Tree

- Tree is a non linear data structure and a hierarchy ~~consiti~~ consisting of a collection of nodes such that each node of the tree stores a value and a list of references to other nodes.

- The data in a tree are not stored in a sequential manner that is data is not stored linearly. Instead They they are arrange on multiple levels or we can say it is a hierarchical structure.

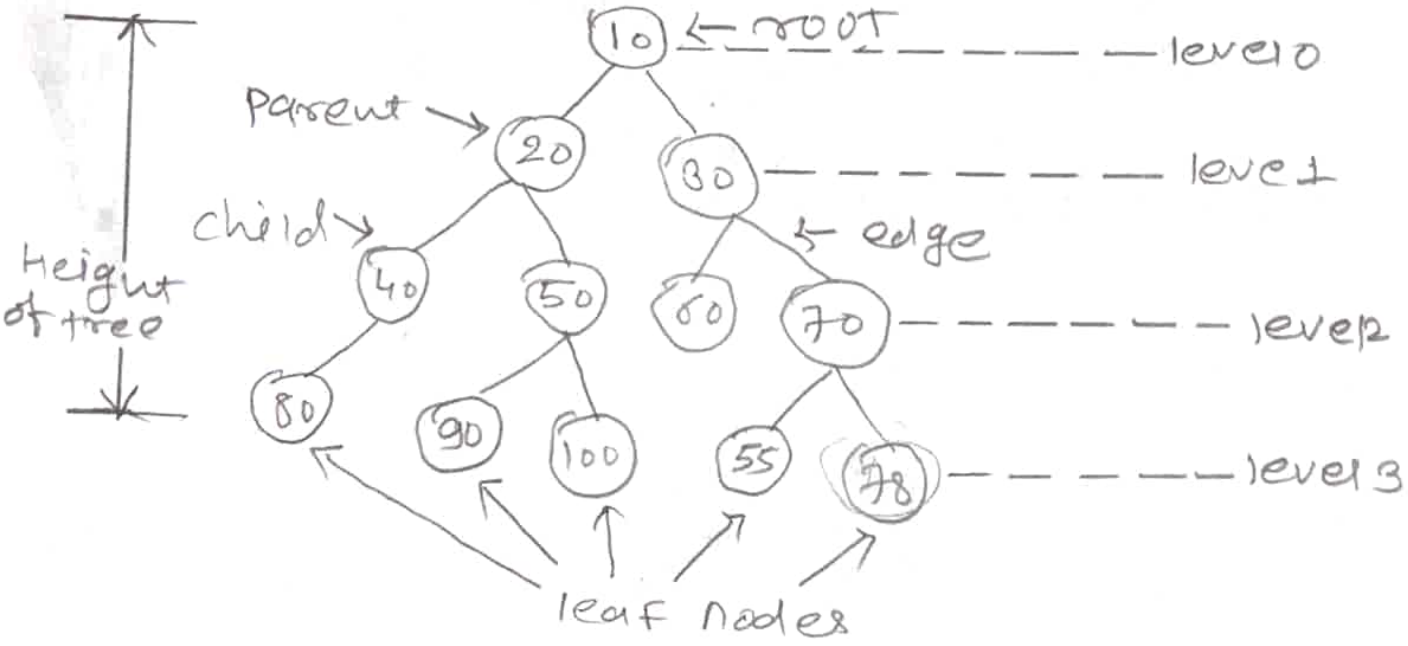


fig - Tree

- Here tree consist of 12 Nodes.
- 10 is a root node of tree.
- 10 is having two child 20 & 30. So 10 is Parent of 20 & 30.
- 20, 30 and are having same parent so they are known as siblings. same way 60 & 70 are also siblings.
- 80, 90, 100, 55, 60, 78 are leaf nodes.

- leaf nodes are also known as external nodes.
- A node which has at least one child is called internal nodes.
- Height of the tree is a length of the longest path from the root of the tree to the leaf node of tree.

---

* Binary Tree

A Binary tree means that the node in the tree can have maximum two childern. Name Binary itself indicate that 'two' therefore, each node can have either 0, 1 or 2 childern.

eg.

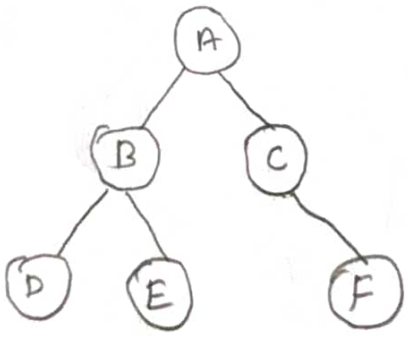


Fig. Binary tree.

Here, tree in the above figure is binary tree as each node have either 2, 0, 1 or 2 child.

* Binary search tree.

- Binary search tree is a binary tree which is either empty or non-empty. if it is non-empty then every node contain a element must ~~follow~~ satisfy following properties.

1. values less than it's parent are placed at left side of parent node.
2. values greater than it's parent are placed at right side of parent node.
3. The Left and Right sub tree are again Binary search tree.

Advantages of BST:

- searching an element in the Binary search tree is easy as we always have hint that which subtree has the desire element.
- Insertion and deletion can be faster in BST.

eg: construct a BST using following data elements - 45, 15, 79, 90, 10, 55, 12

solution:

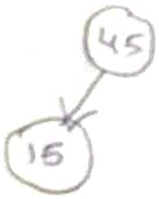
- let's start construction of BST by inserting element 45 as root node.
- Then we will read next node, if it is smaller than the root node, insert it ~~ets the~~ as the left child

- if new node is larger than the root node we will insert it as the right child.

Step 1: Insert 45:

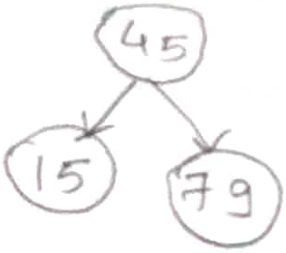
Step 2: Insert 15:

Here 15 is smaller than 45, so insert it in the left side of root node.



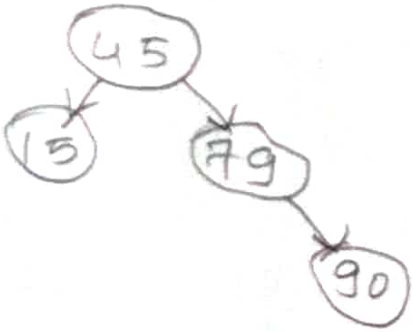
Step 3: Insert 79

- 79 is greater than 45, so insert it in to right of 45.



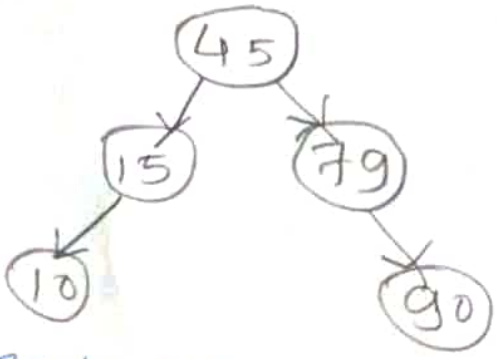
Step 4: Insert 90

Here 90 is greater than 45 and 79 so put it on right side of 79.



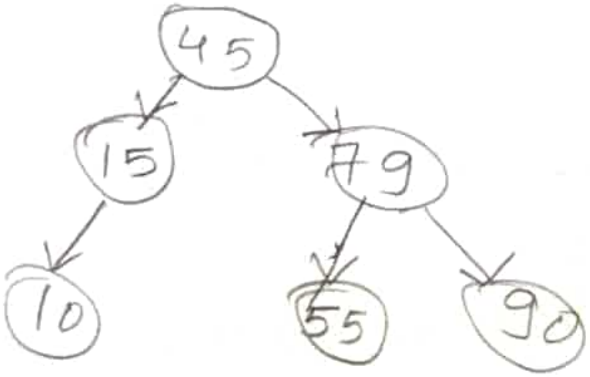
step 5: Insert 10.

10 is smaller than 45 and 15, so we will put 10 on the left side of 15.



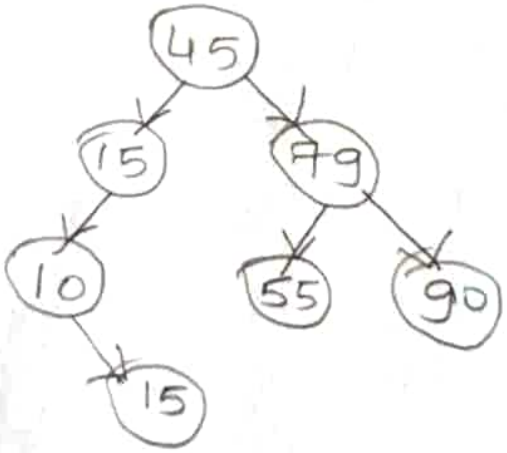
step 6: Insert 55

55 is larger than 45, but smaller than 79 so 55 will be inserted on left of 79.



step 7: Insert 12:

12 is smaller than 45 and 15 but larger than 10 so we can inser 12 ~~12~~ on ~~left~~ right of 10



Q) construct BST using - Jan, Feb, March, April, may, June, July, Aug, sept, octo, Nov, Dec.

Q) construct BST using - 10, 15, 11, 99, 7, 55, 46, 79, 80, 40.

* operations on Binary search Tree.
- we can perform various operations on BST.

1) search a key
2) Insert a key
3) Delete a key
4) Traverse the tree.

~~1) search a key~~ :-

* creating a BST :-

Algorithm to create a BST.

step 1: start

step 2 : root = NULL

step 3: read key and create a new node to store key value.

step 4: if root is NULL then new node is root. t=root

step 5: while (t ≠ null)

case 1: if key < t → data
attach new node to t → left

case 2: if key > t → data
attach new node to t → right

case 3: if t→data = key
print key already exist

step 6: stop.

*Algorithm to search a element is BST

Step 1: start

step 2: initialize t=root, flag = false

step 3: while (t ≠ null)

Case 1: if t.data = key
then
flag = true
[end if]

if key < t→data
then
t= t→ left
[end if]

if key > t →data
t= t→ right
[end if]

Step 4: if flag = true
then
display "key is found at node", t
else
display "key is not exist"
[end if]

step 5: stop

* AVL Tree :-

- AVL Tree can be defined as height balance binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height the height of right sub-tree from that of it's left sub tree
i.e.

Balance factor = [height of left sub tree] - [height of right sub tree]

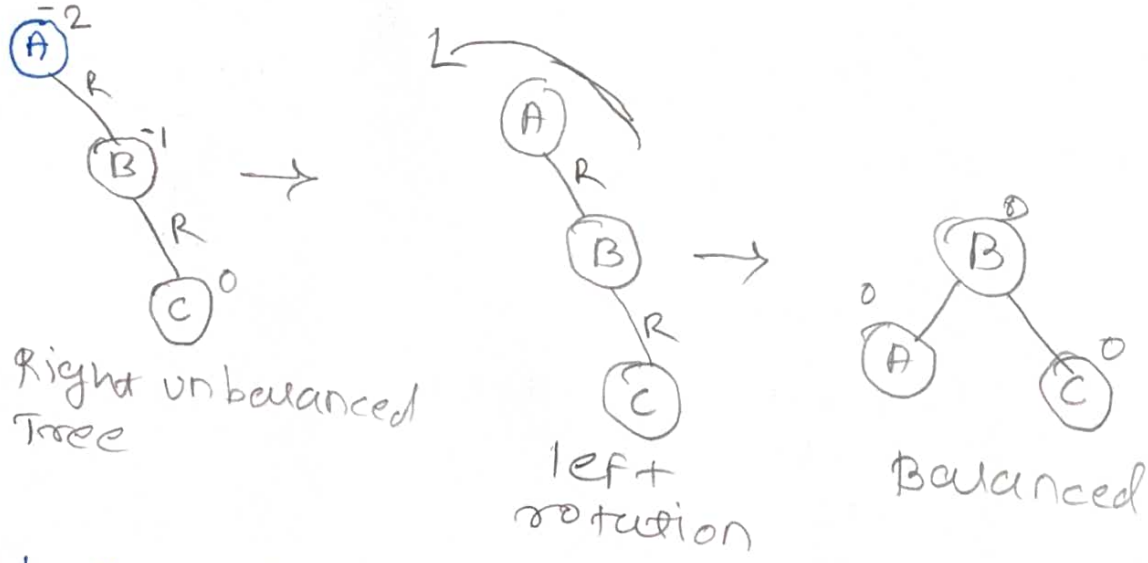
Tree is said to be balance if balance factor of each node is in between -1 to 1 i.e (-1, 0, 1), otherwise three is said to be unbalance and need to be balance.

* Insertion of element is AVL Tree.

- Insertion of element is AVL tree is same as insertion is BST.
- in the AVL tree after insertion of every element, the Balance factor of nodes in the tree may get affected and tree will get unbalanced. so to balance this tree we need to perform some rotations.
- Their are four types of rotations are possible.
  1) LL Rotation.
  2) RR Rotation.
  3) RL Rotation
  4) LR Rotation.

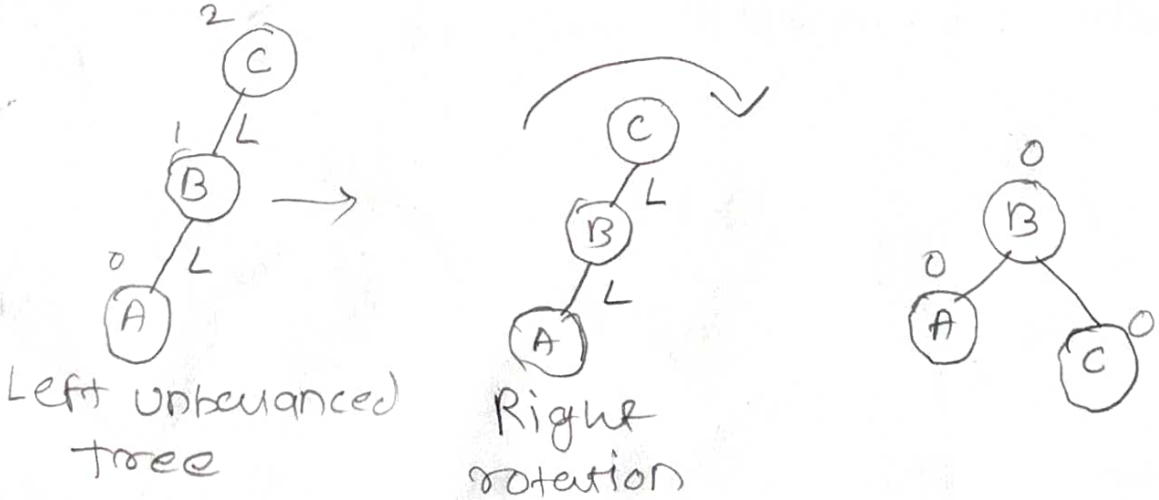
1) RR Rotation :.

- when tree become unbalanced, due to a node is inserted in to the right subtree of the right subtree of A. Then we perform RR rotation.
- RR rotation is an anticlockwise rotation.

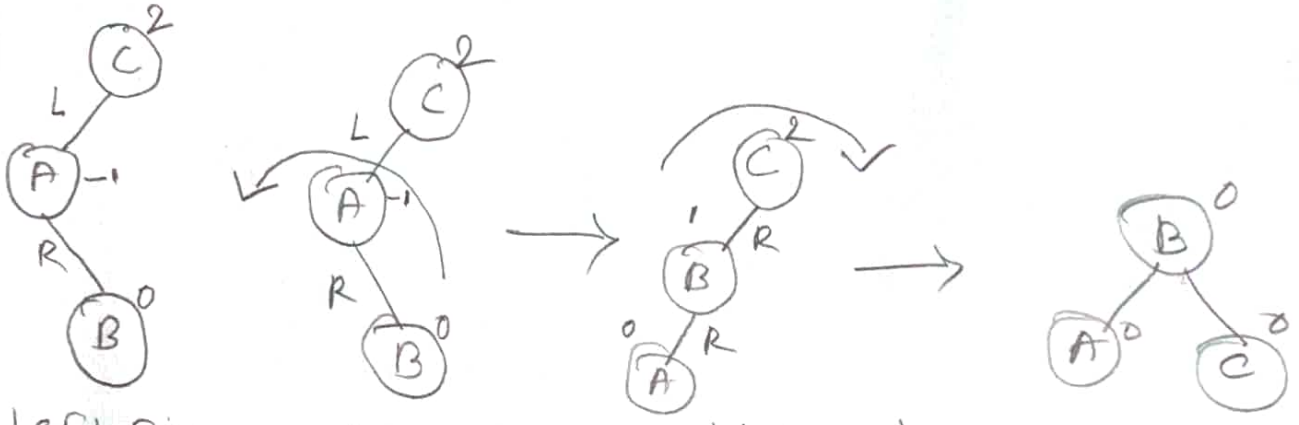


2) LL Rotation :

- when tree become unbalanced, due to a insertion of node in left subtree of left-sub tree of C. Then we perform LL Rotation
- LL rotation is clockwise rotation.



LR Rotation:
- in LR Rotation we need to perform two rotation First RR Rotation is performed on subtree and then LL Rotation is performed on full tree. full tree means first node from the path of inserted node whose balanced factor is other than -1, 0 or 1.

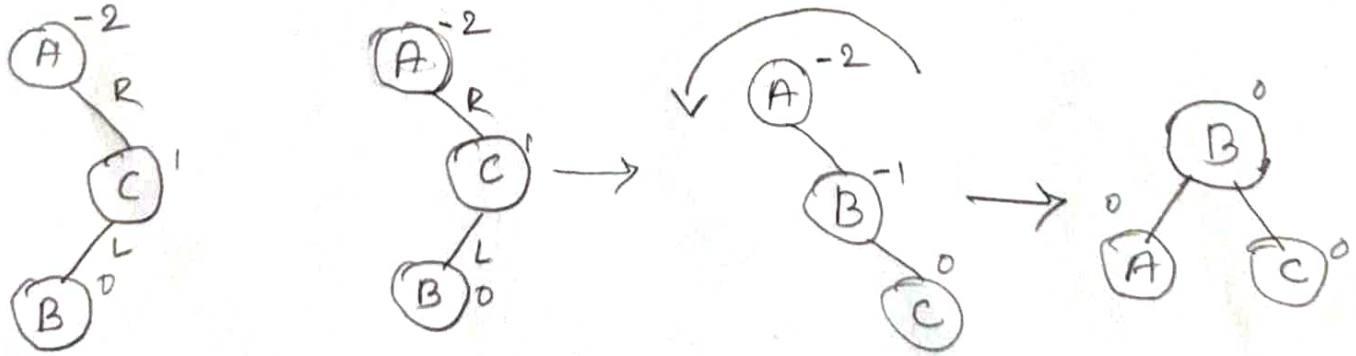


Left Right unbalanced Tree

RR Rotation on subtree

LL Rotation on

RL Rotation :-
- in RL rotation we will perform ~~to~~ 2 Rotation First LL rotation is performed on subtree and then RR rotation is performed on full tree, Full tree means ~~path From~~ the first node from the path of inserted node whose balanced factor is other than -1, 0, 1



Right left unbalanced tree

LL Rotation on subtree

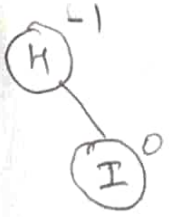
R R Rotation

Q. construct an AVL tree having following element H, I, J, B, A, E, C, F, D

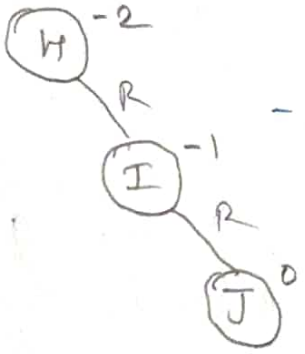
→

Step 1:
(Insert H)

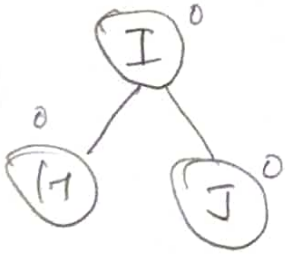
step 2:
(Insert I)



Step 3:
(Insert J)

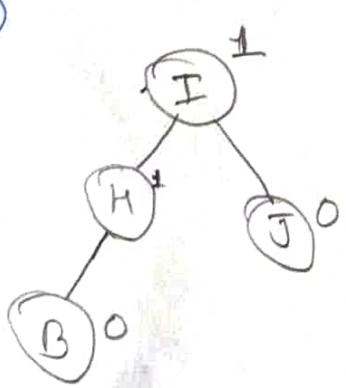


- Here after insertion of J, tree become unbalance as the balance factor of H is -2
- Here we need to perform RR Rotation.

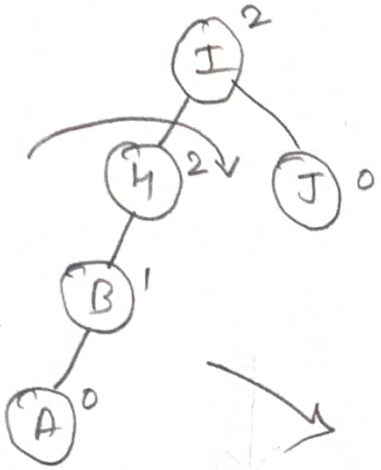


(balanced)

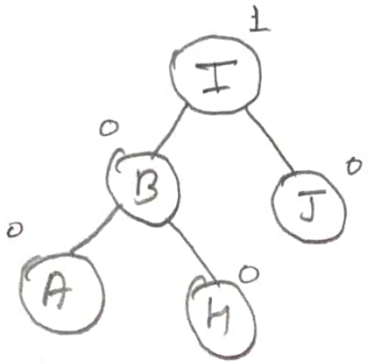
Step 4:
(Insert B)



step 5: (Insert A)

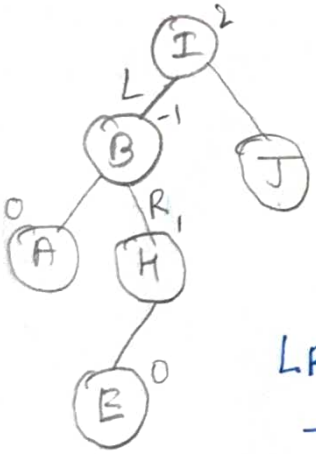


- After insertion of A tree has become unbalanced as the balance factor of H is 2
- so we will perform LL Rotation



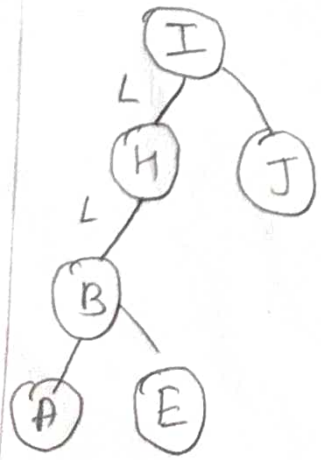
(balanced)

Step 6: (Insert E):

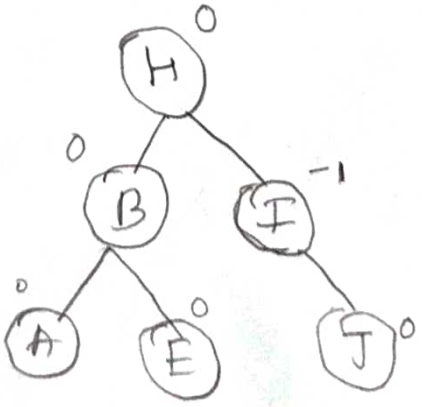


- ~~After~~ Insertion of ~~tree~~ E tree has become unbalanced as the balance factor of I is 2.
- Here we need to perform LR Rotation.
- ~~first rotation is~~

RR Rotation

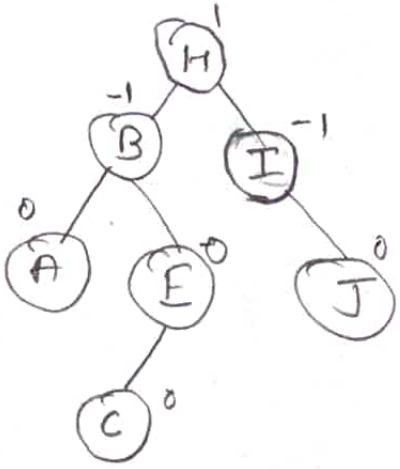


LL Rotation

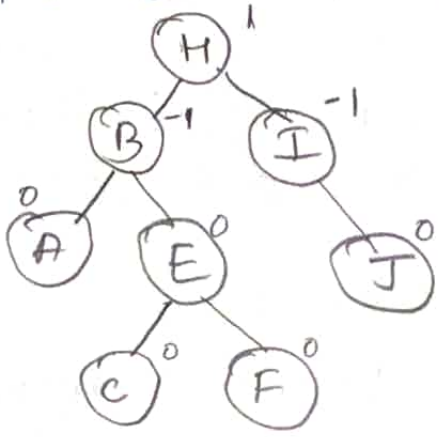


(balanced)

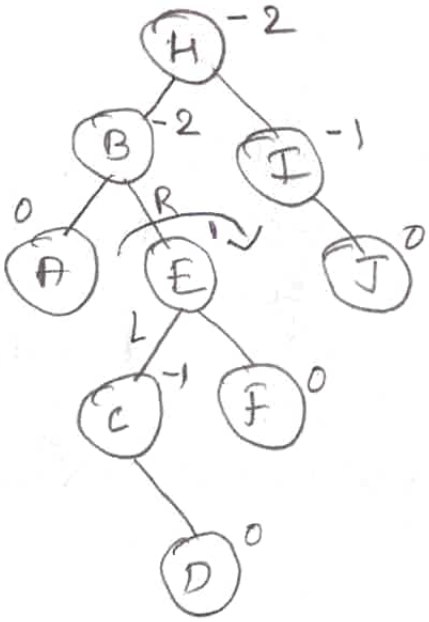
step 7: (insert C)



step 8: (Insert F)

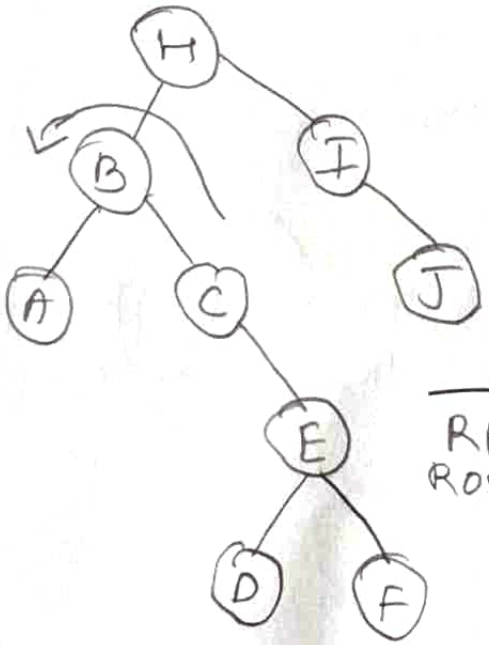


Step 9: (Insert D)

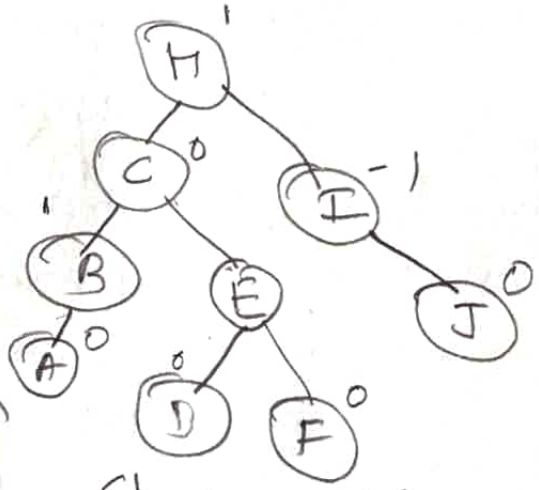


- After insertion of D tree has become unbalanced as the balance factor of B is -2.
- Here we can balance tree by using RL Rotation

LL Rotation.



RR Rotation



(balanced)

Q) construct AVL tree for following elements

1) 50, 100, 20, 40, 80, 50, 90, 70

2) sun, mon, Tue, wed, thur, fri, sat.

3) 7, 99, 88, 50, 28, 42, 72, 10

*segment tree.

- segment tree is basically binary tree used for storing the interval or segments.
- segment tree provide two operations
  1. Update
  2. Query (min/max/sum)
- segment tree is used in case of where there are multiple range queries on Array or modification of the same array.
example - finding the sum of all the elements in an array from indices L to R or finding minimum of all the elements in an array from indices L to R

*construction of segment tree.

- consider Array of size N and corresponding segment tree T.-1
- Then root T will represent whole Array A[0:N-1]
- Then it is broken down in to half interval or segment and two childern of the root represents as

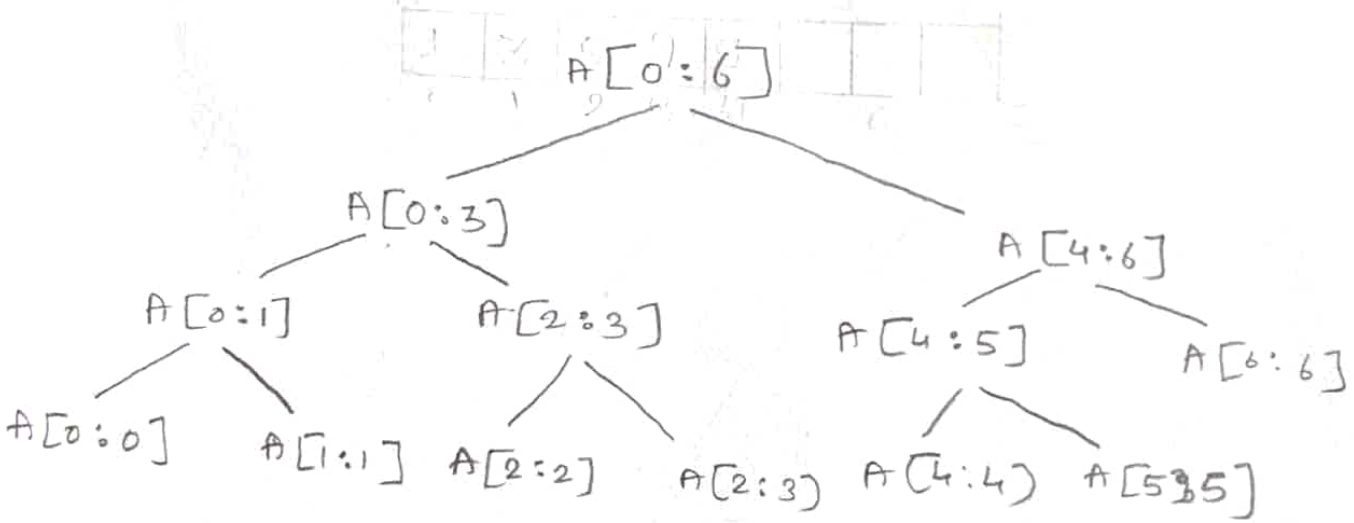
A[0:(n-1)/2]

and

A[(N-1)/2+1:(n-1)]

so, in each step The segment is devided in to half and the two childern represent those two halfs.

- once segment tree is built it's structure cannot be changed.
- Segment tree of Array A of size 7 will look like. ~~A = {1, 3, 5, 7, 9, 11}~~



example:

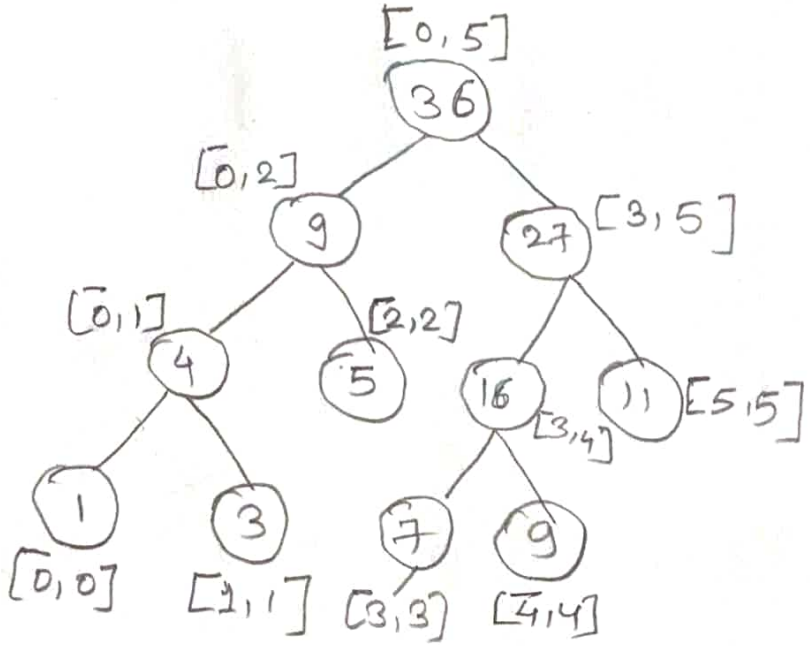
Q> Construct segment tree for A = {1, 3, 5, 7, 9, 11} and Find

1. max(1, 3)
2. max(3, 5)
3. min(2, 4)
4. sum(1, 4)
5. Update: set arr[3] = 12.

=> First we construct segment tree then we will solve queries one by one.

1	3	5	7	9	11
0	1	2	3	4	5

The segment tree for maximum range query is as follow.



1. max(1,3) = 7
2. max(3,5) = 11
3. min(2,4) = 5
4. sum(1,4) = 24
5. Update
   set arr[3] = 12
   [After update element at index 3 will get updated from 7 to 12]

Red Black Tree.

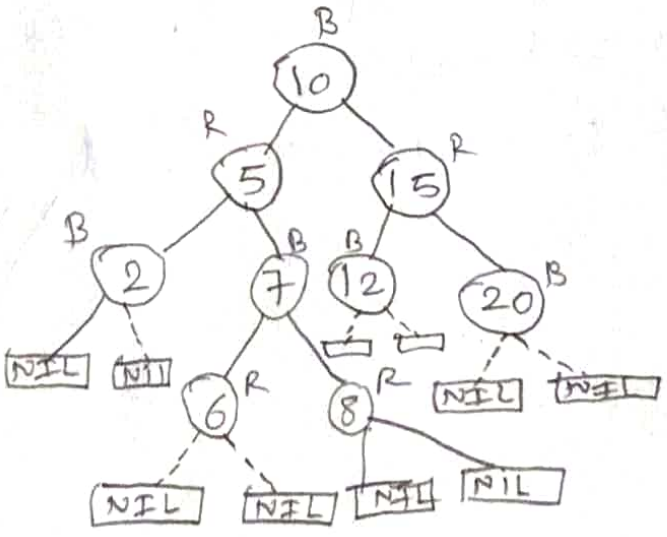
A red black tree is a kind of self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color, that is red or black.

These color are used to ensure that the tree remain balanced during insertion and deletion.

Rules To construct Red-Black Tree:-

1) The root of the tree is always black.
2) Every node has a color either red or black
3) There are no two adjecent red nodes.
4) Every path from a node to any of it's descendant NULL nodes has the same number of black nodes.
5) All leaf nodes are black nodes.

The Figure below shows Red-Black Tree.



* B- Black
R- Red

Fig- Red-black Tree

When our application need to perform frequent insertion or deletion, Then red-black tree should be Preferred. and if insertion and deletion are less frequent then AVL Tree should be Preferred.

eg. Create Red-Black tree by inserting following sequence of elements.

10, 18, 7, 15, 16, 30, 25, ~~40~~

1) if tree is empty, create new node as root node with color black.
2) if tree is not empty, create new node as leaf node with color Red
3) if Parent of new node is black Then exit.
4) if Parent of new node is Red, Then check the color of parent's sibling of new node.
   a) if color is black or NULL Then do suitable rotation and Recolor.
   b) if color is Red, Then recolor and also check if Parent's parent of new node is not root node, then recolor.
5) No two adjecent red nodes
6) count no of black nodes in each node.

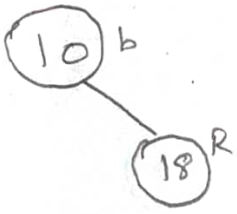
⇒

Step 1:-
Insert (10) -

Here Tree is empty, so insert new node as Root node with black color.

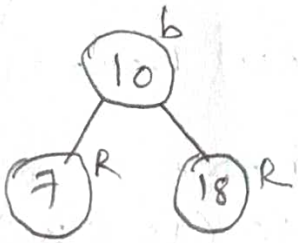
Step 2:
Insert 18

Here Tree is not empty, so insert new node with Red color.



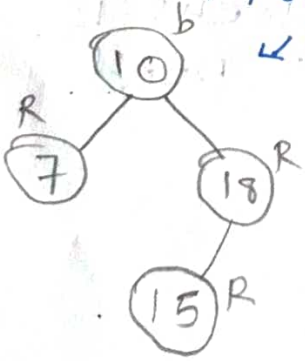
Step 3:
Insert 7

Tree is not empty. So insert new node with Red color.



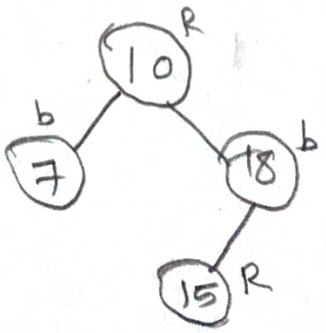
Step 4:
Insert 15

1) Tree is not empty so insert (15) with Red color.



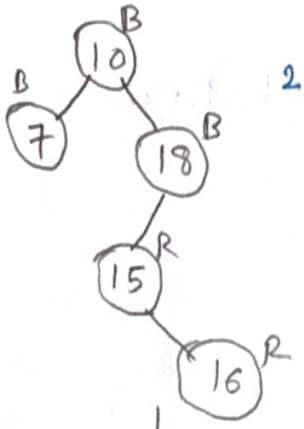
2) Here there are two consecutive Red nodes (15 and 18) & The new nodes Parent's sibling color is red so we Recolor to make it Red black Tree.

Recoloring



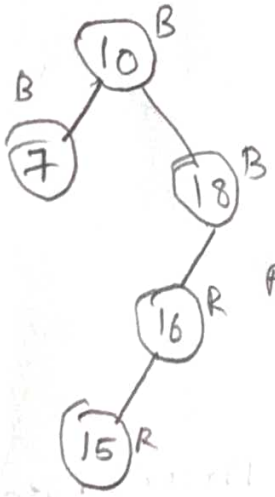
step 5:

Insert 16 - 1. Insert new node with Red color.

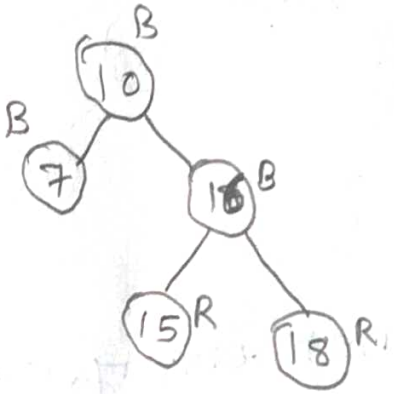


2. Here there are two consecutive Red nodes (15 & 16). Here new node's Parent sibling is NULL so we need rotation.
   - Here we perform LR rotation and recolor.

↓ after left rotation

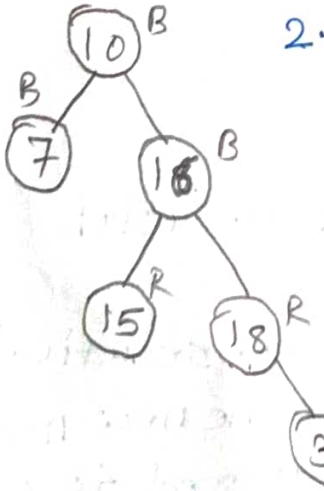


→ After Right rotation



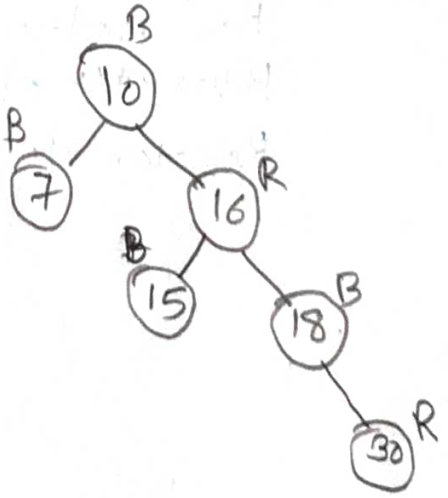
step 6:

Insert 30: 1. Insert new node with Red color.



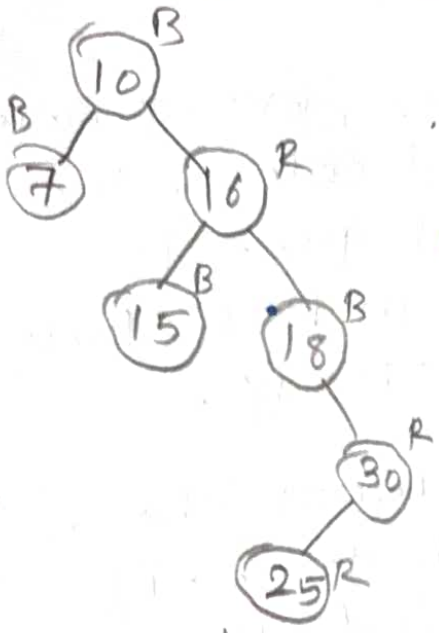
2. Here there are two consecutive Red nodes (18 & 30), parent sibling is Red and parent's parent is not root node, so we need Recolor only

→ After Recolor



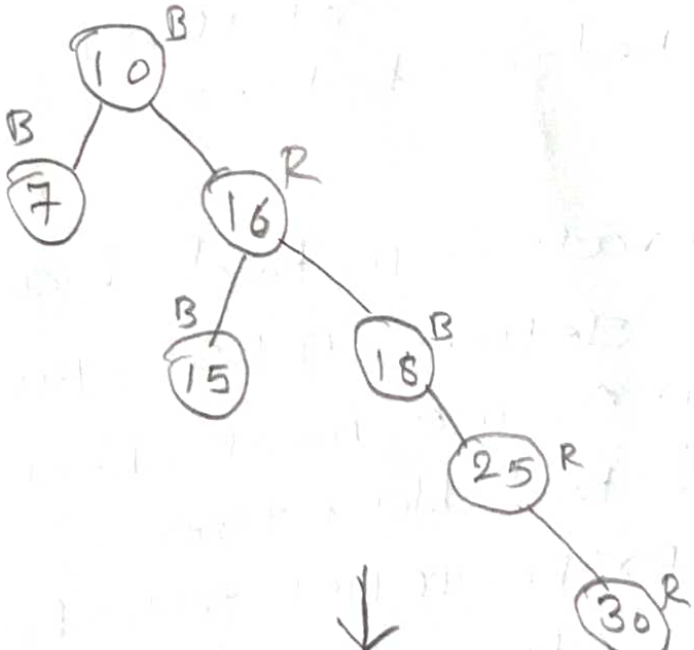
step 7:

Insert 25 Insert new node with Red color.

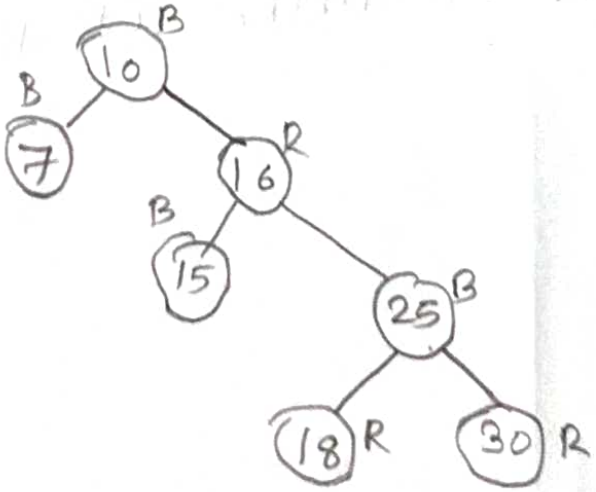


- Here there are two consecutive Red nodes (30 & 25). Parent sobling is NULL, so Here we need RL Rotation & then recolor.

After right rotation



After left rotation



* operations on Red black Tree.

1) Insertion: In a Red black tree, every new node must be inserted with Red color. The insertion operation in Red black tree is similar to insertion operation in BST. But it is inserted with a color property.
- After every insertion, we must check all the properties of red-black tree. if all the properties are satisfied, then we may move further otherwise we perform the operation like rotation, recolor to make it Red-black tree.

2) Deletion :-

- The deletion operation in Red-black tree is similar to deletion in BST. But after every deletion, we must check all the properties of Red-black tree.
- if any any of the property not satisfied ne need to perform operation like rotation, recolor to make it Red-black Tree.