

EXPERIMENT NO. 8

Author: Harsheet Chordiya

Roll No.: 37

Sem and Section: 3 CSEB

Source file: expt08.c

Date: 09/02/2021

=====

Aim: To study AVL tree (the height balanced tree) and implement various operations on AVL tree.

Problem Statement: Consider a AVL tree node `treeNodeAVL`, representing a self-referential structure. Write a menu driven program to-(1) insert a node into AVL tree, (2) delete a node from AVL tree, (3) print traversals (inorder, preorder, postorder) for AVL tree. Write other routines as required by the application to support the mentioned routines. Your program must also allow for creating a pre-allocated list before the start of AVL Tree ADT. You must not use global variables.

=====

THEORY

=====

Introduction to height balance tree:

AVL tree is a self-balancing binary search tree. The key advantage of using an AVL tree is that it takes $O(\log n)$ time to

perform search, insert, and delete operations in an average case as well as the worst case because

the height of the tree is limited to $O(\log n)$.

The structure of an AVL tree is the same as that of a binary search tree but with a little difference.

In its structure, it stores an additional variable called the `BalanceFactor`. Thus, every node has a

balance factor associated with it. The balance factor of a node is calculated by subtracting the

height of its right sub-tree from the height of its left sub-tree. A binary search tree in which every

node has a balance factor of -1, 0, or 1 is said to be height balanced. A node with any other

balance factor is considered to be unbalanced and requires rebalancing of the tree.

Balance factor = Height (left sub-tree) - Height (right sub-tree)

Tree Rotations:

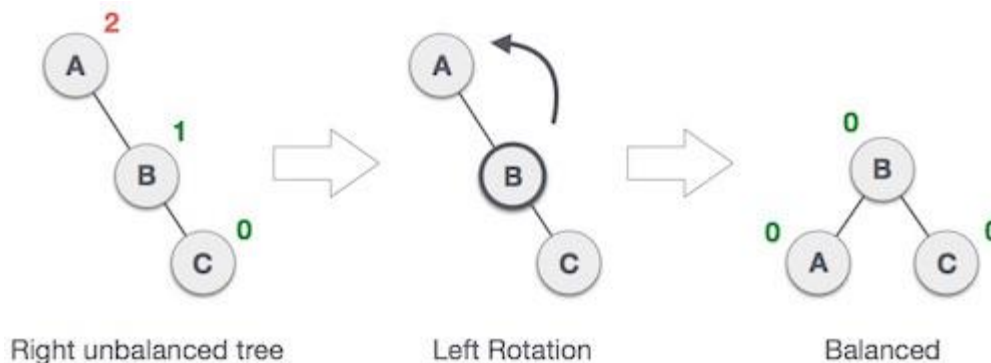
To balance itself, an AVL tree may perform the following four kinds of rotations -

- Left-Left rotation
- Right-Right rotation
- Left-Right rotation
- Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left-Left Rotation

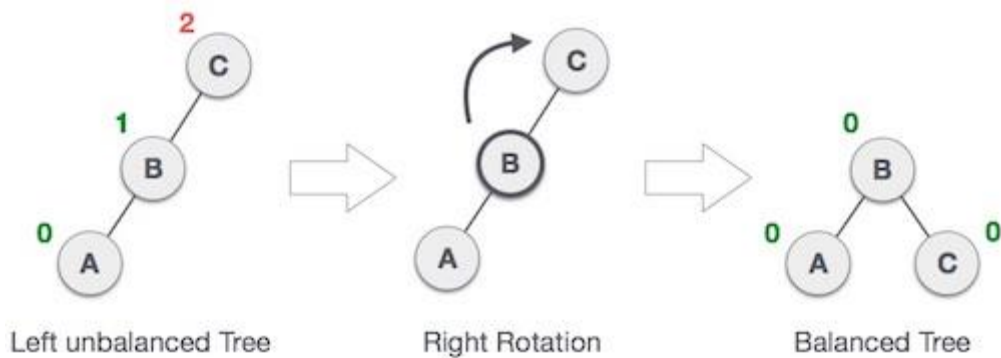
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation -



In our example, node A has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making A the left-subtree of B

Right Rotation

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



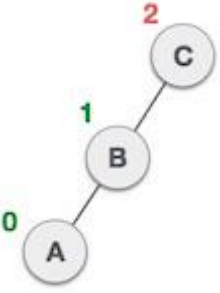
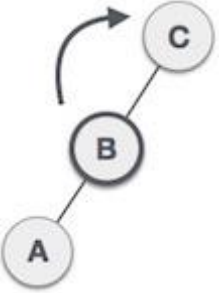
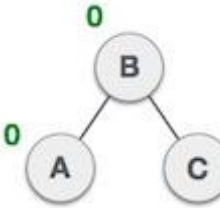
As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

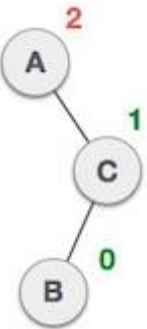
rotation.

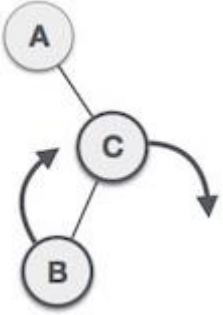
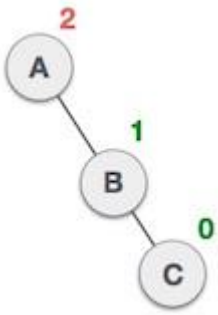
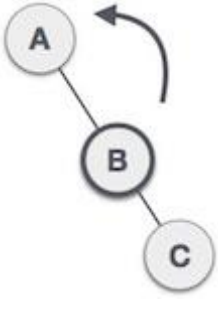
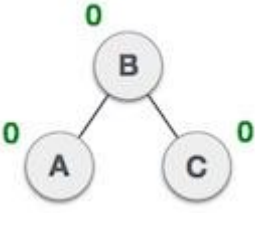
State	Action
	<p>A node has been inserted into the right subtree of the left subtree. This makes unbalanced</p> <p>node. These scenarios cause AVL tree to perform left-right rotation.</p>
	<p>We first perform the left rotation on the left subtree of C. This makes A, the subtree of B.</p>

	<p>Node C is still unbalanced, however now, it is because of the left-subtree of the left-subtree.</p>
	<p>We shall now right-rotate the tree, making B the new root node of this subtree. Node C becomes the right subtree of its own left subtree.</p>
	<p>The tree is now balanced.</p>

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	<p>A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2.</p>

	<p>First, we perform the right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes the right subtree of A.</p>
	<p>Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.</p>
	<p>A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B.</p>
	<p>The tree is now balanced.</p>

=====

ALGORITHMS

=====

Algorithm to insert a node in AVL tree:

1. If ROOT = NULL

Return Call Create_AVL_Node (KEY)

2. If KEY < DATA (ROOT)

LCHILD (ROOT) = Call Insert_AVL (LCHILD (ROOT), KEY) Else If KEY > DATA (ROOT) RCHILD (ROOT) = Call Insert_AVL (RCHILD (ROOT), KEY) I

Else

Return ROOT

3. HEIGHT (ROOT) = MAX (Call Height_AVL_Node (LCHILD (ROOT)), Call Height_AVL_Node (RCHILD (ROOT))) +

4. BAL= Call BALANCE FACTOR (ROOT)

5. BAL > 1 AND KEY < DATA (LCHILD (ROOT)) Return Call Right Rotate_AVL (ROOT)

6. If BAL < -1 AND KEY > DATA (RCHILD (ROOT)) Return Call Left_Rotate_AVL (ROOT)

7. If BAL > 1 AND KEY > DATA (LCHILD (ROOT))

LCHILD (ROOT) = Call Left_Rotate_AVL (LCHILD (ROOT)) Return Call Right Rotate_AVL (ROOT) 8. Balance AVL tree, RL-Case If BAL < -1 AND KEY < DATA (LCHILD (ROOT)) I

RCHILD (ROOT) = Call Right Rotate_AVL (RCHILD (ROOT))

Return Call Left Rotate AVL (ROOT)

9. Return AVL tree, if no balancing required Return ROOT

Algorithm to delete a node in AVL tree:

1. Is the tree empty??

If ROOT = NULL KEY MIN_VAL

Return ROOT//denotes operation failure

2. Traverse the left subtree and right subtree to locate the intended node

If KEY < DATA (ROOT)

 LCHILD(ROOT)= Call Delete_AVL (LCHILD (ROOT), KEY)

Else If KEY > DATA (ROOT)

 RCHILD (ROOT) = Call Delete AVL (RCHILD (ROOT), KEY)

//the Intended Node

Else{

If LCHILD (ROOT) = NULL OR RCHILD (ROOT) = NULL //Single or No Child Node

 If LCHILD (ROOT) <> NULL

 TEMP = LCHILD (ROOT)

Else

 TEMP RCHILD (ROOT)

If TEMP= NULL //(a) No Child

 TEMP ROOT

 ROOT = NULL

Else //(b) Single Child

 DATA (ROOT)=DATA (TEMP)

 HEIGHT (ROOT)= HEIGHT (TEMP) //Copy the Nodes

 Restore TEMP

Else //(c) Node has two children

TEMP = Call Min_Value_Node (RCHILD (ROOT)) //Inorder Successor

DATA (ROOT) = DATA (TEMP)

RCHILD (ROOT) = Call Delete_AVL (RCHILD (ROOT), A(DATA (TEMP)))

3. If the has only one node.

If ROOT NULL

Return ROOT

4. Update the height of ROOT

HEIGHT (ROOT) =MAX(Call Height_AVL_Node (LCHILD (ROOT)),

Call Height AVL_Node (RCHILD(ROOT))) + 1

5. Validate balance factor of ROOT

BAL= Call BALANCE FACTOR (ROOT)

LBAL= Call BALANCE FACTOR (LCHILD (ROOT))

RBAL= Call BALANCE FACTOR (RCHILD (ROOT))

6. Balance AVL Tree, LL-Case

If BAL > 1 AND LBAL >= 0

Return Call RightRotate (ROOT)

7. Balance AVL Tree, LR-Case

If BAL > 1 AND LBAL < 0

LCHILD (ROOT) = Call LeftRotate (LCHILD (ROOT))

Return Call RightRotate (ROOT)

8. Balance AVL Tree, RR-Case

If BAL < -1 AND RBAL < 0

Return Call LeftRotate (ROOT)

9. Balance AVL Tree, RL-Case If BAL<-1 AND RBAL > 0

RCHILD (ROOT)=Call RightRotate (RCHILD (ROOT))

Return Call=LeftRotate (ROOT)

10. No AVL violations, Return tree

Return ROOT

=====

PROGRAM

=====

//Header file delecrations

```
#include<stdio.h>
#include<stdlib.h>
#define MAXOF(x,y)((x)>(y) ? (x):(y))
#define STOP 12345
#define MIN_VAL -999
struct treeNode
{
    struct treeNode *lchild;
    int data;
    int height;
    struct treeNode *rchild;
};
typedef struct treeNode avlNode;
typedef avlNode * avlTree;
```

//Function Declaration/Signatures/Prototypes

```
int heightAVL(avlTree);
avlTree createAVLNode(int);
avlTree insertAVL(avlTree,int);
avlTree rightRotate(avlTree);
avlTree leftRotate(avlTree);
int bFactor(avlTree);
avlTree deleteAvl(avlTree,int);
void inOrderAVL(avlTree);
void postOrderAVl(avlTree);
void preOrderAVL(avlTree);
void traverseAVL(avlTree);
```

```

void addressTrace(avlTree root);

avlTree createAVLTree();

avlTree emptyAVL(avlTree root);

//Driver Functions

int main()
{
    avlTree root=NULL;

    int choice,key;

    printf("===CREATING AVL TREE===\n");

    root = createAVLTree();

    do
    {
        printf("\nWhat do you want to perform ?....\n");

        printf("1. Insert node.  2.Delete node.  3.Print traversals.\n");

        printf("4.Height of AVL tree.   0.Exit\n");

        printf("CODE :: ");

        scanf("%d",&choice);

        switch(choice)
        {

            case 0 :emptyAVL(root);

                printf("\nThank you....\n");

                break;


            case 1 :printf("\n Enter the Node : ");

                scanf("%d",&key);

                root = insertAVL(root , key);

                traverseAVL(root);

                break;


            case 2 :printf("\n Enter the Node you want to Delete : ");

```

```

        scanf("%d",&key);

        root = deleteAvl(root , key);

        traverseAVL(root);

        break;

    case 3 :traverseAVL(root);

        break;

    case 4 :printf("\nThe height of the Binary Tree is : %d \n",
heightAVL(root));

        break;

    default:printf("\nINVALID INPUT.....\n");

        break;

    }

}while(choice!=0);
}

//function defination
int heightAVL(avlTree root)
{
    if(root==NULL)

        return 0;

    return root->height;
}

avlTree createAVLNode(int key)
{
    avlTree neww;

    neww=(avlTree)calloc(1,sizeof(avlNode));

    neww->data=key;

```

```

    neww->height=1;

    neww->lchild=neww->rchild=NULL;

    return(neww);
}

//right rotate subtree rooted at y
avlTree rightRotate(avlTree y)
{
    printf("\n\tright rotation at %d",y->data);

    avlTree x=y->lchild;

    avlTree T2=x->rchild;

    //rotate

    x->rchild=y;

    y->lchild=T2;

    //update height

    y->height=MAXOF(heightAVL(y->lchild),heightAVL(y->rchild))+1;

    x->height=MAXOF(heightAVL(x->lchild),heightAVL(x->rchild))+1;

    //return tree

    return x;
}

//left rotate subtree rooted at x
avlTree leftRotate(avlTree x)
{
    printf("\n\tleft rotation at %d",x->data);

    avlTree y=x->rchild;

    avlTree T2=y->lchild;

    //rotate

    y->lchild=x;

    x->rchild=T2;

    //height update

```

```

        x->height=MAXOF(heightAVL(x->lchild),heightAVL(x->rchild))+1;
        y->height=MAXOF(heightAVL(y->lchild),heightAVL(y->rchild))+1;
        return y;
    }
    int bFactor(avlTree root)
    {
        if(root==NULL)
            return 0;
        return heightAVL(root->lchild)- heightAVL(root->rchild);
    }
    avlTree insertAVL(avlTree root,int key)
    {
        int bal;
        //Usual BST insertion
        if (root == NULL)
            return (createAVLNode(key));
        if (key < root->data)
            root->lchild = insertAVL(root->lchild, key);
        else if (key > root->data)
            root->rchild = insertAVL(root->rchild, key);
        else
            return root; //Equal keys
        //Update the height of ancestor node, root
        root->height = 1 + MAXOF(heightAVL(root->lchild), heightAVL(root->rchild));
        //Check bFactor of root
        bal = bFactor(root);
        //Balance the tree
        //1. Left-Left Case
        if (bal > 1 && key < (root->lchild)->data)
        {

```

```

        printf("\nLL case:\n");
        return rightRotate(root);
    }
    //2. Right-Right Case
    if (bal < -1 && key > (root->rchild)->data)
    {
        printf("\nRR case:\n");
        return leftRotate(root);
    }
    //3. Left-Right Case
    if (bal > 1 && key > (root->lchild)->data)
    {
        printf("\nLR case:\n");
        root->lchild = leftRotate(root->lchild);
        return rightRotate(root);
    }
    //4. Right-Left Case
    if (bal < -1 && key < (root->rchild)->data)
    {
        printf("\nRL case:\n");
        root->rchild = rightRotate(root->rchild);
        return leftRotate(root);
    }
    //If no balancing required
    return root;
}

void inOrderAVL(avlTree root)
{
    if (root)

```

```

        {
            inOrderAVL(root->lchild);
            printf("%4d", root->data);
            inOrderAVL(root->rchild);
        }
    }

void preOrderAVL(avlTree root)
{
    if (root)
    {
        printf("%4d", root->data);
        preOrderAVL(root->lchild);
        preOrderAVL(root->rchild);
    }
}

void postOrderAVL(avlTree root)
{
    if (root)
    {
        postOrderAVL(root->lchild);
        postOrderAVL(root->rchild);
        printf("%4d", root->data);
    }
}

void traverseAVL(avlTree root)
{
    printf("\n\nInorder Traversal : ");
    inOrderAVL(root);
    printf("\n\nPreOrder Traversal : ");

```

```

    preOrderAVL(root);

    printf("\n\nPostOrder Traversal : ");

    postOrderAVL(root);

    printf("\n\n");
}

void addressTrace(avlTree root)
{
    if(root!=NULL)
    {
        addressTrace(root->lchild);

        printf("\n\t %3d [%9lu] (%ld) [LC:%9lu] [RC:%9lu]",root->data,(unsigned
long)root,root->height,(unsigned long)root->lchild,(unsigned long)root->rchild);

        addressTrace(root->rchild);
    }
}

avlTree createAVLTree()
{
    int value;

    avlTree root = NULL;

    printf("\nAVL Tree Creation\n");

    do
    {
        printf("\n\t\tEnter the key (to end, enter %d): ",STOP);

        scanf("%d", &value);

        if (value != STOP)
        {
            root = insertAVL(root, value);

            traverseAVL(root);

            printf("NODE TRAIL..");
        }
    }
}

```



```

        addressTrace(root);

        printf("\n");
    }
else
{
    printf("\tAVL Tree Created\n\tThe Traversal are...\n");
    traverseAVL(root);
}
}while(value!=STOP);
return root;
}

avlTree findMinValueNode(avlTree root)
{
    if (root == NULL)
        return NULL;

    if (root->lchild == NULL)
        return root;

    return findMinValueNode(root->lchild);
}

avlTree deleteAvl(avlTree root,int key)
{
    avlTree temp;

    int bal,lbal,rbal;

    if(root==NULL)
    {
        key=MIN_VAL;
        return root;
    }

```

```

if(key<root->data)
{
    root->lchild=deleteAvl(root->lchild,key);
}
else if(key>root->data)
{
    root->rchild=deleteAvl(root->rchild,key);
}
else
{
    if(root->lchild==NULL || root->rchild==NULL)
    {
        if(root->lchild!=NULL)
        {
            temp=root->lchild;
        }
        else
        {
            temp=root->rchild;
        }
        if(temp==NULL)
        {
            temp=root;
            root=NULL;
        }
        else
        {
            root->data=temp->data;
            root->height=temp->height;
            free(temp);
        }
    }
}

```

```

        }
    }
    else
    {
        temp=findMinValueNode(root->rchild);
        root->data=temp->data;
        root->rchild=deleteAvl(root->rchild,temp->data);
    }
}

if(root==NULL)
{
    return root;
}

root->height=MAXOF(heightAVL(root->lchild),heightAVL(root->rchild))+1;
bal=bFactor(root);
lbal=bFactor(root->lchild);
rbal=bFactor(root->rchild);

//1)ll case
if(bal>1 && lbal>=0)
{
    printf("\nLL case:");
    return rightRotate(root);
}

//2)lr case
if(bal>1 && lbal<0)
{
    printf("\nLR case:\n");
    root->lchild= leftRotate(root->lchild);
    return rightRotate(root);
}

```

```

    }

    //3)rr case
    if(bal<-1 && rbal<=0)
    {
        printf("\nRR case:\n");
        return leftRotate(root);
    }

    //4)rl case
    if(bal<-1 && rbal>0)
    {
        printf("\nRL case:\n");
        root->rchild=rightRotate(root->rchild);
        return leftRotate(root);
    }

    return root;
}

avlTree emptyAVL(avlTree root)
{
    if(root!=NULL)
    {
        emptyAVL(root->lchild);
        emptyAVL(root->rchild);
        printf("\t\t\tReleasing..%d...[%u]\n",root->data,(unsigned)root);
        free(root);
    }
}

```

```
=====
EXECUTION-TRAIL
=====
```

```
===CREATING AVL TREE===
```

```
AVL Tree Creation
```

```
Enter the key (to end, enter 12345): 12
```

```
Inorder Traversal : 12
```

```
PreOrder Traversal : 12
```

```
PostOrder Traversal : 12
```

```
NODE TRAIL..
```

```
12 [ 6688056] (1) [LC: 0] [RC: 0]
```

```
Enter the key (to end, enter 12345): 13
```

```
Inorder Traversal : 12 13
```

```
PreOrder Traversal : 12 13
```

```
PostOrder Traversal : 13 12
```

```
NODE TRAIL..
```

```
12 [ 6688056] (2) [LC: 0] [RC: 6688080]
```

13 [6688080] (1) [LC: 0] [RC: 0]

Enter the key (to end, enter 12345): 15

RR case:

left rotation at 12

Inorder Traversal : 12 13 15

PreOrder Traversal : 13 12 15

PostOrder Traversal : 12 15 13

NODE TRAIL..

12 [6688056] (1) [LC: 0] [RC: 0]

13 [6688080] (2) [LC: 6688056] [RC: 6688104]

15 [6688104] (1) [LC: 0] [RC: 0]

Enter the key (to end, enter 12345): 16

Inorder Traversal : 12 13 15 16

PreOrder Traversal : 13 12 15 16

PostOrder Traversal : 12 16 15 13

NODE TRAIL..

12 [6688056] (1) [LC: 0] [RC: 0]

```
13 [ 6688080] (3) [LC: 6688056] [RC: 6688104]
15 [ 6688104] (2) [LC:      0] [RC: 6688128]
16 [ 6688128] (1) [LC:      0] [RC:      0]
```

Enter the key (to end, enter 12345): 21

RR case:

left rotation at 15

Inorder Traversal : 12 13 15 16 21

PreOrder Traversal : 13 12 16 15 21

PostOrder Traversal : 12 15 21 16 13

NODE TRAIL..

```
12 [ 6688056] (1) [LC:      0] [RC:      0]
13 [ 6688080] (3) [LC: 6688056] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688152]
21 [ 6688152] (1) [LC:      0] [RC:      0]
```

Enter the key (to end, enter 12345): 9

Inorder Traversal : 9 12 13 15 16 21

PreOrder Traversal : 13 12 9 16 15 21

PostOrder Traversal : 9 12 15 21 16 13

NODE TRAIL..

```
9 [ 6688176] (1) [LC: 0] [RC: 0]
12 [ 6688056] (2) [LC: 6688176] [RC: 0]
13 [ 6688080] (3) [LC: 6688056] [RC: 6688128]
15 [ 6688104] (1) [LC: 0] [RC: 0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688152]
21 [ 6688152] (1) [LC: 0] [RC: 0]
```

Enter the key (to end, enter 12345): 55

Inorder Traversal : 9 12 13 15 16 21 55

PreOrder Traversal : 13 12 9 16 15 21 55

PostOrder Traversal : 9 12 15 55 21 16 13

NODE TRAIL..

```
9 [ 6688176] (1) [LC: 0] [RC: 0]
12 [ 6688056] (2) [LC: 6688176] [RC: 0]
13 [ 6688080] (4) [LC: 6688056] [RC: 6688128]
15 [ 6688104] (1) [LC: 0] [RC: 0]
16 [ 6688128] (3) [LC: 6688104] [RC: 6688152]
21 [ 6688152] (2) [LC: 0] [RC: 6688200]
55 [ 6688200] (1) [LC: 0] [RC: 0]
```

Enter the key (to end, enter 12345): 20

Inorder Traversal : 9 12 13 15 16 20 21 55

PreOrder Traversal : 13 12 9 16 15 21 20 55

PostOrder Traversal : 9 12 15 20 55 21 16 13

NODE TRAIL..

```
9 [ 6688176] (1) [LC:      0] [RC:      0]
12 [ 6688056] (2) [LC: 6688176] [RC:      0]
13 [ 6688080] (4) [LC: 6688056] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (3) [LC: 6688104] [RC: 6688152]
20 [ 6688224] (1) [LC:      0] [RC:      0]
21 [ 6688152] (2) [LC: 6688224] [RC: 6688200]
55 [ 6688200] (1) [LC:      0] [RC:      0]
```

Enter the key (to end, enter 12345): 90

RR case:

left rotation at 16

Inorder Traversal : 9 12 13 15 16 20 21 55 90

PreOrder Traversal : 13 12 9 21 16 15 20 55 90

PostOrder Traversal : 9 12 15 20 16 90 55 21 13

NODE TRAIL..

```

    9 [ 6688176] (1) [LC:      0] [RC:      0]
    12 [ 6688056] (2) [LC: 6688176] [RC:      0]
    13 [ 6688080] (4) [LC: 6688056] [RC: 6688152]
    15 [ 6688104] (1) [LC:      0] [RC:      0]
    16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
    20 [ 6688224] (1) [LC:      0] [RC:      0]
    21 [ 6688152] (3) [LC: 6688128] [RC: 6688200]
    55 [ 6688200] (2) [LC:      0] [RC: 6688248]
    90 [ 6688248] (1) [LC:      0] [RC:      0]

```

Enter the key (to end, enter 12345): 75

RL case:

right rotation at 90

left rotation at 55

Inorder Traversal : 9 12 13 15 16 20 21 55 75 90

PreOrder Traversal : 13 12 9 21 16 15 20 75 55 90

PostOrder Traversal : 9 12 15 20 16 55 90 75 21 13

NODE TRAIL..

```

    9 [ 6688176] (1) [LC:      0] [RC:      0]
    12 [ 6688056] (2) [LC: 6688176] [RC:      0]
    13 [ 6688080] (4) [LC: 6688056] [RC: 6688152]
    15 [ 6688104] (1) [LC:      0] [RC:      0]
    16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
    20 [ 6688224] (1) [LC:      0] [RC:      0]

```

```

21 [ 6688152] (3) [LC: 6688128] [RC: 6688272]
55 [ 6688200] (1) [LC:      0] [RC:      0]
75 [ 6688272] (2) [LC: 6688200] [RC: 6688248]
90 [ 6688248] (1) [LC:      0] [RC:      0]

```

Enter the key (to end, enter 12345): 34

RR case:

left rotation at 13

Inorder Traversal : 9 12 13 15 16 20 21 34 55 75 90

PreOrder Traversal : 21 13 12 9 16 15 20 75 55 34 90

PostOrder Traversal : 9 12 15 20 16 13 34 55 90 75 21

NODE TRAIL..

```

9 [ 6688176] (1) [LC:      0] [RC:      0]
12 [ 6688056] (2) [LC: 6688176] [RC:      0]
13 [ 6688080] (3) [LC: 6688056] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
20 [ 6688224] (1) [LC:      0] [RC:      0]
21 [ 6688152] (4) [LC: 6688080] [RC: 6688272]
34 [ 6688296] (1) [LC:      0] [RC:      0]
55 [ 6688200] (2) [LC: 6688296] [RC:      0]
75 [ 6688272] (3) [LC: 6688200] [RC: 6688248]
90 [ 6688248] (1) [LC:      0] [RC:      0]

```

Enter the key (to end, enter 12345): 22

LL case:

right rotation at 55

Inorder Traversal : 9 12 13 15 16 20 21 22 34 55 75 90

PreOrder Traversal : 21 13 12 9 16 15 20 75 34 22 55 90

PostOrder Traversal : 9 12 15 20 16 13 22 55 34 90 75 21

NODE TRAIL..

9	[6688176]	(1)	[LC:	0]	[RC:	0]
12	[6688056]	(2)	[LC:	6688176]	[RC:	0]
13	[6688080]	(3)	[LC:	6688056]	[RC:	6688128]
15	[6688104]	(1)	[LC:	0]	[RC:	0]
16	[6688128]	(2)	[LC:	6688104]	[RC:	6688224]
20	[6688224]	(1)	[LC:	0]	[RC:	0]
21	[6688152]	(4)	[LC:	6688080]	[RC:	6688272]
22	[6688320]	(1)	[LC:	0]	[RC:	0]
34	[6688296]	(2)	[LC:	6688320]	[RC:	6688200]
55	[6688200]	(1)	[LC:	0]	[RC:	0]
75	[6688272]	(3)	[LC:	6688296]	[RC:	6688248]
90	[6688248]	(1)	[LC:	0]	[RC:	0]

Enter the key (to end, enter 12345): 10

LR case:

left rotation at 9

right rotation at 12

Inorder Traversal : 9 10 12 13 15 16 20 21 22 34 55 75 90

PreOrder Traversal : 21 13 10 9 12 16 15 20 75 34 22 55 90

PostOrder Traversal : 9 12 10 15 20 16 13 22 55 34 90 75 21

NODE TRAIL..

9	[6688176]	(1)	[LC:	0]	[RC:	0]
10	[6688344]	(2)	[LC:	6688176]	[RC:	6688056]
12	[6688056]	(1)	[LC:	0]	[RC:	0]
13	[6688080]	(3)	[LC:	6688344]	[RC:	6688128]
15	[6688104]	(1)	[LC:	0]	[RC:	0]
16	[6688128]	(2)	[LC:	6688104]	[RC:	6688224]
20	[6688224]	(1)	[LC:	0]	[RC:	0]
21	[6688152]	(4)	[LC:	6688080]	[RC:	6688272]
22	[6688320]	(1)	[LC:	0]	[RC:	0]
34	[6688296]	(2)	[LC:	6688320]	[RC:	6688200]
55	[6688200]	(1)	[LC:	0]	[RC:	0]
75	[6688272]	(3)	[LC:	6688296]	[RC:	6688248]
90	[6688248]	(1)	[LC:	0]	[RC:	0]

Enter the key (to end, enter 12345): 6

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 55 75 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 34 22 55 90

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 55 34 90 75 21

NODE TRAIL..

```
6 [ 6688368] (1) [LC: 0] [RC: 0]
9 [ 6688176] (2) [LC: 6688368] [RC: 0]
10 [ 6688344] (3) [LC: 6688176] [RC: 6688056]
12 [ 6688056] (1) [LC: 0] [RC: 0]
13 [ 6688080] (4) [LC: 6688344] [RC: 6688128]
15 [ 6688104] (1) [LC: 0] [RC: 0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
20 [ 6688224] (1) [LC: 0] [RC: 0]
21 [ 6688152] (5) [LC: 6688080] [RC: 6688272]
22 [ 6688320] (1) [LC: 0] [RC: 0]
34 [ 6688296] (2) [LC: 6688320] [RC: 6688200]
55 [ 6688200] (1) [LC: 0] [RC: 0]
75 [ 6688272] (3) [LC: 6688296] [RC: 6688248]
90 [ 6688248] (1) [LC: 0] [RC: 0]
```

Enter the key (to end, enter 12345): 76

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 55 75 76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 34 22 55 90
76

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 55 34 76 90 75
21

NODE TRAIL..

```

6 [ 6688368] (1) [LC:      0] [RC:      0]
9 [ 6688176] (2) [LC: 6688368] [RC:      0]
10 [ 6688344] (3) [LC: 6688176] [RC: 6688056]
12 [ 6688056] (1) [LC:      0] [RC:      0]
13 [ 6688080] (4) [LC: 6688344] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
20 [ 6688224] (1) [LC:      0] [RC:      0]
21 [ 6688152] (5) [LC: 6688080] [RC: 6688272]
22 [ 6688320] (1) [LC:      0] [RC:      0]
34 [ 6688296] (2) [LC: 6688320] [RC: 6688200]
55 [ 6688200] (1) [LC:      0] [RC:      0]
75 [ 6688272] (3) [LC: 6688296] [RC: 6688248]
76 [ 6688392] (1) [LC:      0] [RC:      0]
90 [ 6688248] (2) [LC: 6688392] [RC:      0]

```

Enter the key (to end, enter 12345): 43

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 43 55 75 76
90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 34 22 55 43
90 76

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 43 55 34 76 90
75 21

NODE TRAIL..

```

6 [ 6688368] (1) [LC:      0] [RC:      0]
9 [ 6688176] (2) [LC: 6688368] [RC:      0]

```

```

10 [ 6688344] (3) [LC: 6688176] [RC: 6688056]
12 [ 6688056] (1) [LC:      0] [RC:      0]
13 [ 6688080] (4) [LC: 6688344] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
20 [ 6688224] (1) [LC:      0] [RC:      0]
21 [ 6688152] (5) [LC: 6688080] [RC: 6688272]
22 [ 6688320] (1) [LC:      0] [RC:      0]
34 [ 6688296] (3) [LC: 6688320] [RC: 6688200]
43 [ 6688416] (1) [LC:      0] [RC:      0]
55 [ 6688200] (2) [LC: 6688416] [RC:      0]
75 [ 6688272] (4) [LC: 6688296] [RC: 6688248]
76 [ 6688392] (1) [LC:      0] [RC:      0]
90 [ 6688248] (2) [LC: 6688392] [RC:      0]

```

Enter the key (to end, enter 12345): 60

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 43 55 60 75
76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 34 22 55 43
60 90 76

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 43 60 55 34 76
90 75 21

NODE TRAIL..

```

6 [ 6688368] (1) [LC:      0] [RC:      0]
9 [ 6688176] (2) [LC: 6688368] [RC:      0]
10 [ 6688344] (3) [LC: 6688176] [RC: 6688056]

```



```

12 [ 6688056] (1) [LC:      0] [RC:      0]
13 [ 6688080] (4) [LC: 6688344] [RC: 6688128]
15 [ 6688104] (1) [LC:      0] [RC:      0]
16 [ 6688128] (2) [LC: 6688104] [RC: 6688224]
20 [ 6688224] (1) [LC:      0] [RC:      0]
21 [ 6688152] (5) [LC: 6688080] [RC: 6688272]
22 [ 6688320] (1) [LC:      0] [RC:      0]
34 [ 6688296] (3) [LC: 6688320] [RC: 6688200]
43 [ 6688416] (1) [LC:      0] [RC:      0]
55 [ 6688200] (2) [LC: 6688416] [RC: 6713120]
60 [ 6713120] (1) [LC:      0] [RC:      0]
75 [ 6688272] (4) [LC: 6688296] [RC: 6688248]
76 [ 6688392] (1) [LC:      0] [RC:      0]
90 [ 6688248] (2) [LC: 6688392] [RC:      0]

```

Enter the key (to end, enter 12345): 12345

AVL Tree Created

The Traversal are...

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 43 55 60 75
76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 34 22 55 43
60 90 76

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 43 60 55 34 76
90 75 21

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.
4.Height of AVL tree. 0.Exit

CODE :: 1

Enter the Node : 45

RL case:

right rotation at 55

left rotation at 34

Inorder Traversal : 6 9 10 12 13 15 16 20 21 22 34 43 45 55 60
75 76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 20 75 43 34 22 55
45 60 90 76

PostOrder Traversal : 6 9 12 10 15 20 16 13 22 34 45 60 55 43
76 90 75 21

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.
4.Height of AVL tree. 0.Exit

CODE :: 2

Enter the Node you want to Delete : 20

Inorder Traversal : 6 9 10 12 13 15 16 21 22 34 43 45 55 60 75
76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 75 43 34 22 55 45
60 90 76

PostOrder Traversal : 6 9 12 10 15 16 13 22 34 45 60 55 43 76
90 75 21

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.

4.Height of AVL tree. 0.Exit

CODE :: 3

Inorder Traversal : 6 9 10 12 13 15 16 21 22 34 43 45 55 60 75
76 90

PreOrder Traversal : 21 13 10 9 6 12 16 15 75 43 34 22 55 45
60 90 76

PostOrder Traversal : 6 9 12 10 15 16 13 22 34 45 60 55 43 76
90 75 21

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.

4.Height of AVL tree. 0.Exit

CODE :: 4

The height of the Binary Tree is : 5

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.
4.Height of AVL tree. 0.Exit
CODE :: 2

Enter the Node you want to Delete : 12

LL case:

right rotation at 10

Inorder Traversal : 6 9 10 13 15 16 21 22 34 43 45 55 60 75 76
90

PreOrder Traversal : 21 13 9 6 10 16 15 75 43 34 22 55 45 60
90 76

PostOrder Traversal : 6 10 9 15 16 13 22 34 45 60 55 43 76 90
75 21

What do you want to perform ?....

1. Insert node. 2.Delete node. 3.Print traversals.
4.Height of AVL tree. 0.Exit
CODE :: 0

Releasing..6...[6688368]

Releasing..10...[6688344]

Releasing..9...[6688176]

Releasing..15...[6688104]

Releasing..16...[6688128]

Releasing..13...[6688080]

Releasing..22...[6688320]

Releasing..34...[6688296]

Releasing..45...[6688600]

Releasing..60...[6713120]

Releasing..55...[6688200]

Releasing..43...[6688416]

Releasing..76...[6688392]

Releasing..90...[6688248]

Releasing..75...[6688272]

Releasing..21...[6688152]

Thank you....

=====

VIVA-VOICE

=====

1.How do AVL tree differ from BST?

Ans: Binary search tree is a binary tree because both the trees contain the utmost two children , every AVL tree is also a binary tree because AVL tree also has the utmost two children. In BST, there is no term exists, such as balance factor, in the AVL tree,each node contains a balance factor, and the value of the balance factor must be either -1, 0, or 1.

2.With proper schematic explain tree rotations..

Ans: Please Refer tree rotations theory part.

3.State the significance of balance factor in AVL.

Ans: Balance Factor:AVL trees are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the Balance Factor. $\text{BalanceFactor} = \text{height}(\text{left-sutree}) - \text{height}(\text{right-sutree})$ If the difference in the height of left and right sub-trees is more than 1, so balance factor is used to find the imbalance

=====

CONCLUSION

=====

In this experiment we have studied what is AVL tree and its various operations. We have also learnt how to write the algorithms for operations of AVL tree. Sir has explained us in detail the use of AVL tree over BST . Sir also explained us node insertion , deletion, balance factor, height of tree in all cases.