of diagonal view.

0 → d
1 → c, 11, 16, 12, 15, 10, 13, 14
2 → b, 4, 6, 8, 5, 7, 9.
3 → a, 1, 2, 3.

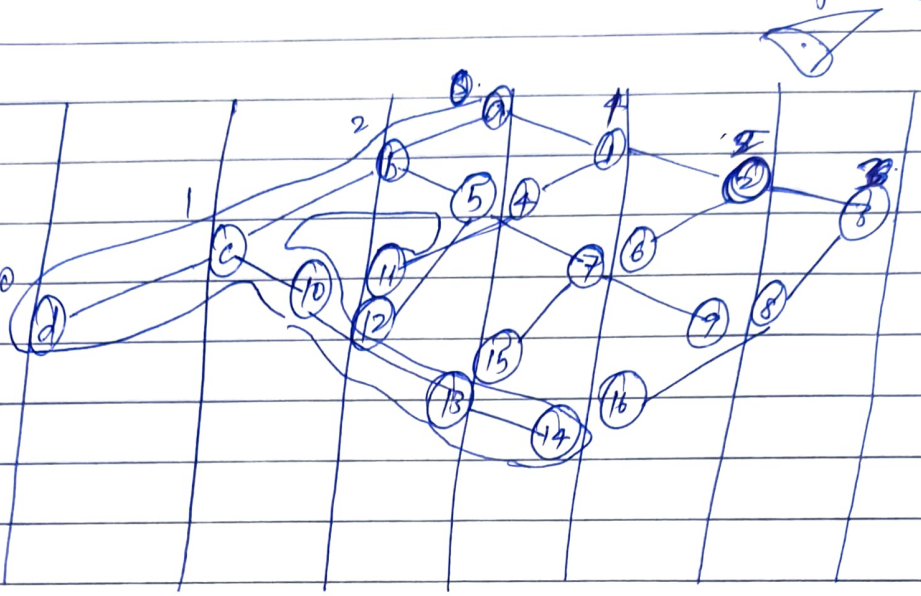
M-2 we can do it in ordered form means eg.

1 → c, (10, 11, 12) (13, 15) (14, 16).

For this we have to put pairs in the queue of coordinates.

APCO
Date :
Page :

Note if we want to observe from right side diagonally. eg.



① Put ~~right~~ left element in same que with same index.

② Put ~~left~~ right in another que with index+1.

a → a, b, c
b → 1, 5, 10, 4, 12, 11
c → 2, 7, 13, 6, 15
d → 3, 9, 14, 8, 16.

Same if we want in ordered formed then put coordinate as well in the pair.

Note M-1 can be done using Recursion also.

```
ans[level].add(node.data);
fn(node.left, level-1, ans);
fn(node.right, level+1, ans);
```

APCO
Date :
Page :

2. Boundry Traversal. (Not done by sir) so please do.

~~gfg~~

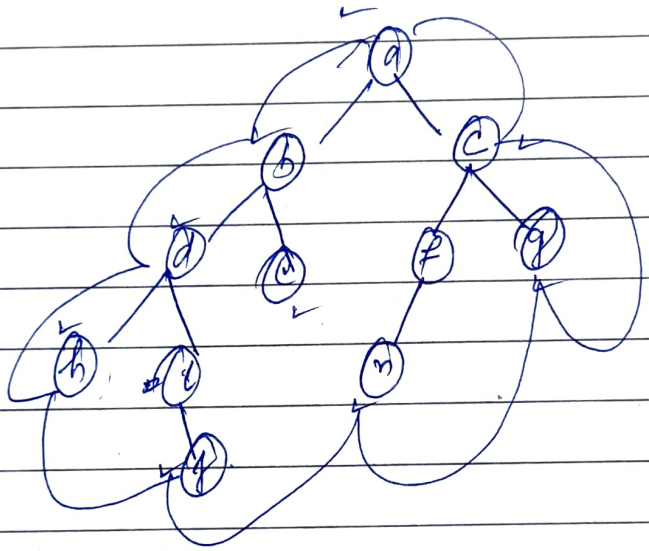
According to gfg.

in boundry traversal we will have.

1. left view node except leaf.
2. Right view node except leaf.
3. leaf node.

boundry.

eg.



lv = a, b, d.

BV = h, j, e, n, g.

rv = c, a

(logically 'e' नहीं आना चाहिए) because bottom view में e दिखाई नहीं देगा

if we consider 90° concept in allignment of nodes.

APCO
Date :
Page :

Dry Run :-

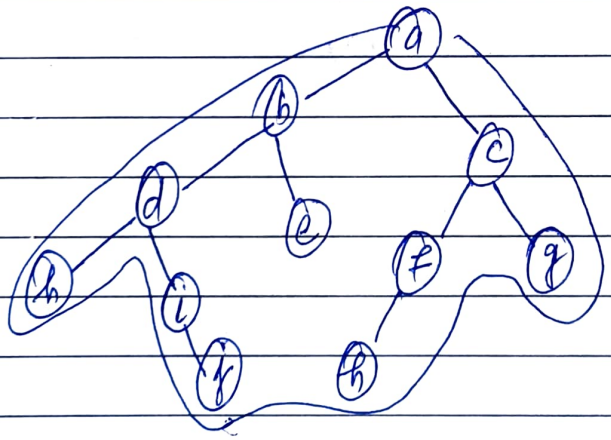
LV : a, b, d, h.
BV : h, d, i, f, h, c, g.
RV : a, c, g.

M.M(Node).

a	✓
b	✓
d	✓
h	✓
i	✓
f	✓
c	✓
g	✓

here we get the.

By Assuming go' concept of nodes.

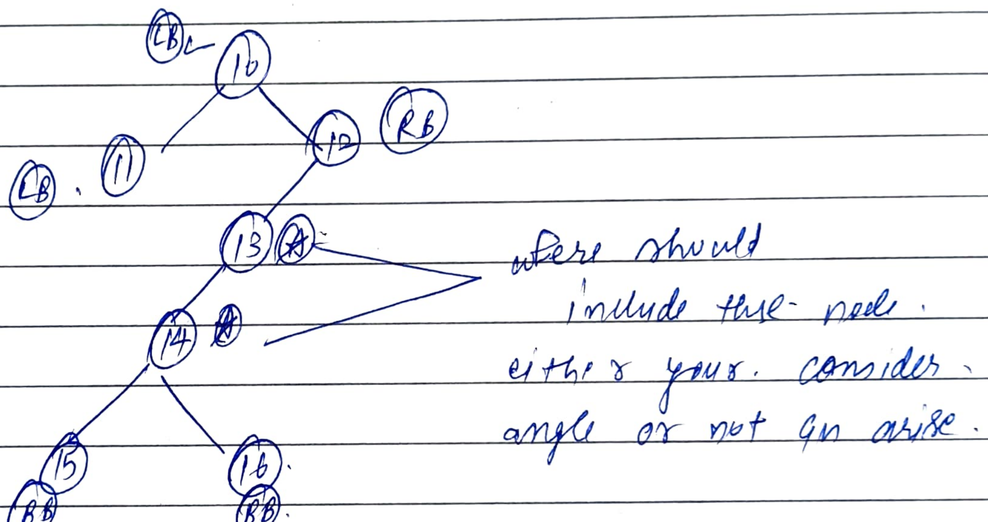


M-2 ~~Boundary traversal pep~~ asked in f2f only.

Boundary traversal gfg solution

① left traversal करो except leaf.
② right " " " "
③ leaf traversal only.
and print combination of all three.

Counter test case for this qn



so its correct solution should be

L.B → 10, 11, 13, 14
B.B → 15, 14, 16.
R.B → 16, 14, 13, 12, 10

and run hashMap.

APCO
Date :
Page :

Agenda:

BST

① construction.
- → max
- → Min.
- → find.

② Valid BST //98.

M-2 Using pair class.

③ 99 // Recover BST. // 99

④ Verify preorder traversal in BST. //255

APCO
Date :
Page :

1. Qu Construction.

What. Have to construct a bst using sorted array.

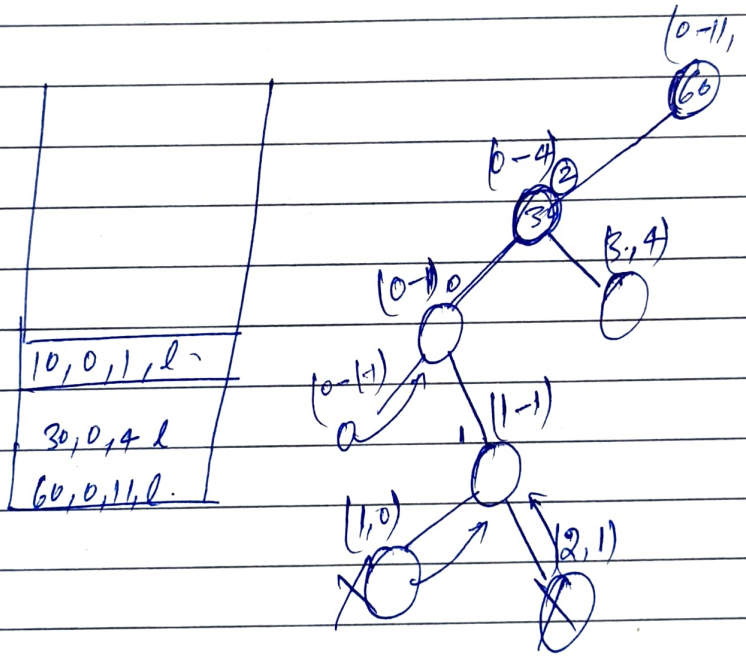
How.

10	20	30	40	50	60	70	80	90	100	110	120
0	1	2	3	4	5	6	7	8	9	10	11

Stack → mid, s, e, direction l/r.

$\frac{11}{2} = 5$

$\frac{4}{2} =$



Algorithm :- ① Find mid
② Call to left for (si, mid-1).
③ Call to right for (mid+1, ei).

APCO
Date :
Page :

// Code.

```
class Node {
    Public:
        int data = 0;
        Node* left = nullptr;
        Node* right = nullptr;

    Node(int data) {
        this.data = data;
    }

};
Node* ConstructTree(Vector<int> &arr, int si, int ei) {
    if(si > ei) {
        return nullptr;
    }
    int mid = (si+ei)/2;

    Node* node = new Node(arr[mid]);
    node->left = ConstructTree(arr, si, mid-1);
    node->right = ConstructTree(arr, mid+1, ei);

    return node;
}
```

APCO
Date :
Page :

2 Qu Max:

Note :- Try to do bst question using iteration instead of recursion because it is way faster and.

if → one people (code in iteration) → selected.
→ Rec → good but not that much.

// code.

```
int maximum (Node * node){.
    Node* curr = node;.
    while( curr->right != null){
        curr = curr->~~next~~ right;.
    }
    return curr-> data;.
}.
```

APCO
Date :
Page :

3.04 ~~Min~~ Find in BST.

```
bool Find(Node* node, int data){

    Node* curr = node;
    while(curr != nullptr){
        if(curr->data == data){
            return true;
        }
        else if(curr->data < data){
            curr = curr->right);
        }
        else
            curr = curr->left;

    return false;
}
```

APCO
Date :
Page :

4.04 ~~Find~~ Minimum in BST.

// Code.

```
int minimum (Node* node) {
    Node* curr = node;
    while (curr -> left != nullptr) {
        curr = curr -> left;
    }
    return curr -> data;
}
```

APCO
Date :
Page :

5. Valid BST 1/98.

What ?

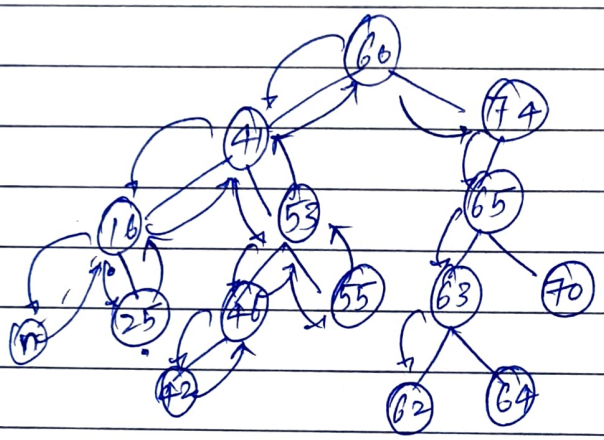
Validate Binary Search Tree.

Note validating condition.

① left subtree must contain value less than node.
② right " " " greater the node value.
③ Both left and right subtree must also be BST.

How :- Make a prev node and traverse and update prev node in inorder manner.

eg.



prev = ~~null 16 25 41 42 46 53 55 60 62~~

. . . . go till 74

if any where we found

prev != null and prev.val > node.val ⇒ return f.

Remember prev वाली concept

// code.

```
TreeNode prev = null;
Public boolean isValidBST (TreeNode root){
    if( root == null) return true;

    if( ! isValidBST (root.left)) return false;

    if( prev != null && prev.val > root.val)
        return false;
        prev = root;
    if( isValidBST (root.right) ) return false;

    return true;

}
```

APCO
Date :
Page :

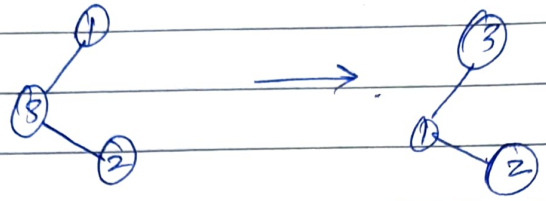
M-2

```
Public class BSTPair {
    boolean isBST = true;
    long min = (long) 1e8;
    long max = -(long) 1e8;

Public BSTPair isValidBST_(TreeNode root){
    if(root == null) return new BSTPair();

    BSTPair left = isValidBST_(root.left);
    BSTPair right = isValidBST_(root.right);

    BSTPair myAns = new BSTPair();

    if(!left.isBST || !right.isBST || left.max >=
        root.val || right.min <= root.val) {
        myAns.isBST = false;
        return myAns;
    }

    myAns.min = Math.min(left.min, root.val);
    myAns.max = Math.max(root.max, root.val);

    return myAns;

}

// Calling Public boolean isValidBST_(TreeNode node){
        return isValidBST_(node).isBST;

    }
```

(6) Recover BST. //99.

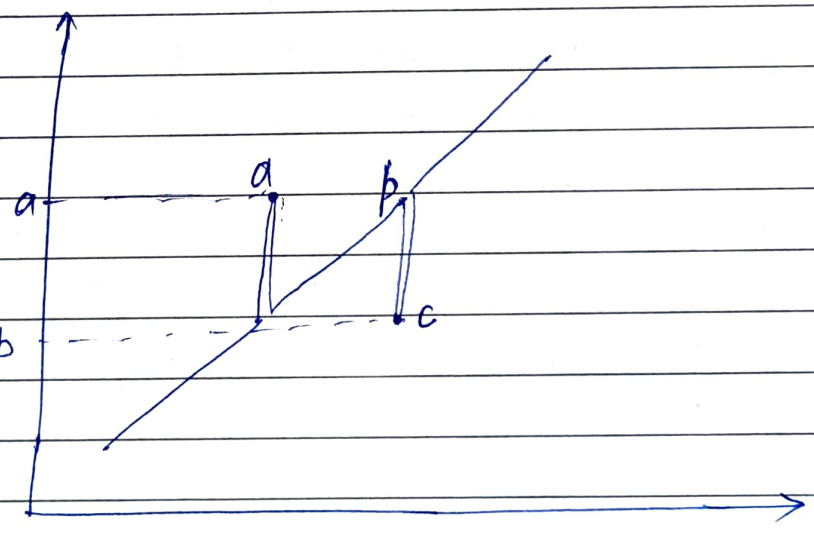
What ? Two elements of BST are swapped by mistake. Recover the tree without changing it structure

eg



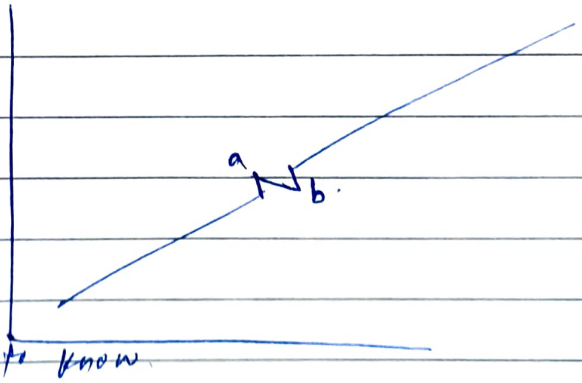
How ? Make 3 variable node a, b and prev. traverse along the tree. if problem occur then. then update 'a' and 'b' if problem occur again in BST & a != null then return and swap a and b.

C-1



C-2

In this condition if Cond" donot fire again and after doing all stuff 9 go to know my change element are adjacent



APCO
Date :
Page :

// Code

```
TreeNode *a, TreeNode* b, TreeNode* prev; = nullptr;
bool recoverTree( TreeNode * root){
    if(root == null){ return false;
    if(recoverTree(root->left)) return true;

    if( prev != nullptr && prev->val > root->val){
        b = root;
        if (a == nullptr)  a = prev;
        else
            return true;

    }
     prev = root;
    if( recoverTree(root->right));
     return true;
    return false;
}
```

// swap a, and b Node's value.

```
void recoverTree( TreeNode * root){
    recoverTree(root);
    if( a != nullptr){
        int temp = a->val;
        a->val = b->val;
        b->val = temp;
    }

}
```

APCO
Date :
Page :

7.Qn verify preorder traversal in BST (//255).

APCO
Date :
Page :

Lecture – 5

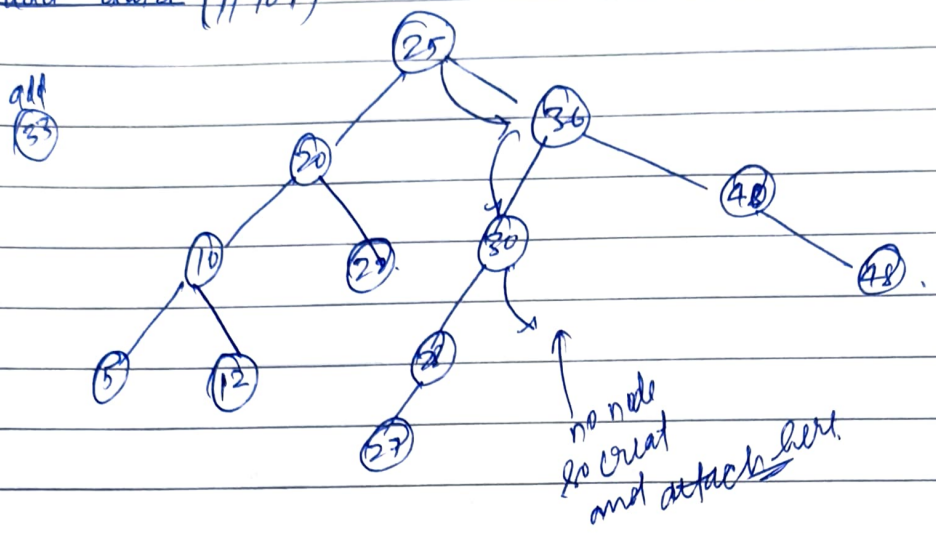
Sept. 27, 2020.

Agenda:

1. add data.
2. Remove data.
3. all solution pairs → in bst and bt
4. 285 // Inorder successor (H.W) such that given value. Not in leetcod locked
5. 510. // Inorder Successor → with given data. locked in leetcode.
6. ~~all solutions~~ ✗
7. BST iteration.
8. LCA in BST.

APCO
Date :
Page :

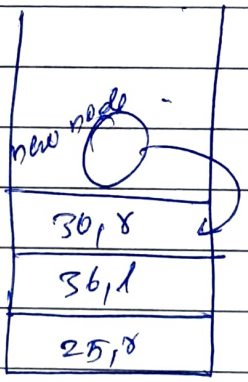
1. add data (1170)



add 33.

How ? ① find the location where you can add (33)
② after finding the location create node and attach it to its prev left/right.

if use in stack.



APCO
Date :
Page :

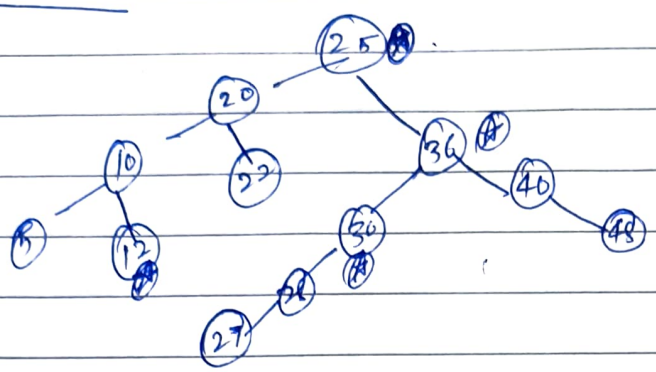
// Code.

```
TreeNode* insertIntoBST(TreeNode* root, int val) {
    if(root == nullptr)
      return new TN(val);

    if( val < root->val).
      root->left = insertIntoBST( root->left, val);.
    else {.
      root->right = insertIntoBST( root->right, val);.

    }.

    return node;.

}.
```

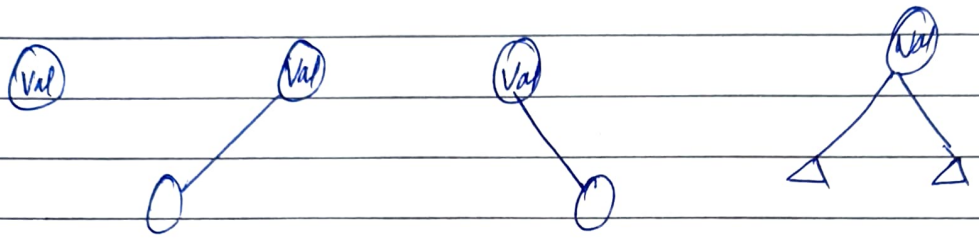
APCO
Date :
Page :

2. Op Remove data

Sub Tree.



Four cases of deletion:



// Code.

```
Node* deleteNode (Node* root, int val){
    if(root == nullptr) return nullptr;

    if( val < root->data)
        root->left = deleteNode (root->left, val);

    else if( val > root->data){
        root->right = deleteNode (Node->right, val);
    }

    else{
        if( root->left == nullptr || root->right == nullptr)
        { Node* node = root->left != nullptr ?
                       root->left : root->right;
          delete root; //not for java.
          return node;
        }

        int mVal = maximum(root->left);   (get max in left subtree)
        root->data = mVal;
        root->left = deleteNode(root->left, mval);
    }

    return root;
}
```

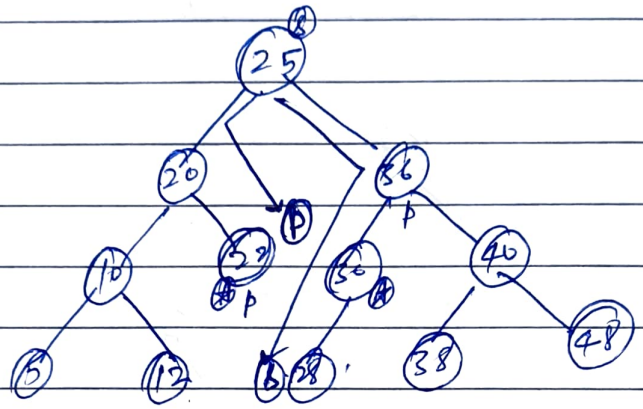
[handle when we don't have left/right child = null]

① add data .
② Remove data .] → very Imp. for avl.

APCO
Date :
Page :

3 Qn predecessor of node → node के पहले वाला
succ nor " " → node के बाद वाला

M-1 ① Inorder traversal करके array में store करे आओ
② traverse array and find

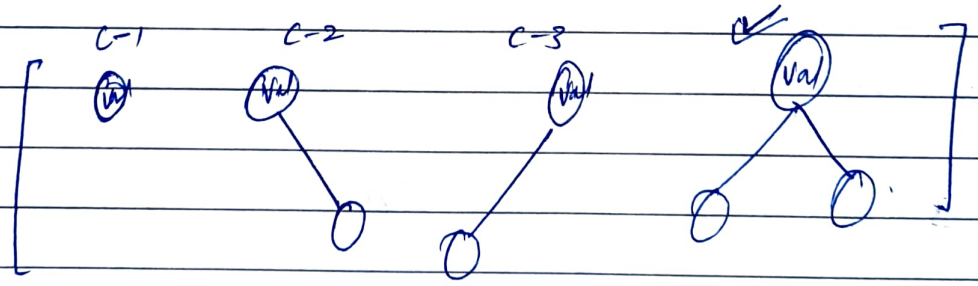


Confirm BST / BT.

if BT. → traverse tree → left → Floor.
↳ right tree.

BST → have to find. predecessor and successor.

V. Imp



Note
→ In BT pred, succ., ceil, floor are different.
→ while in BST.

succ = ceil.
pred = floor.

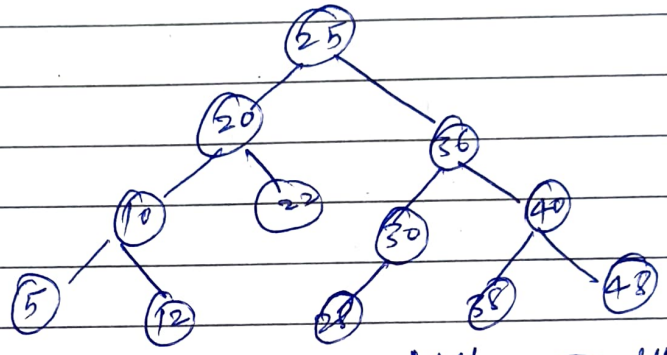
do dry run:

① ~~2 5 1~~ ① In B.T P

Pred, succ, ceil and floor all are different

② while in BST Pred = floor } .
succ = ceil.

eg.



op : ~~2 3~~ 5, 10, 12, 20, 22, 25,

M-1 Array में ~~आर~~ से ले आओ
elem से पहले pred, and बाद वाला succ.

M-2 Solve in tree without space.

for B.T
[T: O(n) → 1.n (pan)
S: Recursive space only

APCO
Date :
Page :

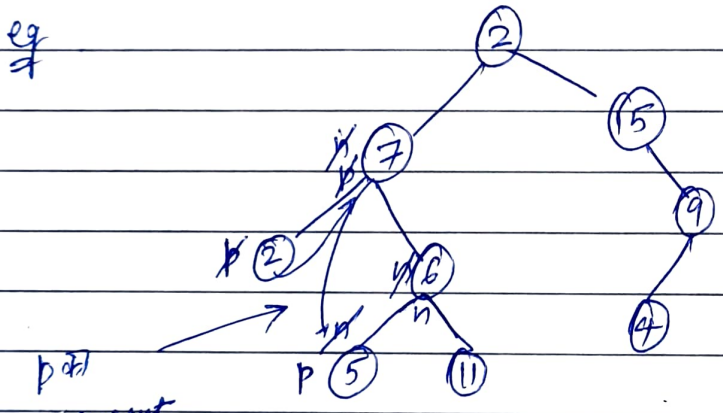
For BST T : o(logn) one pass solution → pred succ
S : o(1)

① For k pred, succ, floo and ceil

How ? ① make a class having.
pred, succ, prev, ceil and floor.

② Traverse in and update in inorder manner.

eg



5,1
6,1
7,8
2,1

ceil → just value.
floor → just val.

APCO
Date :
Page :

~~857. all solution pair.~~

// Code.

```
Public static class allSolPair {
    Node prev = null;
    Node pred = null;
    Node succ = null;
    int ceil = (int) 1e9 ;
    int floor = -(int) 1e9;

}

Pr void allSolPair( Node node, int data, allSolPair ans){

    if ( node == null) return;

    if( node.data > data)
        ans.ceil = Math.min( ans.ceil, node.data);
    if( node.data < data)
        ans.floor = Math.max( ans.floor, data);
```

(आगे दी data से comp करके update ceil & floor)

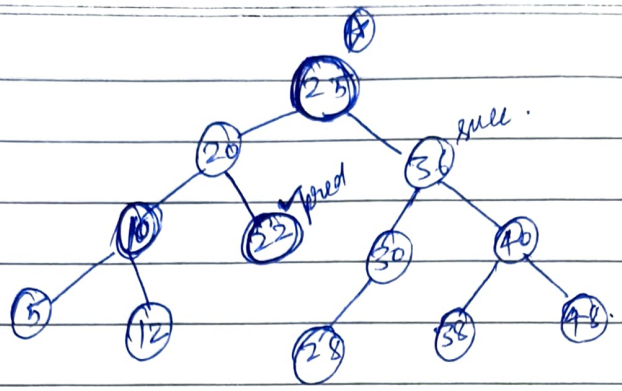
call // allSolution(node.left, data, ans);

```
    if ( node.data == data)
        ans.pred = ans.prev

    if ( ans.prev != null && ans.prev.data == data)
        ans.succ = node;

    ans.prev = node;

// right call // allSoln( node.right, data, ans);

}
```

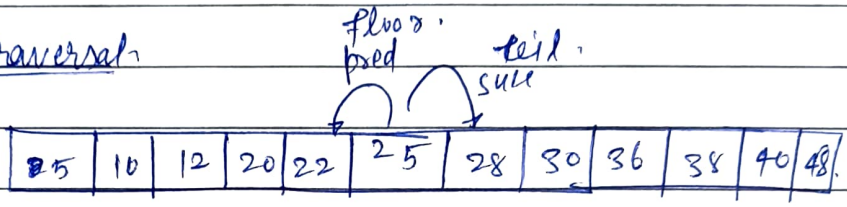
APCO
Date :
Page :

Imp

In BST.



Inorder traversal.



eg for 25

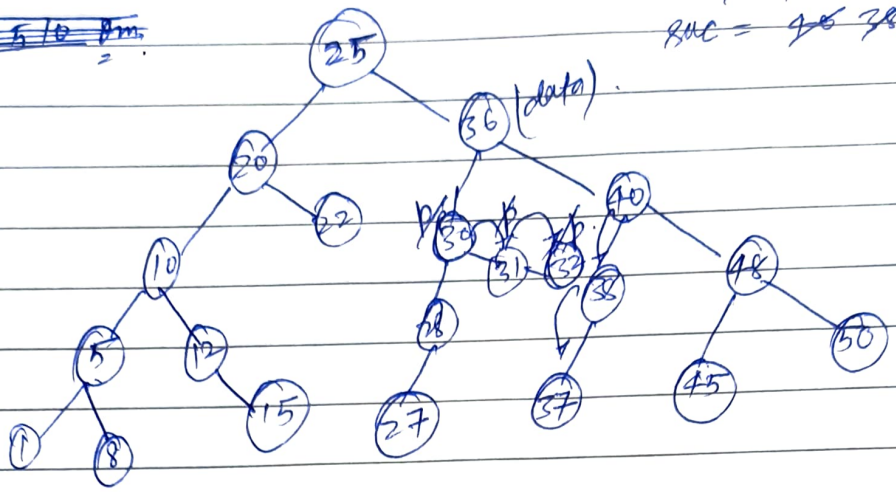
How ? ask wheather B.T or B.S.T

M-1 B.T but T.C : O(n)

M-2 B.S.T T.C : O(log n)

app ① Node ka left me right most pred.
② Node ka right me leftmost succ.

p = 30 ~~31~~ 32
succ = ~~36~~ ~~38~~ 37



how 3 cases arise.

(1) if data > node.data.
right call.

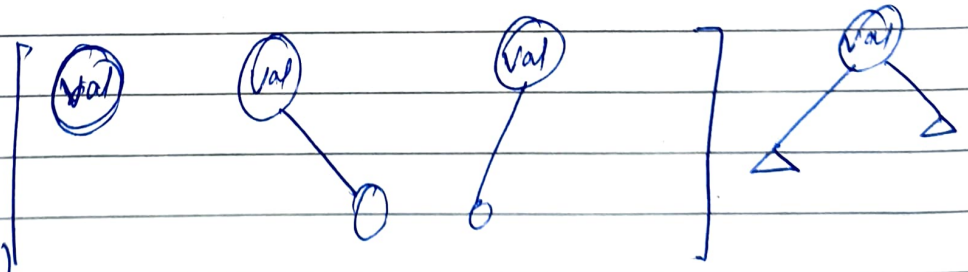
(2) else left call.

(3) data == node.data.

(a) because ~~हमारा~~ pred curr के left का right most होगा
so, update pred till
pred -> right != null.

(b) ~~if~~ succ. curr के right का leftmost ~~होगा~~
so updat till succ का left not null.

Imp T.C



(do dry Run)

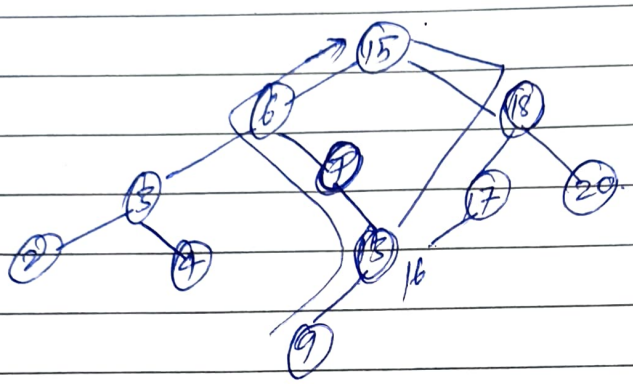
APCO
Date :
Page :

// Code

```
void allSolPairs(Node* node, int data){
    Node* curr = node;
    Node* pred = nullptr;
    Node* succ = nullptr;

    while(curr != nullptr){
        if(curr->data == data){
            if(curr->left != null){
                pred = curr->left;
                while(pred->right != nullptr){
                    pred = pred->right;
                }
            }

            if(curr->right != nullptr){
                succ = curr->right;
                while(succ->left != null){
                    succ = succ->left;
                }
            }

            break;
        }

        else if(curr->data < data){
            pred = curr;
            curr = curr->right;
        }
        else{
            succ = curr;
            curr = curr->left;
        }
    } // while
} // fn
```

Qn 510 // Inorder Successor of with given data value.

What? give a 'node' in a binary search tree find inorder successor of 'node' in BST




```
class Node {
        Public int val;
        Public Node left;
        "      "  right;
        "      "  parent;
}
```

eg Find for node (15) with using its data.

M-1 Comparing data.

app

Here two case arises.

(1) Our successor always lies on node's right's leftmost child (if node's right not null).

(2) lies on node to root path's BST value.

APCO
Date :
Page :

~~Our all solution~~.

C-1 for '15' path-1

C-2. for '9' follow ~~root~~ node to root path. ~~का~~ value answer ~~होगा~~.

// Code

```
Node* inorderSuccessor(Node* node){
    Node* curr = node;
    Node* succ = nullptr;
// node का rightmost access.
    if(node->right != nullptr){
        succ = succ->right;
        while(succ->left != nullptr){
            succ = succ->right;
        }
        return succ;
    }
// parent to root ~~node~~ का access).
    while(curr->parent != nullptr &&
          curr->parent->val < node->val){
        curr = curr->parent;
    }

    return curr->parent != nullptr ? curr->parent :
           nullptr;
}
```

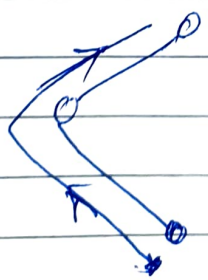
succ = curr.right

APCO
Date :
Page :

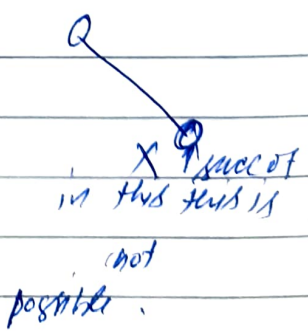
~~LCA in BST~~

M-2 Comparision on the basis of address.

```
Node* inorderSuccessor(Node* node){

    Node* curr = node;
    Node* succ = nullptr;

    if(node->right != nullptr){
        succ = curr->right;
        while(succ->left != nullptr){
            succ = succ->left;
        }
        return succ;
    }
    while(curr->parent != nullptr){
        if(curr->parent->left == curr){
            return curr->parent;
        }
        curr = curr->parent;
    }

    return nullptr;

}
```

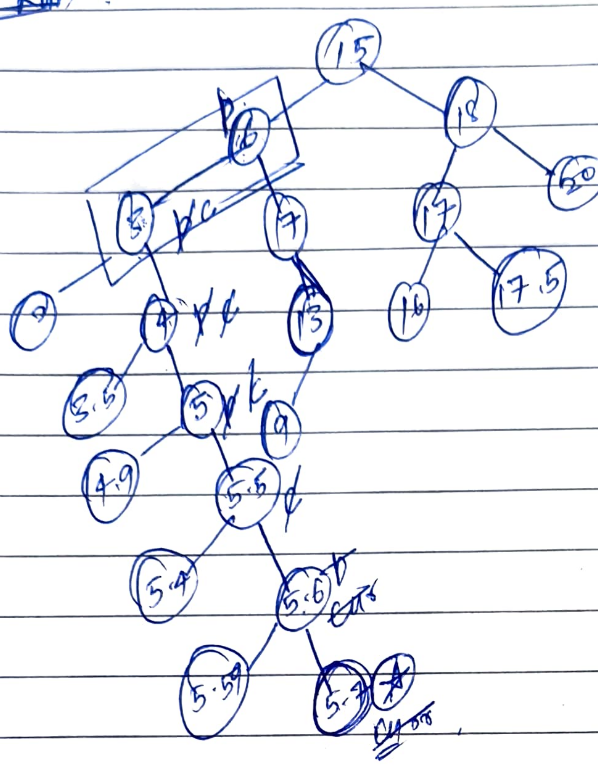


our soln find in only this struct



~~Qn BST iterator.~~

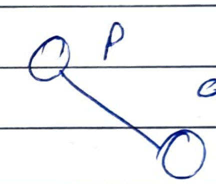
dry Run.



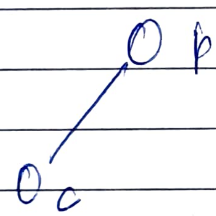
(Parent का पहला left ans होगा)

curr का parent = [illegible]

then return



never get बड़ा val then curr.



only in this st. we can get greater node.

Note here 2 cases arises for successor in ioo manner.

① node ऊपर है then right का leftmost succ होगा

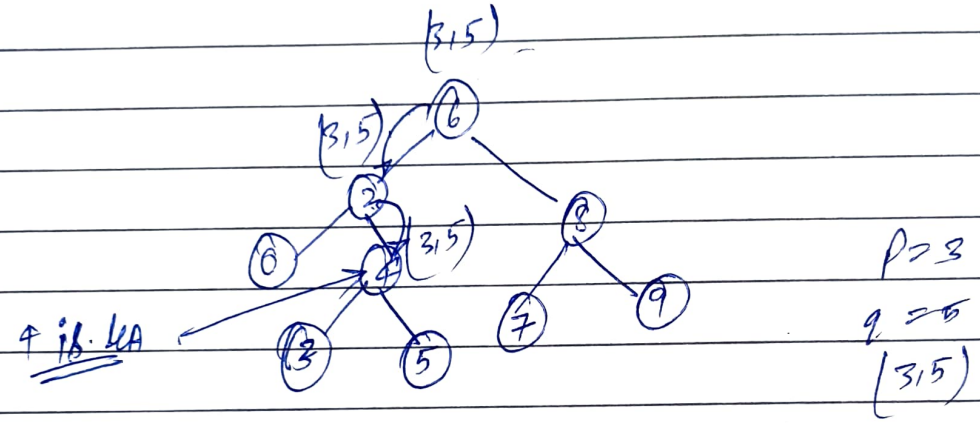
② node नीचे है तो succ node to root path में है।

APCO
Date :
Page :

Q4 235 // lowest common ancestor.

What find LCA of b.t.

How: when we are not able to decide where to call whether left / right then that node become LCA.



// code.

```
TreeNode* LCA( TreeNode* root, TN* p, TN* q ){

    TN* curr = root;
    while( curr != nullptr ){
        if( curr->val < p->val && curr->val < q->val ).
            curr = curr -> right;

        else if( curr->val > p->val && cr->> q->val )
            curr = curr -> right;
        return curr;

    }
}  // fn.
```

APCO
Date :
Page :

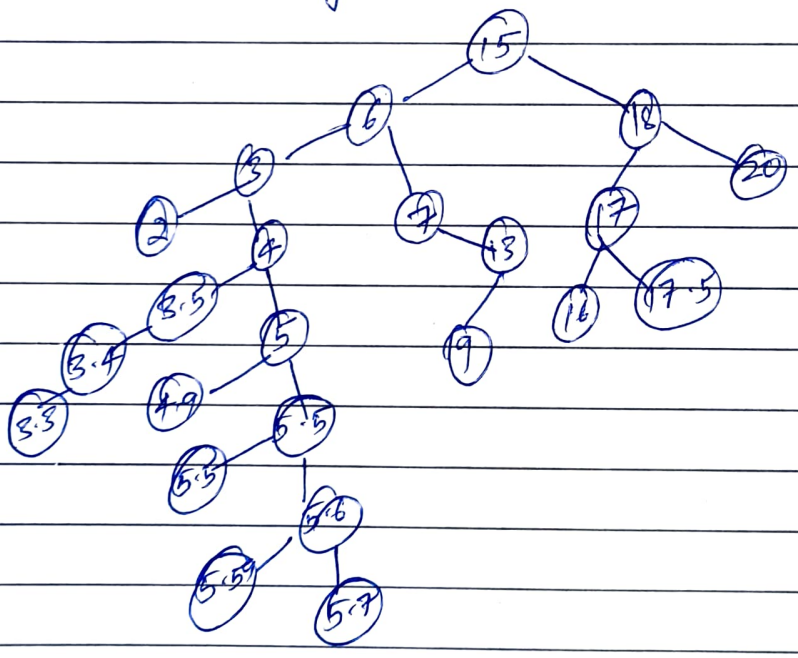
Q4 BST iterator - 11173.

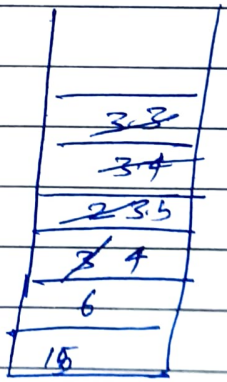
It should have two fn ① calling next() gives next smallest element
② hasnext() tell have more elem or not

How:-

① Put node and all left child into the stack.
② pop() child and put all its right left child into stack. and do until node != null.

dry run.





3.3 3.4 3.5 3 4 6 15 - - -
and so on.

APCO
Date :
Page :

Qu 1/285 Inorder successor.

How ? applied here the same approach as of pred and succ. in one raw.
here only find succ not pred.

APCO
Date :
Page :

(28, Sept, 2020)
(lecture-6)

Agenda:-
AVL

(1) AVL Construction
(2) 1382 // balance bst.

(#) avl is concept to make BST balance so that our comp always remain i.n o(logn).

APCO
Date :
Page :

(1) BST property
(1) lST < Node < rST.
(2) $-1 \le lh - rh \le 1$

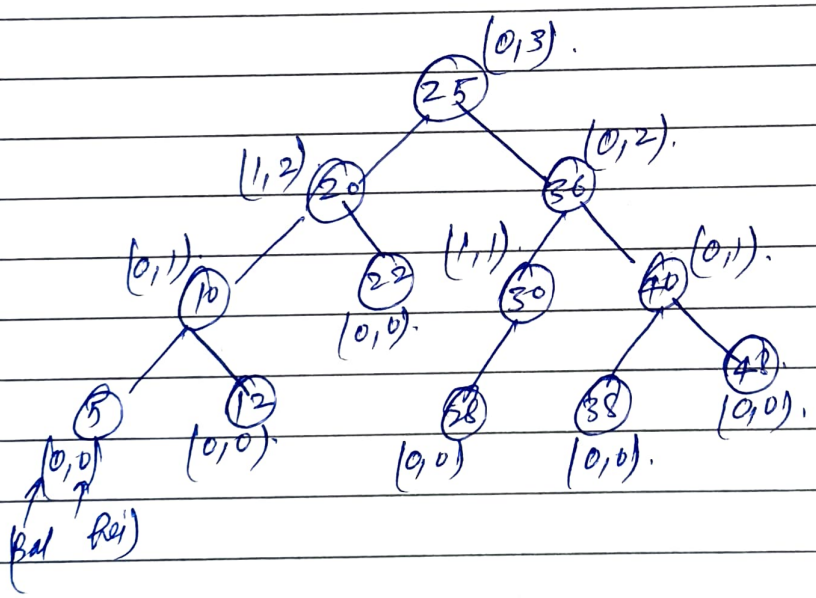
$$|lh - rh| \le 1$$

$(lh - rh) = \{-1, 0, 1\}$.

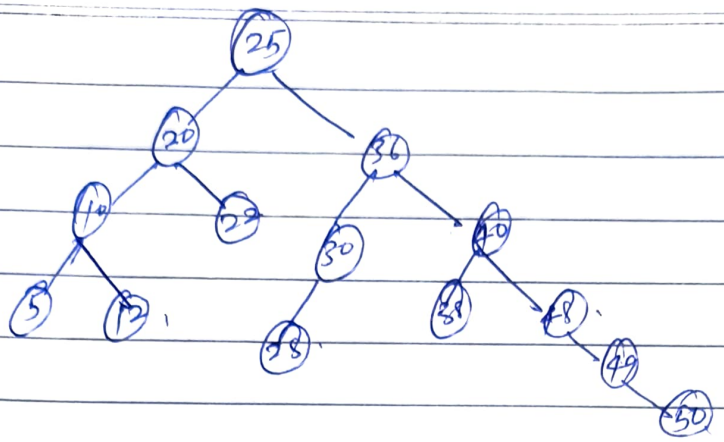
↳ if this property exist ($O(h) \rightarrow O(\log(n))$ होती है)

→ if BST not balanced complexity me जाती है.

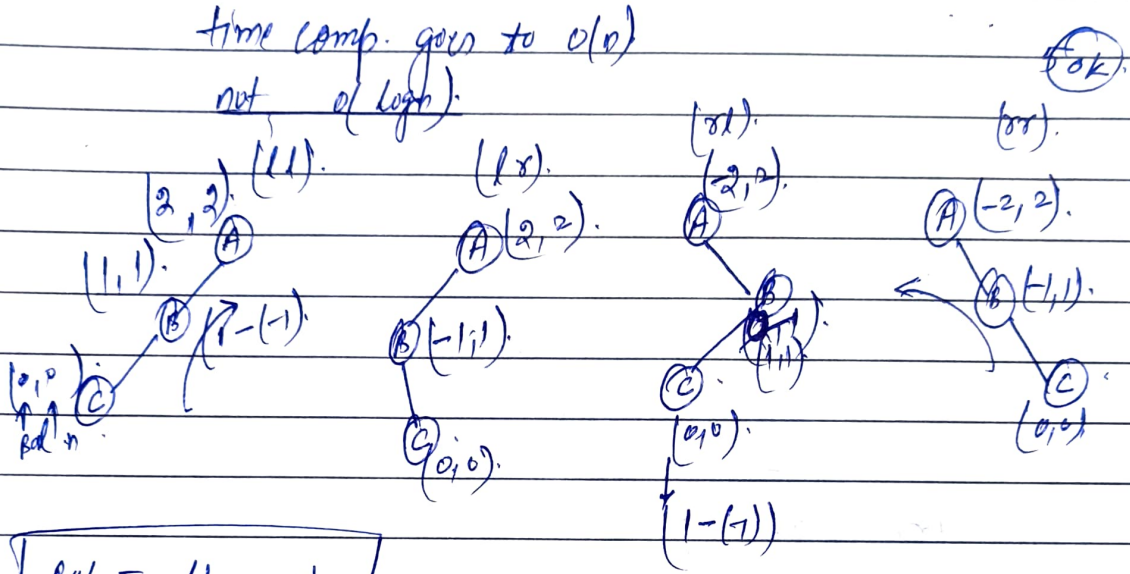
eg.



eg if we add 49, 50, 51, 52 - - - - 5000
all node get add in right most node & comp. goes to.
o(n) not o(logn)
so, we use avl.



time comp. goes to O(n)
not O(logn).



Bal = lh − rh.

A(-1,+1) =
Bal = -1+1
= 0.

Bal = 2 → ll. / lr.

Bal = -2 → rl. / rr.

APCO
Date :
Page :

→ This concept is only useful when we add/ Remove something in Bst and our balancing get distort.

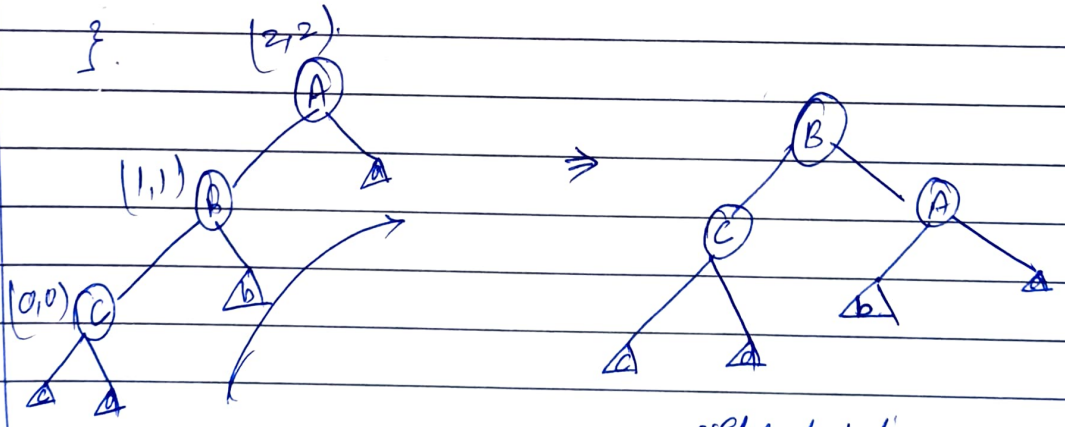
//Code:

```
#include <iostream>.
#include < vector>.

Using namespace std;.

Class Node {.
  Public:
    int data = 0;.
    Node * left = nullptr;.
    Node * right = nullptr;.
    int bal = 0;.
    int height = 0;.

    Node (int data) {.
        this → data = data;.
    };.

void UpdateHeightAndBalance (Node* node) {. // O(1).

    int lh = -1;.
    int rh = -1;.

    if( node → left != nullptr) {.
        lh = n → left → height;.
    }.
```

```
if(node->right != nullptr){
    rh = node->right->height;

node->bal = lh - rh;
node->height = max(lh, rh) + 1;

}
```

③ `Node *rightRotation (Node* A) { // O(1)`

```
Node* B = A->left;
Node BKaRight = B->right;

B->right = A;
A->left = BKaright;

UpdateHeightAndBalance(A);
updateHeightAndBalance(B);
Return B;
}
```



after rotation.

APCO
Date :
Page :

④ Node* leftRotation (Node*A) { //O(1).

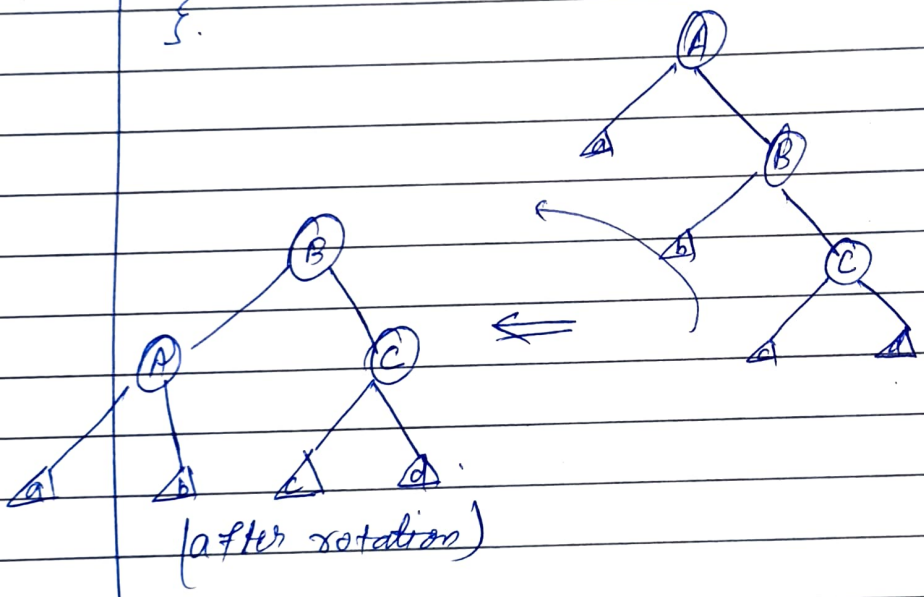
Node* B = A->right;
Node* BKaRightLeft = B->Left;

B->Left = A;
A->right = BKaLeft;

UpdateHeightAndBalance(A);
UpdateHeightAndBalance(B);

return B;

}



(after rotation)

APCO
Date :
Page :

⑤ Node* getRotation(Node* node) { // o(1).

updateHeightAndBalance(node);

if(node->Bal == 2) // ll, lr.

{ if(node->left->Bal == 1) // ll.

{

return rightRotation(node);

}

else{ // lr

node->left = leftRotation(node->left);

return rightRotation(node);

}

else if(node->Bal == -2){ // rr, rl.

{ if(node->right->bal == 1){ // rl.

node->right = rightRotation(node->right);

return leftRotation(node);

}

else{ // rr.

return leftRotation(node);

}

}

return node;

}

APCO
Date :
Page :

(6) display function is simply display of bt.

(7) int maximum(Node* node){.

```
Node* curr = node;
while(curr->right != null){
    curr = curr->right;
}

return curr->data;
}
```

(8) Insert into BST.

is like normal insert fn just call get rotation by passing root as argument at the time of return.

eg

```
root = getRotation(root);
return root;
```

(9) same as delete Node just call getrotation fn at the time of return.

```
root = getRotation(root);
return root;
```

APCO
Date :
Page :

Qu 1382. What :-

we have given a bst and have to balance it.

How.

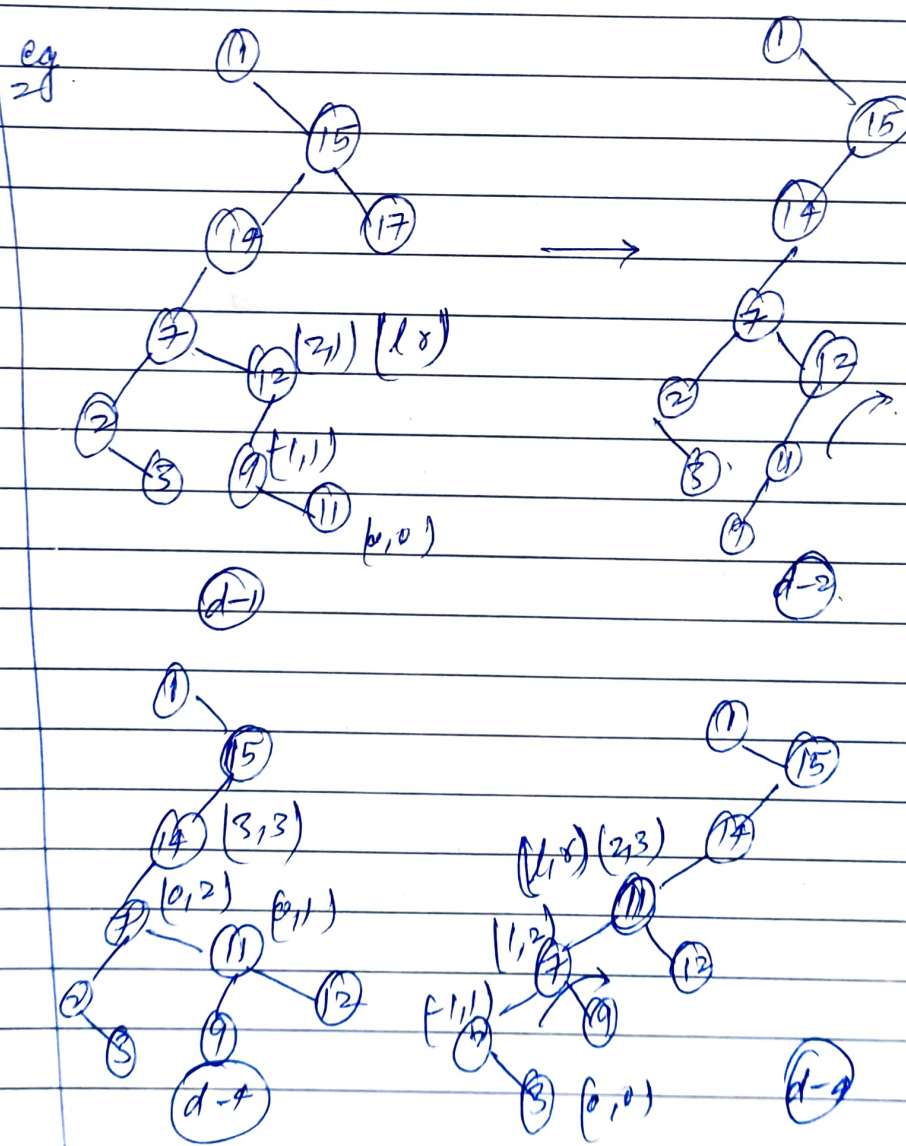
M-1 Using inorder traversal bring element in array and make bst. return node.

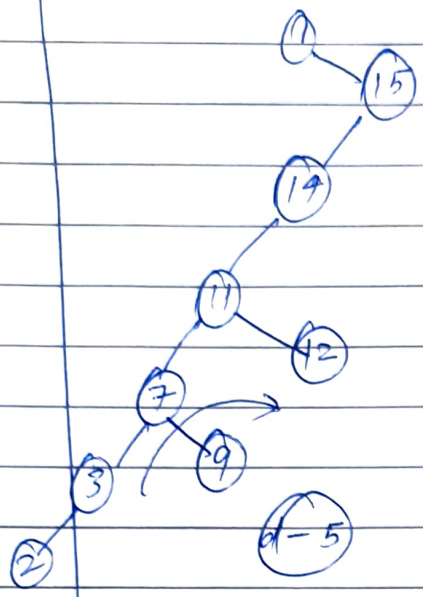
(O(n) Space t: O(n).

M-2

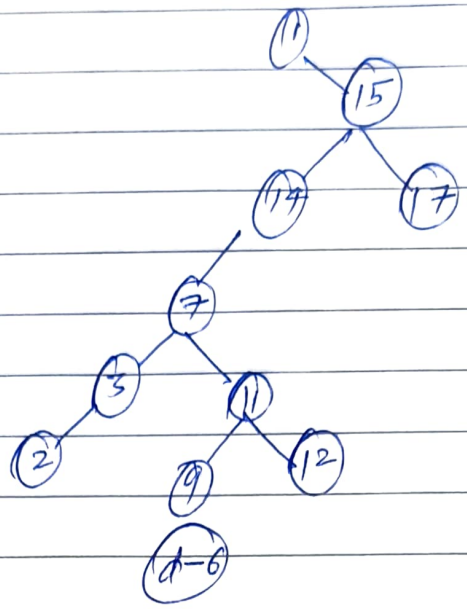
Using Avl Concept

O(1) Space.

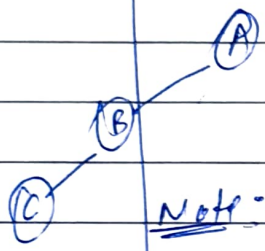




⇒



(get completely balanced tree)



Note :- Here we have to call get rotation on 'A' and 'B' again and again as after rotation height and balanced are altered.

Note 2 :- here we have to check our balanced greater than 1. our balanced get more than two beaz our tree is not balanced.

APCO
Date :
Page :

Lecture - 7
(30/sept/2020).

97/144

Agenda :-

1. Construct B.T from Pre. qn.
2. " " " Post, qn.
3. B.T Canvas. // 968
4. 889 // Pre and Post
5. Tree from pre-order trav (from bst).
6. From post order to bst (H.W).
7. From level order (H.W).

APCO
Date :
Page :

Pg

$T(n) = T(n/2) + T(n/2) + n/2$ ← (while loop searching)

$T(n) = 2T(n/2) + n/2$.

$2T(n/2)$. $2^2T(n/4) + 2(n/4)$.

$2^2T(n/4) = 2^3T(n/8) + 2^2(\frac{n}{2^2})$.

~~Mail Free Mail~~

$T(n) = 2^l T(0) + l(n/2)$.

$n, n/2, \frac{n}{4}, \ldots, 1$ (log n)

$\Rightarrow n = 2^{l-1}$

$\log n = \log 2^{l-1}$

$l = \log_2 n + 1$ ★

$T(n) = 2^{\log_2 n + \log_2 n)} n/2$. ★.

$T(n) = n + \frac{n}{2}(\log_2 n)$. → $\log_2 n$

In worst case (means tree is skew).

$T(n) = T(n-1) + n$.

$T(n-1) = T(n-2) + (n-1)$.

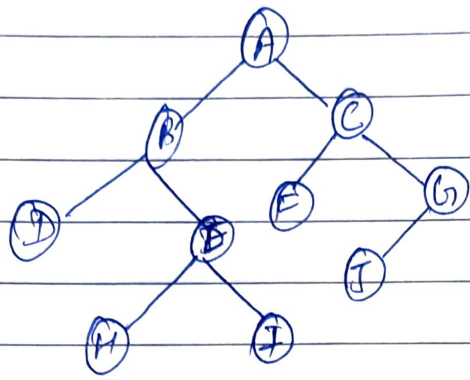
$T(n) = 2 \cdot T(n-1) + n$.

root

n | T

APCO
Date :
Page :

1. Qn Const B.T From pre and In.



LRNR In = D, B, H, E, I, A, F, C, J, G

NLR. Pre = A, B, D, E, H, I, C, F, G, J,
3 4 5 6 7 8 9 10 11 12

Here start index other than '0' eg, 3,4 any.

(3,12)

A

[len = idx - si (Io)]

(main idea)

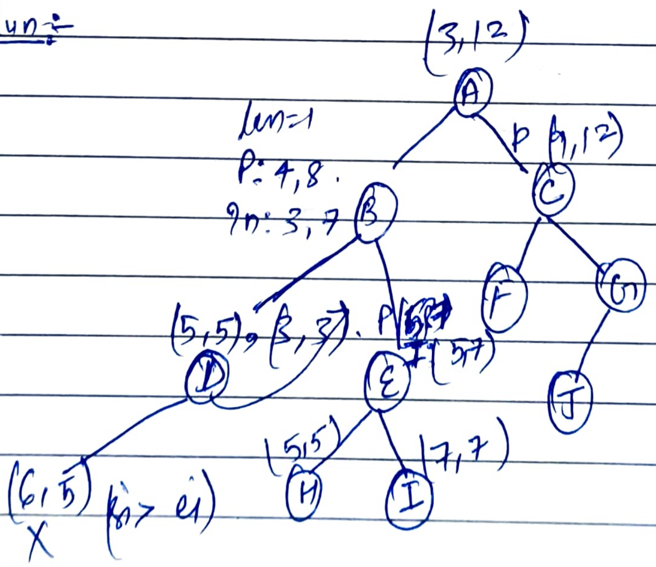
Pre : si+1, si+len

in : si, idx-1

Pre: (si+len+1, ei)

in: (idx+1, ei)

dry Run:-



len = 8-3
= 5

len = idx - isi
= 4-3
= 1

len = 0

len =

APCO
Date :
Page :

// Code :- TN fn(int[] preorder, int psi, int pei, int[] inorder, int isi, int iei) {
if (psi > pei) return null;
d(1) TreeNode node = new TreeNode(preorder[psi]);
int idx = isi;
① — O(n/2) [while(inorder[idx] != preorder[psi]){
idx++;
}
int count = idx - isi; // Count Nodes in lst.
② — T(n/2) → node.left = fn(preorder, psi+1, psi+count, inorder, isi, idx-1);
③ — T(n/2) → node.right = fn(preorder, psi+count+1, pei, inorder, idx+1, iei);
return node;
}

Time complexity :-
In avg case = n log n.
when tree is balanced and node lie in middle of inorder traversal

$T(n) = T(n/2) + T(n/2) + (n/2)$.
$T(n) = 2T(n/2) + n/2$
$2T(n/2) = 2^2 T(n/4) + 2(n/4)$
$4T(n/4) = 2^3 T(n/8) + 2^3 (n/8)$.

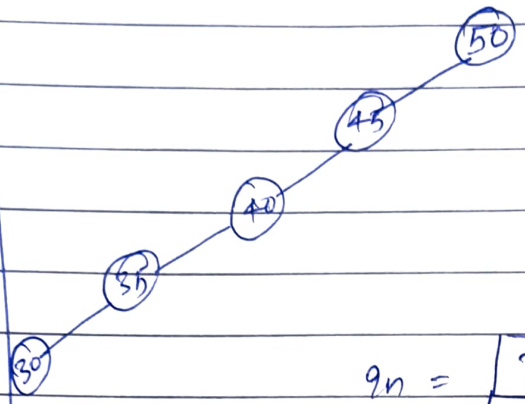
$T(n) = 2^l T(0) + 2^l (n/2 + n/4 + n/8 + \dots)$
$= 2^{\log_2 n} (\log_2 n)$
$= n + n \log_2 n$.
$T(n) = n \log n$. ④

APCO
Date :
Page :

~~Can't be from post and in.~~

C-2 when we have left skewed tree using this method.

T : $O(n^2)$. because



qn =	30	35	40	45	45
Pre =	50	45	40	35	30..

① now we found node at $O(n)$ not $n/2$.

② As no right branch so have to go to left $T(n-1)$ times.

so,

$T(n) = n + T(n-1)$.

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + (n-2)$.

⋮

$T(n) = T(0) + n + (n-1) + (n-2) + \cdots + 1$

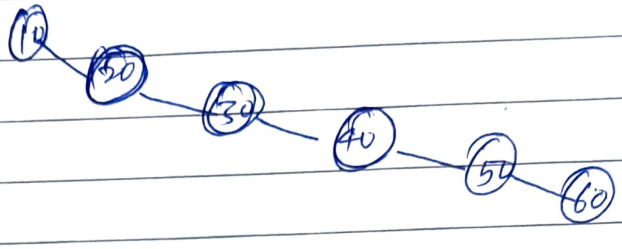
$= \frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$ (*).

$\boxed{T(n) = O(n^2).}$

APCO
Date :
Page :

Q-8 In best case when tree is right skewed.



Pre	10	20	30	40	50	60

In	10	20	30	40	50	60

① Now while ~~if~~ line $O(1)$.

② only have to make call on right side so.

$$T(n) = T(n-1) + O(1)$$
$$T(n-1) = T(n-2) + O(1)$$
$$\vdots \quad - \quad \vdots \quad + O(1).$$
$$\vdots \quad \vdots \quad \vdots$$
$$\vdots \quad T(0) + O(1)$$

$$T(n) = T(0) + n(O(1))$$

$$\boxed{T(n) = O(n).}$$ ⑧

APCO
Date :
Page :

Q4 ~~Const b.t from pre and post~~

M-2 ① put index and node in treemap and retrive $\longrightarrow$ $\log n$ (cost).

② from preorder find node and length.

So combining ① and ②.

$$T(n) = (\log n) \times n$$

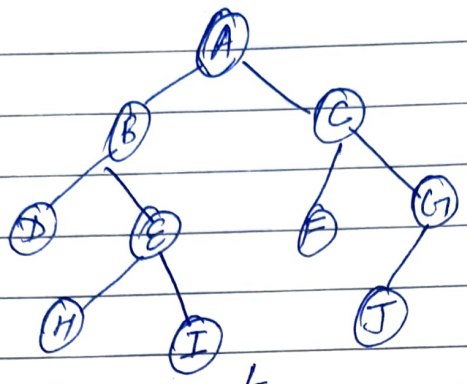
($\log n$: T.Map; n: tree traversal) ⭐

M-3 Using (infinite space).

$\rightarrow$ means array of infinite size as we don't know node value of tree and retrive node index.

APCO
Date :
Page :

2 Qu. Const B.T for Post and IO.

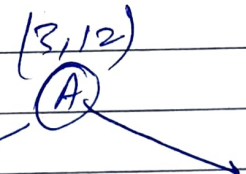


		L				R				N
LRN	Post	D	H	I	E	B	F	J	G	C	A
LNR	In	A	B	D	E	H	I	C	F	G	J
	In	D	B	H	E	I	A	F	C	J	G
		3	4	5	6	7	8	9	10	11	12

idx = 8.

len = 8 − 1 + 1 ⇒ 8 − 3

= 5



post (si, si + count − 1) (idx+1, iei) (si + count, pei)

In (isi, idx − 1). (idx+1, iei)

APCO
Date :
Page :

① only change from prev question is.
compare
while(inorder[idx] != postorder[pei])

② In calling pass given ar guement as prev page.

APCO
Date :
Page :

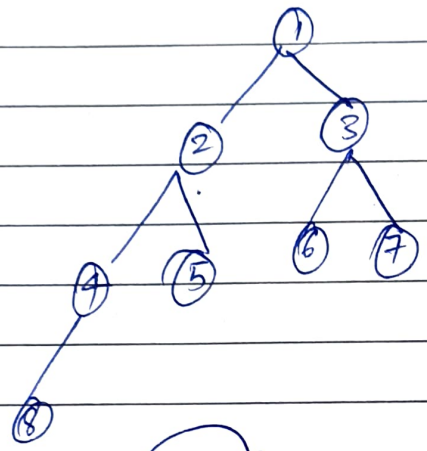
3. Construct tree from preorder and post order.

What ?

Make tree from preorder and postorder traversal.

eg.

How ?



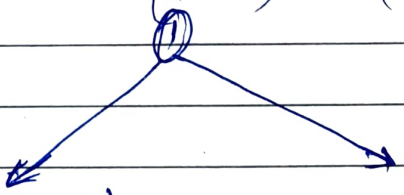
NLR pre

LRN post

1	2	4	8	5	3	6	7
8	4	5	2	6	7	3	1

3 4 5 6 7 8 9 10

(Left child) (Right child)



pre: (psi+1, psi+count) (psi+count+1, pei)

post: (psi, psi+count-1) (psi+count, psei-1)

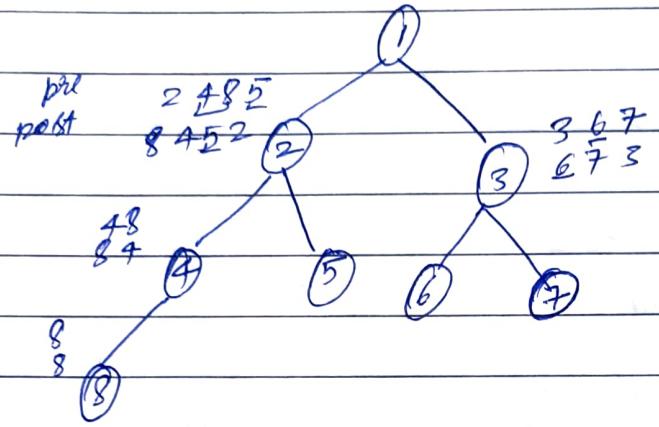
index → post order ka

count = ind - post si

APCO
Date :
Page :

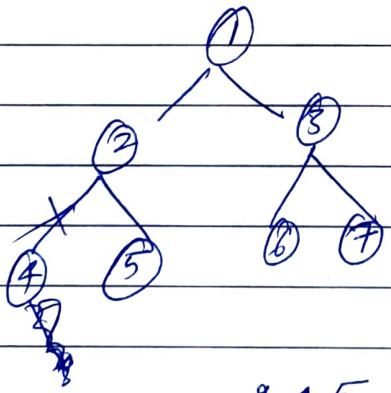
→ search node of next in post order and subtract 8
→ gives count / element present on L.H.S.

dry Run:

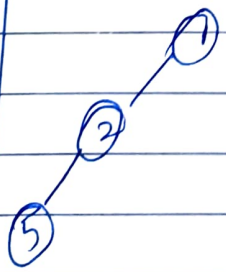


Note ~~If~~ soln in case of vacant left node and valued right node is not possible.
→ because, we will not be able to decide whether it is a left child/ right child.

eg.



→ 2 4 5 we can decide it is ~~left~~
2 5



(while 5 is right child)

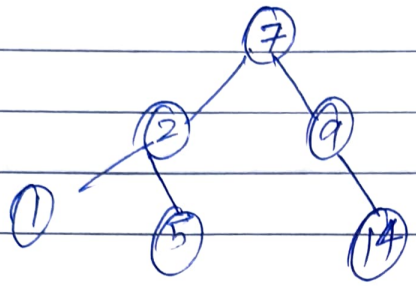
APCO
Date :
Page :

// Code (889)

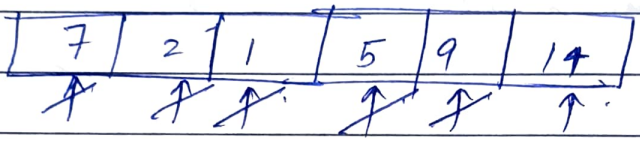
```
Public TreeNode buildTree(int[] postorder, int posi, int poei,
        int[] preorder, int psi, int pei) {

    if(psi > pei) return null;
    if(psi == pei) return new TreeNode(postorder[posi]);

    TreeNode node = new TreeNode(preOrder[psi]);
    int idx = isi;

    while(postorder[idx] != preorder[psi+1]) {
        idx++;
    int count = idx - posi + 1

    node.left = buildTree(postorder, posi, posi+count-1,
                inorder, isi, idx-1);

    node.left = fn(postorder, posi, posi+count-1, preorder,
                psi+1, psi+count);
    node.right = fn(postorder, posi+count, poei-1,
                preorder, psi+count+1, pei);

    return node;

}
```

4. Const BST from preorder.

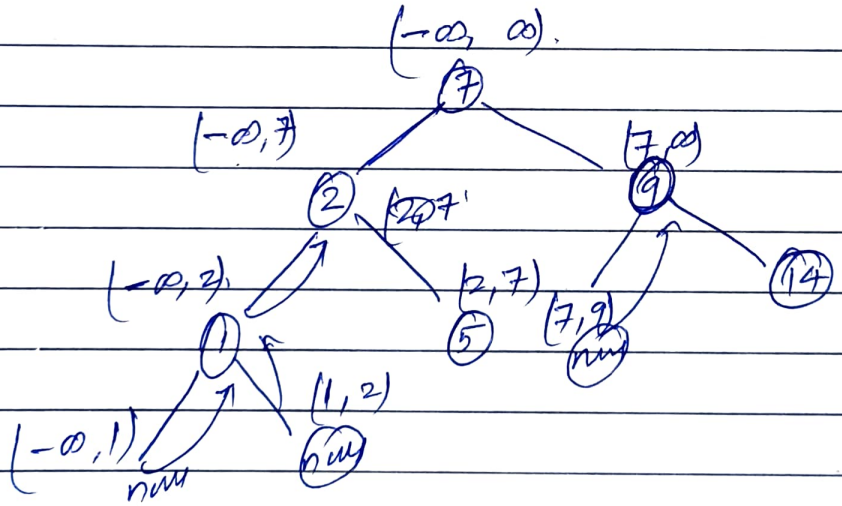
What? Make bst from given pre order array



How



① Set a range and if index value lies within range then make node other wise return (null).



APCO
Date :
Page :

// Code :

Note :- In BST

(left subtree).lst ≤ (node.data) < rst (right subtree).

int idx = 0;

TreeNode* buildTree(vector<int>& arr, int lrange, int rrange){

if(idx >= arr.size() || arr[idx] < lrange || arr[idx] > rrange) return nullptr;

int data = arr[idx++];
TreeNode *node = new TreeNode(data);

node->left = buildTree(arr, lrange, data);
node->right = buildTree(arr, ~~rrange, data~~ data, rrange);

return node;

};

Note :-
if we want to remove int idx then maintain arr of size '1' and ~~put~~ do '1' there.

```
fn( new int idx ){
    int data = arr[ idx[0]++ ];
```

access element at that index.

APCO
Date :
Page :

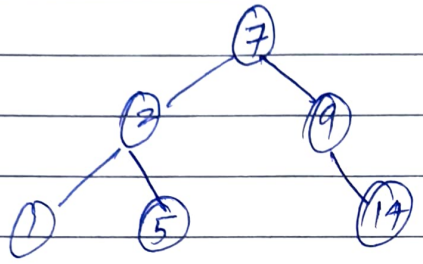
6. Q. Make BST from post order traversal

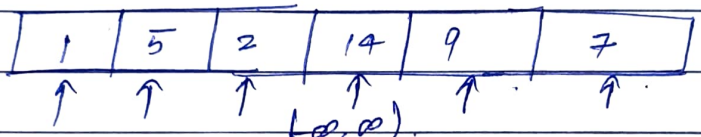
How. Qn is same like previous.

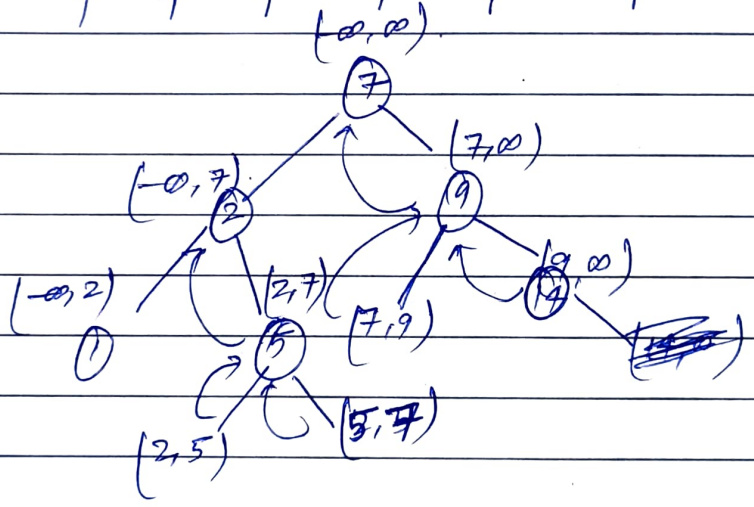
diff

① idx = arr.length-1 and dec idx--,

② 오른쪽 right 먼저 call 하고나서 then left 를 call.







APCO
Date :
Page :

7 Qn BST From level order.

APCO
Date :
Page :

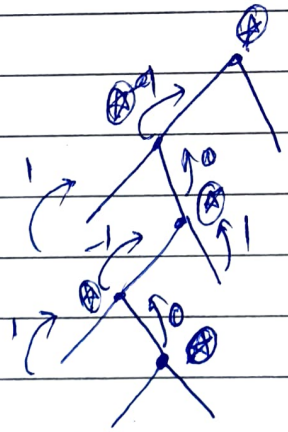
Que. B.T Canvas.

What :- have to plot camera such that all nodes become visible.
1 camera can cover parent and both child.

How :-
Here we will return three things depending up on situation.
① eg camera ≠ = 0
② camera need. = 1
③ don't need camera = -1

Note : We'll never put camera on leaf node.

eg.



APCO
Date :
Page :

Teaches how to use multiple return.

-1 = I'm covered by camera.
0 = I'm Camera.
1 = I need Camera.

// Code

```
int camera = 0;
Public int minCameraCover(TreeNode root){
    if(minCamCover(root) == 1) camera++;
    return camera;
}
```

→ because if root get req of camera then no of to fulfill
so handle this case specially.

```
Public int minCamCover(TreeNode root){
    if(root == null) return -1; // need cam

    int lres = minCamCover(root.left);
    int rres = minCamCover(root.right);

    if(lres == 1 || rres == 1){
        camera++;
        return 0;
    }
    if(lres == 0 || rres == 0){
        return -1;
    }
    return 1;
}
```

10, 20,

APCO
Date :
Page :

[Lecture-8] 2 Oct 2020.

Agenda :-

1. B.T to dll. (gfg)
2. B.T to cdll. (gfg).
3. Morris traversal (Inorder) -]
4. Pre order "
5. ~~post~~ "] नहीं होगा
6. traversal (using iteration pre, post, in).
7. Zig zag traversal (11.N).
8. Kth smallest element in bst // 230.
9. ~~655 // TW.~~

S	Rec	logn	o(1)
T	o(n)	o(n)	o(n) → Morris traversal IO

8. Two Sum IV - input in BST. // 653
9. check if pre, in and post traversal are of same tree.
10. Find tree diameter using pre and ~~post~~ in. With out making tree.
11. Print all k sum path in a b.t

APCO
Date :
Page :

1. Qu. Binary tree to dll.

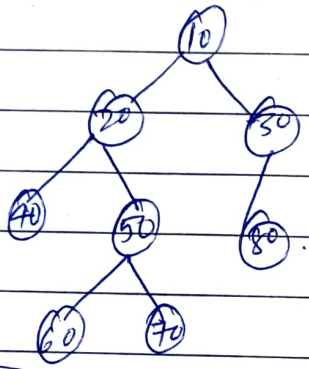
what Convert a given b.t to dll

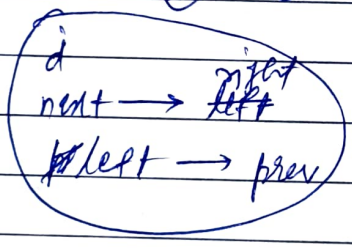
How.?

M-1 ① Do inorder traversal of Binary tree and bring all the node in an array.

② when you get array of nodes just link them. and return head node.

eg.





change करें

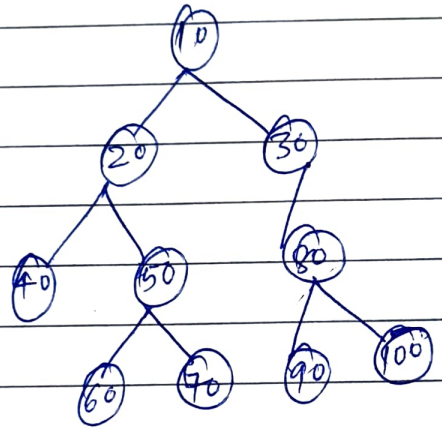
APCO
Date :
Page :

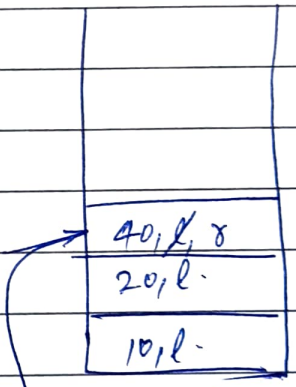
M-2 Using prev pointer and with out extra space.

How :- Traverse in inorder and update prev in inorder manner.

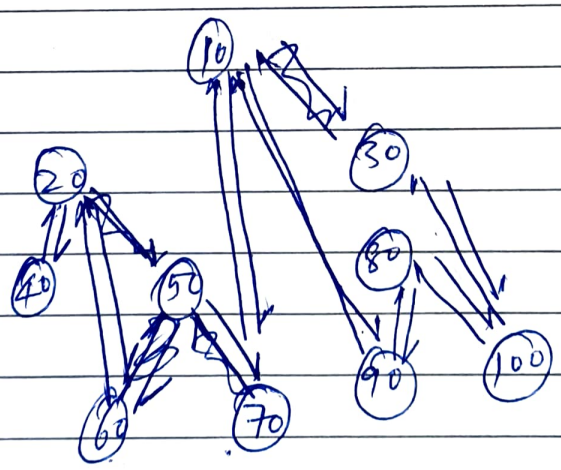
```
prev -> next = curr
curr -> prev = prev.
```

eg





(right की call लगाने से just पहले prev and curr पे pointing होती है then call लगती है)



we will get :-

APCO
Date :
Page :

// Code.

(1) Fn

```
Node prevDLL = null;
Node headDLL = null;

void btoDLL-(Node node){
    if(node == null) return;

    btoDLL-(node.left);
    if(prevDLL == null) headDLL = node;
    else{
        prevDLL.right = node;
        node.left = prevDLL;
    }
    prevDLL = node;
    btoDLL-(node.right);

}
```

// Calling fn of dll and return head node.

(2) Fn

```
Node btoDLL(Node root){
    btoDLL-(Node root);
    return headDLL;
}
```

APCO
Date :
Page :

2. Qu Convert bst to circular dll

HOW :- ① Here app. remain same as of prev qn. and methods are also valid.

② just return करने से पहले

head.left = prev.
prev.right = head }.

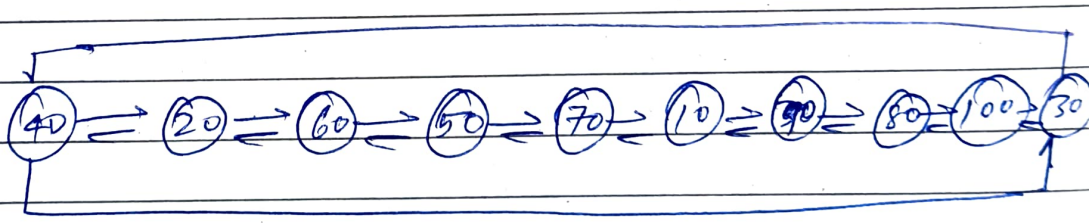
① Fn ① as of prev.

② Fn ②

Public Node bTreeToCList (Node root){.
bToDLL (root);.
prevDLL.right = headDLL;.
headDLL.left = prevDLL;.

return head ;.

}.



(*) (1) Threaded B.T (I will do during Revision time).
(2) Uses concept of threading.

APCO
Date :
Page :

Morris traversal very Important approach.

Morris traversal uses concept of threading.
it uses two condition. ~~curr = null तक ही~~.

(1) अगर left node का right most null है
   (a) Create thread b/w rmost and curr.
   (b) अगर rmost.right != null है then break thread and print curr का data.

(2) If curr का left null है then print data and move curr to curr.right.

~~How~~ benefit O(1) space and O(n) time.
while rec. O(n) space and O(n) time.

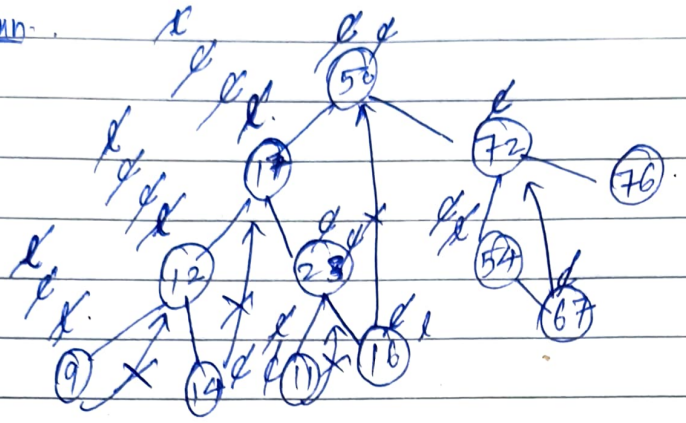
Note this traversal can be use in qn we are asked to find solution in O(1) space complexity

eg. Kth smallest element (5) qn.

APCO
Date :
Page :

Inorder

dry Run.



Inorder traversal

9 12 14 17 11 23 16 50 54 67 72 76.

// Code.

①
```
Public static Node rightMostNode(Node node, Node curr){
    while(node.right != null && node.right != curr){
        node = node.right;
    }
    return node;
}
```

②
```
Public static void Morristraversal(Node node){
    Node curr = node;
    while(curr != null){
        Node leftNode = curr.left;
        if(leftNode == null){ // left null
            sysoln(curr.data);
            curr = curr.right;
        }
```

```
else {
    Node rmost = rightMostNode(leftNode, curr);
    if(rmost.right == ~~curr~~ null) { // thread creation
        rmost.right = curr;
        curr = curr.left;
    } else {          // thread break;
        rmost.right = null;
        System(curr.data);
        curr = curr.right;
    }
}
}
} // while का
} // fn का
```

APCO
Date :
Page :

④ Pre order - Morris traversal:

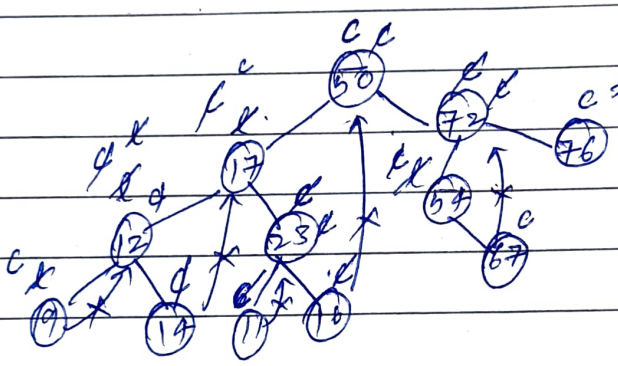
In How?

Here we will print data at two cases

① thread creation time. ✓

② when node ka left null ho ✓.

Remaining all the other thing remain same.



c = curr.next

loop break

∵ curr. ~~next~~ = null.

Pre order.

50 17 12 9 14 23 11 16 72 54 67 76.

// changed code from prev.

```
else {
    Node rmost = rightMostNode(leftNode, curr);
    if(rmost.right == null) { // thread creation.
        Sysout(curr.data);
    }
    rmost.right = curr;
    curr = curr.left;
}
else { // thread break.
```

```
        rmost.right = null;
        curr = curr.right;
    }
}  // outer else
```

// Note :-

Post order morris traversal is not possible.

APCO
Date :
Page :

(5) Or Traversal Using iteration :

Using iteration we maintain stack which work as recursive stack.
→ do using ustack whenever said to do recursive go using iteration.

How ? make a pair class containing
① node.
② left done. } boolean
③ right done. }
④ self done. }

Mark them true accordingly as per need. which kind of traversal we want.

(2) ① For pre order. working go as.
① self & print data.
② left
③ right
④ remove.

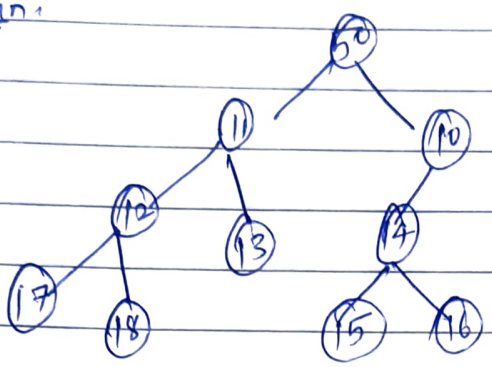
② for post order.
① ld ② rd ③ sd. & print ④ pop

③ for inorder
① ld ② sd & print ③ rd ④ remove. from stack.

→ Always write example of qn on copy and think about → to writing code take hardly 5 min.

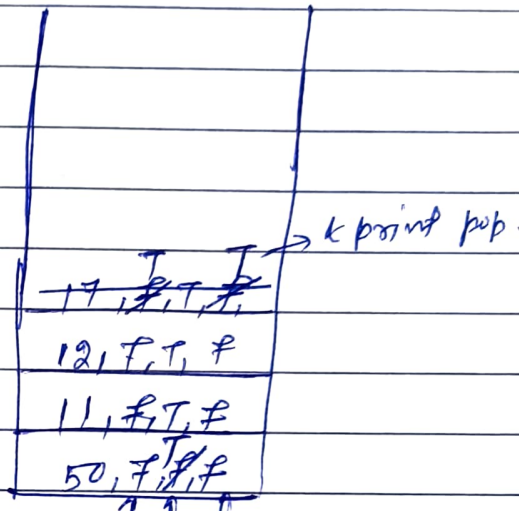
APCO
Date :
Page :

dry Run:



one node is visited at most 4 times.

① S | L | R
② Left
③ right
④ pop from stack.



Post order:

17, 18, 12, 13, 11, 15, 16, 14, 10, 50.

// Code -

① Pair class.

```
Ps class tpair {
    Node node = null;
    boolean selfdone = false;
    boolean leftdone = false;
    boolean rightdone = false;

tpair ( Node all 4 ) {
    this.selfdone = selfdone;
    this.leftdon  = leftdon;
    this.rightdon = rightdone;
}
```

(all three 4 as args)

APCO
Date :
Page :

Pre order Iterative traversal

② Fn :

```
Public static void preTraversal (Node node) {

    Stack <tPair> st = new Stack <> ();
    st.push( new tPair (node, false, false, false));

    while (st.size() != 0) {
        tPair rp = st.peek();

        if (!rp.selfdone) {
            rp.selfdone = true;
            sysoln( rp.node.data + " ");
        } else if (!rp.leftdone) {
            rp.leftdone = true;
            if (rp.node.left != null)
                st.push( new tPair( rp.node.left, false,
                        false, false));
        }

        else if (!rp.rightdone) {
            rp.rightdone = true;
            if (rp.node.right != null) {
                st.push( new tPair( rp.node.right, f, f, f));
            }
        }

        else {
            st.pop();
        }
    } // while
} // fn
```

APCO
Date :
Page :

(Q) for 9n and post
just ऊपर - नीचे कर दो if elsif elseif else
वाली condition को.

APCO
Date :
Page :

Qu ⑦ k^{th} Smallest Element in BST // 230

① o(n) space ⟶ means we can store all n nodes.
② logn space ⟶ root → leaf तक के node को store कर सकते हैं.
③ o(1) means only one node is space allowed है.

3 Method

	Rec.	Iteration	Morris
S	o(n).	o(logn).	o(1).
T.	o(n)	o(n).	o(n).

M-1 Using iteration.

S: o(log n)
T: o(n).

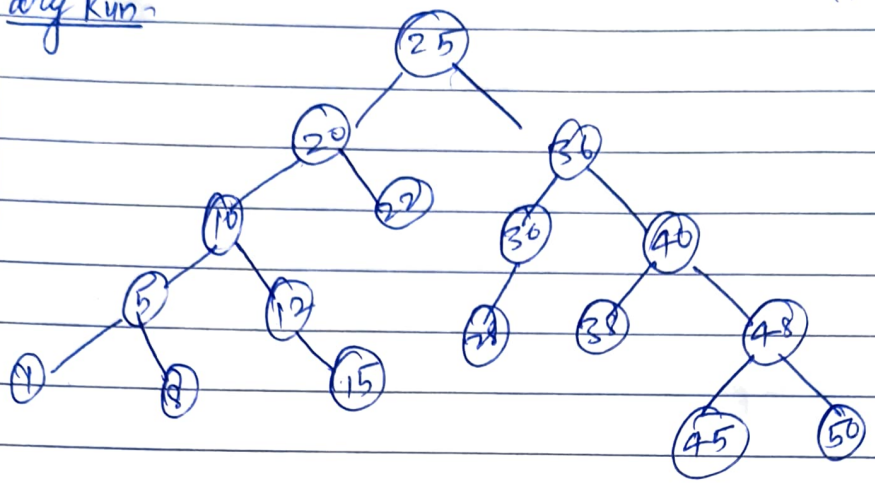
How? ① Stack में put element तब तक उसका left null न हो जाए

② pop करो and उसके right को डालो कि वो खुद के अपने child को stack में जाए वे तब तक null नहीं होता left

→ This solution is just simulation of stack #pair. that is quite easy than normal stack of node only

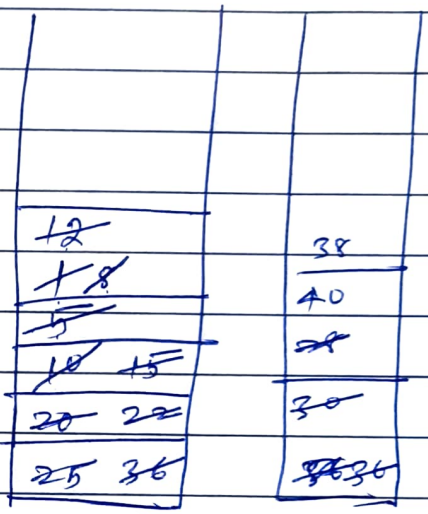
→ using only node of stack we need 2 stack diff, will so leave it

dry Run:



count = ~~1 2 3 4 5 6 7 8 9 10 11 12~~

36 is the 12th element so return '36'.



① // Code:

```
void addAllLeft(TreeNode* node, stack<TN*>& st){
    while(node != nullptr){
        st.push(node);
        node = node->left;
    }
}

int KthSmallest(TreeNode* root, int k){
    stack<TreeNode*> st;
    addALLLeft(root, st);
```

```
while(--k>0){
    TreeNode *rn = st.top();
    st.pop();
    addAllLeft(rn->right, st);
}
return st.top()->val;
```

M-2 Using Inorder morris traversal

T : O(n).
S : O(1).

How :- वहाँ जो print है वहाँ put k--
and if(--k==0){
    break / return कर दो

APCO
Date :
Page :

// code

diff from morris traversal at

~~if(left~~

fn(TreeNode node, int k) {

if(leftNode == null){ //
if(--k == 0) break;
curr = curr.right;
}
else{
if ( ) // thread create.
same.

else{ // thread break.
rmost.right = null;
if(--k == 0) break;

curr = curr.right;
}

return curr.val;

APCO
Date :
Page :

M-3 Do using Recursion.

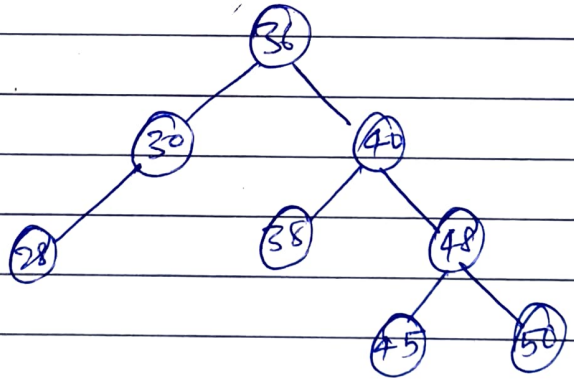
APCO
Date :
Page :

Q. Qu ~~Two Sum - IV // 653~~ Qn.
check if pre, in and post traversal are of same tree

What given pre, post and ino traversal array tell whether they are of same tree / not?

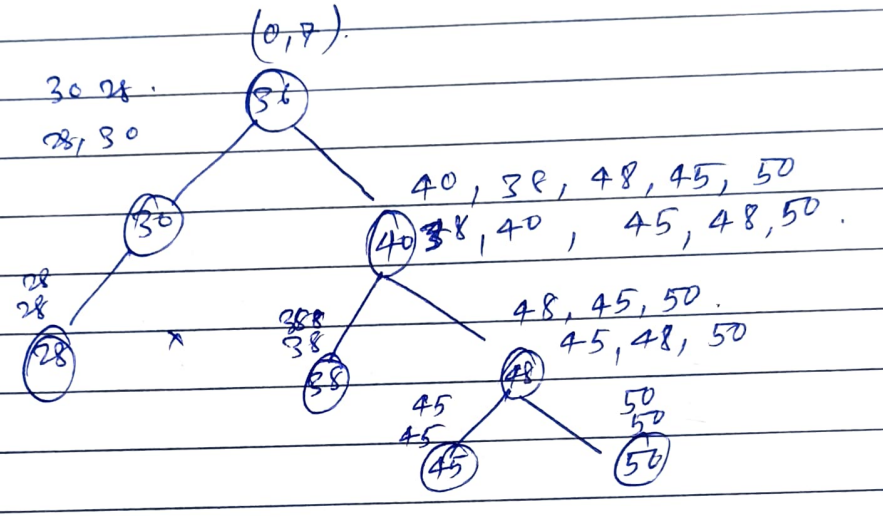
How.? ① Here we'll usame same method as of Create of tree from pre and in ~~and~~.
② when reach at leaf node we'll match wheather leaf node is equal to post or not
③ if not match or element of pre o not found in in return false.

dry Run.



APCO
Date :
Page :

	↓							
Pre	36	30	28	40	38	48	45	50
in	28	30	36	38	40	45	48	50
post	28	30	38	45	50	48	40	36
	0	1	2	3	4	5	6	7



Note · Here we'll not create node just do recursion, and check condition (if fail)
→ not creating not becoz take space (to optimize it)

APCO
Date :
Page :

// Code -

```
int idx = 0;
Public boolean checkTree (int [] pre, int psi, int pei, int[] in,
        int isi, int iei, int post[]) {

    if( psi > pei ) return true;
    int idx = isi;
    boolean flag = false;
    while ( idx < iei ) {
        if( in [idx] == pre [psi] ) {
            flag = true;
            break;
        }
        idx++;
    }
    if ( ! flag ) return false;
    int count = idx - isi; // Count of node in left tree.

    if( fn( pre, psi+1, psi+count, in, isi, idx-1, post)
        return false;
    if( fn( pre, psi+count+1, pei, in, idx+1, iei, post)
        return false;

    if( pre [psi] != post [idx++]) return false;

    return true;

}
```

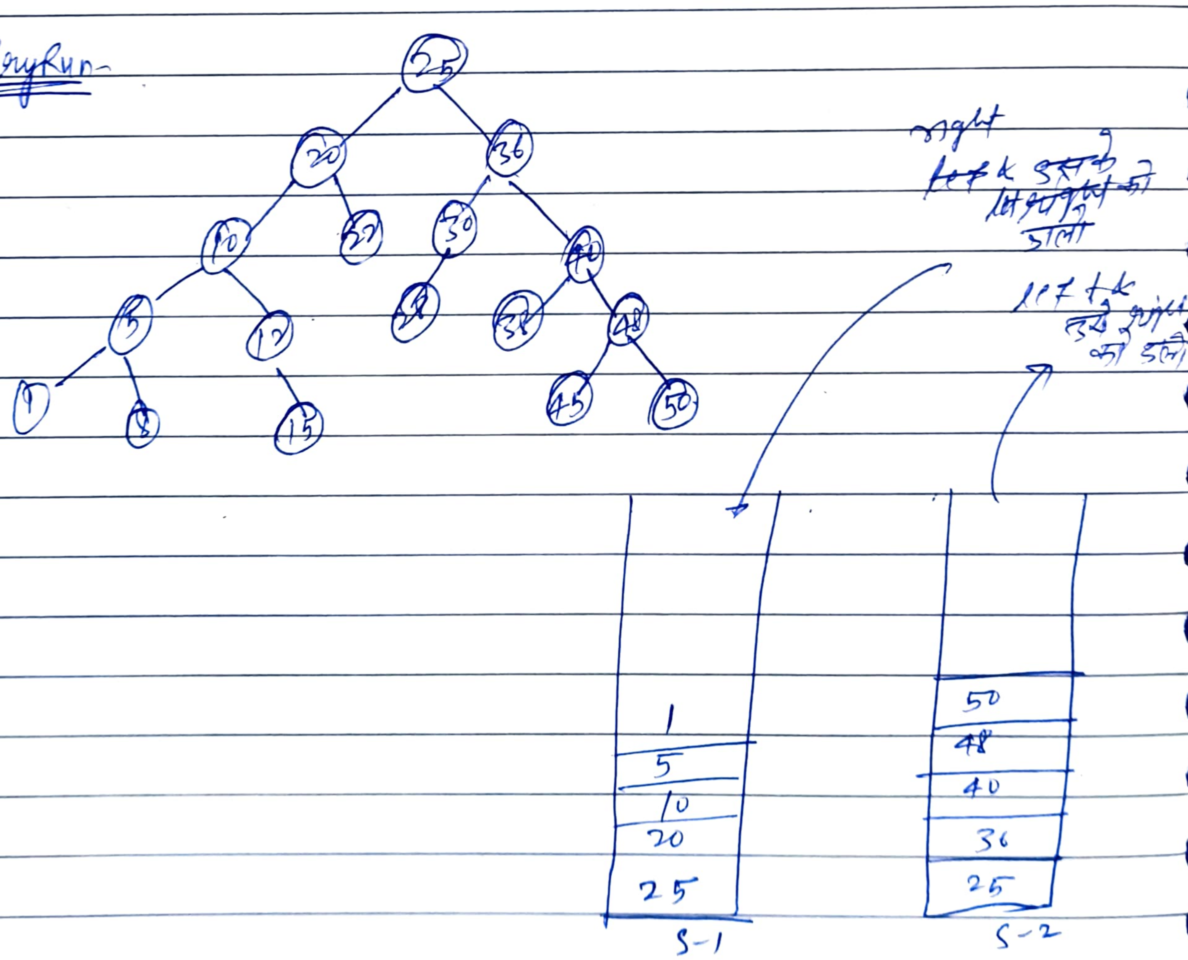
8 Que. Two sum – IV // 653 leet code. (do code by self)

~~ek~~ what check wheath given target ~~element~~ is ~~elet and~~ of sum of two BST element or not.

How?

① Maintain two stack.
②. S-1 मे left elemt and S-2 मे के right elem.
③ if sum > tare then S-2 से pop करो else S-1

dryRun-



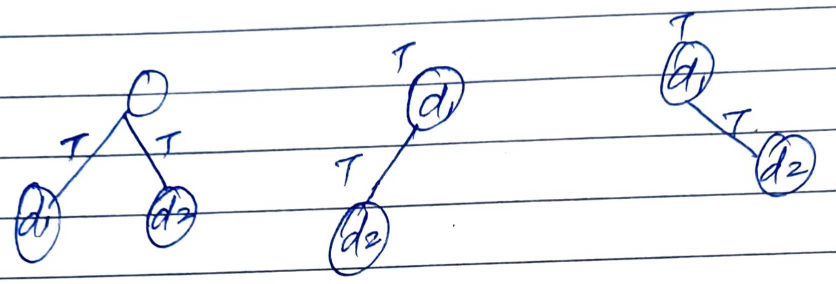
Imp

LCA in O(1) space complexity (B.T).

How :-

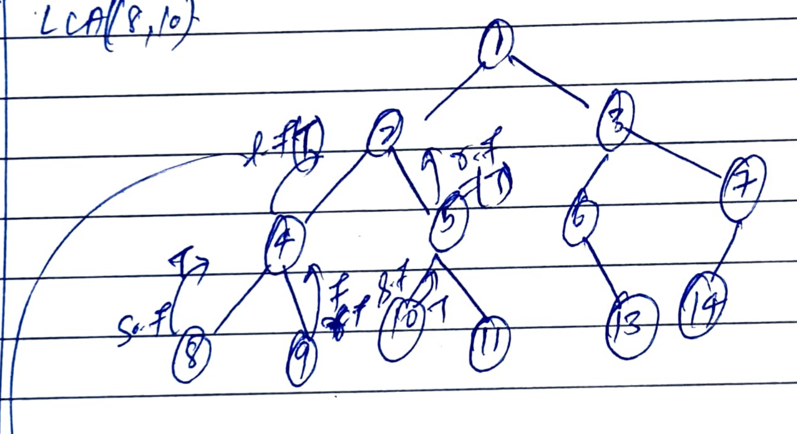
Our LCA can exist in 3 case.

C-1



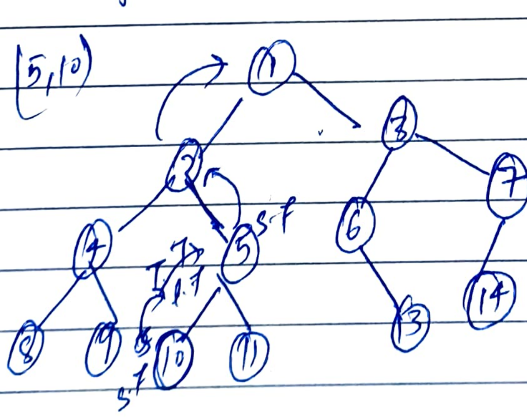
Here we'll have 3 variable:

C-1 LCA(8,10)



if get found and set node = 2.

C-2 (5,10)



C-3 similar for (5,11)

C-4 (11,110) (never got lca)

APCO
Date :
Page :

```
static Node LCA = null;

Public static boolean LCA_Better( Node node, int d1, int d2){
    boolean sf = false;
    if( node.data == d1 || node.data == d2)
        sf = true;

    boolean lf = LCA_Better (node.left, d1, d2);
    if(LCA != null) return true;

    boolean rf = LCA_Better(node.right, d1, d2);
    if(LCA != null) return true;

    if( (lf && rf) || (lf && sf) || (rf && sf))
        LCA = node;

    return lf || rf || sf;

}
```

javap java.util.Stack → give all fn.

[only BST and BT]

~~APCO~~ ~~Inf~~

Date: / /

lecture-9 (Not Imp).

(generic Tree) 4/oct/2020

[It is part of graph and not asked].

Agenda.

1. Construct
   - → display.
   - → Size.
   - → Height
   - → Find.
2. Root to Node path (of Node).
3. Find LCA.
4. Min ~~stuff~~ no. of switches. = Edge Dist(node 1) + ED(n2) − 2 * LCA [H.W].
5. Find LCA ~~in~~ without using AL. [H.W].
6. check tree is foldable/not.
7. Check mirror imag.
8. Linearize the tree t: O(n) S: O(1) + rec.

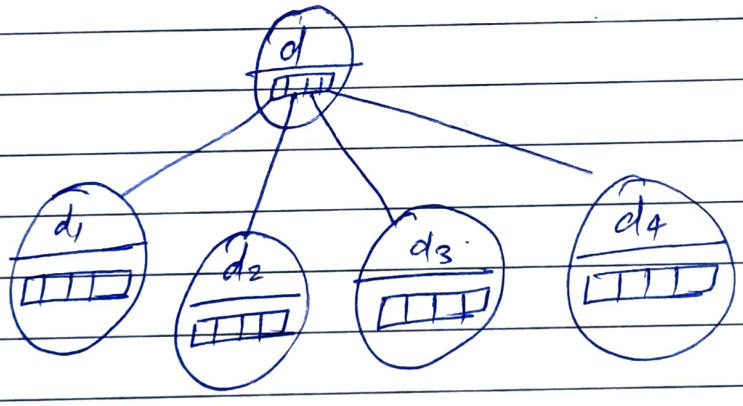
APCO
Date: / /

if you think

soln	(optimization)
s : O(n)	O(n).
↓	O(1).
t : O(n)	

→ Always tell solution is higher to complexity to lower
→ and do that that sol which you are 100% confident in online round
→ which satisfy constraints.

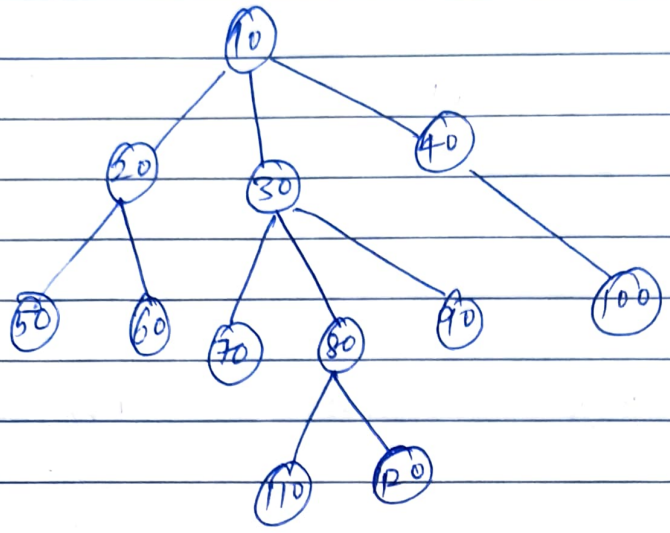
① Construct tree



Node will have
① data
② Arraylist of Node.

serialize form of tree = representation in array form of tree

APCO
Date: / /



Serialization.

here '-1' represent no more children of that node.

Preorder traversal of tree :-

10	20	50	-1	60	-1	-1	30	70	-1	80	110	-1	120	-1	-1	90

-1	-1	40	100	-1	-1	-1

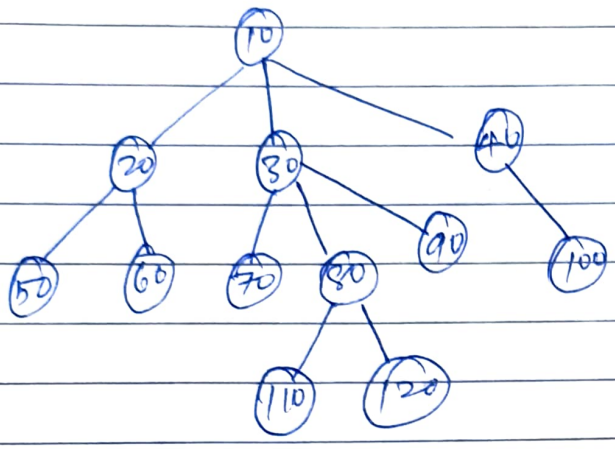
Dry Run.

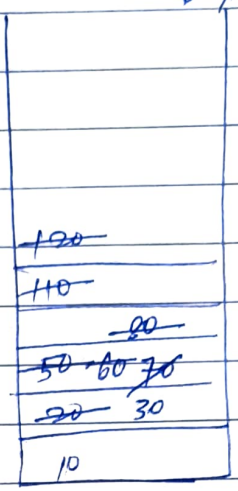
How ? ① Take a stack and put element till you get '-1' and

② → pop and add element to top of stack.

APCO

Date:





Analysis - ① top of stack पर parent होगा ही
② '-1' मिलने पर attach popped element to top of stack.

// Construct

①

```
Public static class Node {
        int data = 0;
        AL<Node> Childs;

        Node (int data) {
            this.data = data;
            childs = new AL<>();
        }

    }
```

APCO
Date: / /

① Construction fn.

```
Public static Node ConstructTree(int[] arr){
    Stack<Node> st = new Stack<>();
    for(int i=0; i<arr.length-1; i++){
        int ele = arr[i];
        if(ele != -1) st.push(new Node(ele));
        else{
            Node node = st.pop();
            st.peek().childs.add(node);
        }
    }
    return st.pop();
}
```

here i go from 0 — n-1 and return node at last

② Public static void display(Node node){

```
StringBuilder sb = new StringBuilder();
sb.append(node.data + "->");

for(Node child : node.childs)
    sb.append(child.data + " ");

System.out(sb);
for(Node child : node.childs)
    display(child);

}
```

APCO
Date: / /

③ height

```
// in terms of edge
Public static int height(Node node) { //
    int h = -1;    // for node h = 0

    for(Node child: node.childs)
        h = Math.max(h, height(child));

    return h+1;
}
```

④ Size().

```
Ps int size(Node node){
    int s = 0;
    for(Node child: node.childs)
        s += size(child);

    return s+1;
}
```

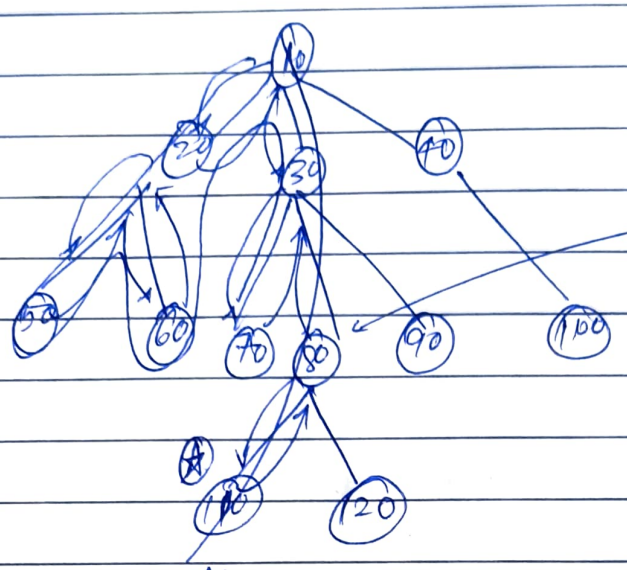
End.

⑤

```
Ps boolean find(Node node, int data) {
    if(node.data == data)
        return true;

    for(Node child: childs){
        if(find(child, data))
            return true;

    return false;
}
```

APCO
Date: / /

(2) Q4 Find Root to Node path.



path.

o/p : 10, 30, 80, 110.

f = f || T.

(if so have more child then loop go for all child but for default call)

// code.

actually it is node to Root (as our name)

```
P S boolean rootToNodePath (Node node, int data,
        AL<Node> list ) {
    if( node.data == data ){
        list.add( node );
        return true;
    }

    boolean res = false;

    for(Node child : node.childs) {
        res = res || rootToNodePath(child,
                data, list);
    }
```

// Note one res set to true then fn don't get called further.

APCO
Date: / /

if (res) list.add(node);.

return res;.

}.

3. LCA

approach / How.

take two AL. and iterate from end till no do not get diff value;.

eg. l_1 = | 10 | 30 | 50 | 110 | 80 | 30 | 10 |

l_2 = | 10 | 30 | 70 | 30 | 10 |

// code.

```
Public static Node LCA(Node node, int d1, int d2){
    AL<Node> l1 = new AL<>();
    AL<Node> l2 = new AL<>();

    rootToNodePath(node, d1, l1);
    rootToNodePath(node, d2, l2);

    Node LCA = null;
    int i = l1.size()-1;
    int j = l2.size()-1;
```

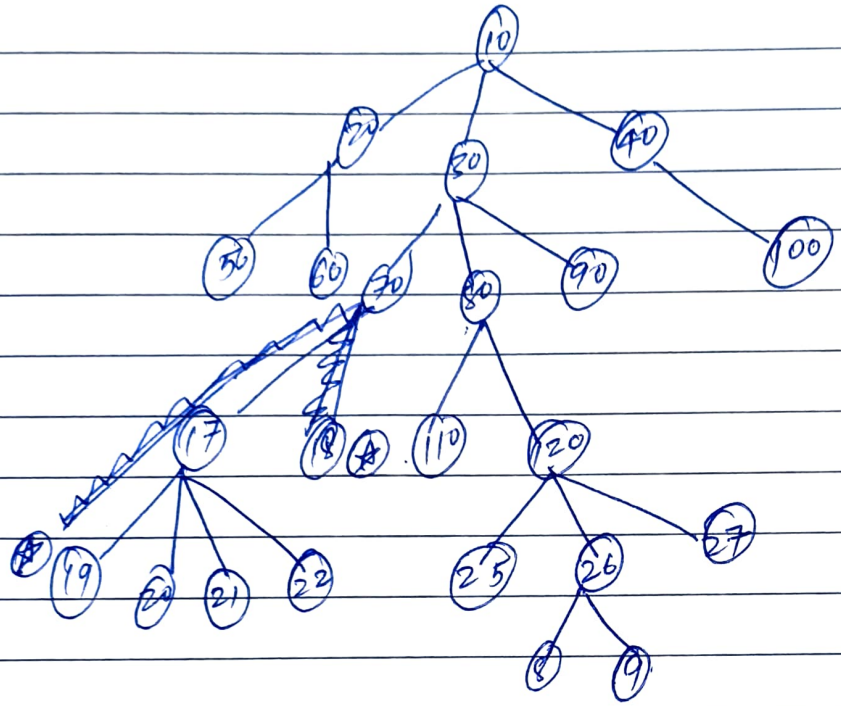
APCO
Date: / /

```
while(i>=0 && j>=0){
    if(l1.get(i) != l2.get(j)) break;

    LCA = l1.get(i);
    i--;
    j--;
}
return LCA;
}
```

APCO
Date: / /

4.ou Min no. of switches required.

What ? Count switches b/w two given nodes.



Suppose have to count switch b/w. 19 and 18

So.

How :- solution is same as lca just maintain count in while loop and.

return list1.size() + l(size2) - 2(lca count - 1)

=

APCO
Date: / /

4. P-2

what How many backward and how many fwd switch Required.

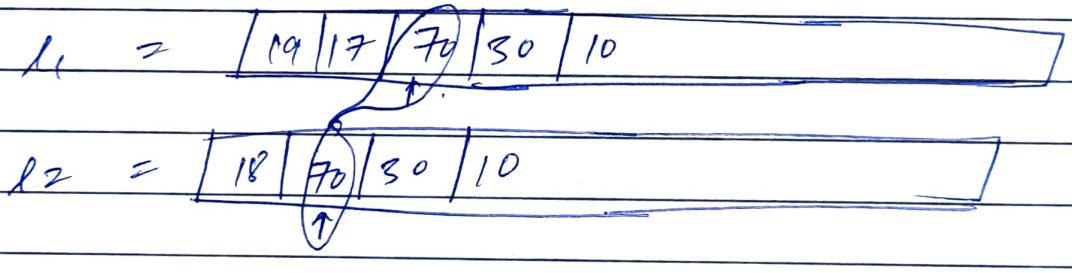
How ?

suppose we have to move from d_1 to d_2.

then:

bck count = l1.size − lca count − 1 − 1

fwd count = l2.size − 1 − lca count − 1



APCO
Date: / /

5. Find LCA without using DL. & O(1) Space.

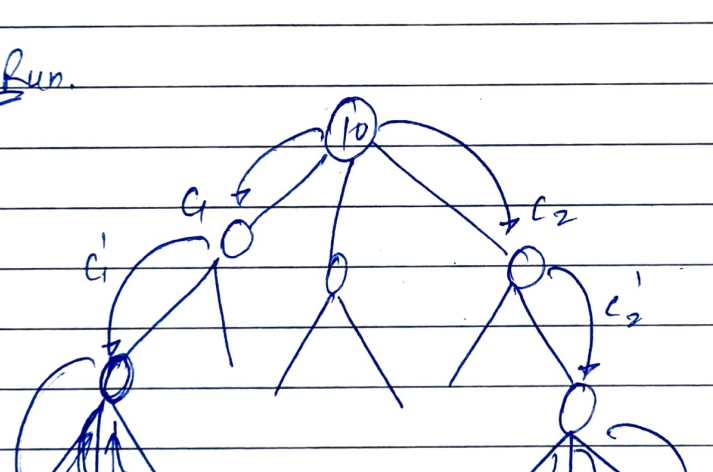
APCO
Date: / /

6.Qu Check tree is foldable or Not

What ? In foldable we only have to check Structure. not data.

How : At any node we pass leftmost and rightmost child and check their size
if size != then return false.
else true;

dry Run.



fC_2
$f(C_1', C_2')$
$f(C_1, C_2)$
$f(n, n)$

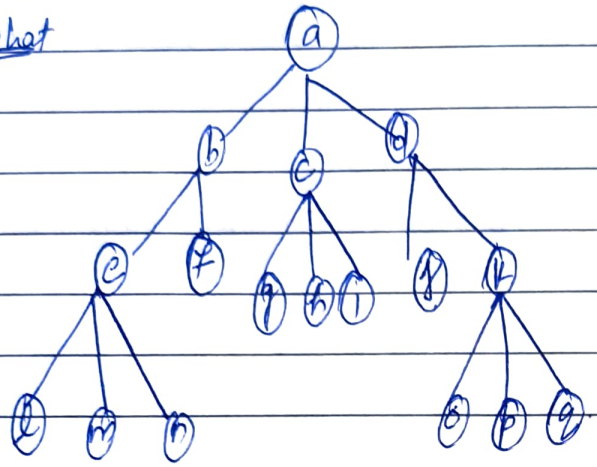
APCO
Date: / /

7.(a) Check Mirror Image of gt.

APCO
Date: / /

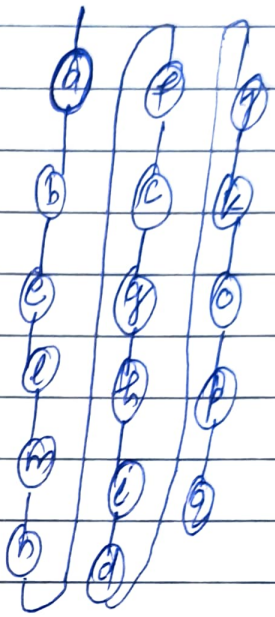
8.04 Linearize generic tree

What



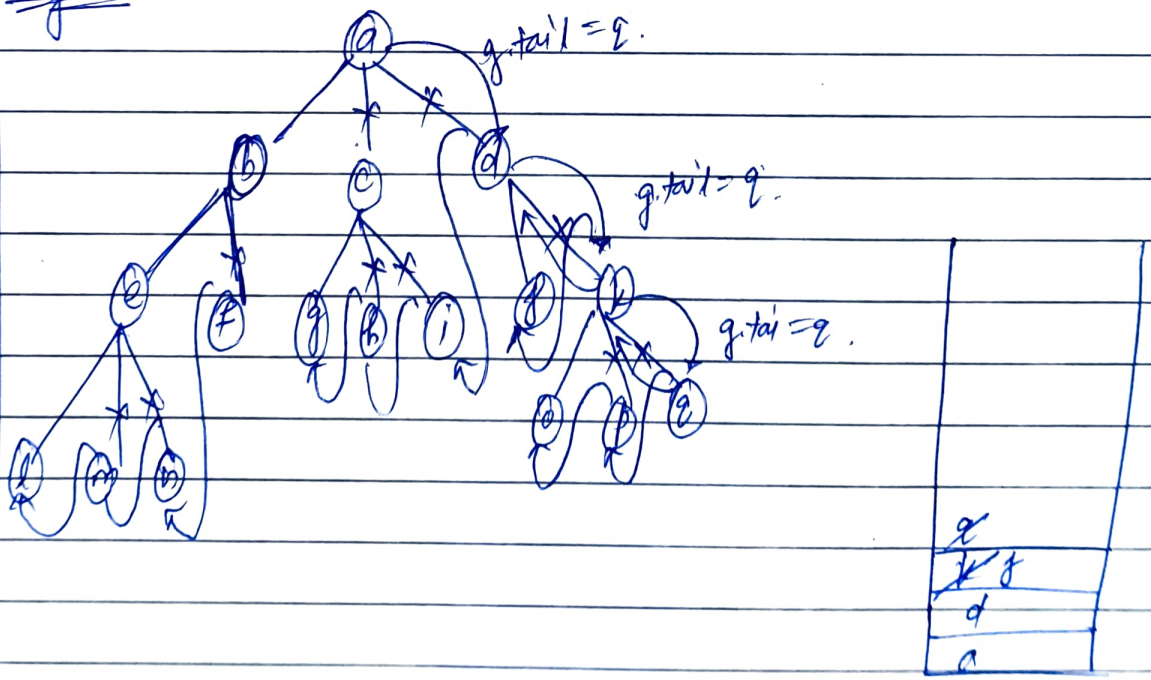
Inorder linearization.

o/p



How? we faith our linearize function to return its tail and linearigated version of it.

dry Run



APCO
Date: / /

// code:

```
Ps Node linearize(Node node){
    if(node.childs.size()==0) return node;

    int n = node.childs.size();
    Node gtail = linearize(node.childs.get(n-1));

    for(int i = n-2; i>=0; i--){
        Node tail = linearize(node.childs.get(i));
        tail.childs.add(node.childs.get(i+1));
        node.childs.remove(i+1);
    }
    return gtail;

}
```