ASSIGNMENT 2 AVL TREE EXPLAINATION AND OUTPUTS

1. **INSERTION**

Let x be a node in the AVL tree T and xl is left subtree and xr is right subtree of node x .Then

1. If xl > xr, then x is **left high**. In this case, xl = xr +1 and balance_factor(x)=1 and the tree is balanced tree
2. If xl = xr, then x is **equal high** and balance_factor(x)=0 and and the tree is balanced tree
3. If xr > xl, then x is **right high**. In this case, xr = xl +1 and balance_factor(x)= -1 and the tree is balanced tree

Critical node: if node x is having a balance factor other than -1 , 0 and 1 then tree is not balanced and x is calleda critical node and we apply rotations to make the tree balanced again.

While inserting a new node it first search the correct position through the root ,after insert it backtracks to root node through the same path and changes the balance factor , if balance factor of any node is other than -1 , 0 or 1 than we apply rotations to make the tree balance.

Balance factor while inserting

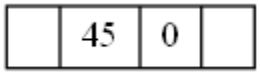
If node x does not have any children :

- **Inserting into left side make its balance factor(x) =1**
- **Inserting into right side make its balance factor(x) =-1**

If node x have left children (balance factor(x) =1) and we are inserting into right of x then balance factor(x) =0
If node x have right children (balance factor(x) =-1) and we are inserting into left of x then balance factor(x) =0

- **AVL_Insert(int k)**

Case1: when tree is empty: we insert a node and mark it as root and make its balance factor as =0



Inserting more nodes:36 and 63

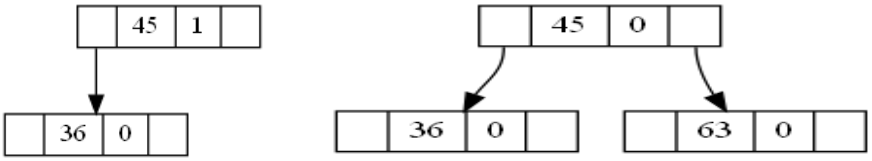


Fig1

Inserting 27 and 39 in fig1

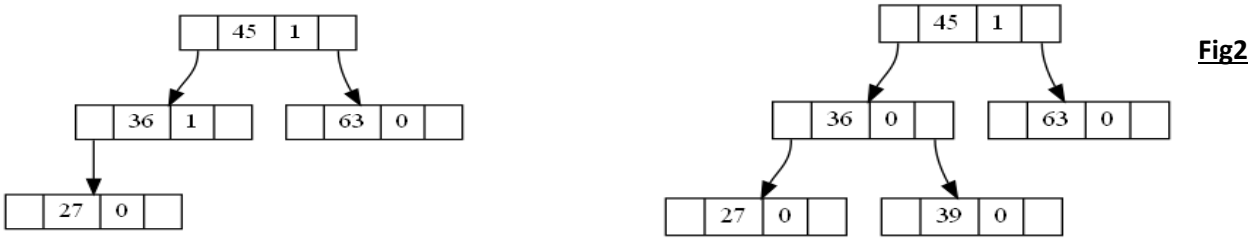


Fig2

Inserting 54 and 72 in fig 1

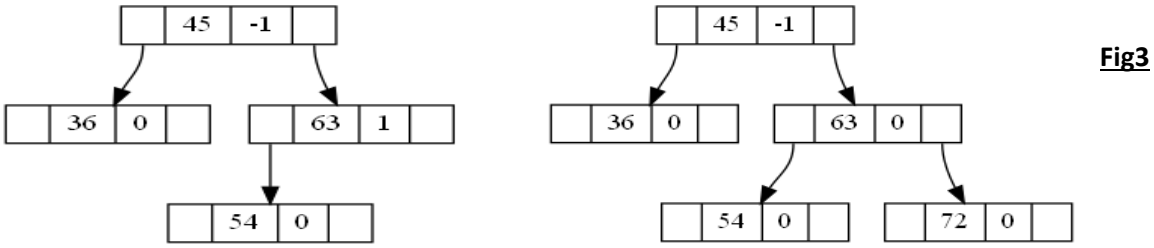
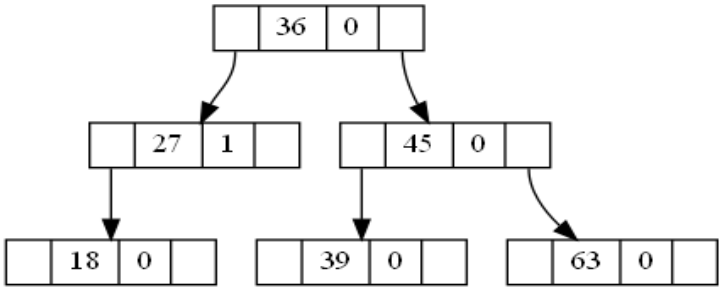


Fig3

Case2 : New node inserting into left node of left subtree

In fig2, now insert 18 with bf(18)=0 , after serching its position 18 will get inserted into left of 27 then it will backtracks to 27 changing its bf from 0 to 1 as it's a left insert then again backtracks to 36 changing its bf from o to 1 then again backtracks to reach 45 ,now 45 was alredy left high and we have inserted new node into left subtree of 45 this makes the tree unbalanced ,now node 45 is critical node and we apply rotaion on node 45.

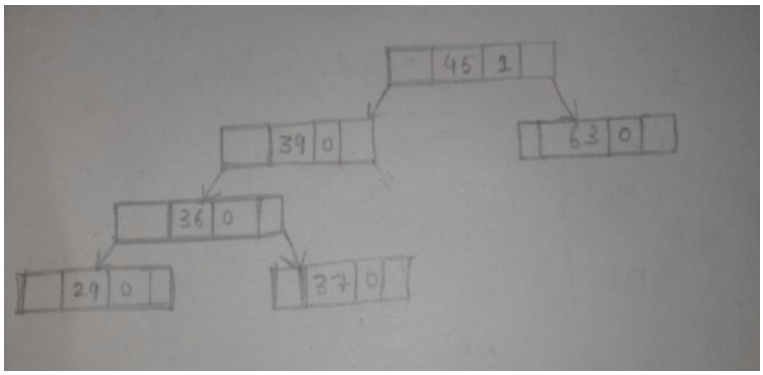
Rotation: as 45 was left high we call balancefromleft(45) which change bf of 45 and leftchild of45 as 0 and then call rotatetoright(45) which balances the tree finally



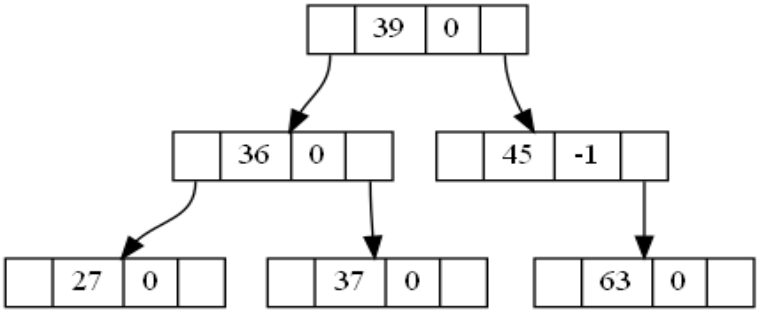
Case3 : New node inserting into right node of left subtree

In fig2 , now insert 37 with bf(37)=0 , after serching its position 37 will get inserted into left of 39 then it will backtracks to 39 changing its bf from 0 to 1 as it's a left insert then again backtracks to 36 changing its bf from 0 to -1 then again backtracks to reach 45 ,now 45 was alredy left high and we have inserted new node into left subtree of 45 this makes the tree unbalanced ,now node 45 is critical node and we apply rotaion on node 45.

Rotation: as 45 was left high we call balancefromleft(45) which change bf of 45 to 1 and leftchild of 45 as 0 and bf of rightchild of leftchild of 45 to 0 and then call rotatetoleft(36) which makes the tree as:

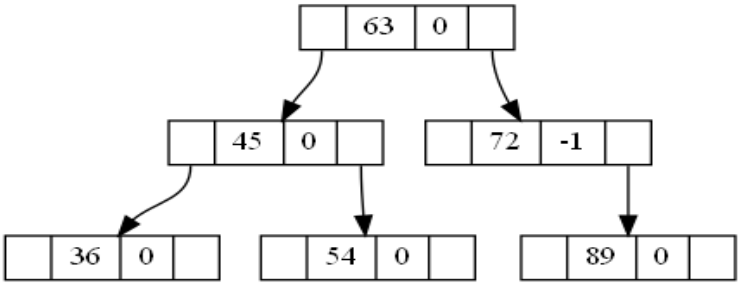


Then we call rotatetoright(45) which finally balances the tree.



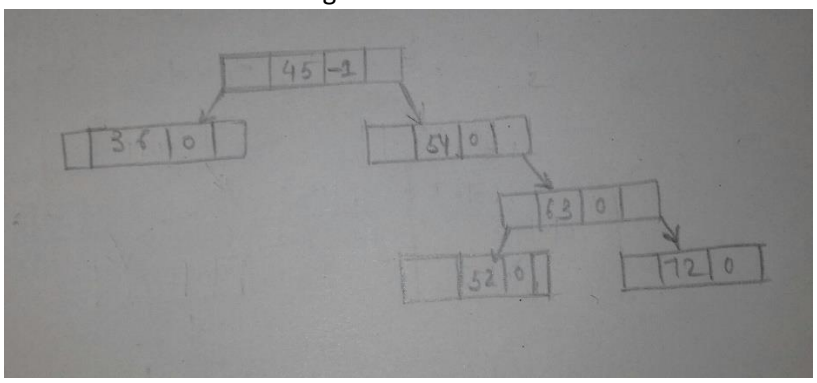
Case4 : New node inserting into right node of right subtree

In fig3, now insert 89 with bf(89)=0 , after serching its position 89 will get inserted into right of 72 then it will backtracks to 72 changing its bf from 0 to -1 as it's a right insert then again backtracks to 63 changing its bf from 0 to -1 then again backtracks to reach 45 ,now 45 was alredy left high and we have inserted new node into right subtree of 45 this makes the tree unbalanced ,now node 45 is critical node and we apply rotaion on node 45.
Rotation: as 45 was right high we call balancefromright(45) which change bf of 45 and right child of45 as 0 and then call rotatetoleft(45) which balances the tree finally

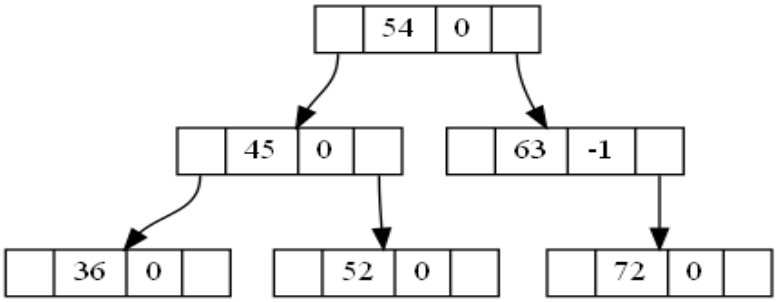


Case5 : New node inserting into left node of right subtree

In fig3 , now insert 52 with bf(52)=0 , after serching its position 52 will get inserted into left of 54 then it will backtracks to 54 changing its bf from 0 to -1 as it's a right insert then again backtracks to 63 changing its bf from 0 to -1 then again backtracks to reach 45 ,now 45 was alredy right high and we have inserted new node into right subtree of 45 this makes the tree unbalanced ,now node 45 is critical node and we apply rotaion on node 45.
Rotation: as 45 was right high we call balancefrom right (45) which change bf of 45 to -1 and rightchild of 45 as 0 and bf of leftchild of rightchild of 45 to 0 and then call rotatetoright (63) which makes the tree as:



Then we call rotatetoleft(45) which finally balances the tree.



2. DELETION

- **Suppose that the node to be deleted, say x, has a left and a right child.**
- **We find the immediate predecessor, say y of x. Then, the data of y is copied into x and now the node to be deleted is y.**
- **To delete the node, we adjust one of the pointers of the parent node. After deleting the node, the resulting tree might no longer be an AVL tree**
- **We traverse the path (from the parent node) back to the root node. For each node on this path, sometimes we need to change only the balance factor, while other times the tree at a particular node is reconstructed**
- **We use the bool variable issmaller to indicate whether the height of the subtree is reduced.**
- **Let p be a node on the path back to the root node. We look at the current balance factor of p.**

1. **If the current balance factor of p is equal high, the balance factor of p is changed according to if the left subtree of p was shortened or the right subtree of p was shortened. The variable issmaller is set to false.**
2. **Suppose that the balance factor of p is not equal and the taller subtree of p is shortened. The balance factor of p is changed to equal high, and the variable issmaller is left as true.**
3. **Suppose that the balance factor of p is not equal and the issmaller subtree of p is shortened. Further suppose that q points to the root of the taller subtree of p**

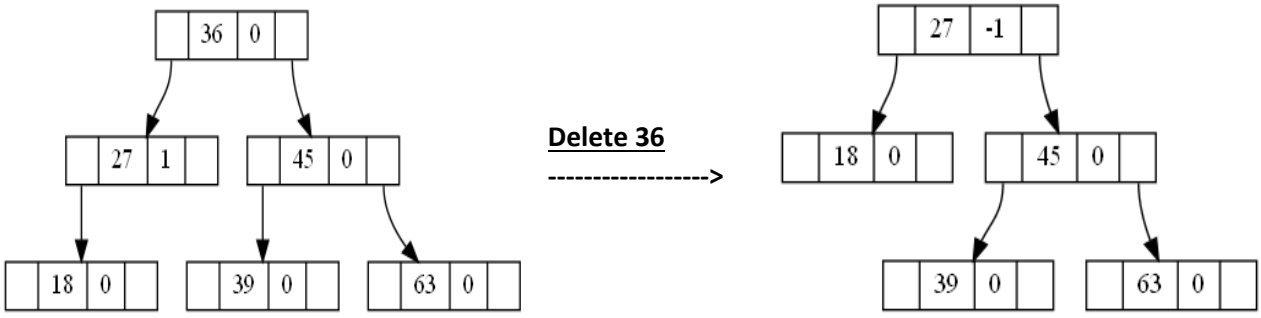
   a. If the balance factor of q is equal, a single rotation is required at p and issmaller is set to false.

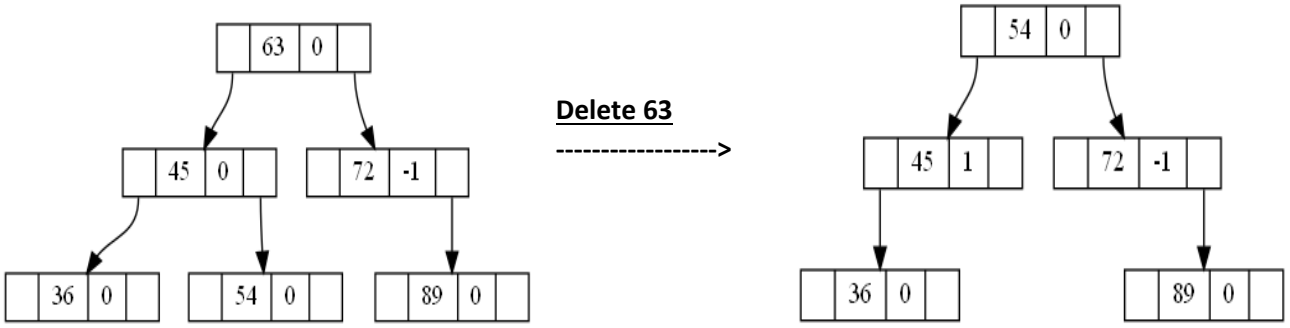
   b. Ifthe balance factor ofq is the same as p, a single rotation is required at p and issmaller is set to true.

   c. Suppose that the balance factors of p and q are opposite. A double rotation is required at p (a single rotation at q and then a single rotation at p). We adjust the balance factors and set issmaller to true.

- **AVL_Delete(int k):**

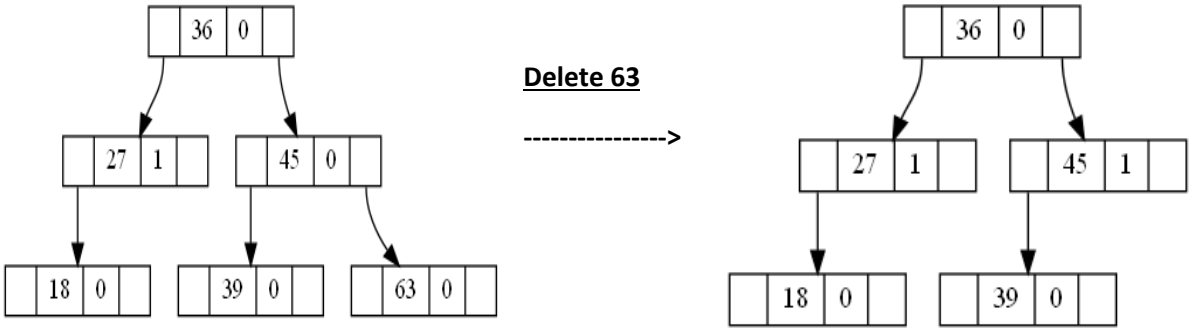
Case 1: DELETING THE ROOT

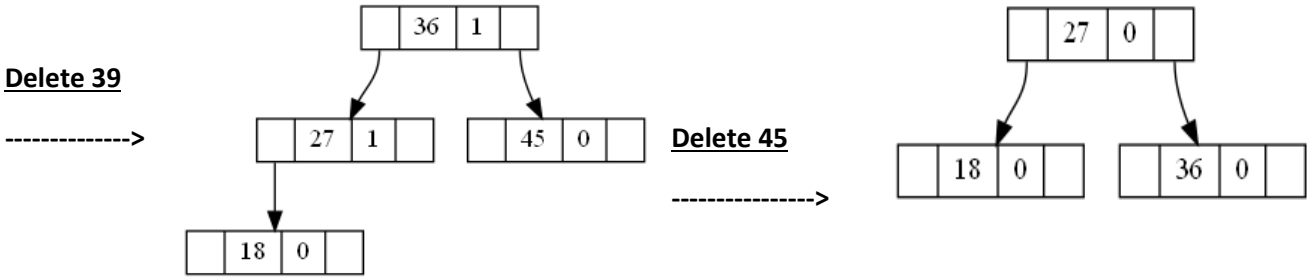




Case 2: DELETING FROM THE RIGHT SUBTREE

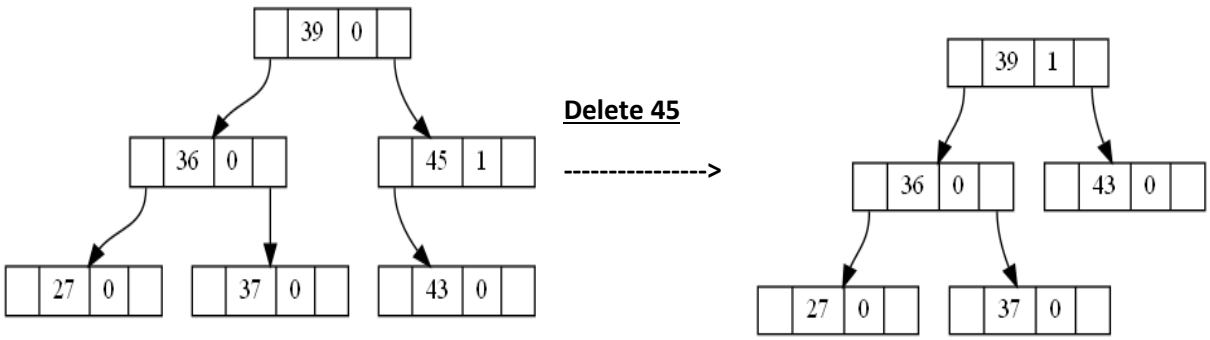
2.1 deleting leaf nodes



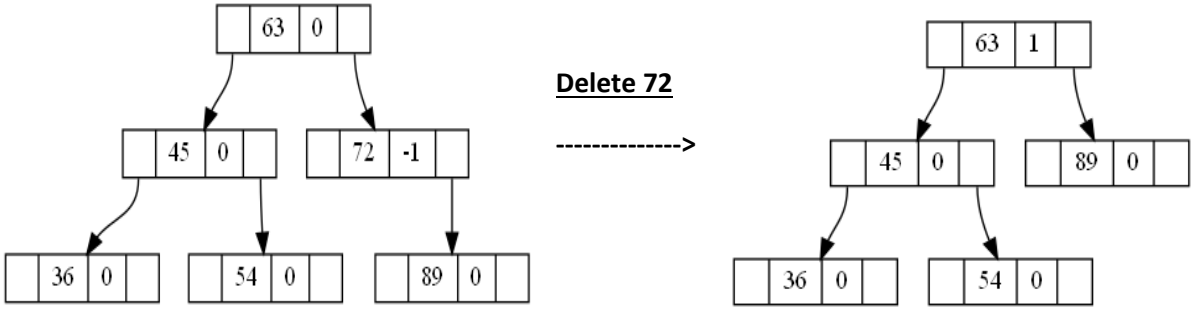


2.2 deleting non leaf nodes

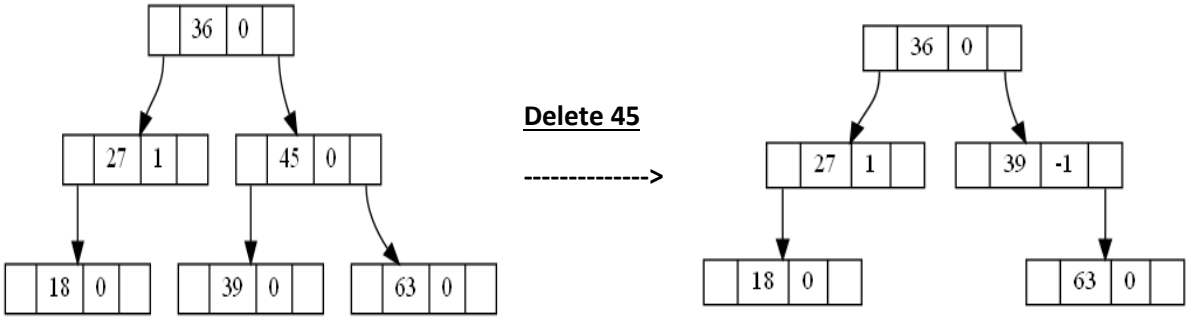
2.2.1 deleting nodes having only left child



2.2.2 deleting nodes having only right child

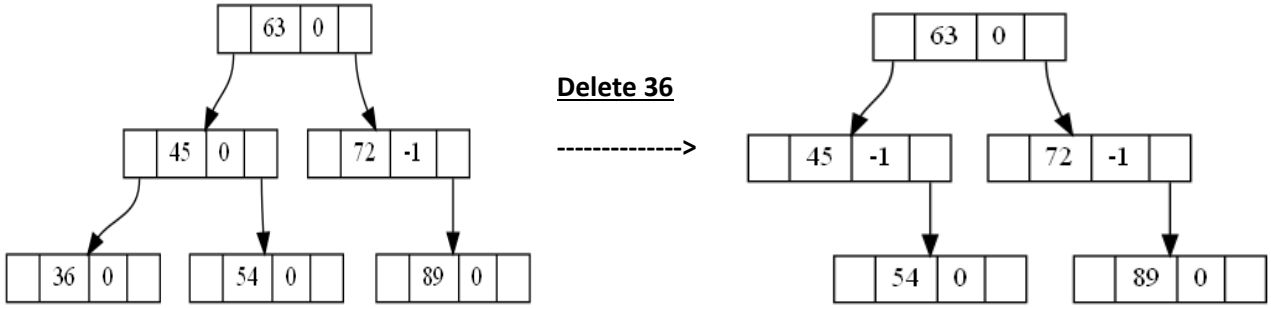


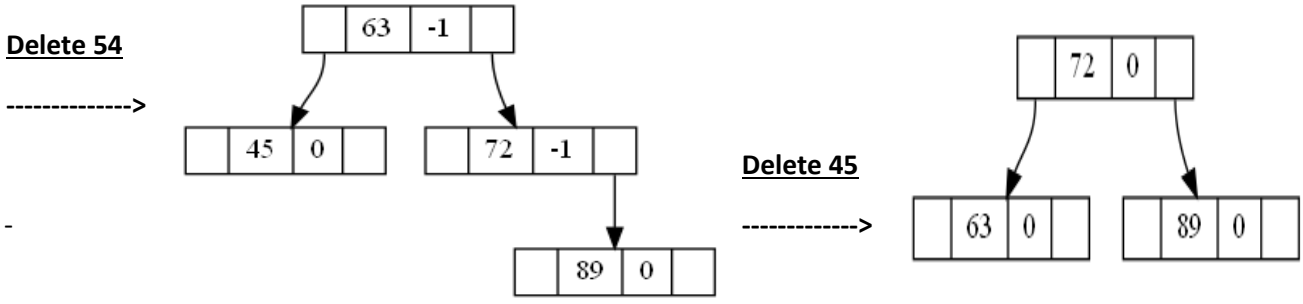
2.2.3 deleting nodes having both child



Case 3: DELETING FROM THE LEFT SUBTREE

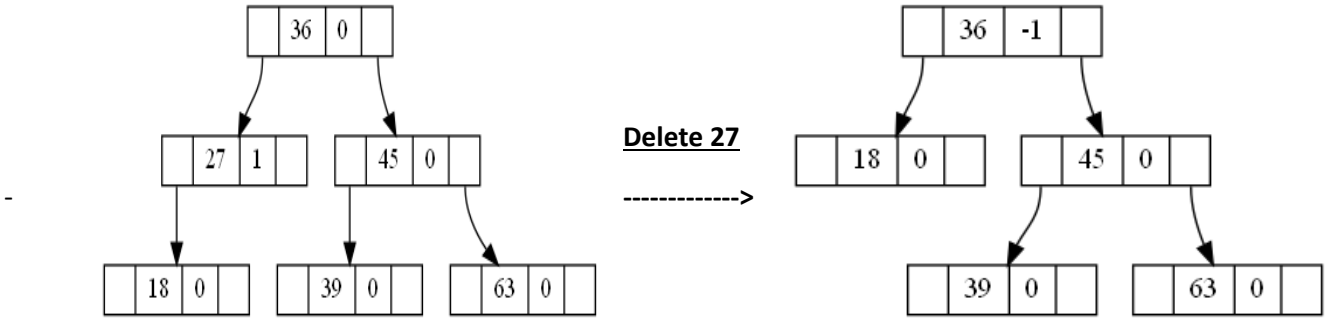
3.1 deleting leaf nodes



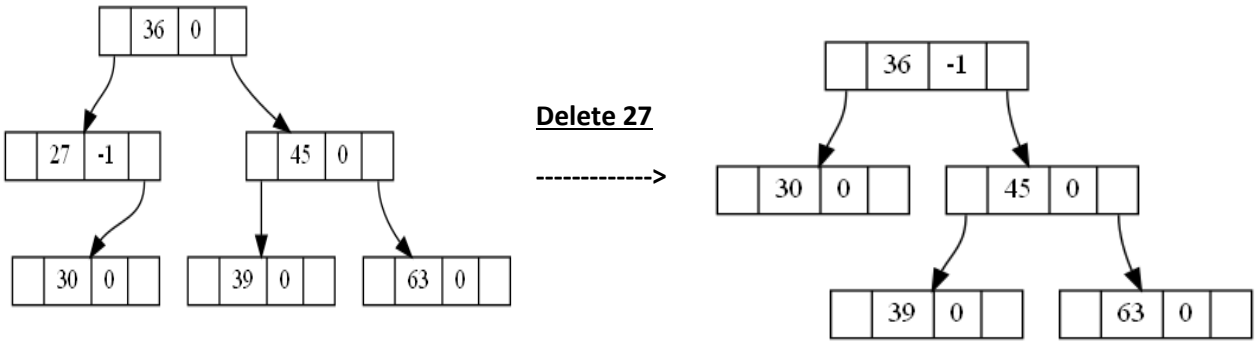


3.2 deleting non leaf nodes

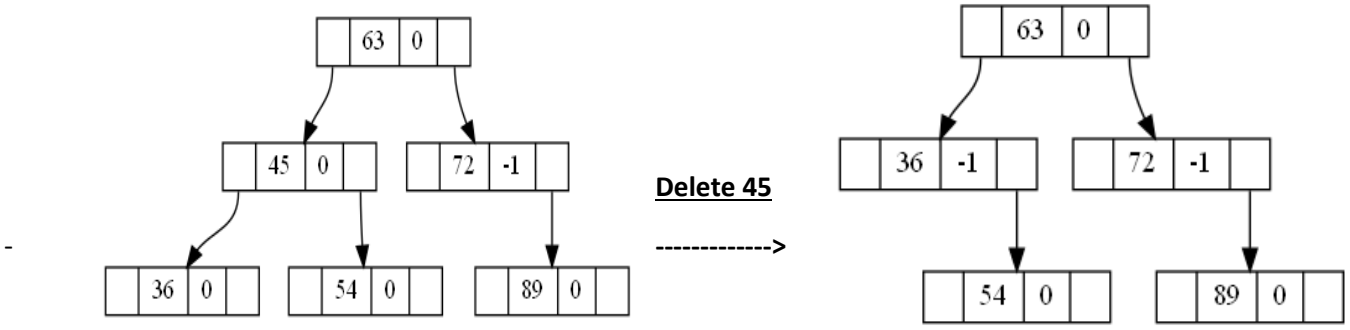
3.2.1 deleting nodes having only left child



3.2.2 deleting nodes having only right child



3.2.3 deleting nodes having both left and right child



3 SEARCH

- **AVL_serach(int k)**
- **Searching elements in the tree : 45 36 63 27 39 37 50 1 90 5 35 38 42 46 60**
  1. **Case1 when k is present**

```
AVL_Operations
1. Insert
2. Delete
3. Search
4. Print
Enter Your Choice: 3
Enter integer element to search: 90
Element 90 found in the tree
Do you want to continue (Type y or n): y
```

2. **Case2 when k is not present**

```
AVL_Operations
1. Insert
2. Delete
3. Search
4. Print
Enter Your Choice: 3
Enter integer element to search: 100
Element 100 not found in the tree
```

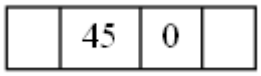
4 PRINT

- **AVL_Print(const char *filename)**
  This is basically PREORDER TRAVERSAL of the AVL Tree so that we get a proper .png file using graphviz which shows output tree after insertion or deletion.

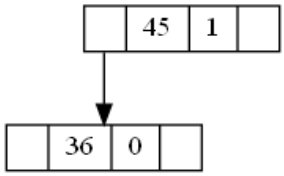
SOME MORE OUTPUTS FOR INSERTIONS AND DELETIONS:

1.INSERTION

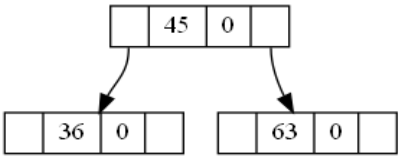
Insert 45>



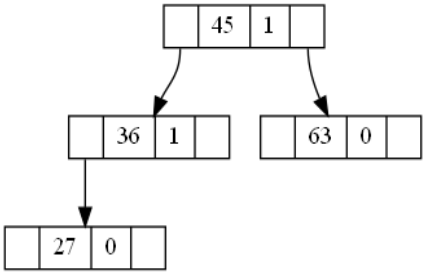
insert36>



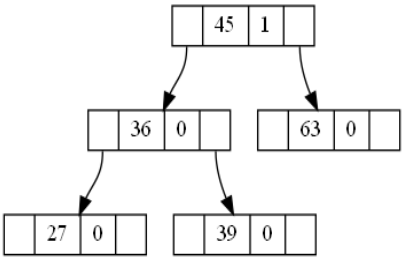
insert63>



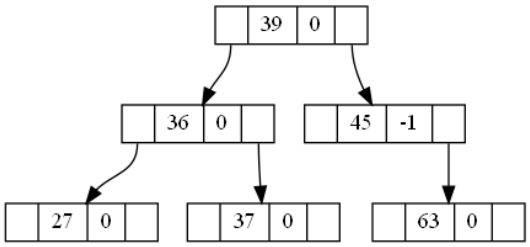
Insert27>



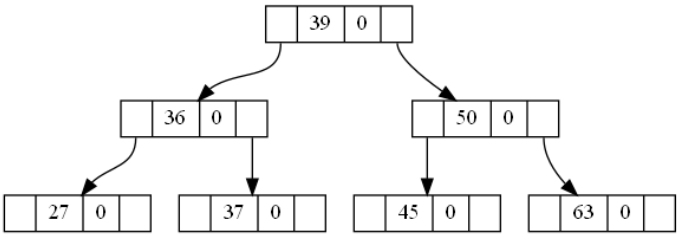
insert39>



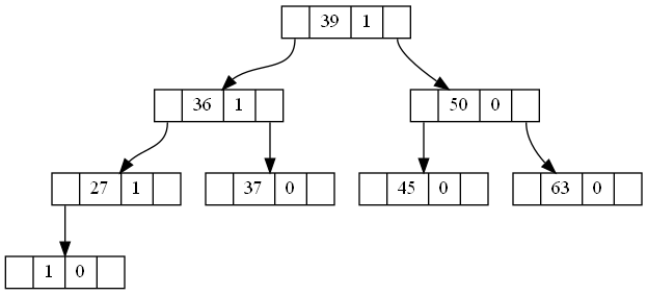
insert37>



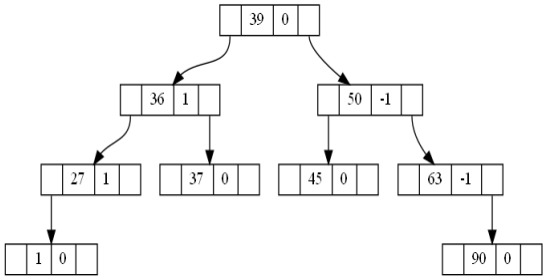
insert50>



insert1>



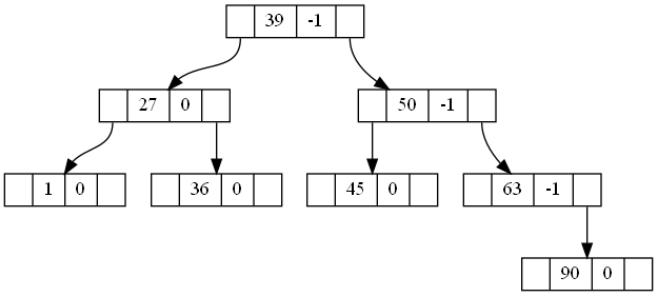
insert90>



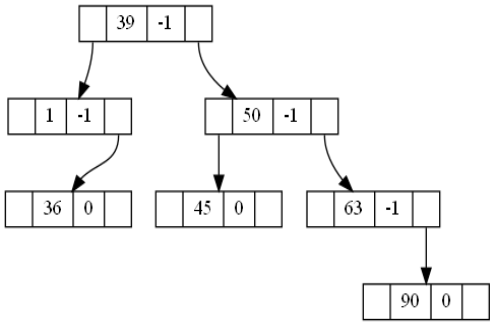
2.DELETION

(using above tree after insert 90)

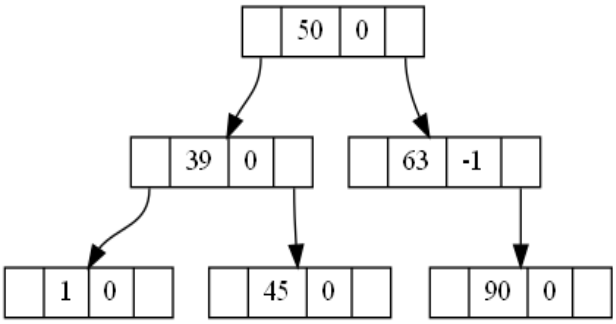
Delete37>



delete27>



Delete36>



delete39>