UNIT-4

Splay Trees : Simple idea, splaying
Red black Trees : Definition, insertion & deletion operations with examples.

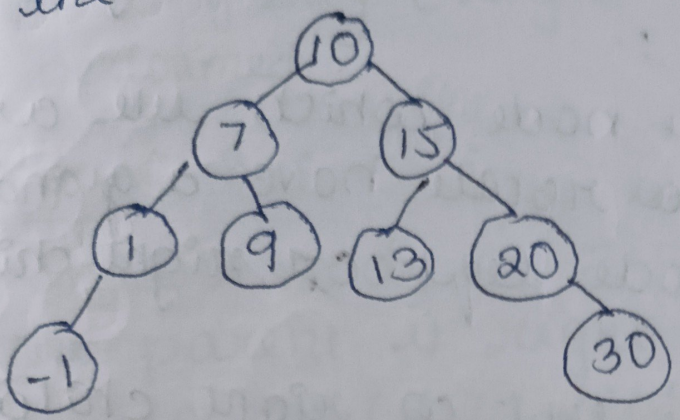
Splay Tree :- Simple Idea

Splay trees are self balancing or self adjusted binary search trees. In other words we can say that splay trees are the variants of binary search trees the prerequisite for splay trees that we should know about the binary search trees.

As we already know, the time complexity of a binary search tree in the average case is O(log n) & the time complexity in the worst case is O(n). In a binary search tree the value of the left subtree is smaller than the root node & the value of right subtree is greater than the root node; in such case the time complexity would be O(log n). If binary tree is left skewed or right skewed then the time complexity would be O(n). To limit the skewness, the AVL & red black tree came in to the picture having O(log n) time complexity for all the operations in all the cases.

Let's understand the search operation in Splay tree

Suppose we want to search 7 element in the tree which is shown below



To search any element in Splay tree first we will perform standard binary search tree operation. As 7 is less than 10 so we will come to the left of the root node. After performing the search operation, we need to perform splaying here splaying means that the operation that we are performing on any element should become the root node after performing some rearrangement of the tree will be done through the rotations

Rotations :-

1. zig rotation (right rotation)
2. zag rotation (Left rotation)
3. zig zig rotation (zig FOLLOWED BY TWO right rotation)
4. zag zag rotation (Two left rotations

5. zigzag (zig followed by zag)
6. zag zig (zag followed by zig)

factors required for selecting a type of rotation :-

→ Does the node which we are trying to rotate have a grandparent
→ Is the node left or right child of the parent
→ Is the node left or right child of the grandparent.

Cases for the rotations :-

Case 1 :-

If the node does not have a grandparent & if it is right child of the parent, then we carry out the left rotation otherwise the right rotation is performed

Case 2 :-

If the node ~~is the right~~ has a grandparent, then based on following scenarios the rotation would be performed.

Scenario 1 :- If the node is right of the parent & the parent is also right of its parent then zig zig (right right) rotation is performed

Scenario 2 :-

If the node is left of a parent, but the parent is right of a parent then zig zag (right left) rotation is performed.

Scenario 3 :-

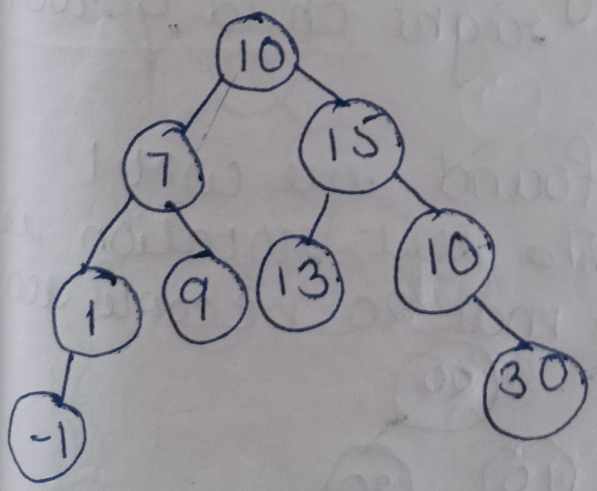
If the node is right of the parent & the parent is right of its parent, then zig zig (left left) rotation is performed.

Scenario 4 :-

If the node is right of the parent but the parent is left of its parent then zig zag (right left rotation is performed

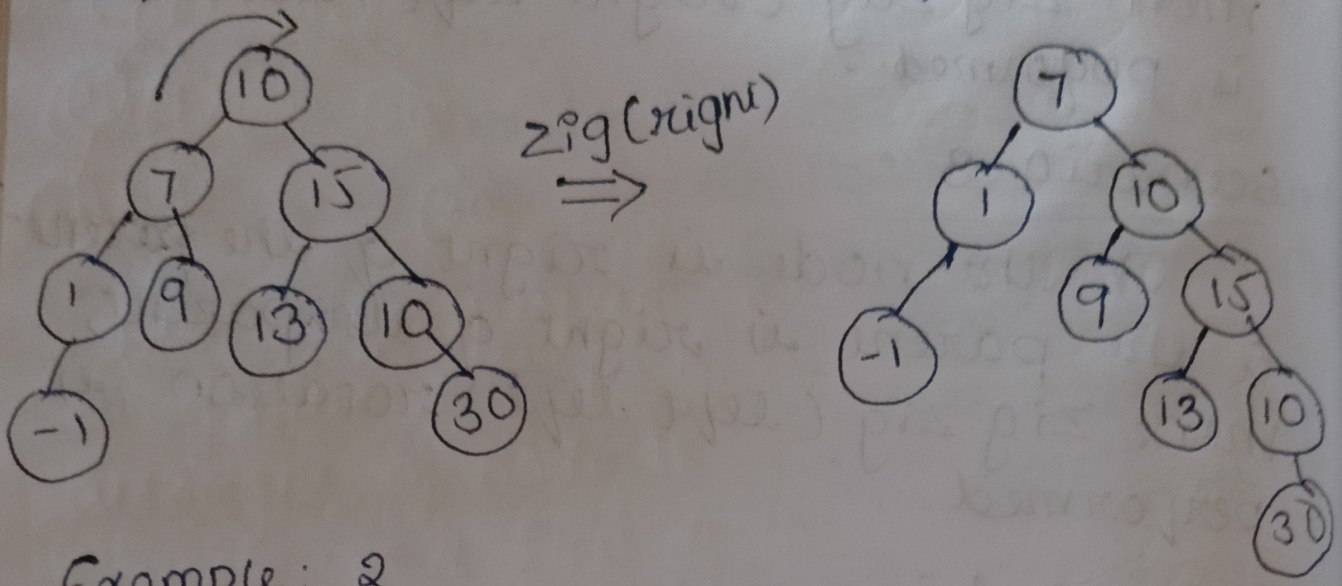
Example :-

Seach 7 in below tree



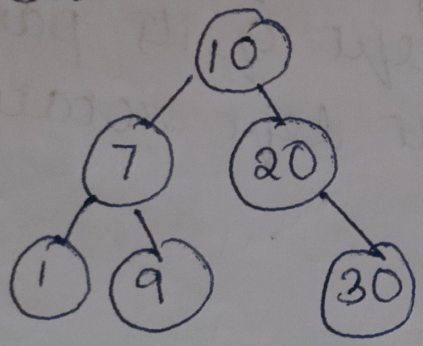
→ First we compare 7 with root node As 7 is less than 10, so it is a left child of root node.

Step 2: once the element is found we will perform splaying. The right rotation (zig) is performed so that 7 becomes root node of tree



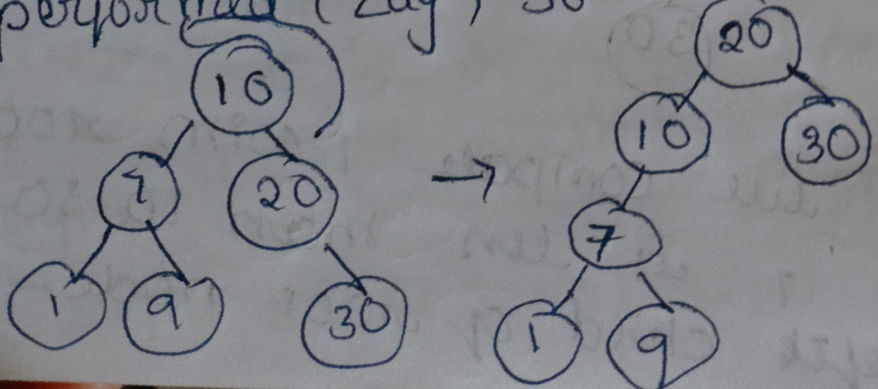
Example: 2

Seach 20



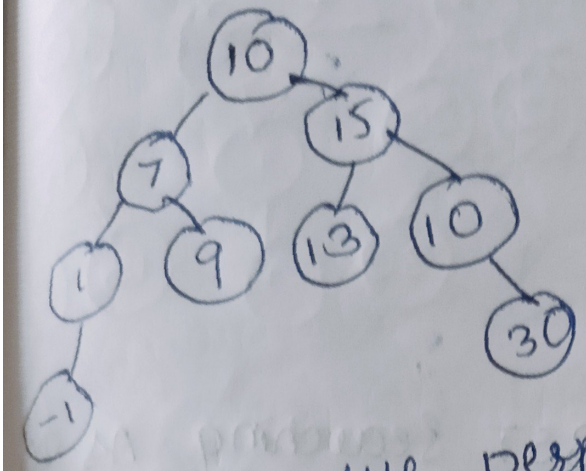
→ First we compare 20 with root node. As 20 is greater than the root node so it is right child of root node.

→ once element is found we will perform splaying the left rotation is performed (zag) so that 20 becomes root



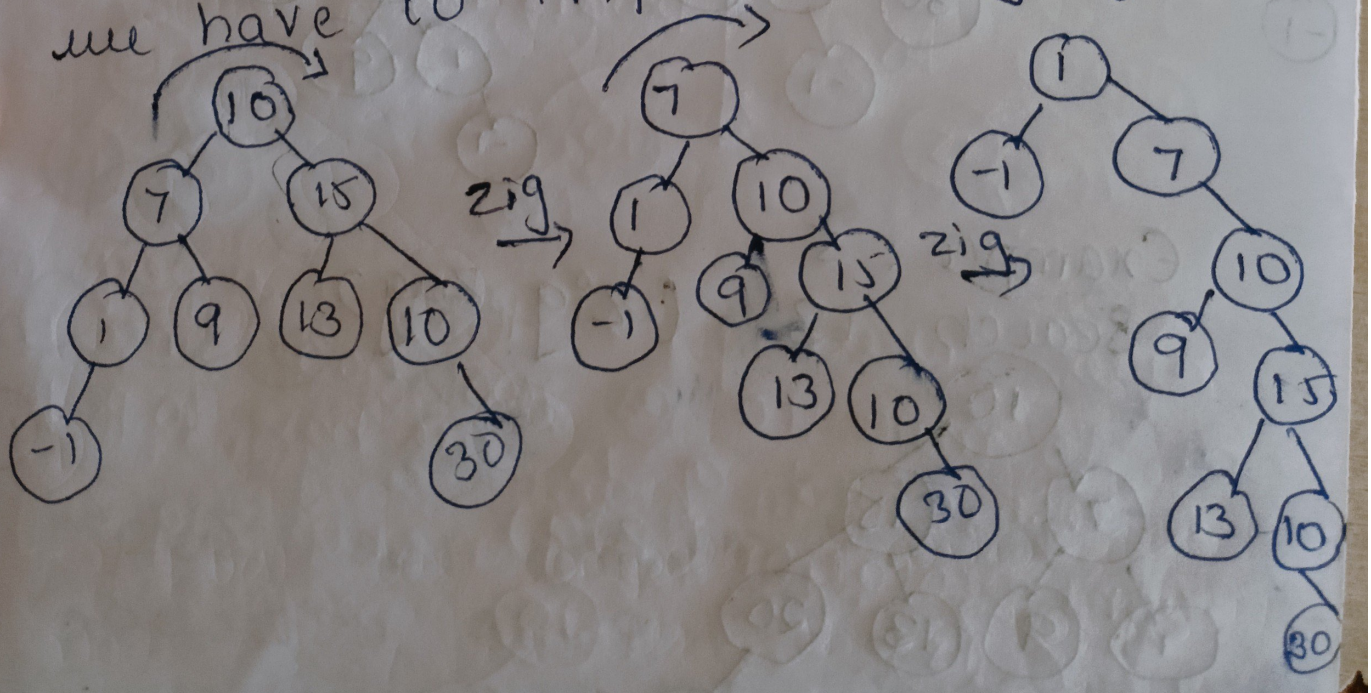
Example 3

Search 1 in tree



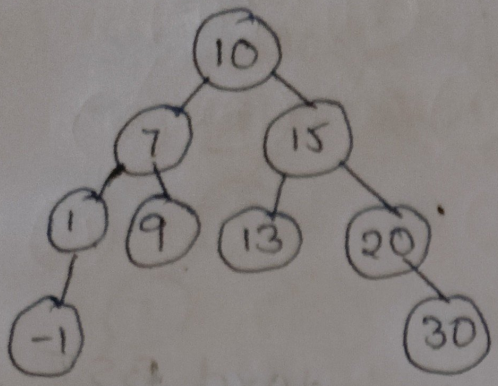
→ First we perform a standard BST searching operation in order to search the 1 element. As 1 is less than 10 & 7 so it will be at left of node 7 therefore element 1 is having a parent i.e 7 as well as grandparent i.e 10

→ In this step we have to perform splaying we need to make node 1 as a root node with help of some rotations. In this case we cannot simply perform a zig or zag rotation we have to implement- zig-zig rotation



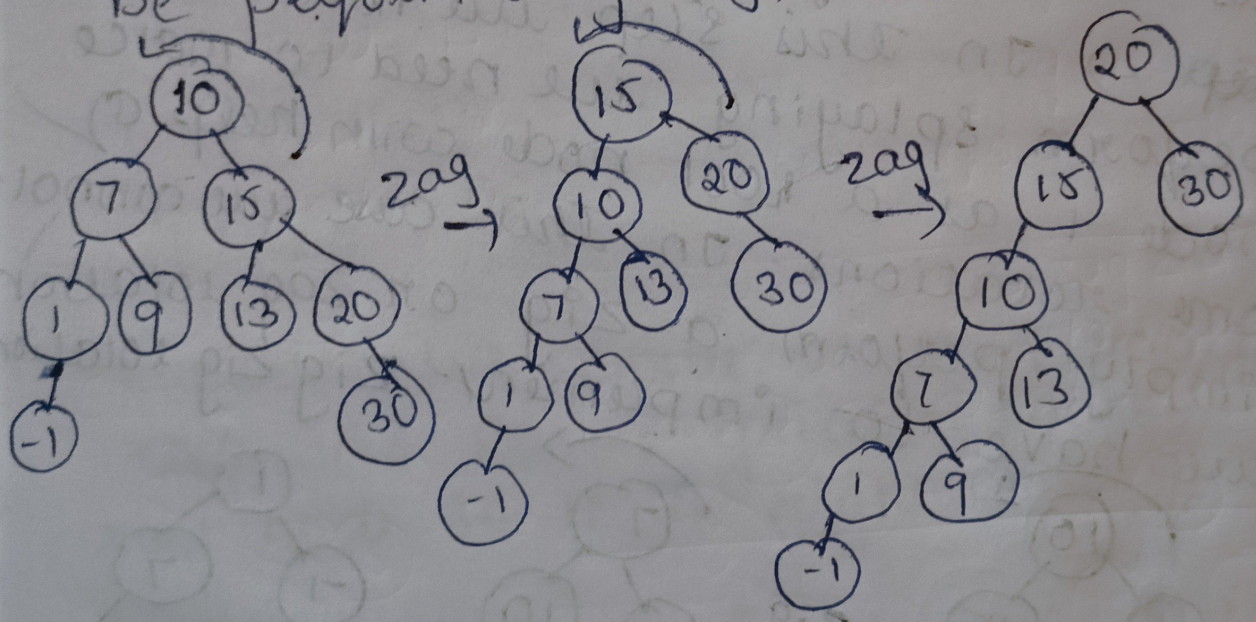
Example 4 (zag zag)

Search 20



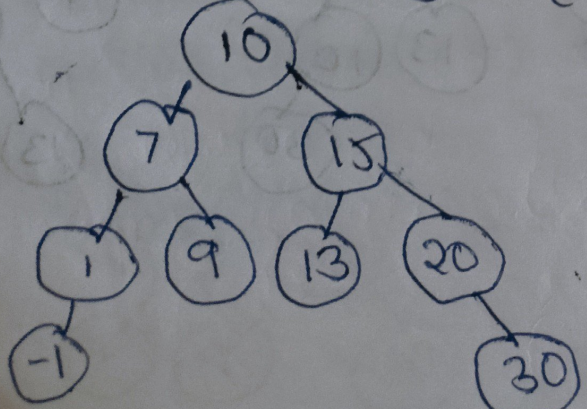
First we perform BST searching. As 20 is greater than 10 & 15 so it will be at right of node 15

→ In second step we to perform splaying In this case two left rotations would be perform (zag zag)

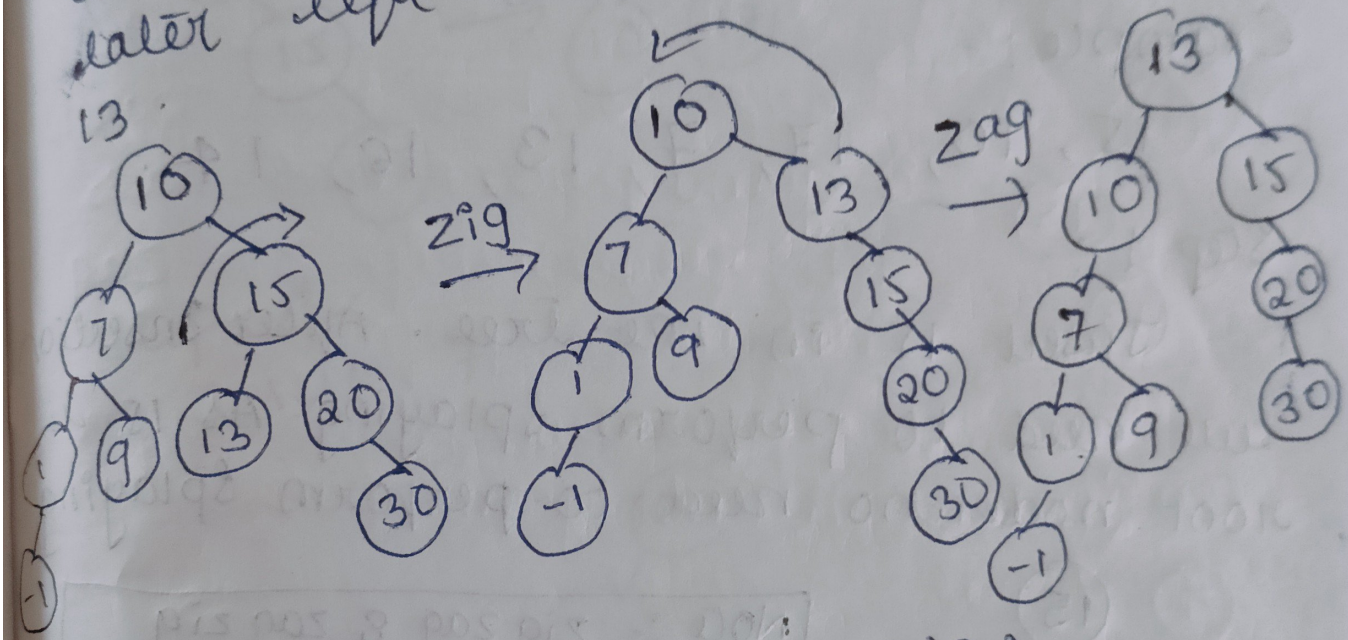


Example : 5

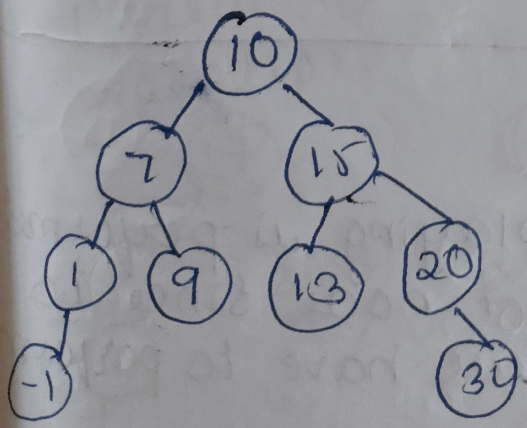
Search 13 (zig zag)



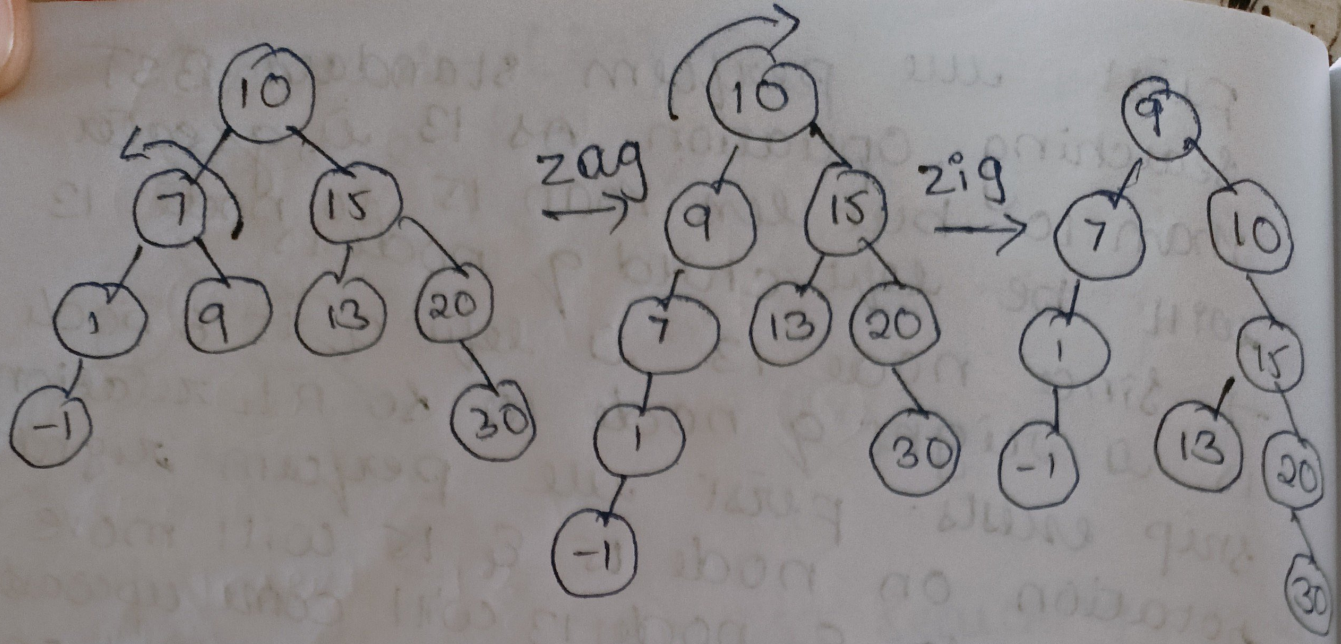
First we perform standard BST searching operation. As 13 is greater than 10 but less than 15 so node 13 will be left child of node 15.
→ Since node 13 is left of 15 is a node is a right of node 10 so RL relation ship exists. First we perform right rotation on node 15 & 15 will move downwards & node 13 will come upwards later left rotation is performed on 13.



Example : 6 (zag zig)
Search 9



First we perform standard BST operation as 9 is less than 10.



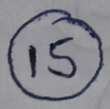
Insertion in Splay Trees :-

Example :-

15, 10, 17, 7, 13, 16, 14;

Step 1 :-

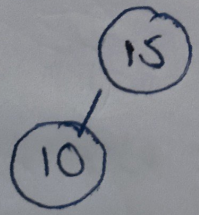
Insert 15 in the tree. After Insertion we need to perform Splaying. As 15 is root node no need to perform Splaying



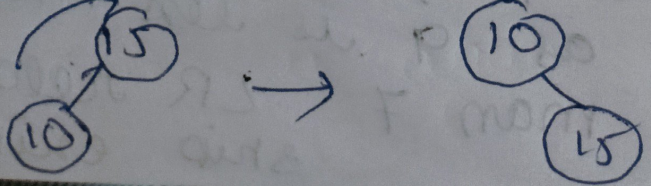
NOTE :- zig zag & zag zig operation is perfomed on parent not grand parent.

Step 2 :-

Insert 10

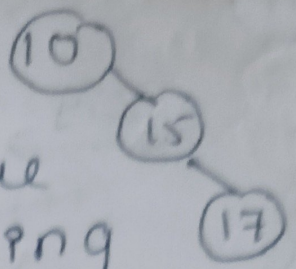


after Inserting 10 Splaying is perfomed on to make 10 as root node since 10 is left child of 15 we have to perform right rotation (zig)

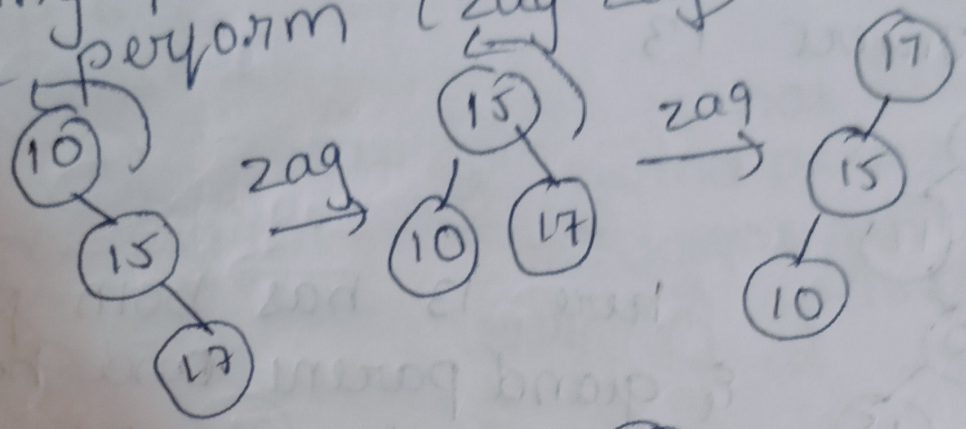


step 3 :-

Insert 17 →

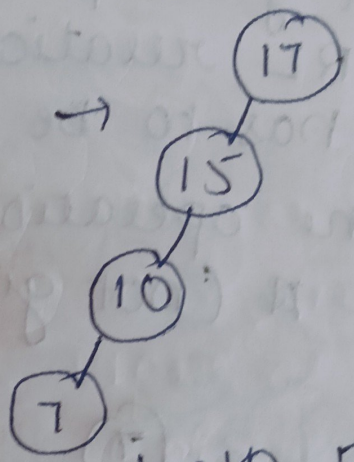


After Inserting 17 we have to perform splaying to make 17 as root node here 17 is having parent & grand parent- here we perform (zag zag)

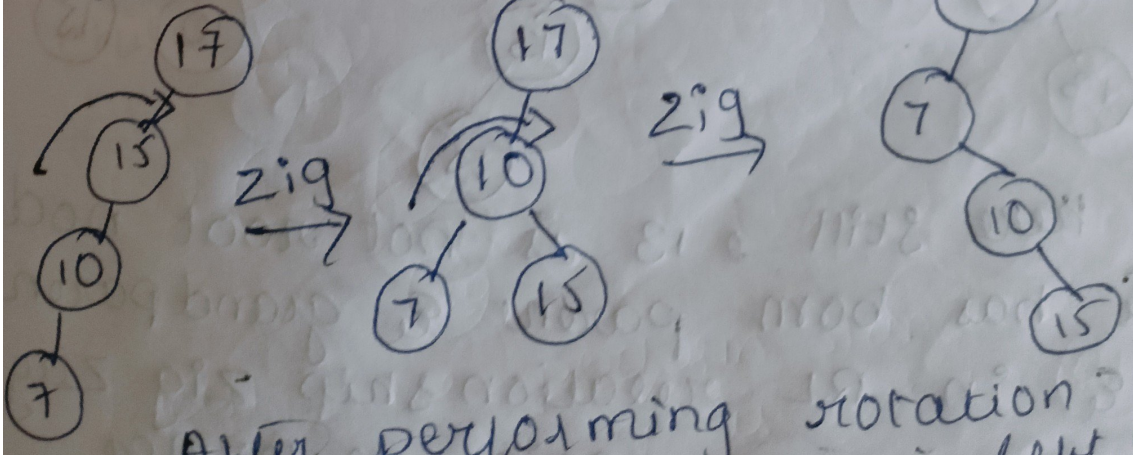


step 4 :-

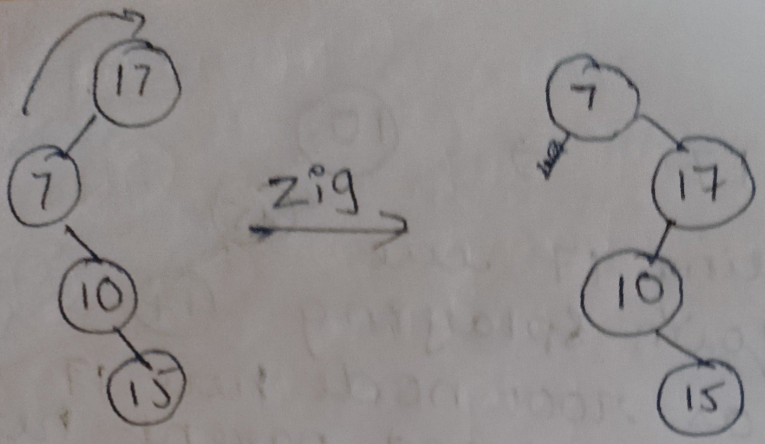
Insert 7 →



Here 7 has both parent & grand parent so we perform (zig zig) rotation

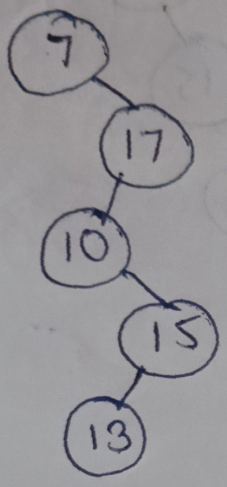


After performing rotation still 7 is not root node 7 is left child node so perform zig rotation



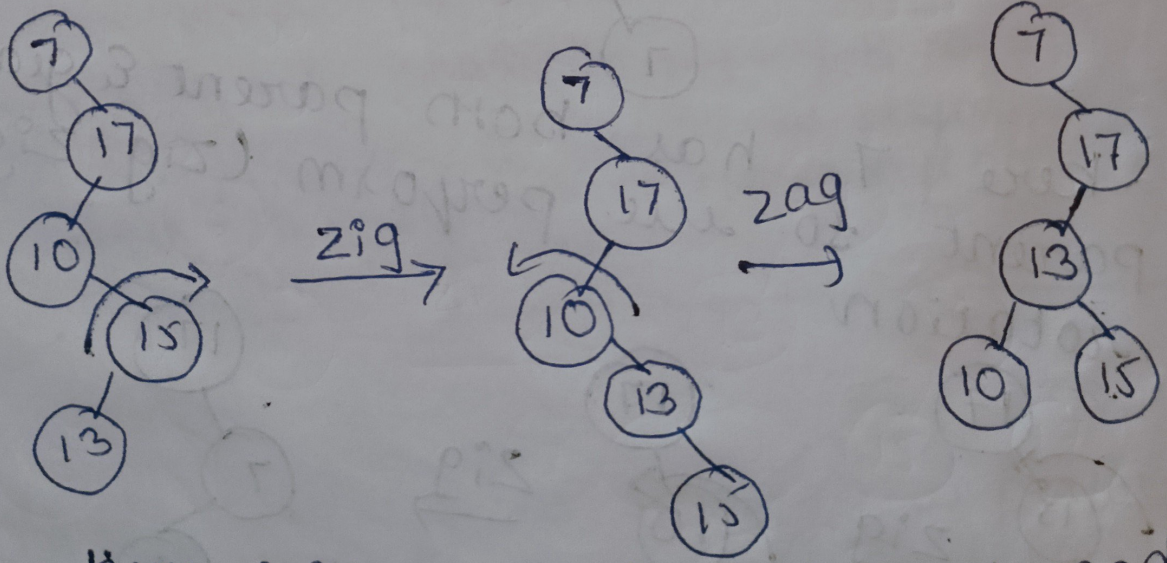
Step 5:-

Insert 13

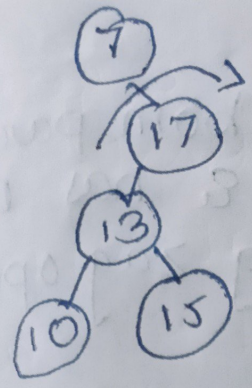


here 13 has bom parent & grand parent and has RL relationship zig zag has to be peyormed.

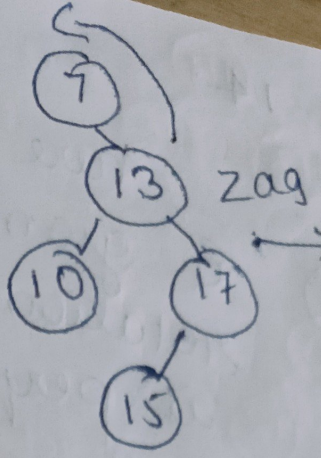
here the operation peyormed yirst is on parent (not grand parent



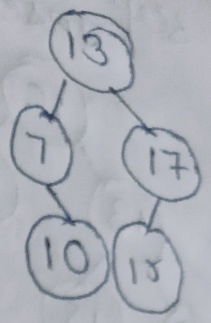
Here still 13 is not root node & has bom parent & grandparent & in RL relationship zig zag rotation - is again peyormed



zig

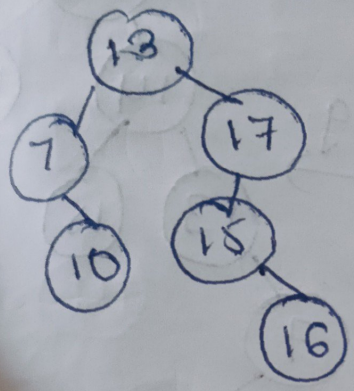


zag

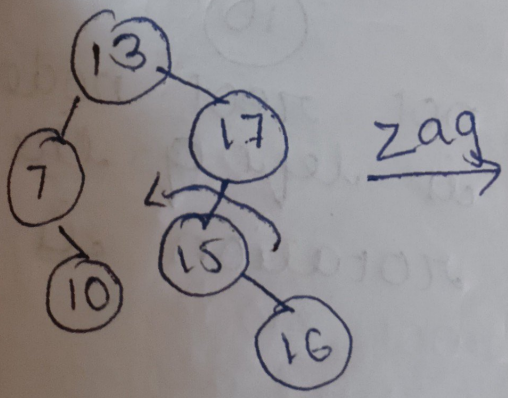


step 6 :-

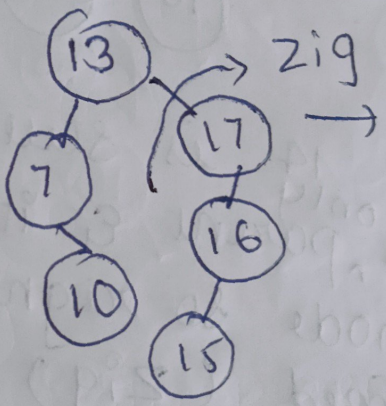
Insert 16



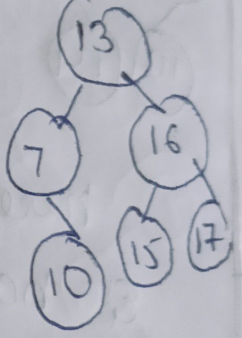
here 16 has bom parent & grand parent & has LR relationship so zag zig operation is performed.



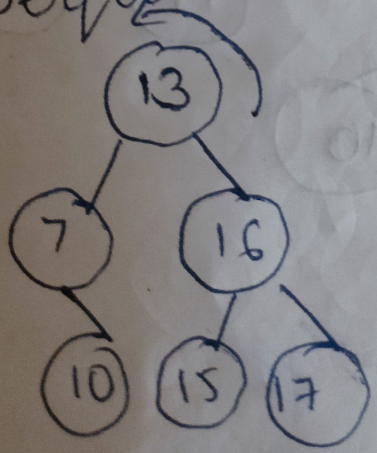
zag



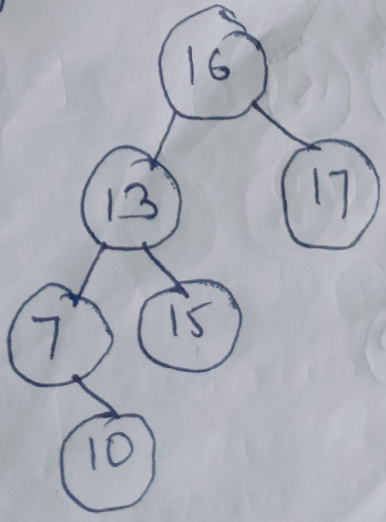
zig



here 16 is still not root node & has only parent & it is right child of root so left rotation is performed (zag)

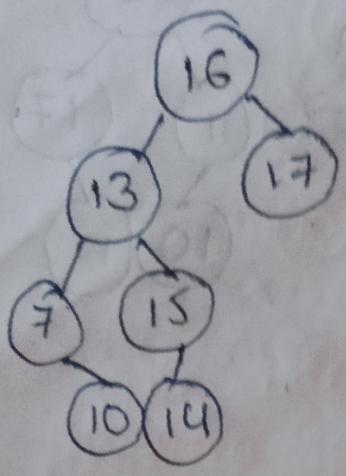


zag

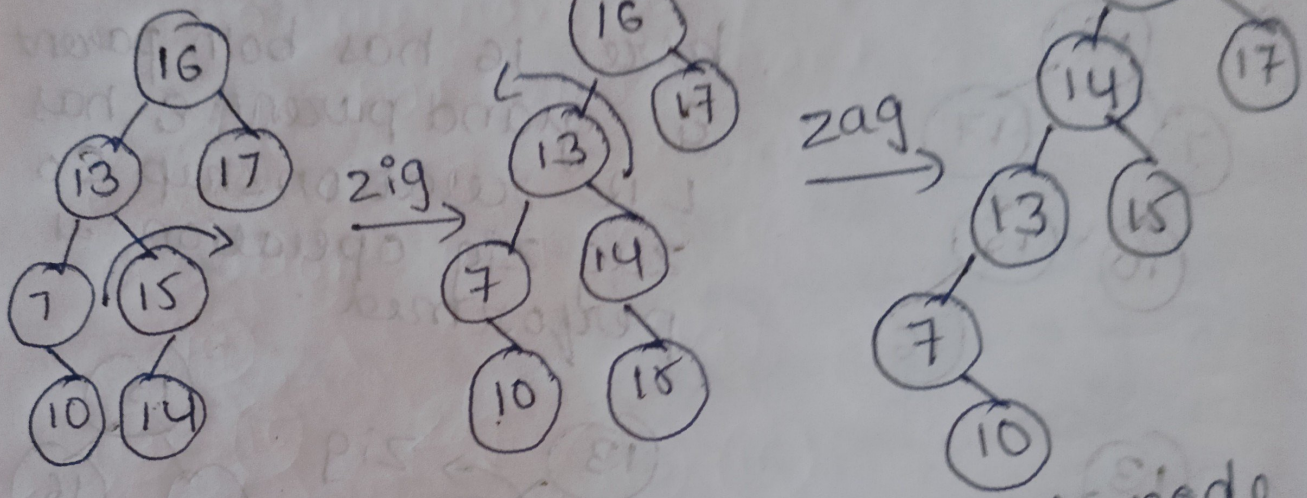


Step : 7

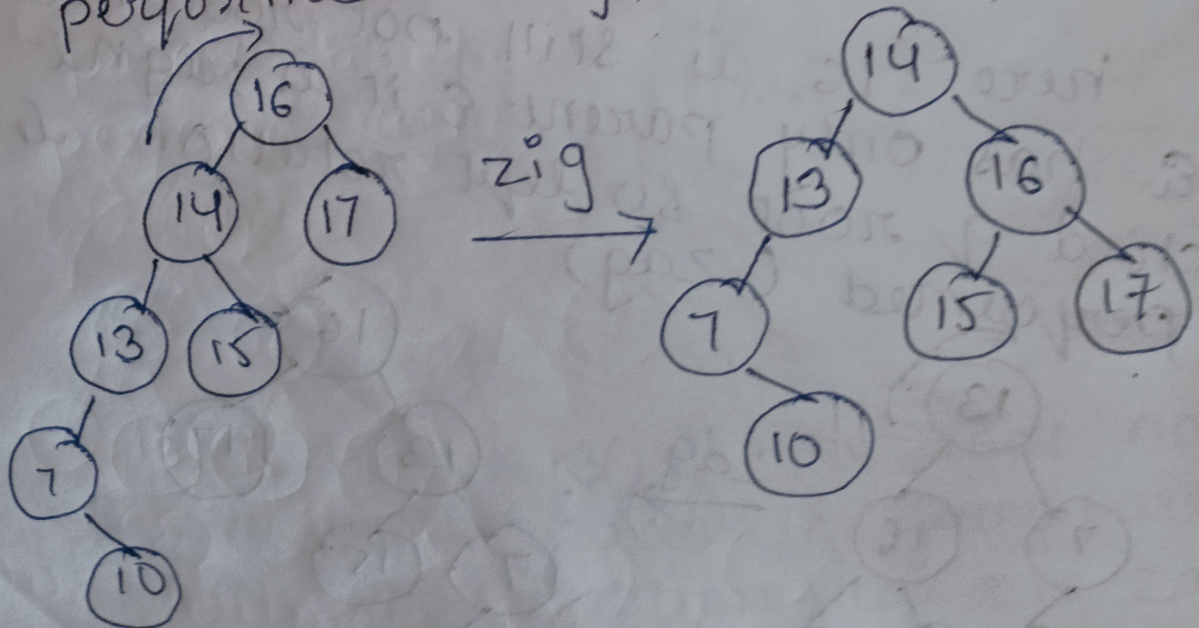
Insert 14.



here 14 has both parent & grand parent & has RL rotation so zig zag operation is performed.



here 14 is still not root node & has only parent & it is left of the root node so right rotation is performed (zig).



Deletion in Splay trees :

Splay Trees are the variants of the binary Seach tree, so deletion operation in the Splay Tree would be similar to the BST but the only difference is that the delete operation is followed in Splay tree by splaying operation

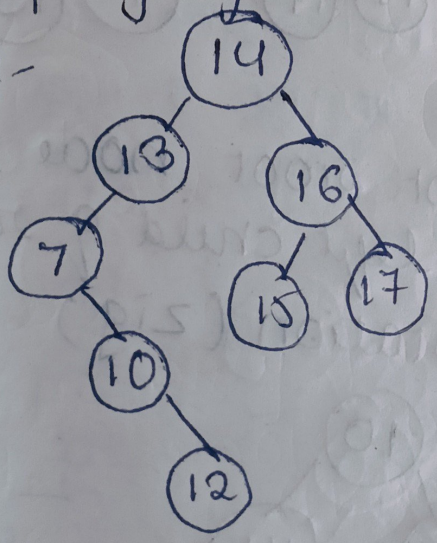
Types of deletions :

1) Bottom up Splaying
2) Top down Splaying

Bottom up Splaying :-

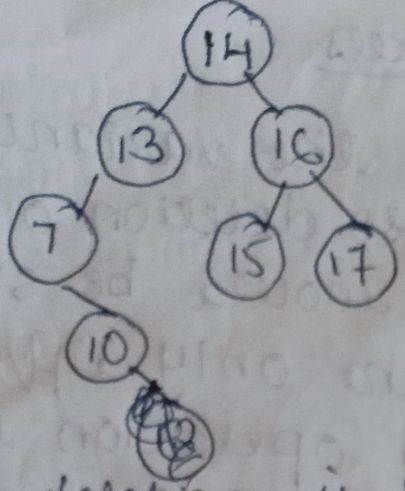
In this method first we delete the element from the tree & then we perform the splaying. on the deleted node.

Ex :-



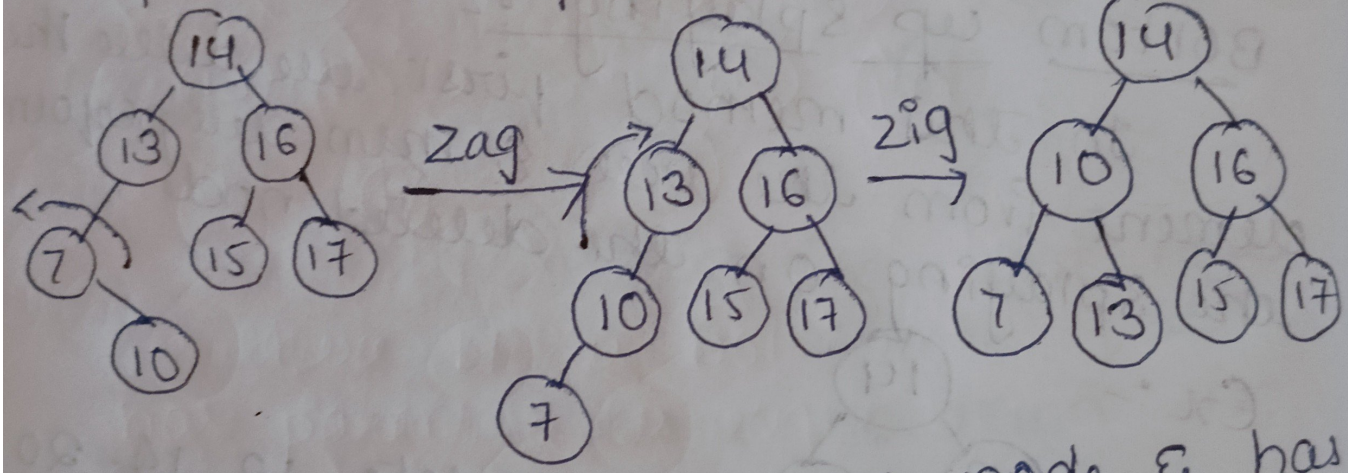
Delete 12, 14, 20 ;

First we simply perform Standard BST deletion operation to delete 12 element. As 12 is leaf node so we simply delete the node from the tree

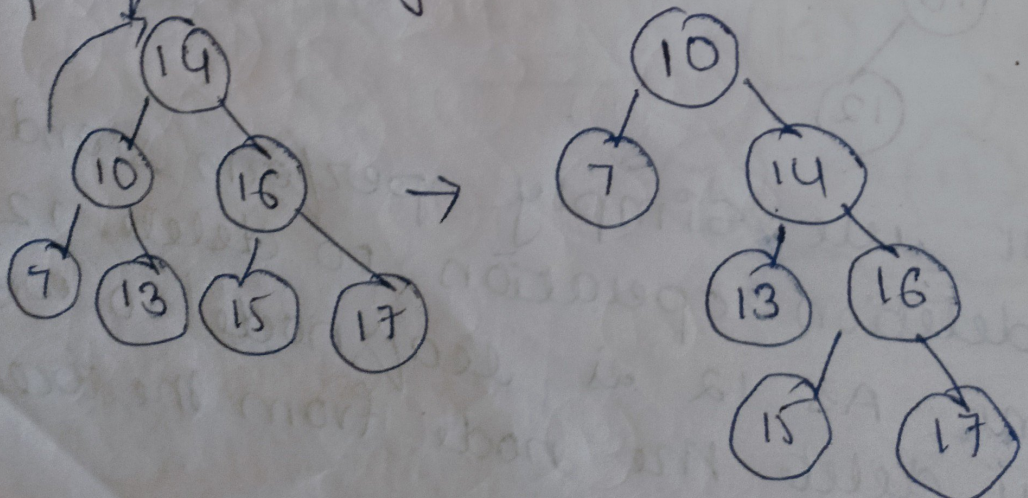


The deletion is still not completed we need to splay the parent of delete node, i.e 10 we have to perform Splay (10)

10 has both parent & grandparent & it in LR relation where zag zig operation is performed.

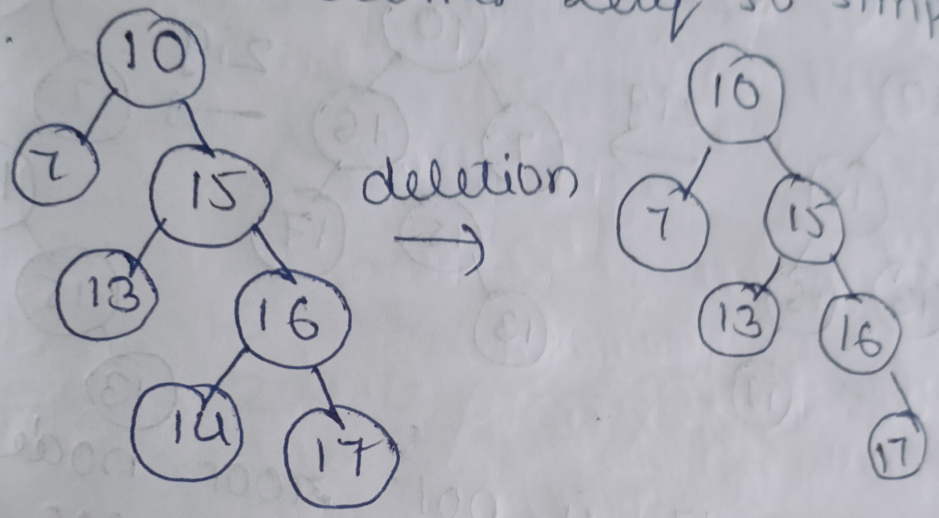


Still 10 is not root node & has parent & it is left child of root so perform right rotation (zig)



Delete 14

As we cannot simply delete the internal node we will replace the value of the node either using inorder predecessor or inorder successor. Suppose we use inorder successor in which we replace the value with lowest value that exist in right subtree of node 14 which is 15 so we replace 14 with 15 since node 14 becomes leaf so simply delete it.

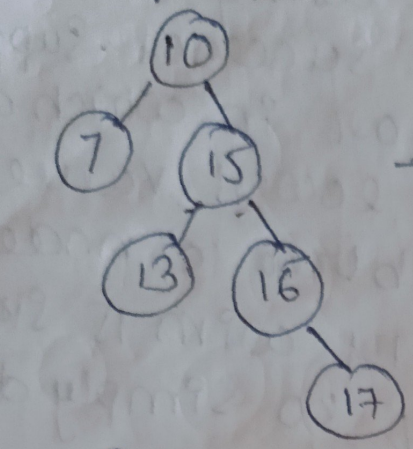


deletion is not completed we need to perform one more operation i.e splaying in which we need to make the parent of deleted node as root node. before deletion the parent of node 14 is 10 as 10 is already root node splaying is not required

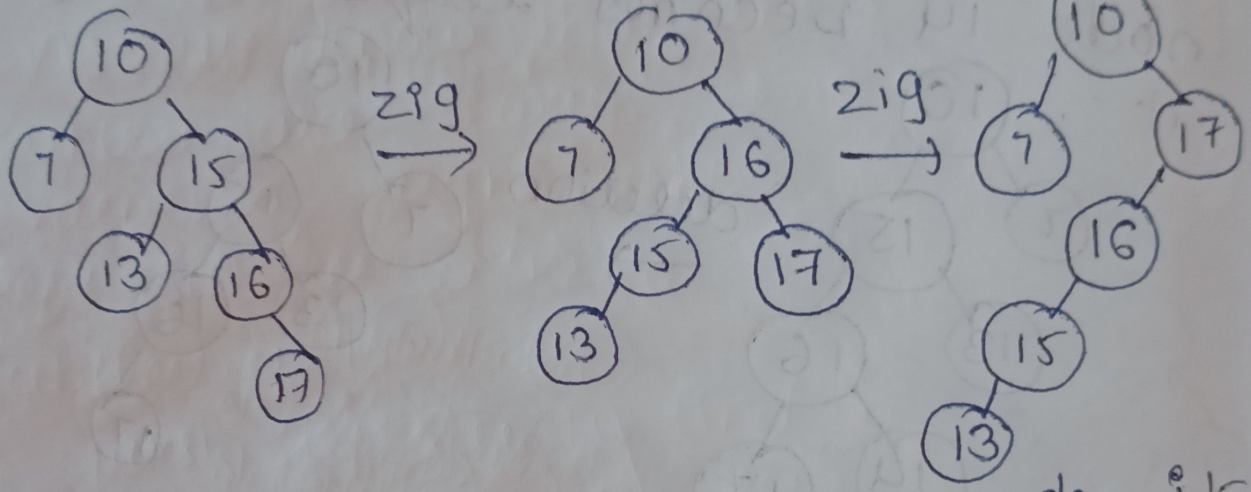
Delete 20

As 20 is not present in the tree there is no element to delete but splaying is done.

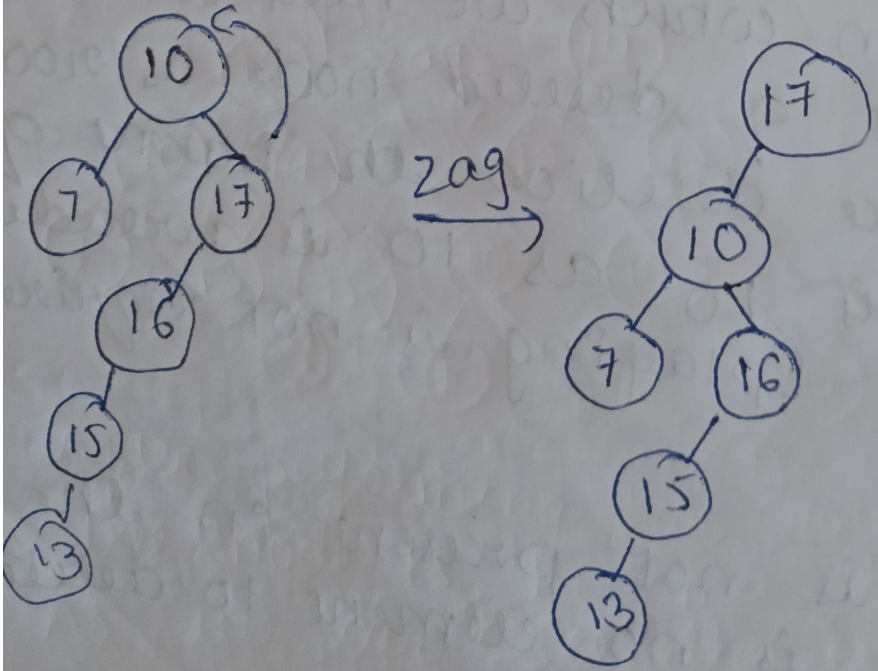
here while Searching for 20 we will traverse up to 17. So splaying will be performed on 17



here 17 as parent & grand parent & has RR relationship o hue zig zig rotation is performed

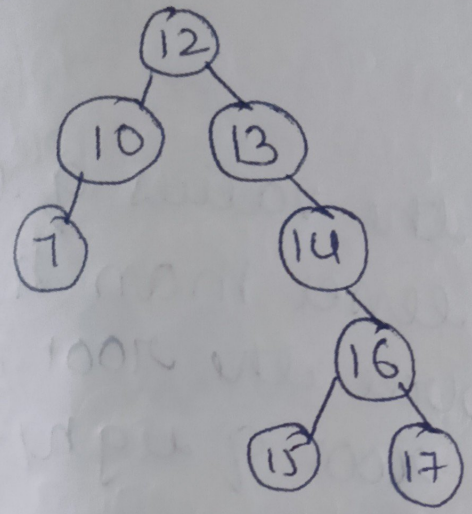


here 17 is not root node it has parent & is right of root node so left operation (zag) is performed



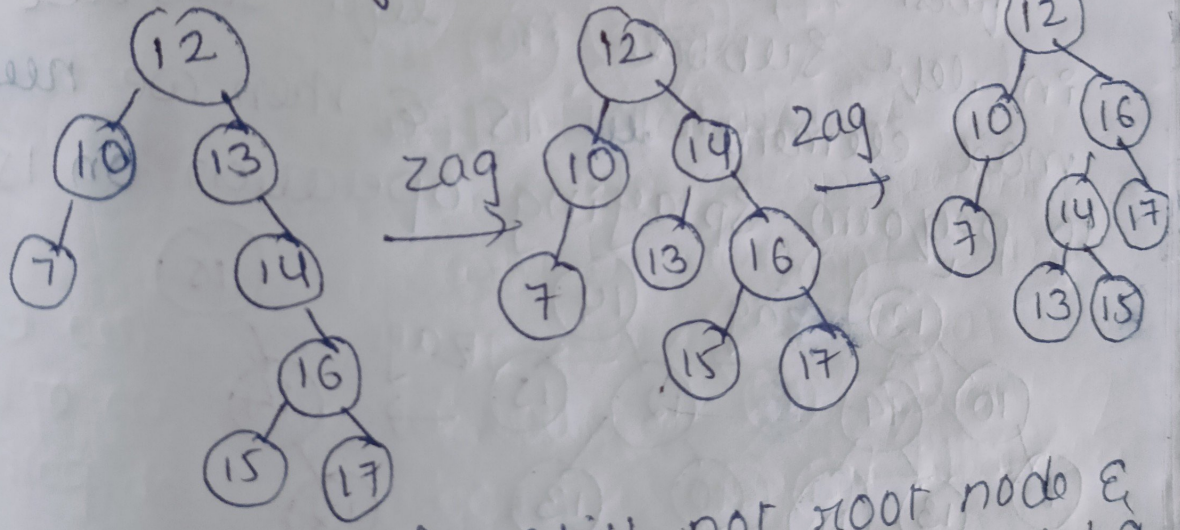
Top down Splaying :-

In top down splaying we first perform splaying on which deletion is to be performed & then delete node from the tree once the element is deleted we will perform join operation

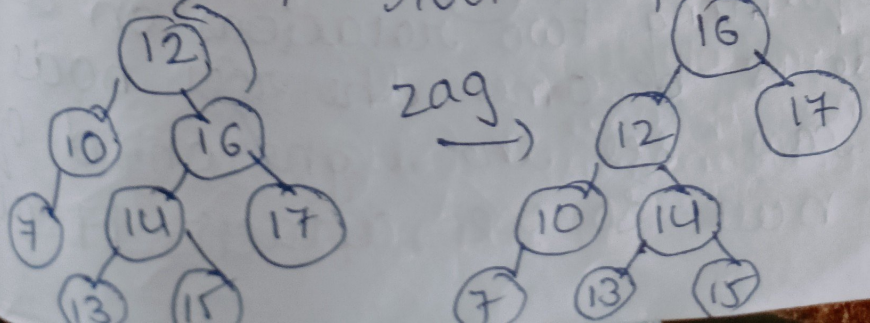


Delete 16

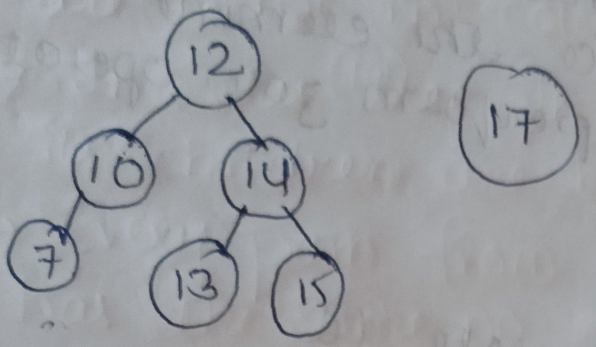
Node 16 has both parent & grand parent zag zag rotation is performed



here 16 is still not root node & has parent which is right child of root so perform zag rotation

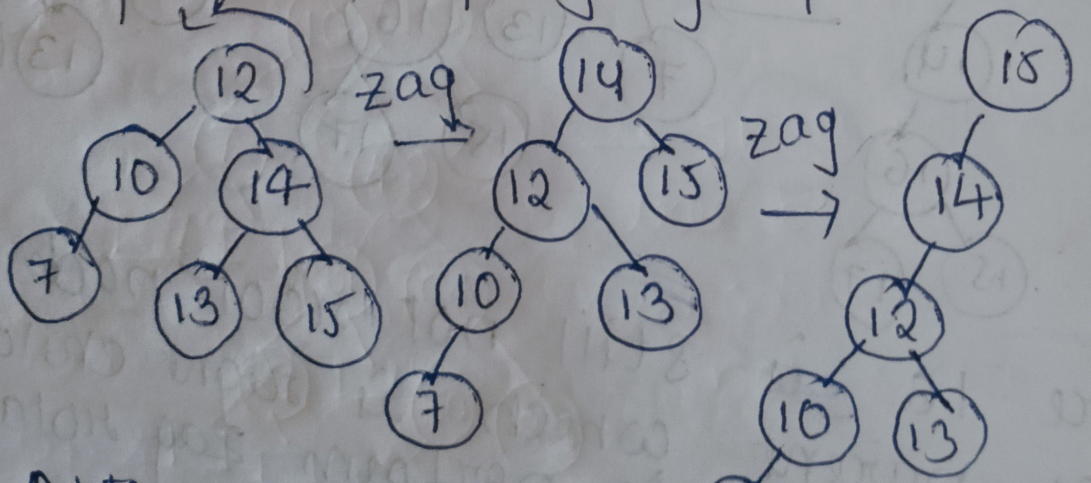


once node 16 becomes root node we will delete the node 16 & we will get two different trees i.e left subtree & right subtree.



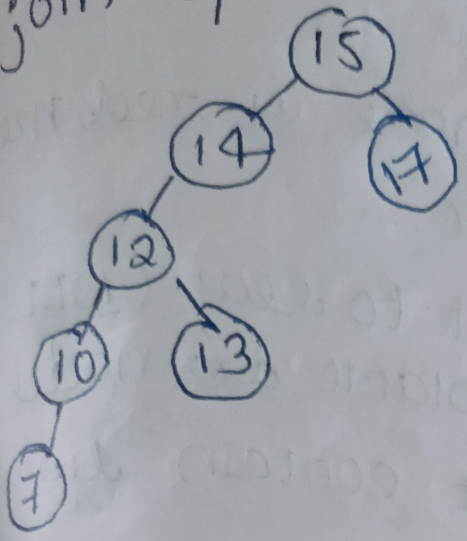
As we know that the values of left subtree are always lesser than the values of right subtree the root of left subtree is 12 & root of right subtree is 17.

First step is to find max element in left subtree in left subtree max element is 15 & then we need to perform splaying operation on 15



After performing two rotations on the tree node 15 becomes the root node. As we can see the right child of 15 is null so we attach 17 at-

the right part of the is as shown this operation is known as a join operation.



Red black trees :-

It is a self-balancing binary search tree. here self balancing means that it balances the tree itself by either doing the rotations or recoloring the nodes.

This tree data structure is named as a red black tree as each node is either red or black in color. Every node stores one extra information known as a bit that represents the color of the node. For example 0 bit denotes black color and 1 bit denotes the red color of the node. This information stored by the node is similar to the binary tree i.e data part, left pointer & right pointer

Rules

→ In this tree every node is either red or black.

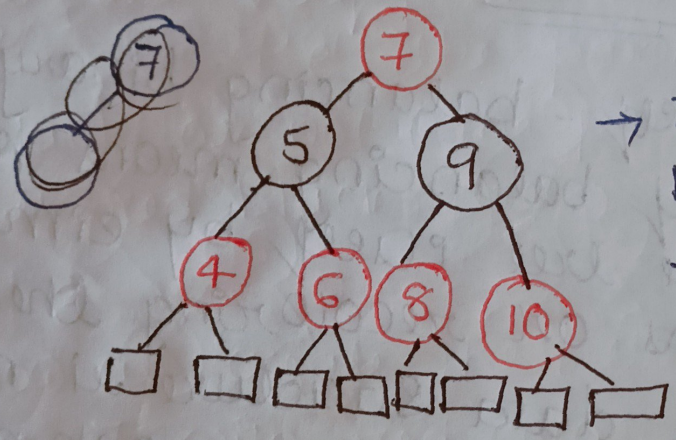
→ In this tree root node & NIL nodes are black in color.

→ In this tree if a node is red, then its children are black.

→ Every path from root to leaf (NIL) has the same no. of black nodes

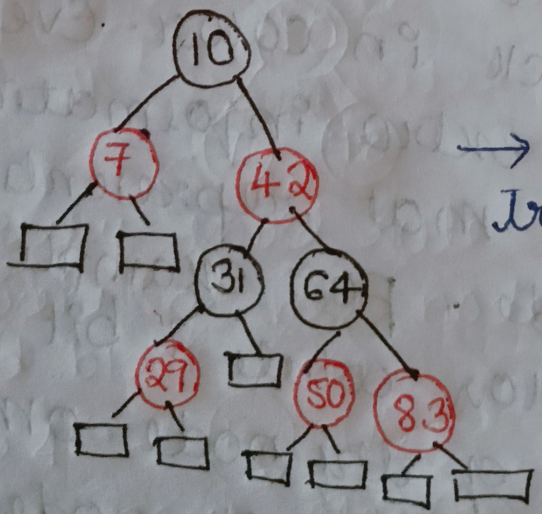
→ Every path does not contain two consecutive red nodes.

Ex:



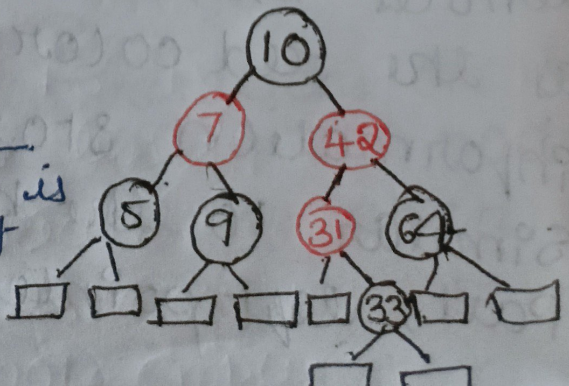
→ It is not a Red black tree because root node is not black.

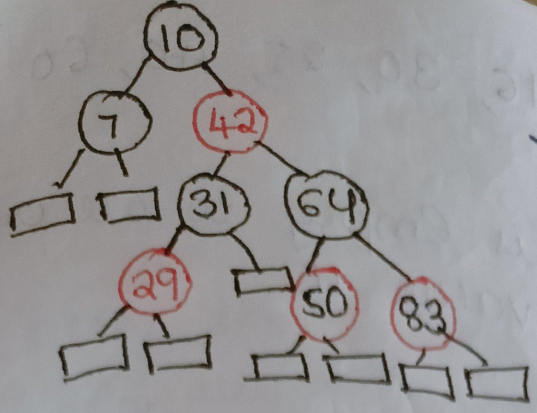
Ex



→ It is not a red black tree because no. of black nodes in each path is not same.

It is not red black tree because if node is red it's children must be black





→ It is a red black tree

Insertion in Red black Tree :-

Rules used to create red black tree

1. If the tree is Empty then we create a new node as a root node with color black.
2. If the tree is not empty then we create a new node as a leaf node with color red.
3. If the parent of a new node is black then exit.
4. If the parent of a new node is red, then we have to check the color of the parents sibling of a new node.
   a) If the color is black or nill then we perform rotations & recoloring
   b) If the color is red then we recolor the node (both parent & sibling) will also check whether the parents parent of a new node is in root node or not; if it is not a root node will recolor & recheck the node

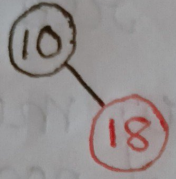
Ex: 10, 18, 7, 15, 16, 30, 25, 40, 60

Step: 1

Initially the tree is empty so we create a new node having value 10.

Step: 2

The next node is 18 As 18 is greater than 10 so it will come at right of 10

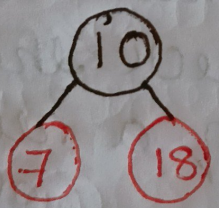


If tree is not empty then newly created node will have red color

Now we verify the third rule of red black tree i.e the parent of new node is black or not (the parent is black so it is red black tree)

Step: 3

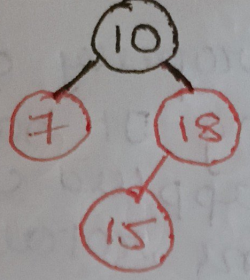
Now we create the new node having value 7 with red color



Verify third rule of red black tree the parent of new node is black so it is red black tree.

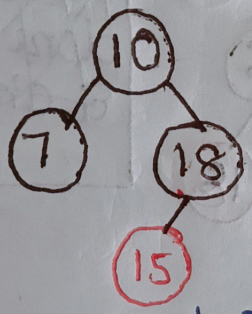
Step 4 :-

The next element is 15 & 15 is greater than 10, but less than 18 so the new node will be created at the left of node 18. The node 15 would be red in color as tree is not empty



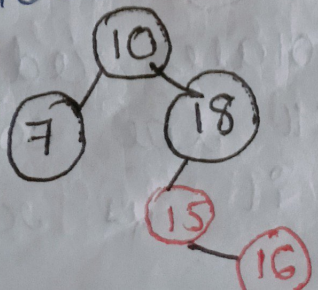
here the parent of a new node is red color we have to check the color of the parent color sibling of new node new node is 15 parent of new node is 18 & sibling of the parent node is node 7 As the parent sibling is red in color we apply rule 4b for this case.

we have to recolor both parent & parents sibling so both node 7 & 18 would be recolored.

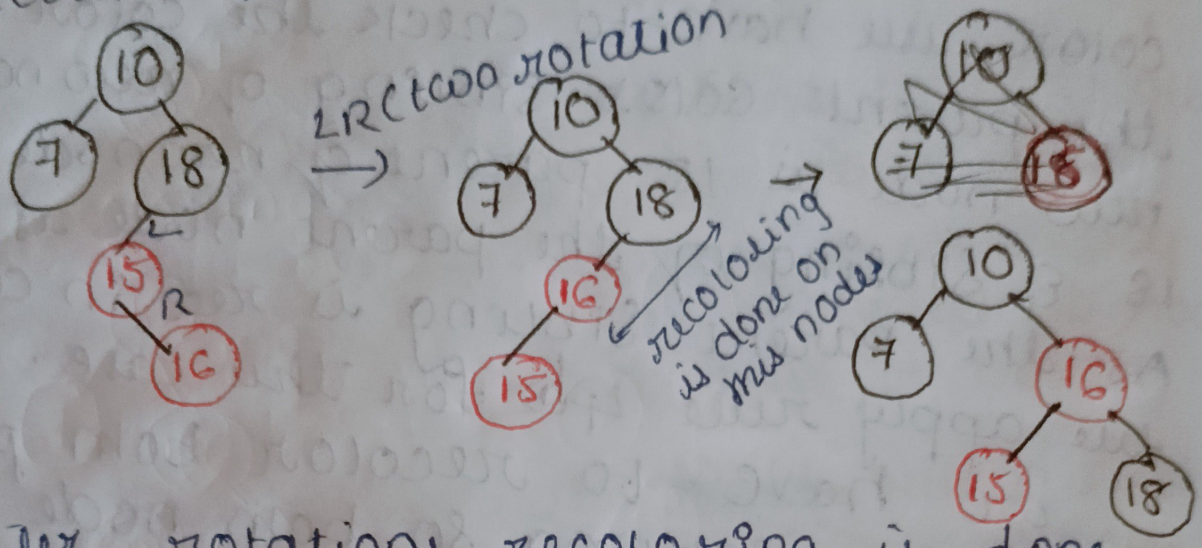


Also we have to check whether the parents parent of new node is root node or not. (If it is root node) so no need to recolor it.

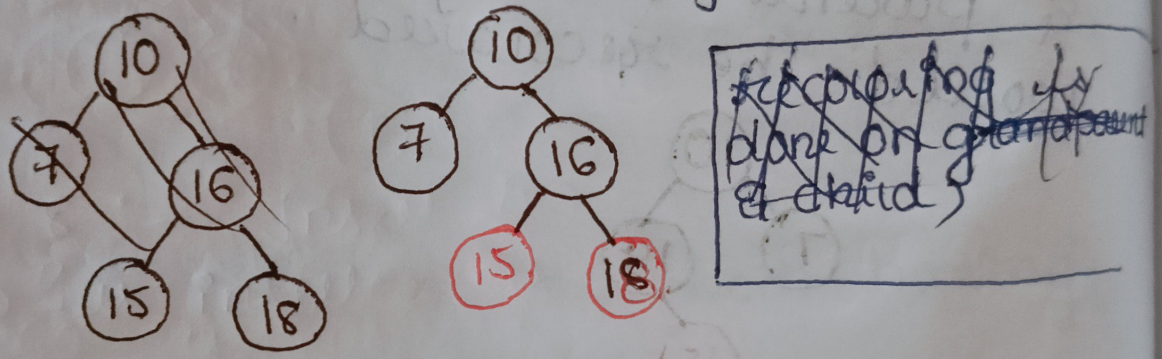
Step 5: next element is 16. As 16 is greater than 10 but less than 18 & grater than 15, so node 16 will come at right of node 15



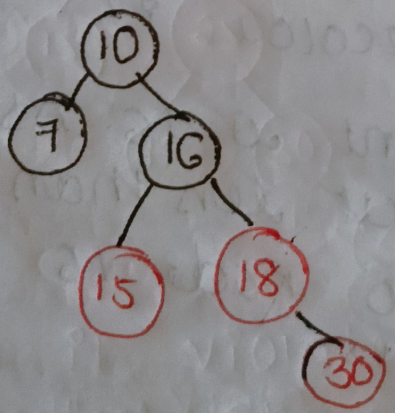
here it violates the property check the color of parent sibling it has no sibling so rule 4a will be applied where we have to perform some rotation & recoloring



After rotations recoloring is done

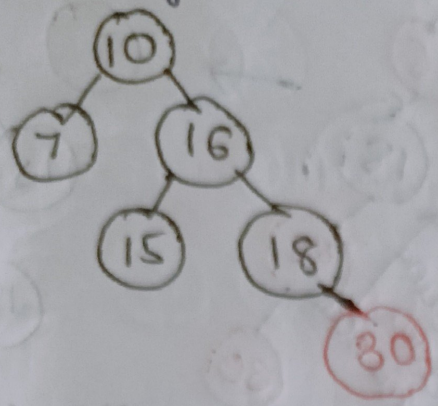


Step 6 :- Insert 30
It is inserted as right child of 18

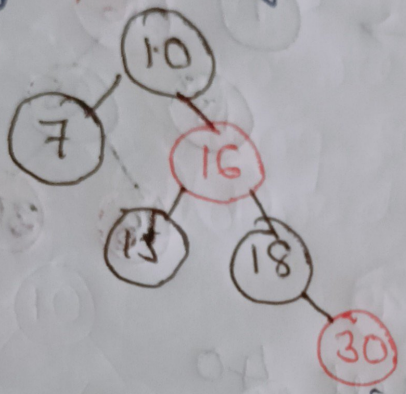


here it violates red black tree property check the color of parent sibling here it is red so we

color the bom parent & parent sibling



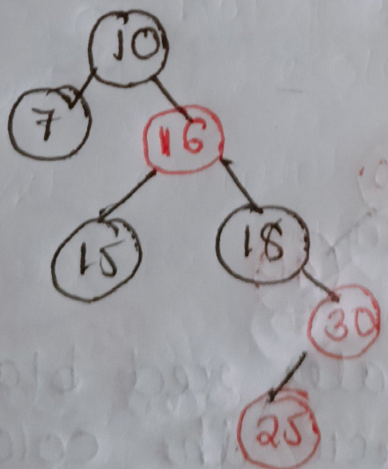
check whether the parent parent is root node here it is not root node so recolor & again check for the properties



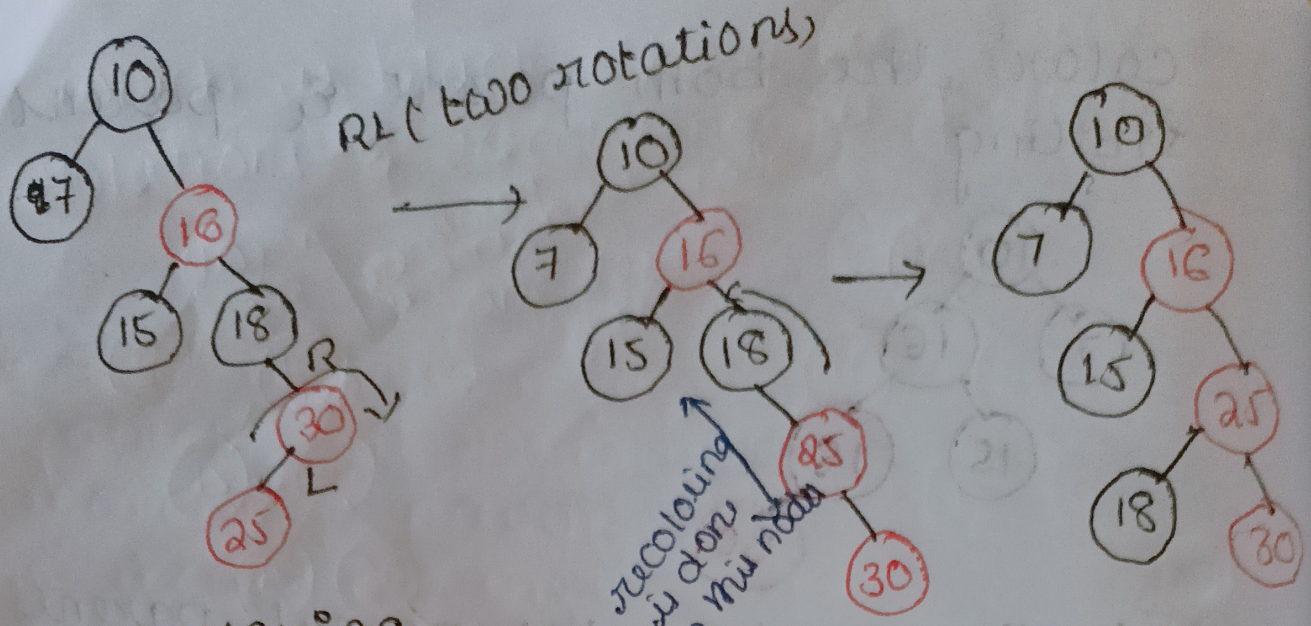
here 16 parent is root node so insertion is done

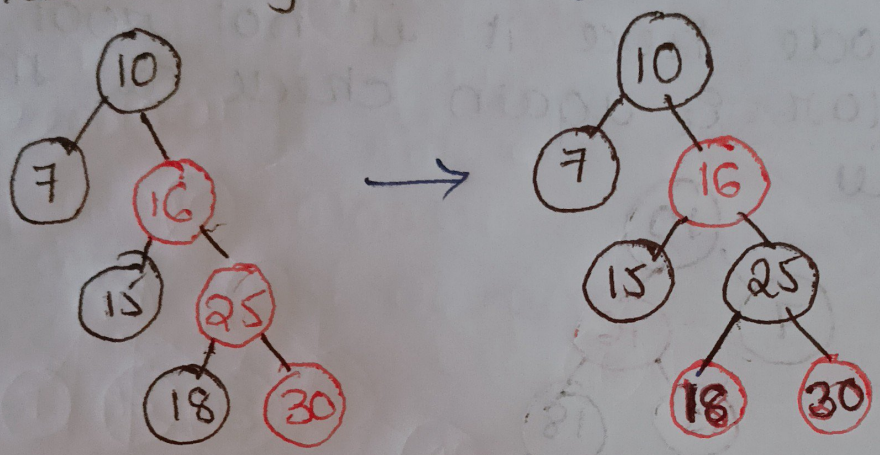
step 7 :- Insert 25

here 25 is inserted at left of 30



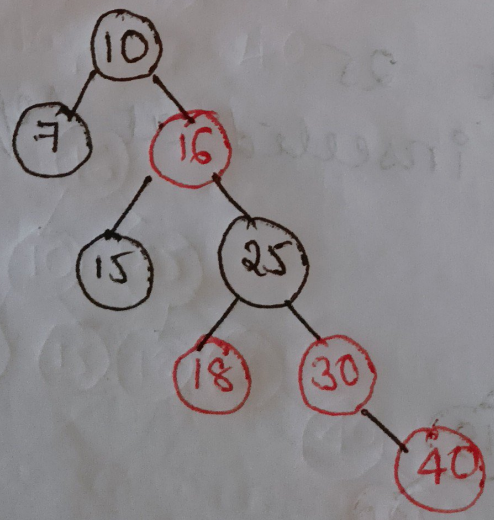
here it violated red black tree property so check the color of parent sibling here it is nil so we perform rotations & then recolor



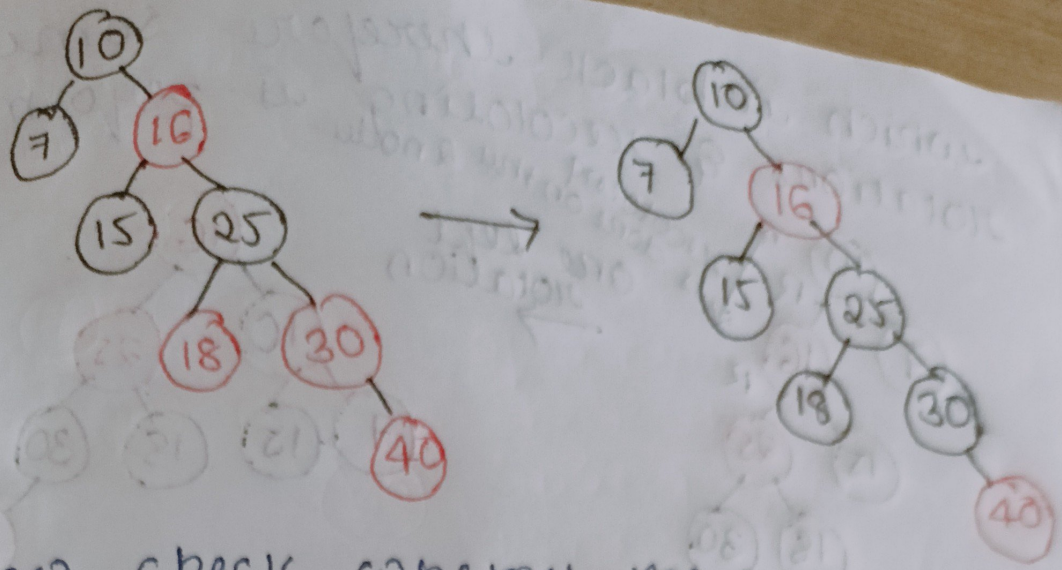


Step 8 :- Insert 40

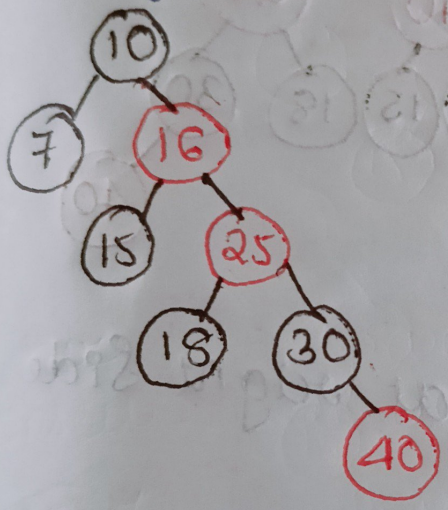
40 is inserted as the right child of 30



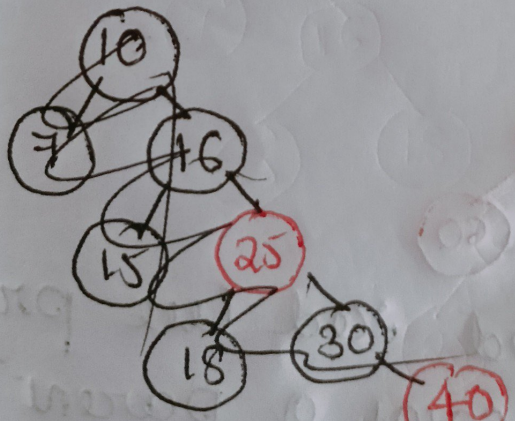
here it violates red black tree properties check the color of parent sibling if it is red so recolor parent & parents sibling



Now check whether the parent of parent is root node or not here 25 is not root then recolor the node & again check the property of tree

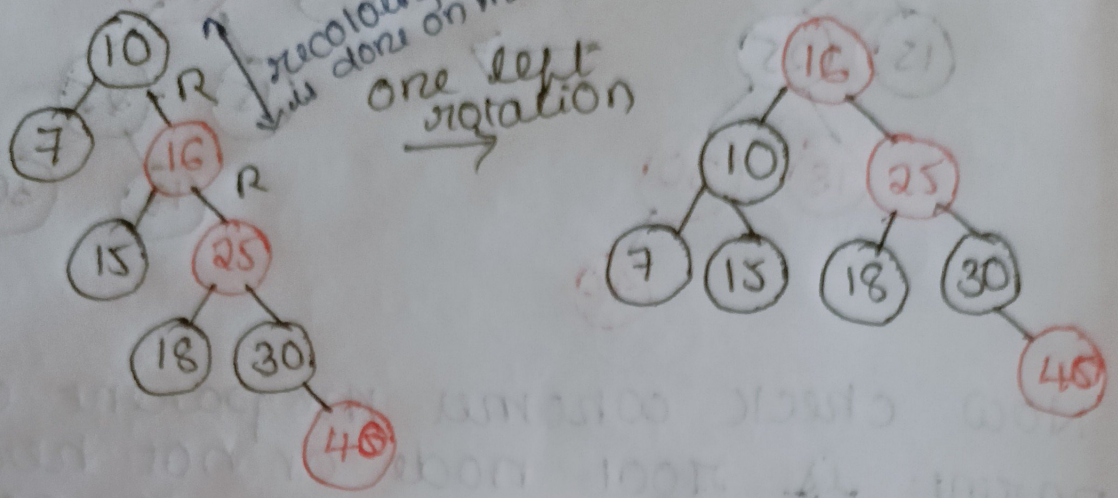


here check again, if parent which is 16, 16 is not root node so recolor & check the property

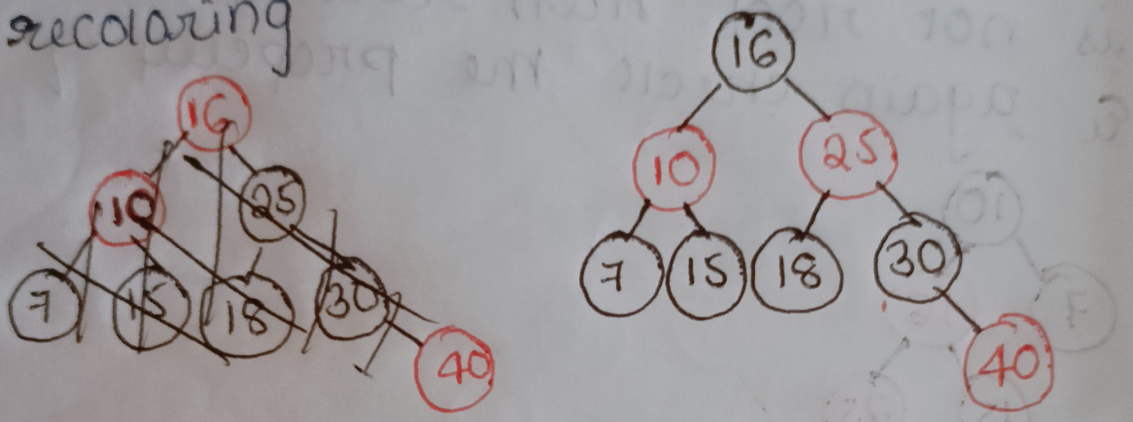


here now new node is 25 the parent of new node is red in color find the color of parent sibling

which is black therefore some rotations & recoloring is performed

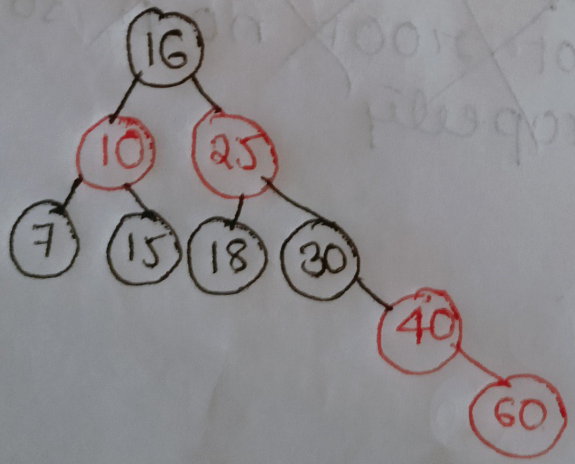


After recoloring

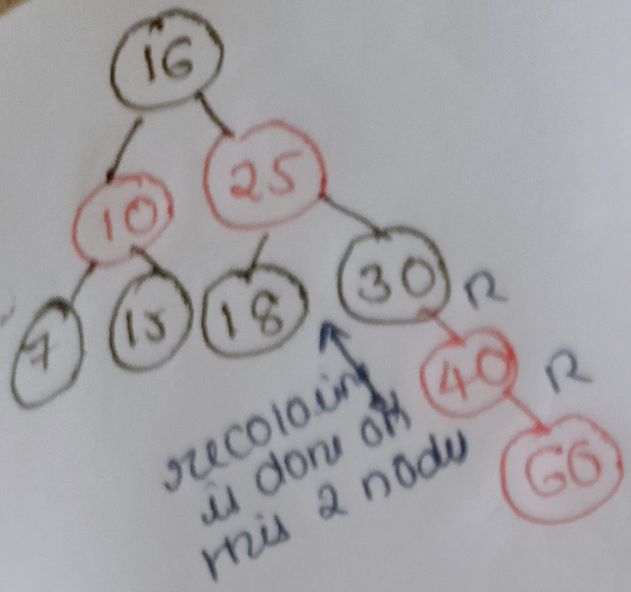


Step 9 :- Insert 60

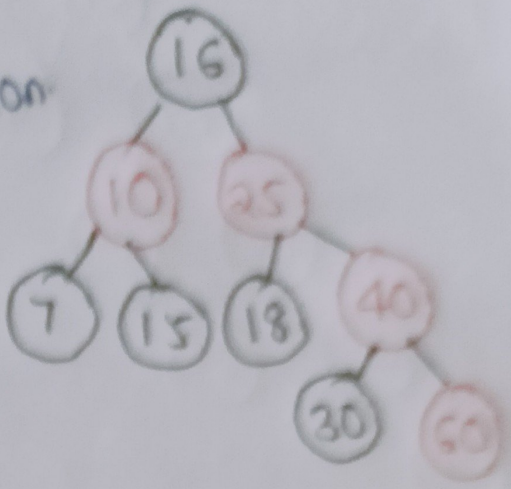
60 is inserted at right side of node 40



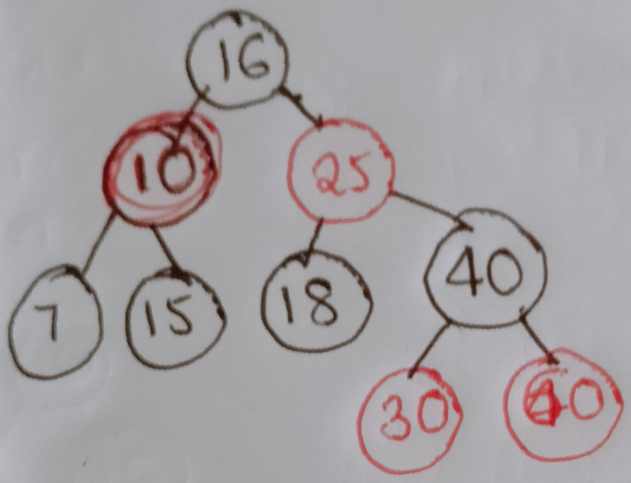
here it violated red tree property here find the color of parent sibling here it is null or Empty so we perform some rotations & recoloring



one left rotation →



After recoloring



It satisfies red black tree properties hence it is a red black tree

Deletion in Red black Trees :-