Lecture # 21 Wednesday 26/5/2021

⇒ Array and linked ~~list~~ based implementation of binary tree.

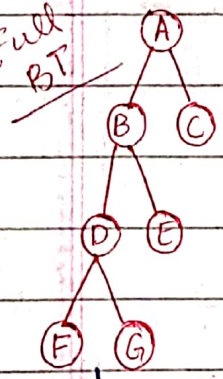
Structurally, binary trees are of three types:-

→ Full binary Tree (Each node has either '0' child or 2 children)
→ Complete BT
→ Perfect BT

extremes

Full BT: • Each node has either zero/no child or has 2 children.
• No ~~or~~ node in the tree has 1 child.

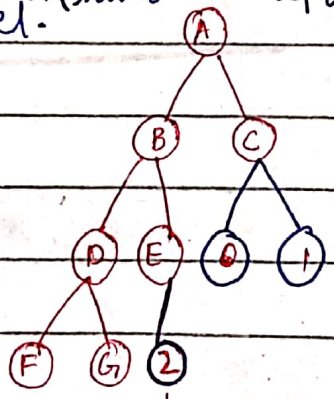
Full BT



Complete BT.

⇒ Leaf node is allowed only on last and second last level.
⇒ Single child is allowed only on last level. → (should be left child)

converting it to 'complete' BT. ⟹

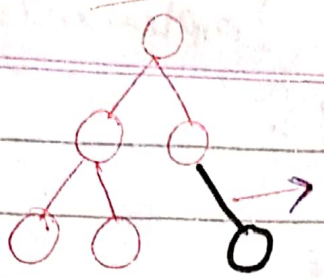


Complete BT

↳ single child is allowed at last level.

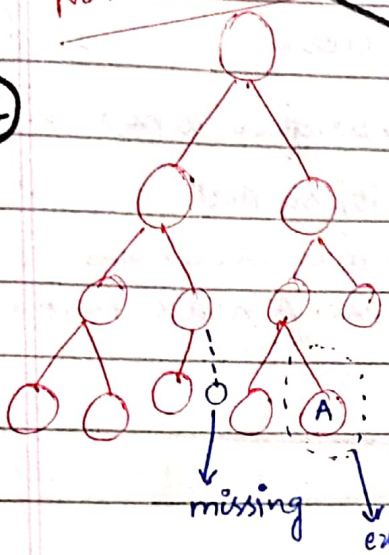
Examples :-

NOT complete

① 

This is not a complete BT. To make it complete, the single node should be left child and not right.

NOT complete

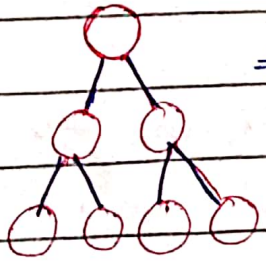
② 

This is not a binary tree (complete), because a condition for complete BT is that The nodes at last level should be as left as possible. In this tree, node A could be missing. But jahan tuk nodes hain, wahan tuk poori filled honi chahiyen.

Perfect BT:

⇒ Each node can have either zero child or 2 children.

⇒ All leaf nodes should only be at last level.



⇒ Internal nodes have 2 children.

⇒ leaf nodes have 0 child.

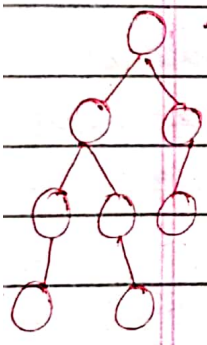
⇒ **Every perfect binary Tree is also a full binary tree**

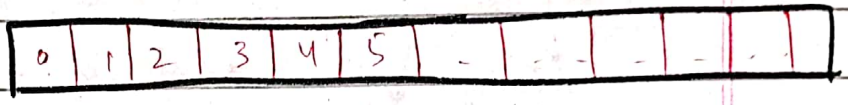
Implementation of BT

Array-based :-

```
Class BT
{
    T * arr;        → stores tree
    bool * status;  → Tells whether a node is present or not.
    int height;     → Keep it, so that we may calculates no. of nodes through it.

    Methods ...
}
```





* Height = 4.
* So, max. nodes = $2^h - 1 = 15$ nodes
* So, create an array of 15 size in the constructor.

① constructor ();

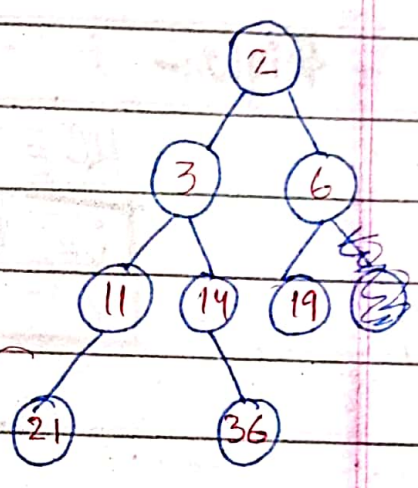
② * setRoot (v);

* Left child of any node = $2i+1$.
  Right child = $2i+2$

③ * void setLeftChild (parent index/i , val) ;

④ * void setRightChild (" ") ;

* void setLeftChild (i, val)
{
left-index = 2i+1;
arr[left-index] = val;
status[left-index] = true;
}



Tree arr:

2	3	6	11	14	19	-	21	-	-	36	-	-	-	-

Status arr:

T	T	T	T	T	T	F	T	F	F	T	F	F	F	F

* setRightChild is same except formula 2i+2.

5. * getLeftChild (parent index);
6. * getRightChild (" ");

To get parent when child is given:

Left:

$LC = 2i + 1$

$$\boxed{i = \frac{LC - 1}{2}}$$

LC = leftchild
i = parentindex = ?

(7)

* getParent (LC)
{ return arr[$\frac{LC-1}{2}$]; }

Right:-

$RC = 2i + 2$

$$\boxed{i = \frac{RC - 2}{2}}$$

* getParent (RC)
{ return arr[$\frac{RC-2}{2}$]; }

Demerit of array based implementation:-

No resize function.

⇒ New array of data & Status is to be formed everytime which is space & time taking.

⇒ Hence, linked list based implementation is preferred.

Linked list based :

```
Struct TreeNode
{   int data;
    TreeNode* left;
    TreeNode * right;
}
```

```
Class BinaryTree
{
  TreeNode* root;
public:
① * BinaryTree()
    { root = NULL; }
② *BinaryTree(int v)
    { root = new TreeNode(v); }
    OR
    root->data = v;
}
```

```
③ *BinaryTree(TreeNode*)
{   root = R;
}
```

```
④ void setLeftChild(TreeNode *parent, int v)
  { p->left = new TreeNode(v); }
```

```
⑤ void deleteLeftChild(TreeNode *p)
  {   if (p->left)
        delete p->left;
      p->left = NULL;
  }
```

```
⑥ void deleteNode(TreeNode *N)
  {   if (N->left)
        delete N->left;
      if (N->right)
        { delete N->right; }
  }
```

STEP 1

STEP 2

STEP 3 Point to NULL that pointer of parent of N, which is pointing to N.

For this purpose, we need to traverse the Tree.

* Stack or recursion will be used.

Pre-order Traversal

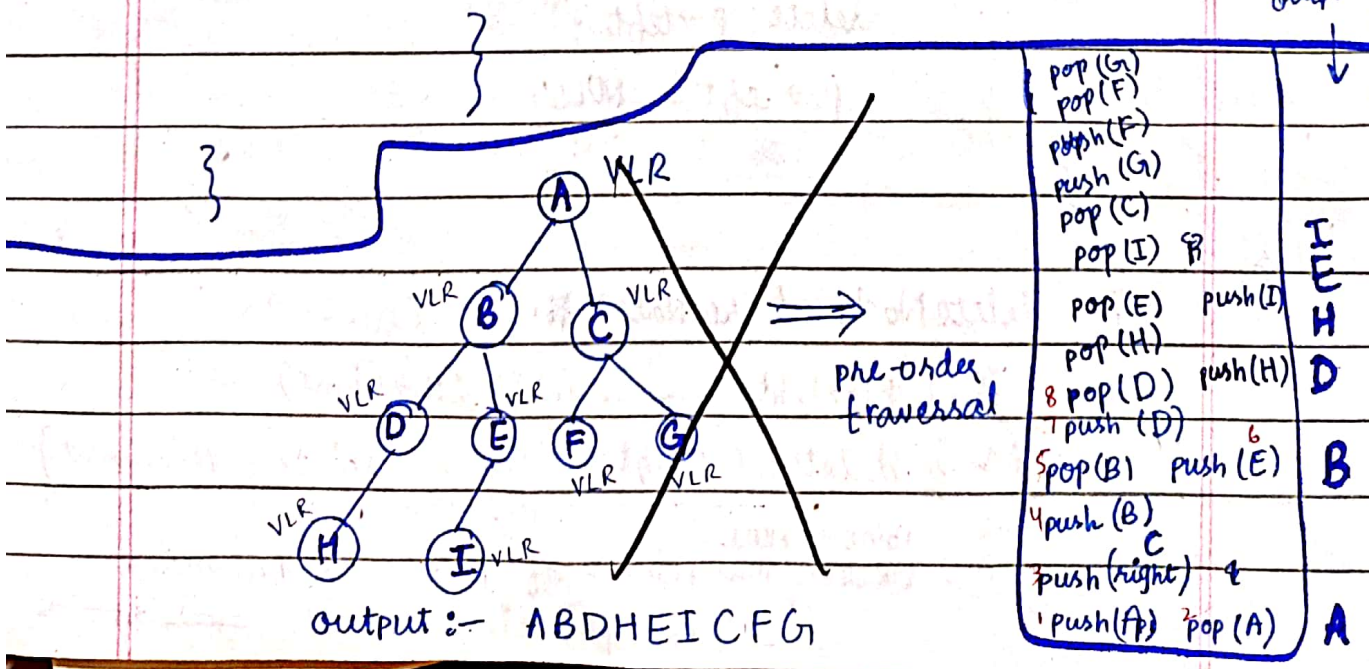
Recall ... theory

Iterative

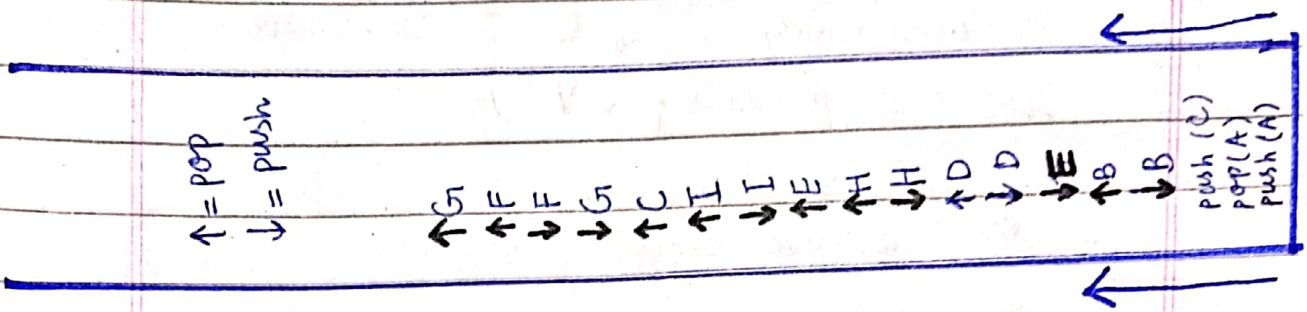
implementation :- Through stack

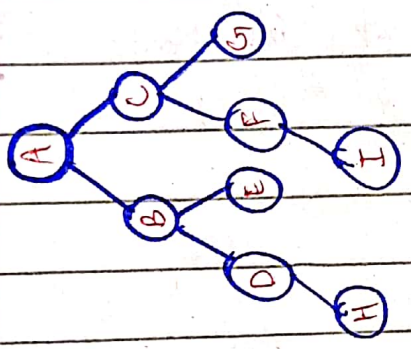
```
void preOrder()
{
    stack<TreeNode> S;
    TreeNode *temp = root;
    if (temp)
    {   s.push(temp);
        while (!s.isEmpty())
        {
            temp = s.pop();
            cout << temp->data;
            s.push(temp->right);
            s.push(temp->left);
```

output



How preOrder function works with Stack:-





Output should be:

ABDHEICFG

* Right child ko left se pehly push krna hai taa k pop (print) krty huye sahi wala order aaye

Through Recursion (Pre order)

```
void PreOrder ( TreeNode * root)
{ if (! root)
      return;
  cout << root → data ;        ⇒ V
  PreOrder (root → left) ;     ⇒ L
  PreOrder (root → right) ;    ⇒ R
}
```

} Pre order.

```
PreOrder (root → left) ;     ⇒ L
cout << root → data;         ⇒ V
PreOrder (root → right) ;    ⇒ R
```

} → inOrder

PreOrder (left) ; ⇒ L
PreOrder (right) ; ⇒ R
cout << temp → data ; ⇒ V
} Post order

* iterative = using stack
* recursion = whi chota wala code

* Recursive implementation of in order :-

```
void InOrder (Tree Node * Root)
{
    if (! root)
        return;
    InOrder (root → left);      (L)
    Cout << root → data;        (V)
    InOrder (root → right);     (R)
}
```

* Youre to write iterative implementation of 'In Order' (Do yourself)

* iterative implementation of level order → (Do yourself using queue)

Completed

Lecture # 22 :-

Two implementations of Tree :-

→ Array - based } → revise
→ Link based }

~~p = 2i + 2~~

* left - child = 2 * parent + 1
* right - child = 2 * parent + 2

* Level order Traversal :-
  ① Enqueue a parent
  ② Dequeue it.
  ③ Enqueue left & right children

Code :-

```
Queue <TreeNode * > q;
q . enqueue (root);
While !(q . isEmpty())
  { dequeue ;
    cout ;
    Enqueue left child ;
    Enqueue right child ;
  }
```

* BSTs are optimized for searching
  For simple binary Tree :- O(n) is time complexity
  For BST :- O(log n) is time complexity.

* BST :-
  1. They are binary
  2. Parent > left child
  3. Parent <= right child
  4. Property 2 & 3 apply for sub-Trees too.

* insertion in BST (revise)
* search in BST (revise) (efficient)

* Tail recursive function (internet)
* searching of BST is in O(h) ⇒ (worst bhi kam hota hai)
* smallest value → left most leaf
* largest value → right most leaf
* Degree of smallest and largest children are 1 or 0 and can't be 2.

* n is the max height of BST. (skewed Trees)
* log(n) is min / ideal height

* LVR (inorder traversal) on BST provides us the sorted data.

* implementation of BST (revise)
* If you insert data in a BST in increasing sorted order ⇒ right skewed Tree.
* insertion in decreasing sorted order ⇒ left skewed order
* If h = n, complexity of insertion & search is O(n).
* If height is not balanced (skewed Trees), then O(n).
* If height is balanced then O(h), O(logn) between O(logn) - O(n).

(Lec#22 55 min)

* Dangling pointer (internet)
* BST mein jab parents ~~me~~ marta hai, to children ko ancestors adopt kr lety hain
* jisy delete krna hai, us k parent ko dangling bony se bchana hai.

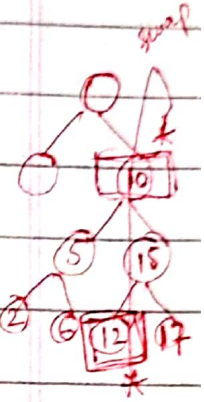
* Deletion in BST:-
  - case 1:- No child (degree 0)
  - case 2:- One child (degree 1)
    - ↓
    - ancestor adopt krlega

→ height balanced tree

case 3: (two children) degree = 2

How to delete such a node? ? ?

Us K right child ki tree me se smallest number dhoond K us K sath swap kr dena hai or phir jo needy chali gyi, smallest wali, usy delete kr dena hai



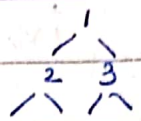
Swap 10 & 12 and then delete 10.

~~Lecture # 23~~

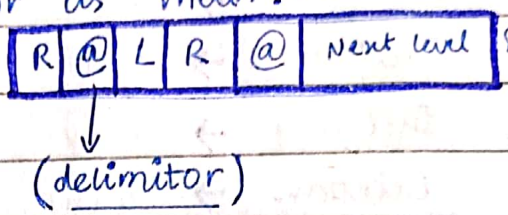
→height balanced Trees AVL
→concept & terminology: balance factors, LL, LR, RL, RR. (rotations)

31/5/2021 Monday Lecture# 23

Print BST Functions:



① Next level kb shuru horaha hai? Is k liye :-
Queue rkh lain or us main :-



& so on

After each level, queue me delimitor rkh den.

② Spaces of kesy manage krna hai?
Tab spaces (Use escape sequences)

---

* Binary Tree → Void remove function → post order traversal will be used for cascade delete. (cruel approach) parent mer jaye to sary bachy bhi maar do.

'h' can be as good as (log n) and as bad as (n).

→ Search operation in simple binary tree costs O(n) (n = no. of nodes)

→ While in BST, it takes O(h) where 'h' = height.
worst case : O(n) ∵ h = n
Best case : O(log n)

→ FindSmallestElement() ⇒ O(h) (left most child)
→ FindLargestElement() ⇒ O(h) (right most child)
→ InOrderTraversal() gives sorted data.
→ insertion() ⇒ O(h)
→ deletion() ⇒ O(h)

Time complexity of operations in BST

searching → O(h)
FindMin → O(h)
FindMax → O(h)
Insertion → O(h)
Deletion → O(h)

Best case :-
$h = \log n$
Worst case :-
$h = n$ → skewed trees

Best case ⇒ $h = \log n$ → Minimum height with num of nodes = n
Worst case ⇒ $h = n$ (Skewed Tree)

⇒ We are to apply some properties on BST to take its complexity to $\log n$ for all cases. (As that of binary search)

⇒ We will use such properties which will ensure that height of BST will always be $\log n$.

To ensure that $h = \log n$:

Demerit of BST:

Time complexity is same as that of simple binary Tree. Worst case : $O(n)$
It should be $O(\log n)$.

AVL Trees

Balanced height
$h = \log n$ (around $\log n$)

* Those BSTs whose height is balanced.
* AVL is BST with some extra properties.
* BST is binary Tree with the property that parent → left is less than parent and parent → right is greater than parent.

AVL: For n number of nodes, height of tree is around (log n).

* Whenever a node will be inserted or deleted, it would be ensured that height is not get unbalanced.

AVL → Adelson, Velski, Landis } → people who made AVL Trees

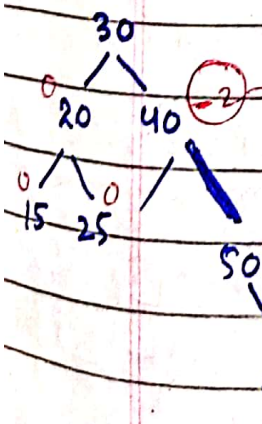
→ Height should always be around log n.

Balance Factor

* Each node has its own balance factor.
* = height of left sub tree − height of right subTree
* Bal. Factor is always an integer.
* If value of balance factor = −1 or 0 or 1, then it means that height of Tree is balanced, else unbalanced.
  - 0 = perfectly balanced
  - 1 = when left side ka ek level right subTree se ek ziyada hota hai
  - −1 = when right subTree ka ek level left subTree se ek ziyada ho
* If unbalanced, we balance Tree through some methods.

On each node, we check balance factor.

⇒ **Example:** Check whether height of this tree is balanced or not?



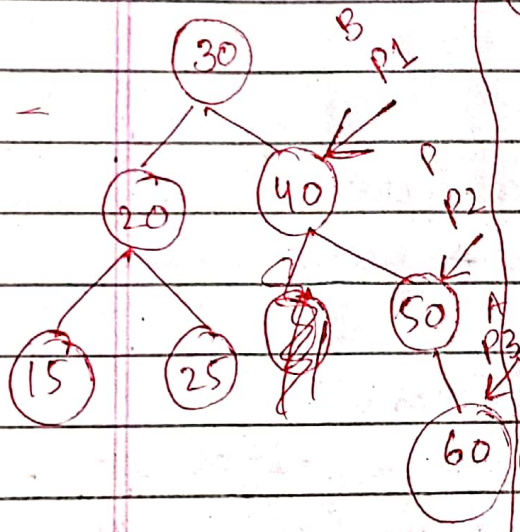
→ We got that Tree is unbalanced, so we will now first balance it. → Methods →

Ans

dy go

Rotations are used to ~~fix~~ check balance factor.

- → LL
- → RR
- → LR
- → RL



(1) Jis node pe balance factor kharab hua, us pe ek pointer point krwa den

(2) Us node ki left or right node ko check krna hai, k kis ki waja se balance kharab hua, jis taraf height ziyada hogi, us ki waja se. Us child pe bhi ek pointer painter krwa den.

(3) Ab jis child pe painter paint kia hai, us k left subtree ya right subtree mein se jiski height ziyada hai, teesra painter us child pe paint krwa dena hai.

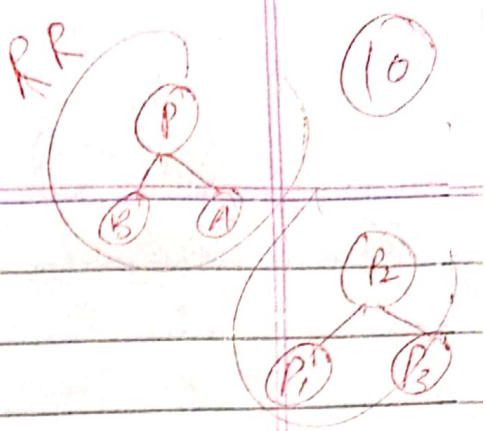
(4) Now, we are to figure out, that which rotation to be applied.

If:

P1 → right = P2 → right = P3 ↓ RR

P1 → right = P2 → left = P3 ↓ RL

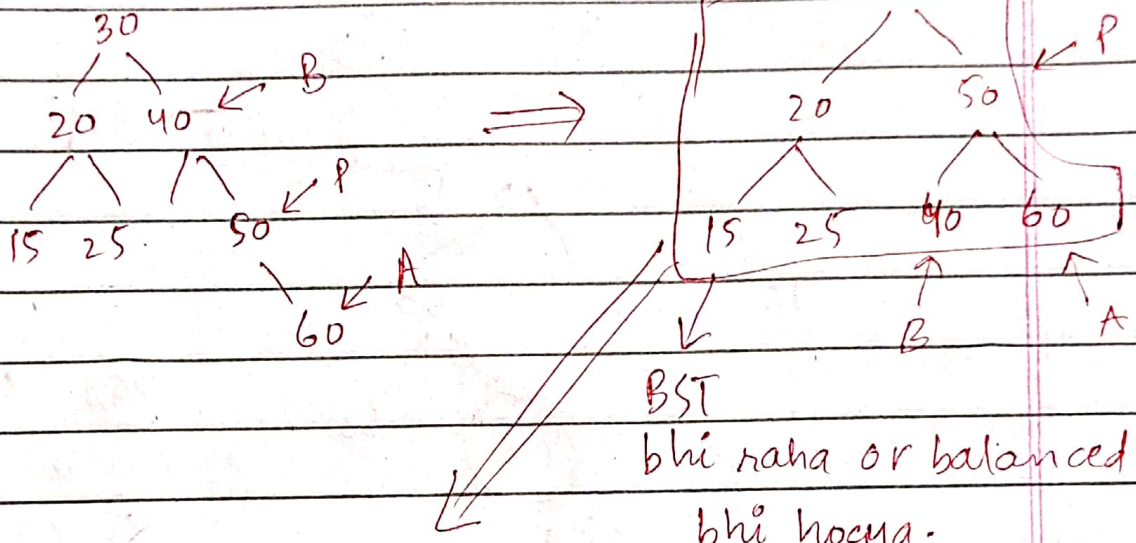
LR
LL ✓



Make P node parent.
Make B its left child
Make A its right child

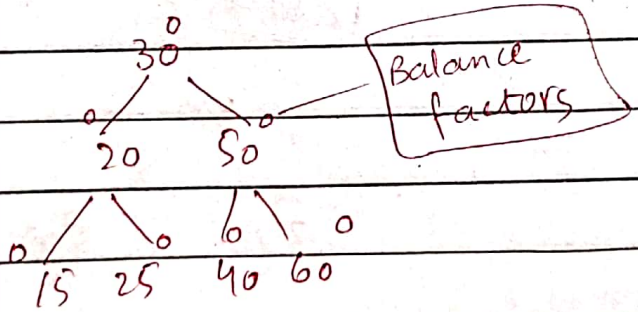
↓

Lets apply it to our unbalanced Tree.



BST bhi raha or balanced bhi hogya.

Lets check either its height is balanced or not.



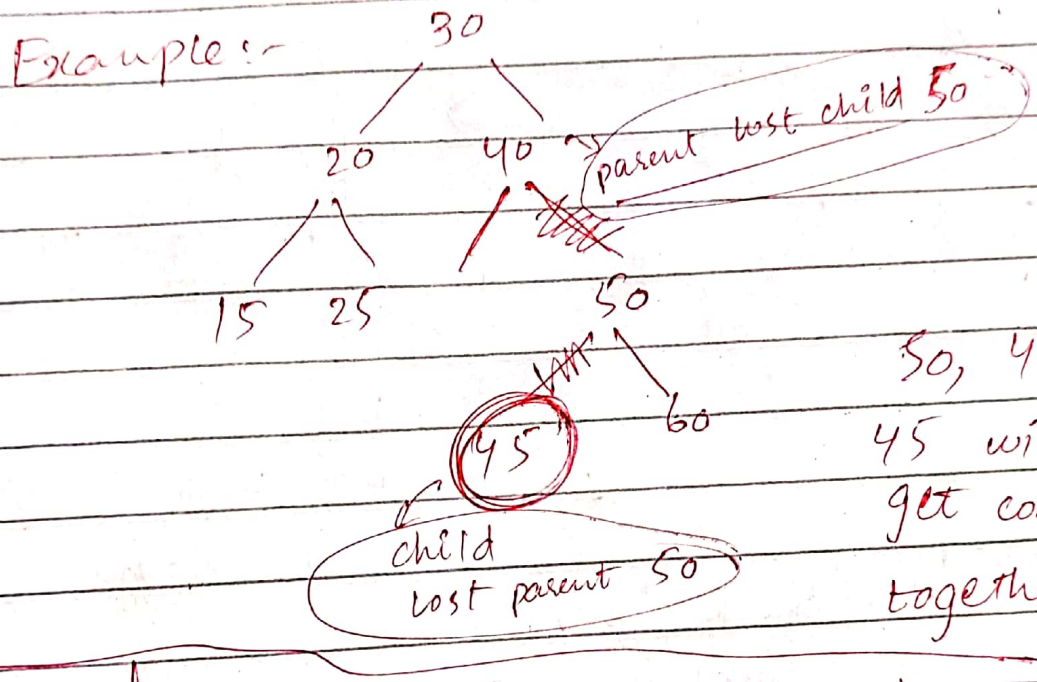
Parent (P)

Aditi's Question:- what if (50) already has a left child. Where will 40 go?

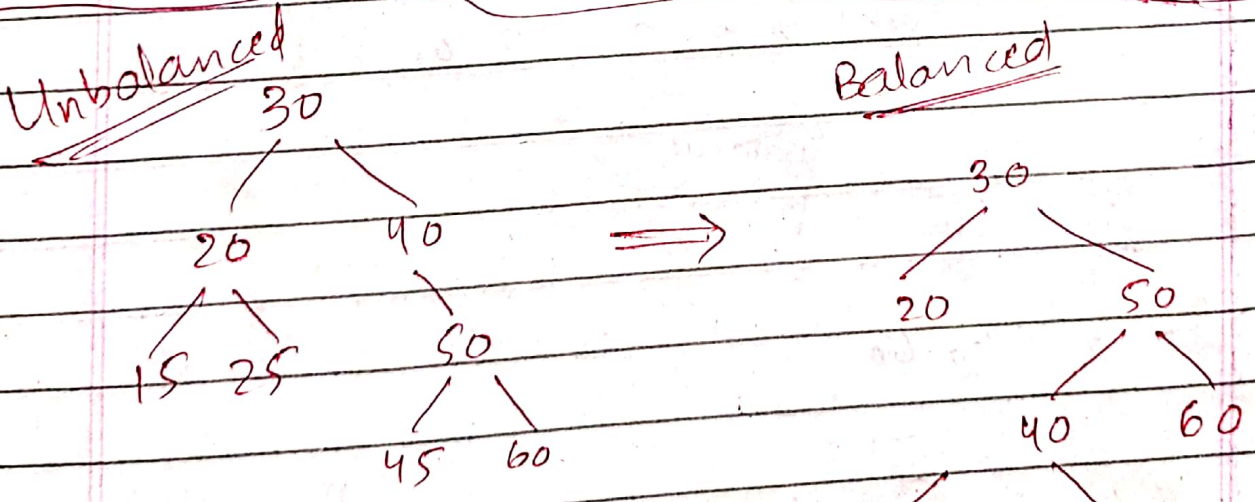
Answer →

⇒ BST wala concept apply hoga. K agr ek node ne apna child lose kia hai or ek ne apna parent lose kia hai, to wo dono parent or child apas me connect hojaty hain. Matlab ancestors adopt kr letay hain grand children ko.

Example:-



50, 40 & 45 will get connected together.

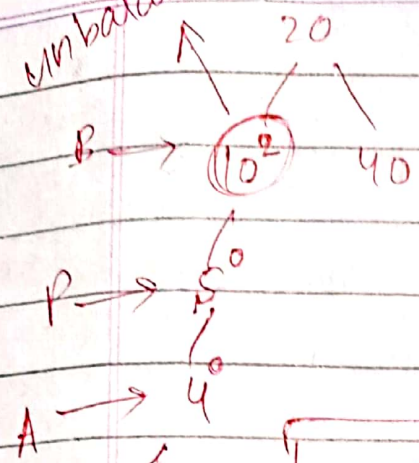


Jesa k (40)ne apna right ~~child loose kia tha~~ tou usy new right child mil gya.

* Steps to balance height of tree *
* Rotations K tareeky

LL Rotation

unbalanced

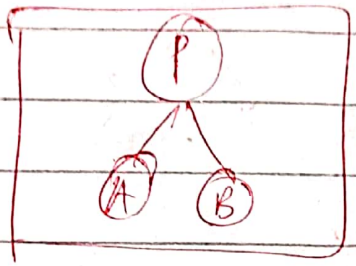


B → left = P → left = A

Means

A < P < B

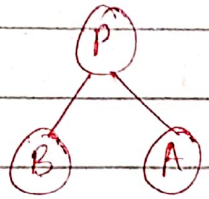
Property of BST



LL

1

RR



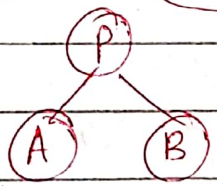
Because it is right side

B < P < A

Right most wali node yaani K (A) subse bari hogi or isliye wo right child bn jaye gi

2

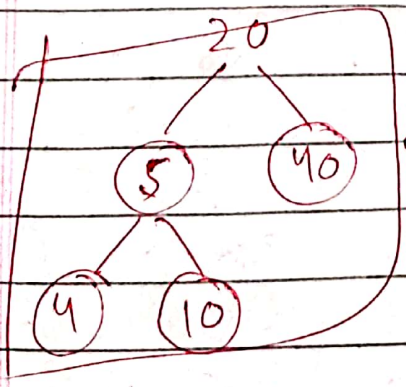
LL



Because it is ~~right~~ left side

A < P < B

Left most wali node ki value sub se choti hogi (A) or isliye wo left child bn jaye gi

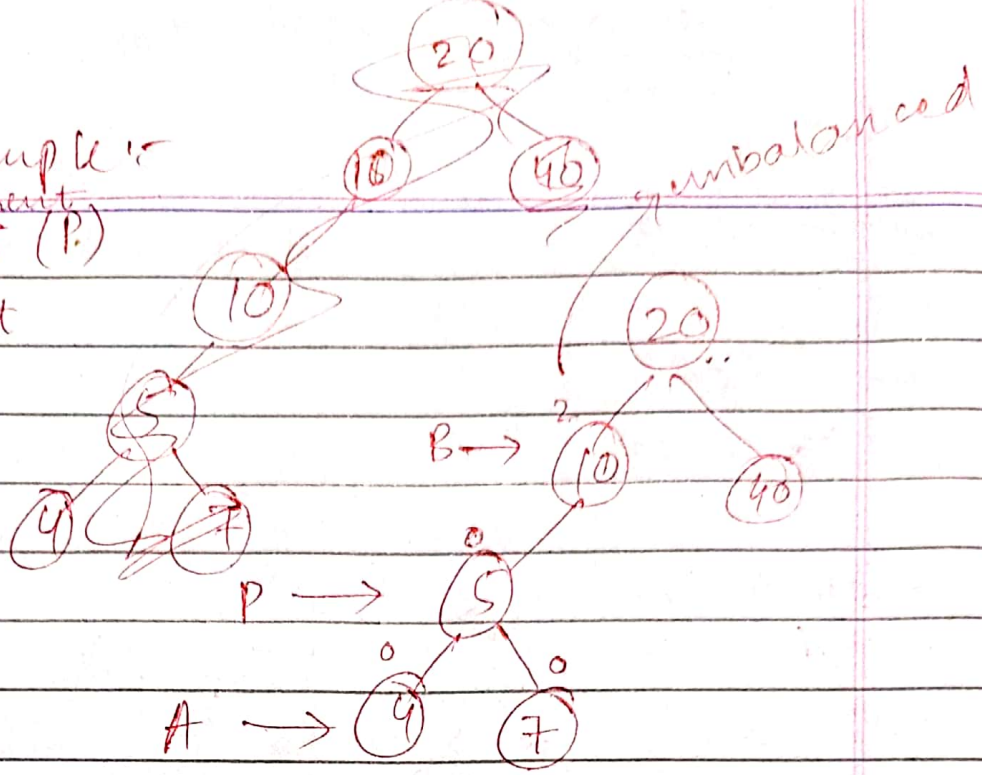


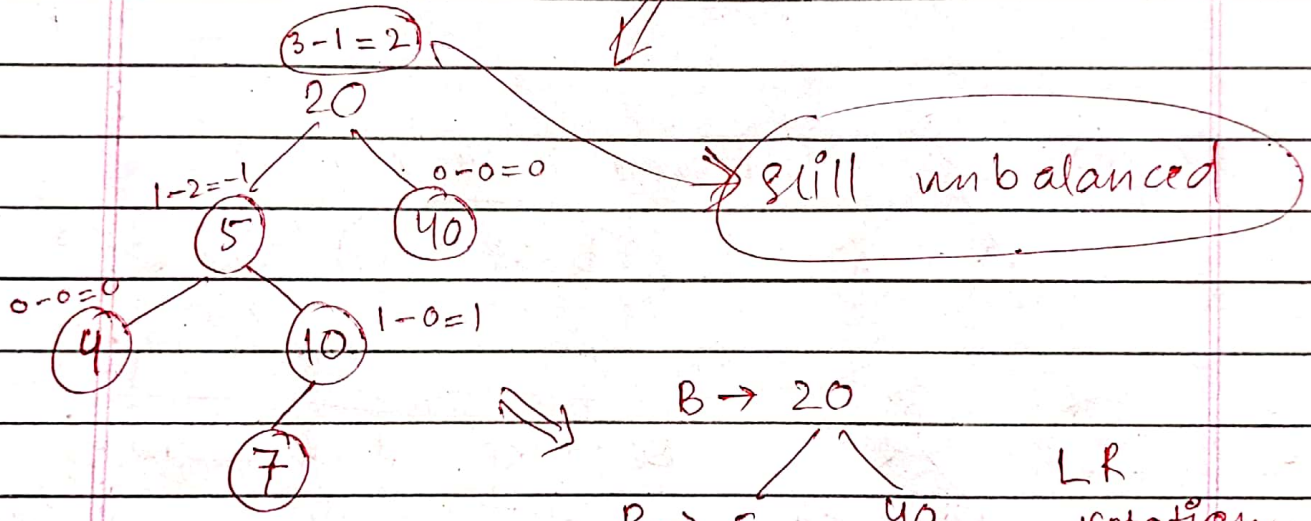
Balanced now using LL

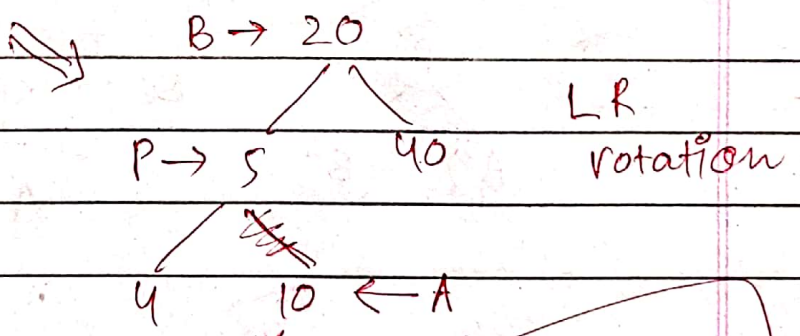
Scanned with CamScanner

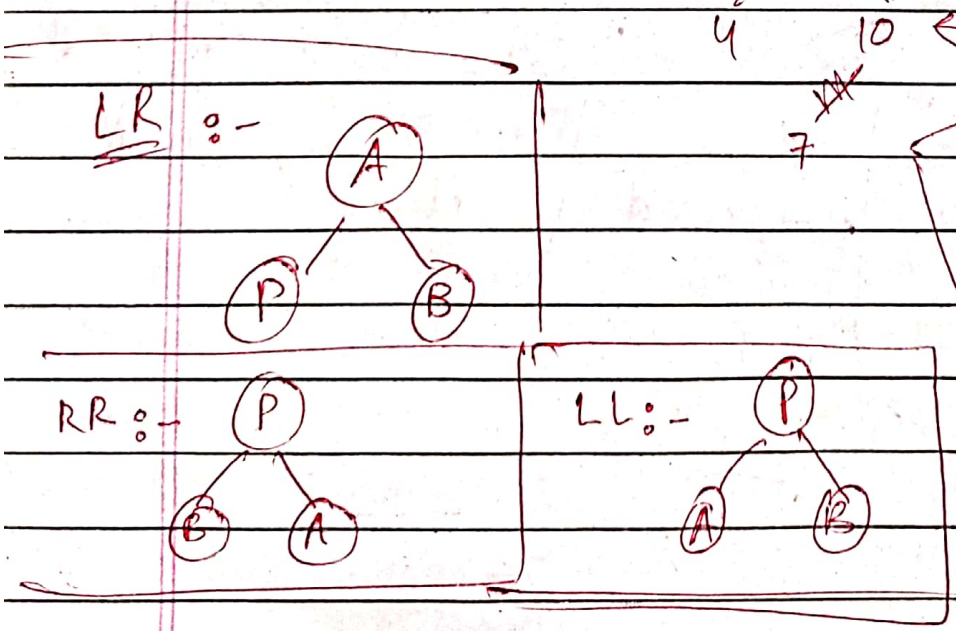
Example :-

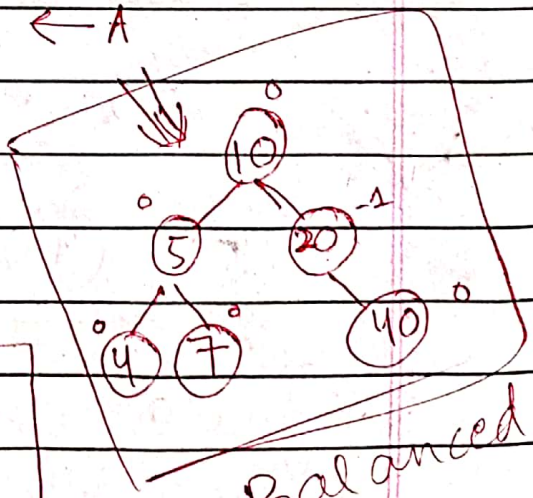
What if 5 (P) has a right child too?







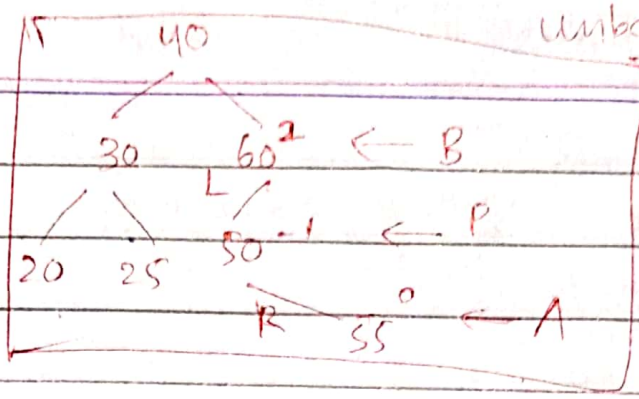




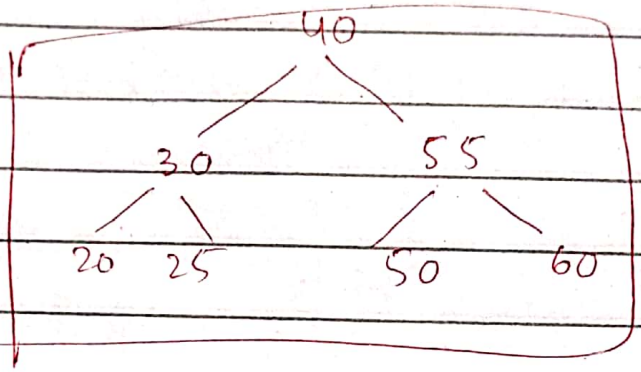
RR rotation

LR

LR rotation example



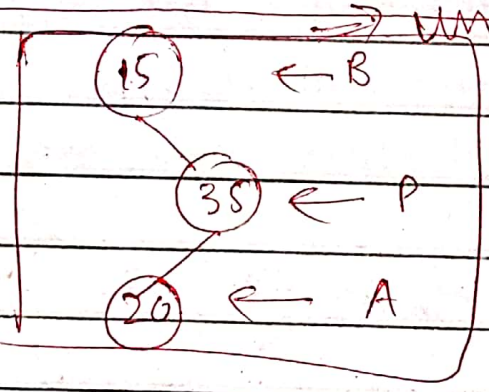
unbalanced (1)



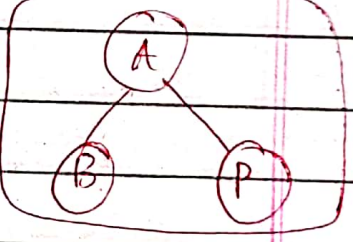
⇒ balanced

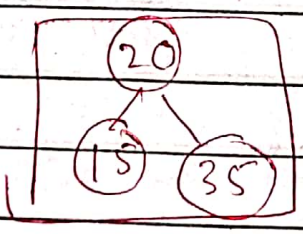
⇒ unbalanced

RL rotation



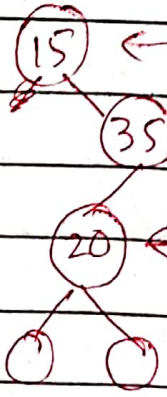
B < A < P





⇒ balanced

What if (A) parent already has a child?



Whi process jo peechy use krty aa rhy hain

Ancestor adapt kr le ga wala process