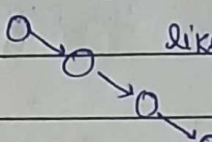
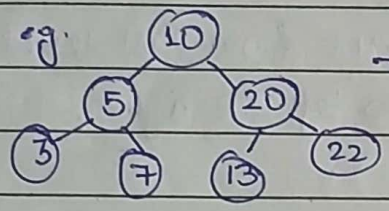
Date 17 01 25
Page No.

*** Binary Trees – Short Notes

- Used bcoz of efficient inserting and deleting ∴ O(logN)
  except for unbalanced binary Tree/Skewed Tree → O(n). like linkedlist.



- Binary Search Tree → smaller than left
  greater than right

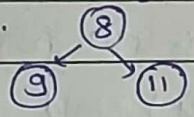


→ finding an item(node) = O(logN)
but for skewed it is O(n) in BST.
∴ we use AVL

∴ Limitation: unuseful for sorted data.

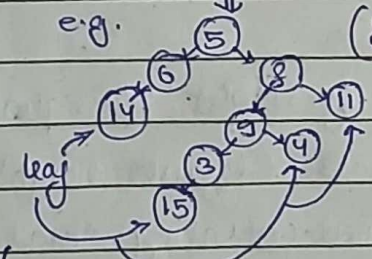
- L.L → class Node{ int value; Node next; }
  ; BT → class Node{ int value; Node left; Node right; }

- Properties: 1. Size of Tree = total no. of nodes

2. 

here 8 = parent, (9,11) = children

3. If two children node have same parent ∴ they are siblings.
4. Lines that connect two nodes = edge
5. Leaf = bottom/last nodes
6. Height (max. of no. of edges blw that node ~~to root~~ leaf)



(8) → from leaf 14 → 3
" 15 → 3
" 4 → 2
" 11 → 1
→ max. = 3 Ans.

7. Level = Diff. of height of that ~~node~~ node from ~~root~~ root
→ here level of 8 = (4 − 3) = 1

Teacher's Signature.....

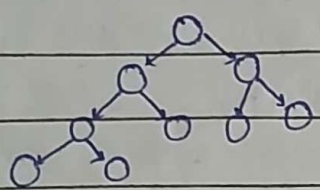
Date
Page No.

8. Ancestor and Descendent
from a node to leaf node → starting node = Ancestor
last " = descendent

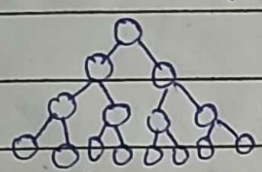
9. Degree = no. of children that node has.
(degree of leaf = 0)

- Types of Binary Tree :-

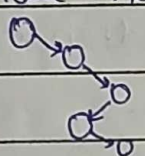
1. Complete = all levels filled apart from last level & in last level filling from left to right.



2. Full/Strict = either 0 or 2 children for each node
3. Perfect = all leaf nodes on same level & all levels are filled.



4. Balanced = when avg. Height is O(log N) → opposite of ~~un~~ skewed.
5. Skewed = every node has only 1 child

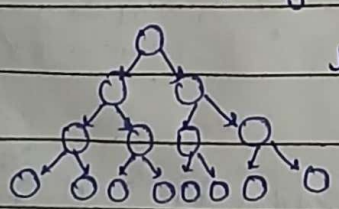


6. Ordered = every nodes have some properties not just randomly inserted. e.g. BST nodes of BST

- Imp. observations :-

1. Total nodes in perfect BT of height 'H' equals $2^{(H+1)} - 1$.

e.g.



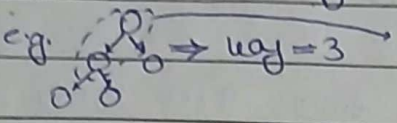
height = 3 (default height is ~~0~~ of root)
On ~~cal~~ counting total nodes = 15
also,
$2^{(3+1)} - 1 = 2^4 - 1 = 16 - 1 = 15.$

Teacher's Signature.....

Date | Page No.

2. For perfect BT → total no. of leaf nodes = 2^h ✓ prev. eg. $h=3 \therefore 2^3=8$ ✓

3. For strict BT → no. of leaves = no. of internal nodes − 1.

eg.  → leaf = 3 internal nodes = 2 ✓ (3−1)

* Binary Tree implementation :-

```
Class BinaryTree {
create non-parameterized const^r of BT
    private static class Node {
        int value;
        Node left;
        Node right;
create int value const^r of Node
    }
private Node root; → declare root only (just like head in LL)
```

- Inserting : 1. inserting root

(create this in main class)

```
public void insert (Scanner sc) {
    sout("enter root node");
    int value = sc.nextInt(); → no need of creating obj. of scanner first.
    root = new Node(value); → first value of Nodes goes to root
```

call for other node

insertions → `insert(sc, root);`

2. inserting other nodes (at first root is passed here.)

```
private void insert (Scanner sc, Node node) {
    sout("Do you want to enter left of :" + node.value);
    boolean left = sc.nextBoolean();
```

∵ true → `if (left) {`

```
        sout("Enter left of :" + node.value);
        int value = sc.nextInt();
```

Teacher's Signature.....

create a node out of it → node.left = new Node(value);

call again for this node now → insert(sc, node.left);

}

Ily, same for right

no base condition in end since tree kitna bhi bada ho sakta hai.

- **Display**

```
public void display() {                    → for spacing / indentation
    display(this.root, "");
}                                          → starting with the root first

private void display(Node node, String indent){
    if(node == null){                      → base condition.
        return;
    }
    sout(indent + node.value);             → more spacing alongwith indent
    display(node.left, indent + "  ");     left →
    display(node.right, "  " + "  ");      right →
}
```

⇓

Dont write code write in words for short nodes.

better
- create non-parameter display fn. in which call for node ie. root alongwith some indentation like " ", → display (same name) begin with
- In that fn. if node is null ∴ return.
- Else, print that node with indentation.

Ily call for left & right. (next nodes after root).

Teacher's Signature.....

Date | Page No.

* BST implementation

I. Basic

- in public class BST make a Node class for value, left, right and int height.
- make Node constr for value.
- " " " (non-parameter) for getvalue in which return value.
- Declare root node then make non-parameter constr of BST.

II. Height

- create height method returning integer and pass Node in it.
- if node is null return -1;
- else node.height.

[• isEmpty() → when root == null (boolean)]

III. Balance

- make a non parameterized boolean type method & call for balance in & first pass the root node.
- create other balance method for node
- if node is null ∴ true ∵ already balanced

(Balance condition → height of left & right should not differ more than by 1.)

- return absolute value of (height(node.left) - height(node.right)) <= 1 (*and) do same for left & right.

call ↗

insert before balancing (must)

III. Insert

- create an void method of inserting & pass value
- Call for other insert method with given parameter → root.insert(value, root)
- In other method → if node == null ∴ create a node (of value) & return it.
- Now if value of node that you inserted in earlier method (int value) is less than node.value → then node.left = insert(value, node.left)

reached at that node.

Teacher's Signature.....

Date
Page No.

- Ily for right ✓
- at the end update height ∴ node.height = Math.max (height(node.left), height(node.right)) + 1;

return node;

IV. Display

- non-parameter void display ↑ init call for other void display & pass this.root (first node), "root node".
- In it's other method if node == null ∴ return.
- Sout (details:+ node.value); // O/P root node : X
- Now to display left :-

display (node.left, "Left child of" + node.value + ":");
(Node node; prev. node; String details)

Ily for right.

V. Multiple insert altogether

⇒ public void populate (int[] nums){
for(int i=0; i<nums.length; i++){
this.insert(nums[i]);
}
}

Therefore in main fn. rather that inserting individually, just create an array & pass into populate.

→ VI. Populate sorted → used for mid values of arrays insertions ~~or~~ to protect from skewed tree.
[Prefer → AVL Trees]

Teacher's Signature.....

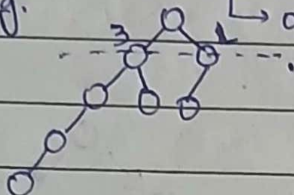
Date 21 01 25
Page No.

* AVL Trees → v.easy topic

why this in picture: for sorted node values it takes O(n) but BST's were meant to do it in O(logN) which is not happening (for sorted) ∴ AVL used
(Adelson Velskii and Landis)

*** We know that for BST → tree should be a Balanced tree.
i.e. H(l) - H(r) = -1, 0, 1 only.
→ on same level

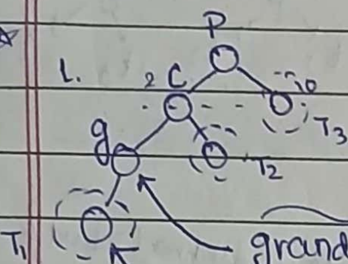
e.g.



∵ 3-1 = 2 ∴ unbalanced.

***** (Just read)

1. ∴ (2-0 = 2) insert normally node n.



already (select!) / to do

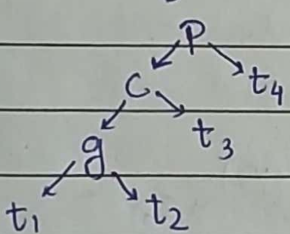
grandchild (before ↓) and in the line of P and C.
[the node due to which tree is not balanced]

2. Name of the case (4 cases) is determined as per the location of child and grandchild.
3. Main motivation is to make P-C-g in a single line then only apply the rule.
4. Therefore rotate (C/P) ~~to get~~ ~~P g C, etc. (together)~~ to balance (c/g → always on top at the end)

***** (check properly v.easy)

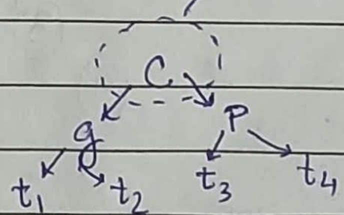
4 rules :- ~~∴ but rotations are fixed for 4 cases~~

1. Left - Left
↓ P ↓ g (on same side left)



∴ right rotate (P) → to bring 'c' on top

now insert others according to basic knowledge BST.



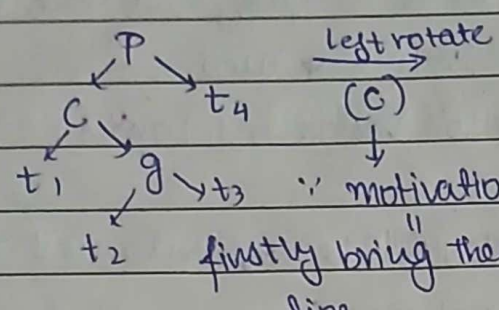
v.v.v. easy

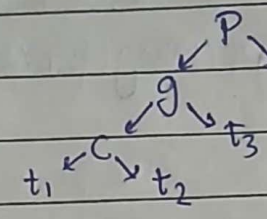
Teacher's Signature..............................

How to 'p' is left or right coz uo to bich m top t hota hai (of subtree)
∴ according to c → its side is decided.

Date | Page No.

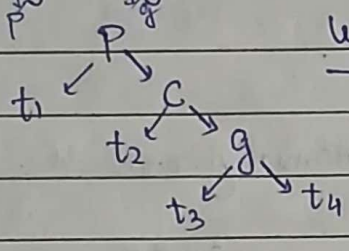
(ii) Left - Right
P g



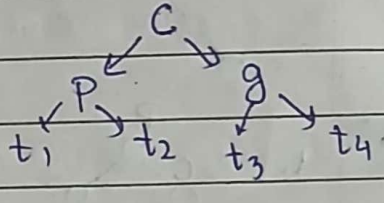


→ Now same as (i)
∴ Solve

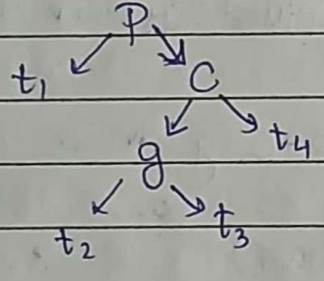
Khud socho yaar! ← (iii) Right - right
P g



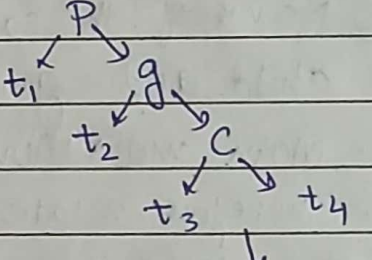
Left rotate (C)
→ to bring 'c' on top coz already in same line (PCg)



(iv) Right - left
P g

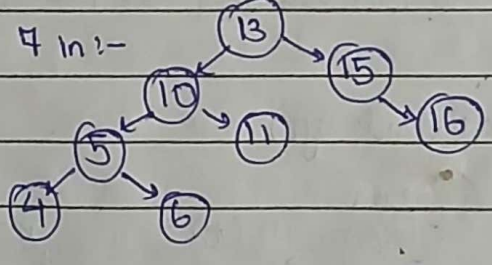


jarori naa pade ye likhna (khud se socho)
(Left rotate (C))
∴ same line

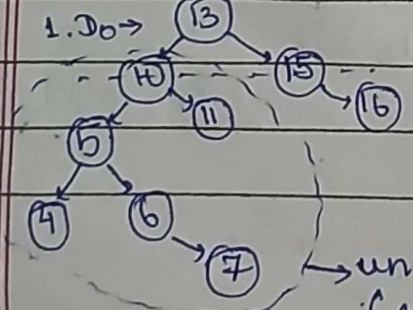


Now same as (ii) ∴ Solve

★ .gfg example → insert 4 in :-

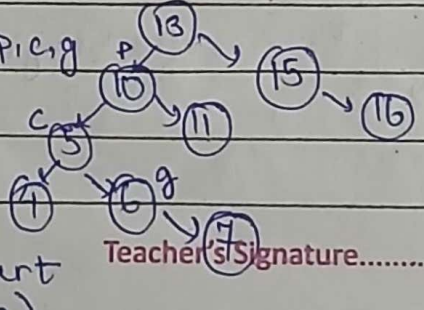


1. Do →

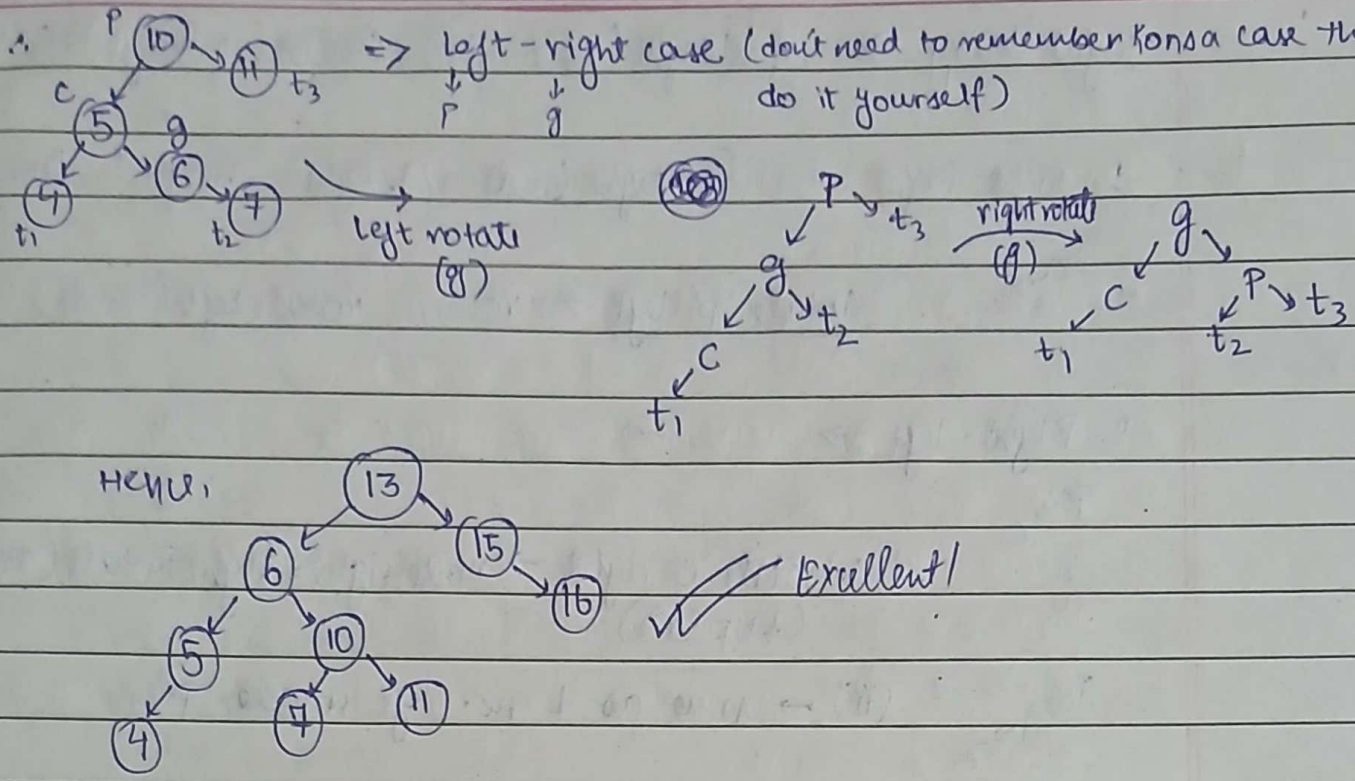


→ unbalanced part

2. Point p, c, g



Teacher's Signature.....



⇒ left-right case (don't need to remember konsa case tha do it yourself)

* Implementation:-

1. Basics 2. Height 3. Empty 4. Balance 5. Insert (after insert + rotate) 6. Display

Insert → only different at end:
return rotate(node);

V. Rotate

private Node rotate (Node node){

Left Heavy → if ((height(node.left) − height(node.right)) > 1) {

1. Left-Left: if ((height(node.left.left) − height(node.left.right)) > 0) {
   (left of g) (right of g)
   rotate 'p' right → return rightRotate(node);
   (P)
   }

2. Left-right if ((height(" " right) − height(" " .left)) > 0) {
   (i). rotate 'c' left → node.left = leftRotate(node.left);
   (and update)
   (or just change this & copy above)
   (ii). same as 1. i.e. return rightRotate(node);
   }
   }

Teacher's Signature.....

Right Heavy → if ((height(node.right) - height(node.left)) > 1) {

1. Right-Right → if ((height(node.right.right) - height(node.right.left)) > 0) {

P→C→g rotate 'p' left → return rotateLeft(node);

2. Right-left → " " " " <0) {

P→C
g←C i. rotate 'c' right → node.right = rightRotate(node.right); (& update)

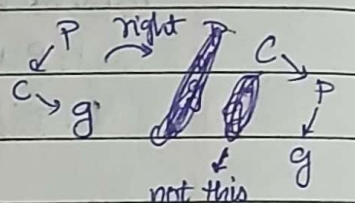
P→g→C (ii) → same as 1 i.e. left rotate 'p'.

VI. Right Rotate

- Node type & pass node p.

  Node c = p.left;

  Node g = c.right;



- rotate ⇒ ∴ (about C)
  - c.right = p; → Do this
  - ~~c.left~~
  - p.left = g; → obviously

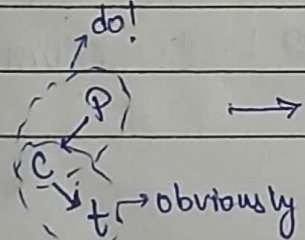
iss case # ~~fg~~ kab se chota ho gya 'c' se)

- update heights: p.height = Math.max(height(p.left), height(p.right)) + 1
  (∵ heights changed after rotation) c. " = " " " c " " c " "

- After rotation 'c' is the node (at top) ∴ return c.

VII. Left Rotate

1. pass C → 2. C→P→t



→ obviously

→ 3. update height of p+c → 4. return p

Teacher's Signature.....

4 1-2=1
2 4 =2<-1

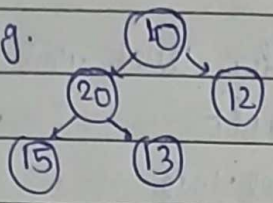
learning: ~~[illegible]~~ → R always after left
∴ N → first = 1, middle = 2, last = 3

Date
Page No.

* **Traversal Methods** → used for converting String/Array/LL/etc ⇌ Trees

1. **Pre-Order**
   ↳ Node-left-right

e.g.



⇒ To be written as: 10 → 20 → 15 → 13 → 12
(10: Node; 20 → 15 → 13: left; 12: right.)

2. **In-Order**
   ↳ L-N-R

for same example :→ 15 → 20 → 13 → 10 → 12 (easy)
(15 → 20 → 13: left; 10: N; 12: right)

3. **Post-Order**
   ↳ L-R-N

⇒ ~~12 → 1~~ 15 → 20 → 13 → 12 → 10
(15 → 20 → 13: left; 12: R; 10: N)

- **Implementing pre-order (N-L-R)**
  - In non-parameter void type method (preOrder()) call for same method name & pass root node in it.
    Overloading
  - In that method, if node == null ∴ return
  - else → Print Node → sout(node.value + " ");
    call L & do same → preOrder(node.left);
    " R " " " → " " (" .right);
    }

(Basically same for in-order & post-order — Do it yourself very easy)

Teacher's Signature.....

Date
Page No.

Binary Trees Questions

1. Binary tree level order traversals

- create a list of list of result for level wise nodes value.
- if root == null then return result.
- Make of a queue of type treeNode.
- Push/offer root into it.
- While the queue is not empty, create a level size of queue.size();
- Make a list of integer for current level values pass size levelSize into it.
  - In that particular list → for i=0 to levelsize, make a current Node of type treeNode = queue.poll();
  - add the value of current node into current level.
  - if currentNode.left != null → add that value in queue. Ily for right.
- finally after while loop add all those currentlevel into result.
- return result.

2. Average of levels in Binary Tree.

→ return the avg. value of nodes on each level.

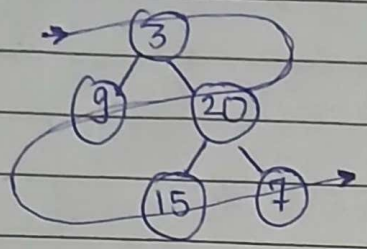
⇒ Concept :→ for average of each level → same code as of BFS

- Create Double type list of result for storing values.
- ~~Declare~~ initialize double type avglevel as 0 inside while loop.
- "for" getting each values of current node.val store it into averagelevel i.e. averagelevel += currentNode.val
- at the end (after every for loop) averagelevel = averagelevel / levelsize;
- result.add(averagelevel)
- return result.

Teacher's Signature.....

Date
Page No.

3. Zigzag level order Traversal



O/P ⇒ [[3],[20,9],[15,7]]

- ~~create a list of list of type integer~~ same code as of BFS
- Instead of creating a queue, make Deque & add root
- boolean reverse = false
- In for loop → if (!reverse) → create a TreeNode type → currentNode = queue.pollFirst(); and add this value into currentLevel
- Do for left and right use addLast
- ~~at~~ the end reverse = !reverse;
- result.add (current level); → return result.
- else { vlly create a TreeNode but here for pollLast (); currentLevel.add & vlly do addFirst for left & right.

4. Binary tree level order Traversal II

→ Just return nodes breadth wise from bottom to Top.

One concept ⇒ in BFS code instead of ~~while~~ result.add(currentLevel) → use result.add(0, currentLevel)
↓
last se add karega

Teacher's Signature.....