NAME: ADWAIT S PURAO
UID: 2021300101
EXP NO. :7
AIM: To perform insertion in an AVL Tree

THEORY:

AVL TREES:

AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1.

AVL tree got its name after its inventor Georgy Adelson-Velsky and Landis.

Why AVL Trees?

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that the height of the tree remains O(log(n)) after every insertion and deletion, then we can guarantee an upper bound of O(log(n)) for all these operations. The height of an AVL tree is always O(log(n)) where n is the number of nodes in the tree.

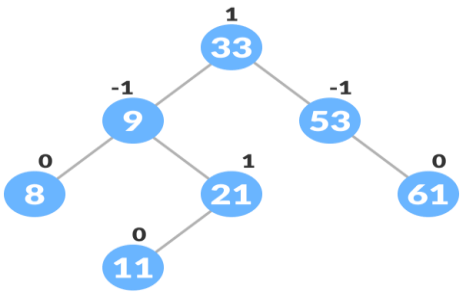
Balance Factor

Balance factor of a node in an AVL tree is the difference between the height of the left subtree and that of the right subtree of that node.

Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)

The self balancing property of an avl tree is maintained by the balance factor. The value of balance factor should always be -1, 0 or +1.

An example of a balanced avl tree is:



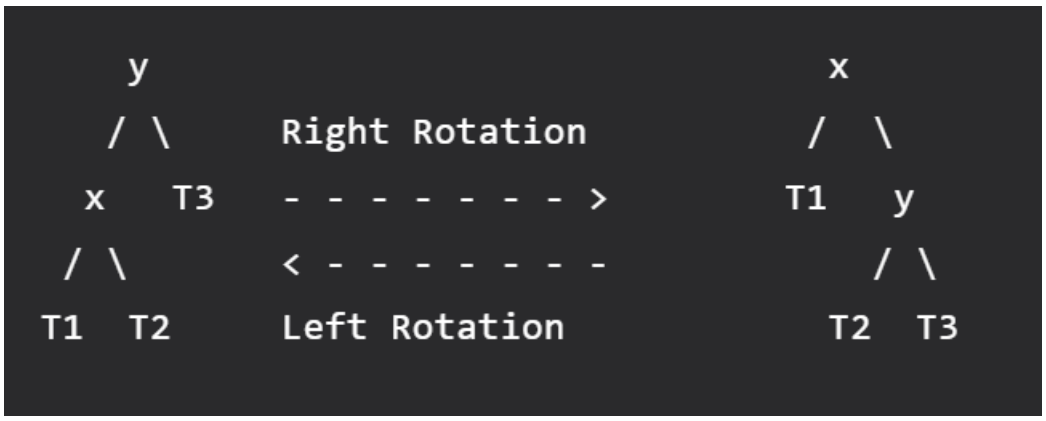
Insertion in AVL Tree:

To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing.

Following are two basic operations that can be performed to balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

- Left Rotation
- Right Rotation

T1, T2 and T3 are subtrees of the tree, rooted with y (on the left side) or x (on the right side)



Keys in both of the above trees follow the following order

keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)

So BST property is not violated anywhere.

Steps to follow for insertion:

Let the newly inserted node be w

Perform standard BST insert for w.

Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that need to be handled as x, y and z can be arranged in 4 ways.

Following are the possible 4 arrangements:

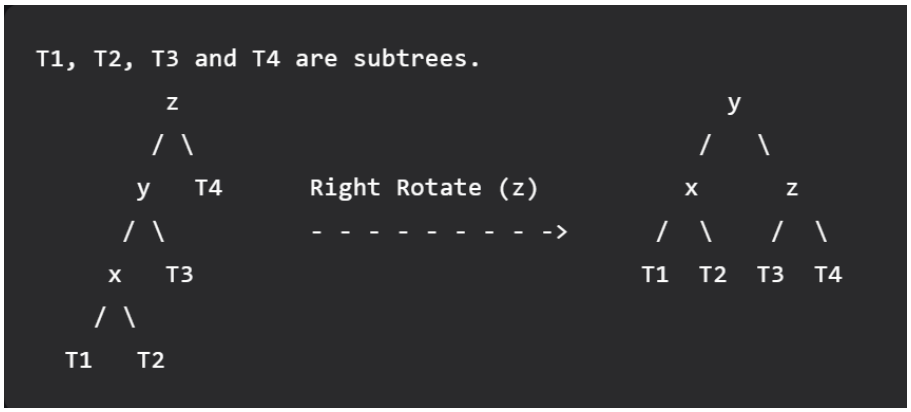
y is the left child of z and x is the left child of y (Left Left Case)

y is the left child of z and x is the right child of y (Left Right Case)

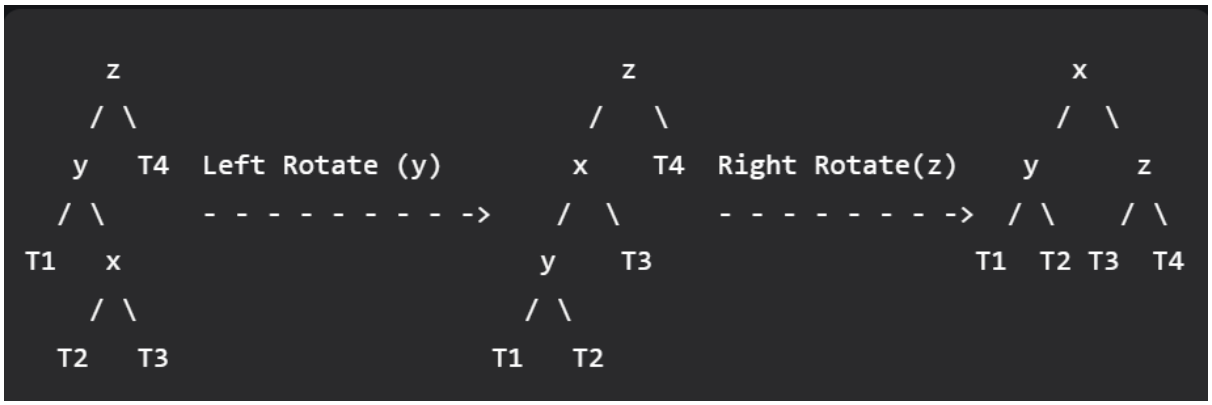
y is the right child of z and x is the right child of y (Right Right Case)

y is the right child of z and x is the left child of y (Right Left Case)

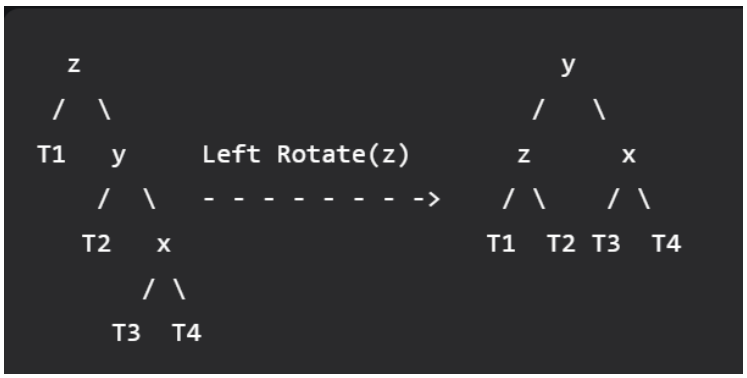
1. Left Left Case



2. Left Right Case

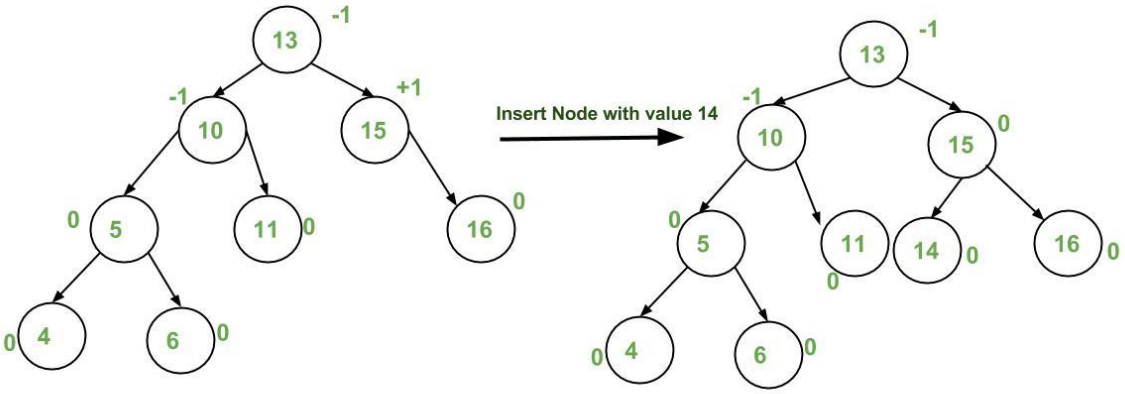


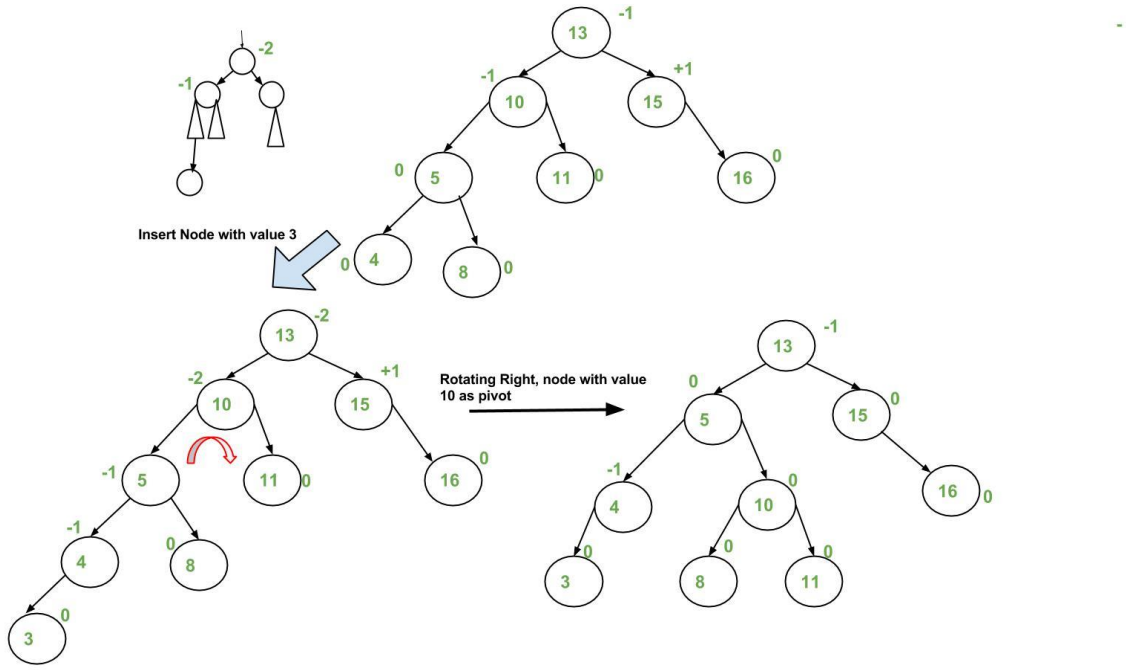
3. Right Right Case

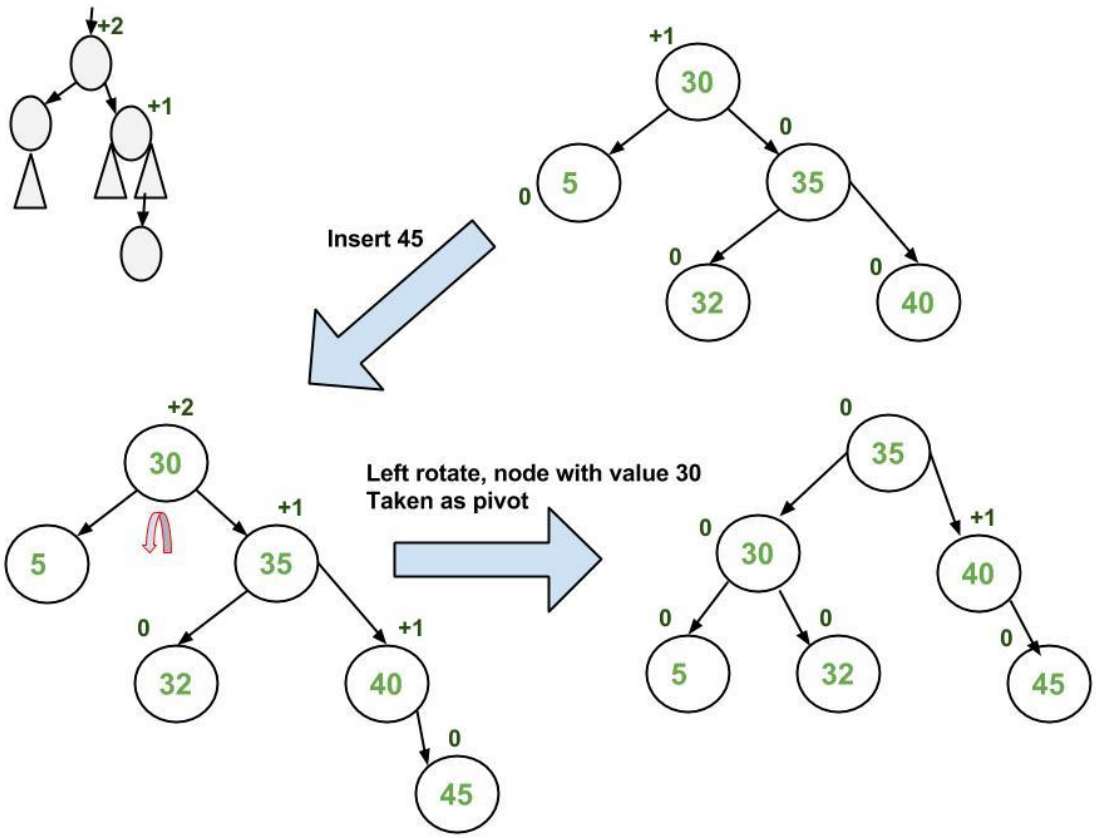


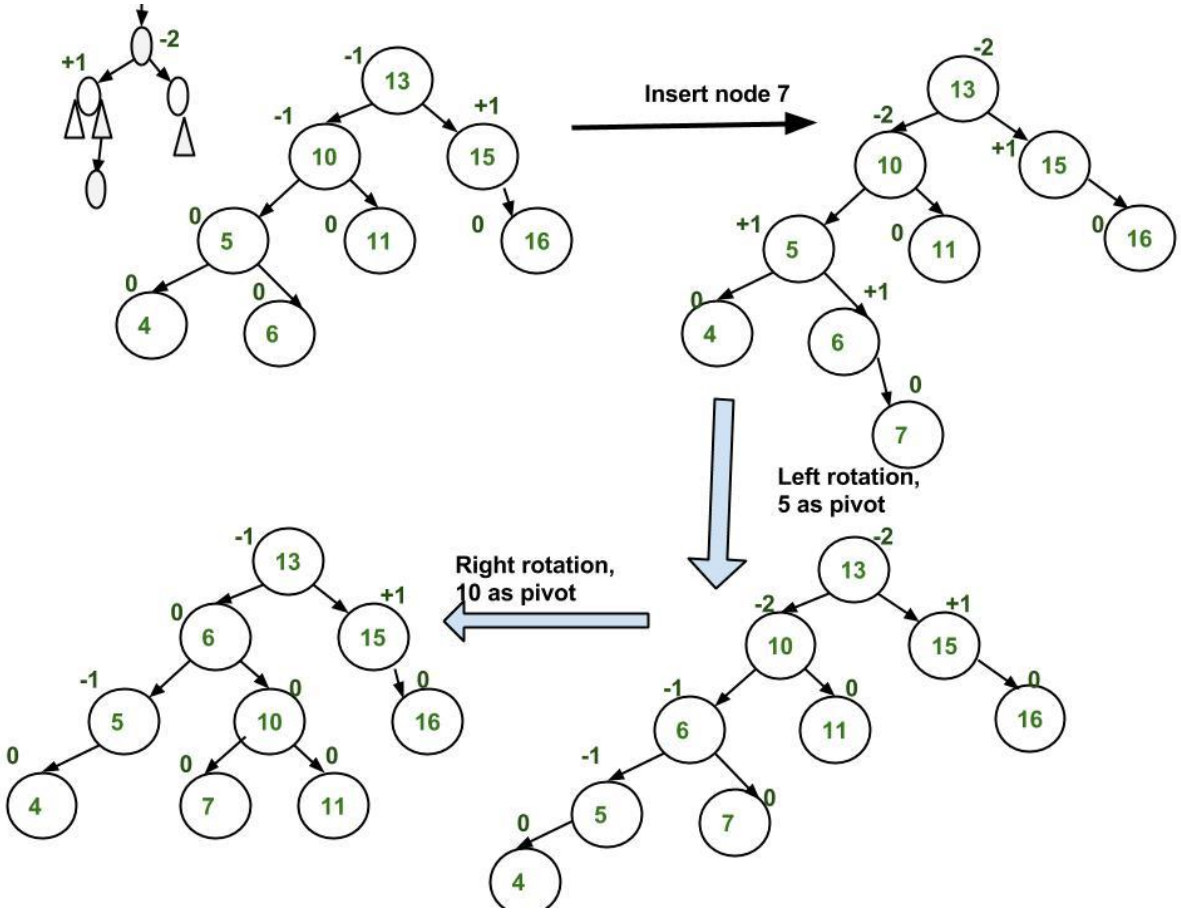
4. Right Left Case

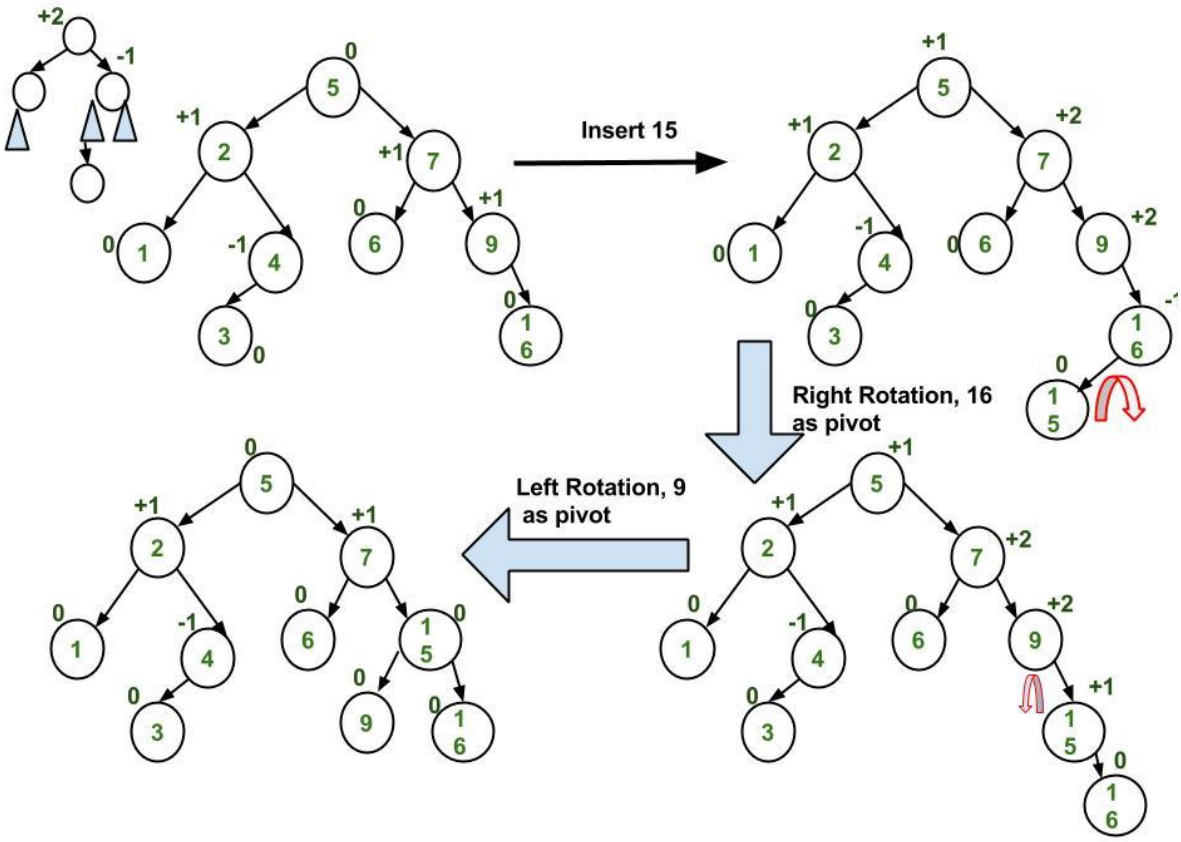
```
   z                                      z                                      x
  / \                                    / \                                    /  \
T1   y   Right Rotate (y)    T1   x      Left Rotate(z)   z      y
    / \  - - - - - - - - - ->    /  \   - - - - - - - - ->  / \    / \
   x   T4                      T2   y                     T1  T2  T3  T4
  / \                              /  \
T2   T3                           T3   T4
```






+2
+1
Insert 45
+1
30
0
5
35
0
0
32
0
40
+2
30
5
+1
35
0
32
+1
40
0
45
Left rotate, node with value 30
Taken as pivot
0
35
0
30
+1
40
0
5
0
32
0
45


+1
-2
-1
13
-1
10
+1
15
0
5
0
11
0
16
0
4
0
6
Insert node 7
-2
13
-2
10
+1
15
+1
5
0
11
0
16
0
4
+1
6
0
7
Left rotation,
5 as pivot
-1
13
0
6
+1
15
-1
5
0
10
0
16
0
4
0
7
0
11
Right rotation,
10 as pivot
-2
13
-2
10
+1
15
-1
6
0
11
0
16
-1
5
0
7
0
4



ALGORITHM:

1) Struct node
   - Int data
   - struct node *left
   - struct node *right
   - int height
2) Function int height(struct node *new_node)
   - if new_node == NULL
     - return 0
   - return new_node->height
3) Function struct node *create(int data)
   - Allocate memory for struct node * ptr
   - ptr->data = data
   - ptr->left = NULL
   - ptr->right = NULL
   - ptr->height = 1
   - return ptr
4) Function int findMax(int n1, int n2)
   - if (n1 > n2)
     - return n1
   - else

return n2

5) Function int balanceFact(struct node *new)

```
if (new == NULL)

    return 0

return height(new->left) - height(new->right)
```

6) Function struct node *leftRotation(struct node *root)

```
rtright = root->right

rtrl = rtright->left

rtright->left = root

root->right = rtrl

root->height = 1 + findMax(height(root->left), height(root->right))

rtright->height = 1 + findMax(height(rtright->left), height(rtright->right))

return rtright
```

7) Function struct node *rightRotation(struct node *root)

```
rtleft = root->left
rtlr = rtleft->right
rtleft->right = root
root->left = rtlr
root->height = 1 + findMax(height(root->left), height(root->right))
rtleft->height = 1 + findMax(height(rtleft->left), height(rtleft->right))
return rtleft
```

8) Function struct node *Insertion(struct node *root, int data)

```
if root == NULL

    return (create(data))

else if root->data > data

    root->left = Insertion(root->left, data)

else if root->data < data

    root->right = Insertion(root->right, data)

else

    return root
```

root->height = 1 + findMax(height(root->left), height(root->right))

int bal = balanceFact(root)

//LL Rotation

if (bal > 1 && root->left->data > data)

return rightRotation(root)

//RR Rotation

if bal < -1 && root->right->data < data

return leftRotation(root)

//RL Rotation

if (bal > 1 && root->left->data < data)

root->left = leftRotation(root->left);

return rightRotation(root);

//LR Rotation

if bal < -1 && root->right->data > data

root->right = rightRotation(root->right)

return leftRotation(root)

return root

9) Function void Inorder(struct node *root)

if root != NULL

Inorder(root->left)

printf("%d bf=%d, ", root->data,balanceFact(root))

Inorder(root->right)

10) Main function

Set struct node* root =NULL

Initialize integer variables val and ch

Set flag=0

Do while flag!=1

Take input of the choice

Switch (ch)

Case 1:

Take input the value to be inserted

Call function Insertion

Print Inorder Traversal of Tree

Break

Case 2:

Set flag to 1

Print program finished

Break

Default:

Invalid choice

Break

PROBLEM SOLVING ON CONCEPT:

Name: Aswath .S. Purao
UID: 2021300101

Date
Page

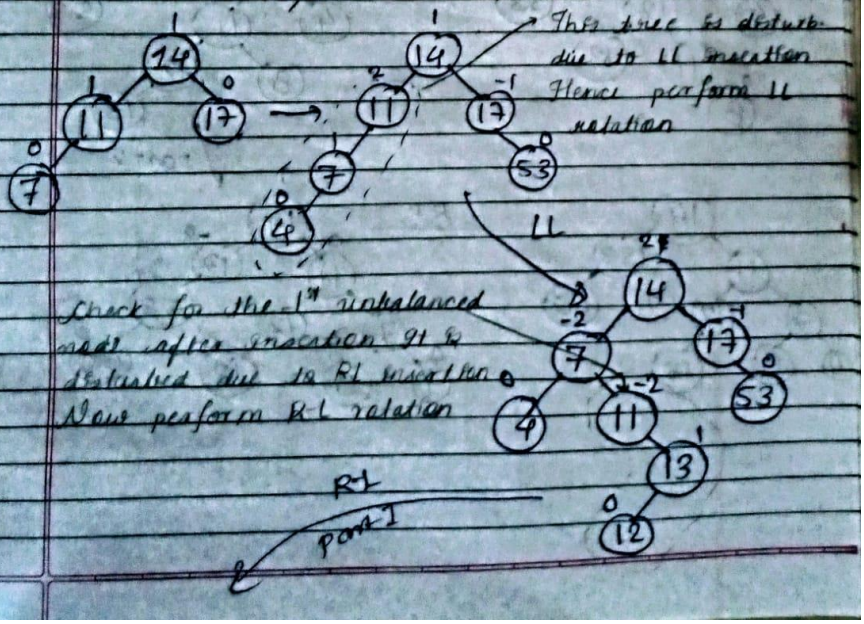
AVL Tree Insertion:

Let the node that needs balancing be

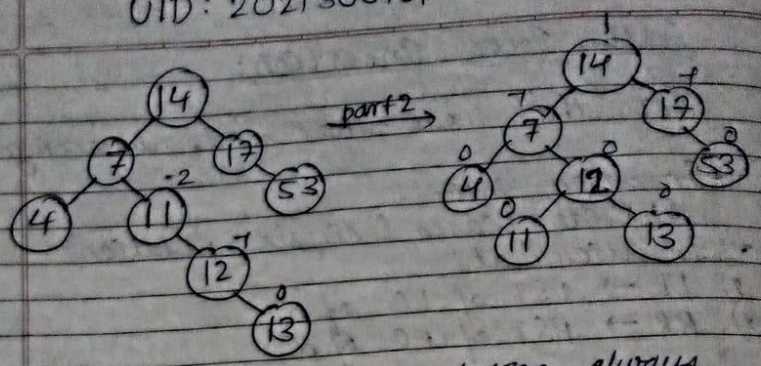
→ ~~Insertion into~~
Outside cases (Single rotation)
→ Insertion at:
1) LL → LST of LC of α
2) RR → RST of RC of α

Inside cases (Double rotation)
1) RL → RST of LC of α
2) LR → LST of RC of α

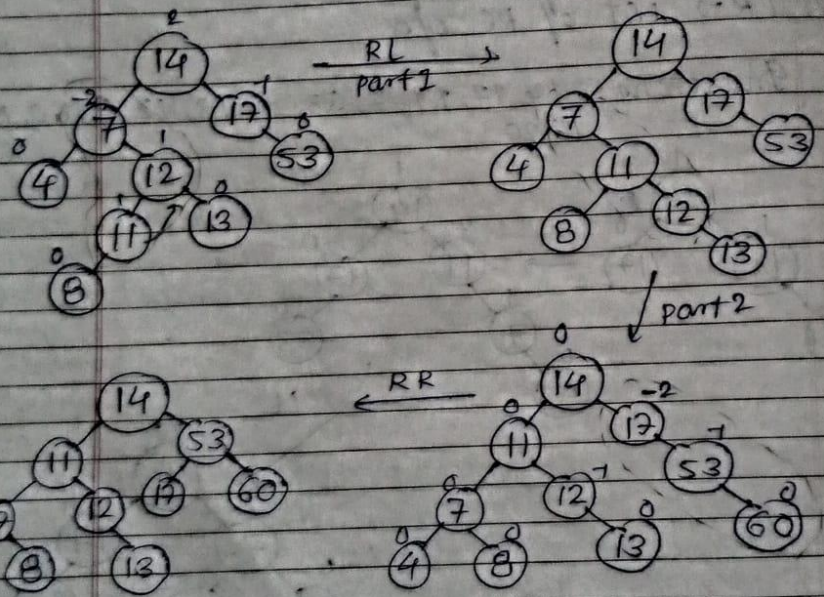
Insertions: 14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20

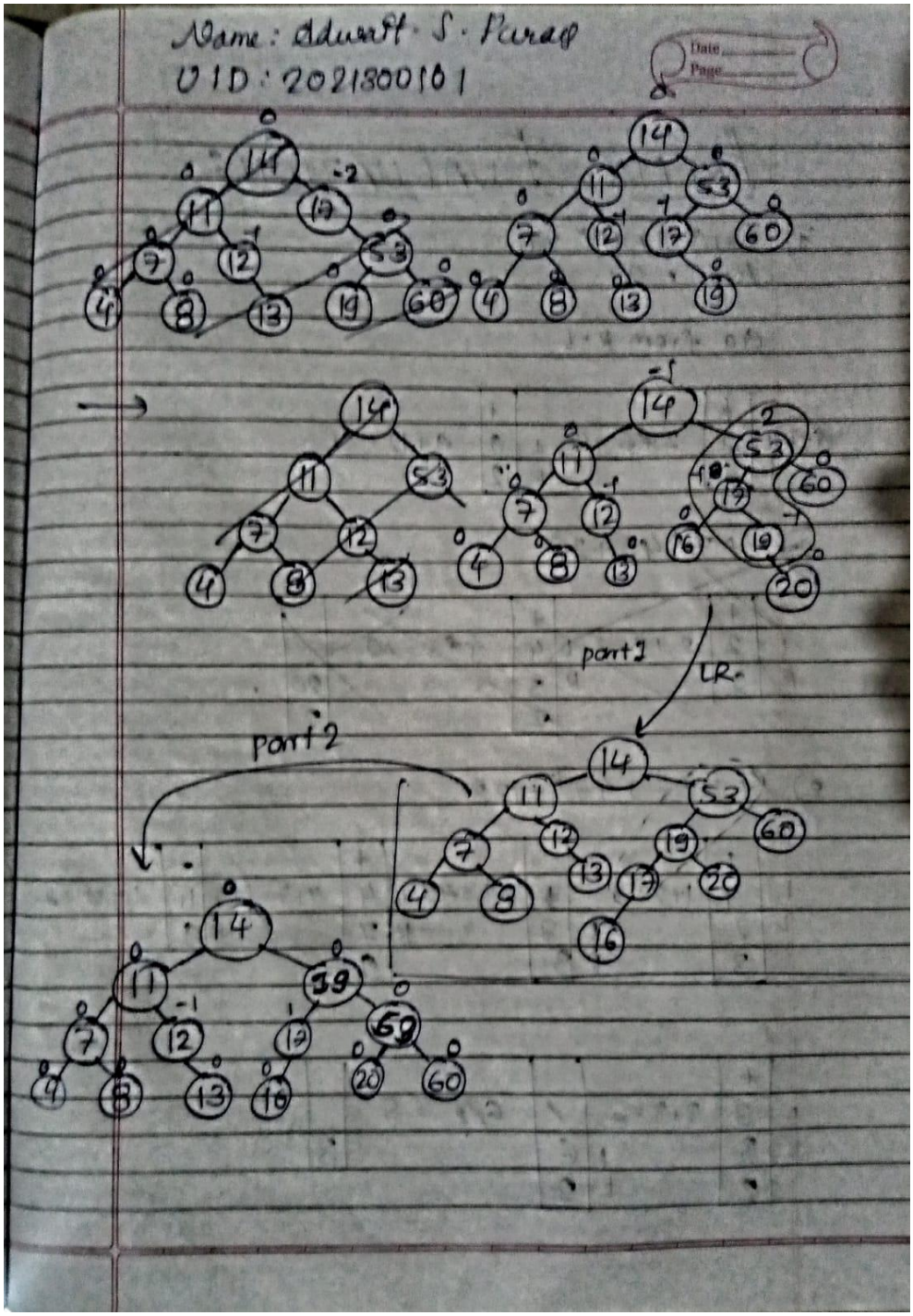


Name: Aswath.S.Purao
UID: 2021300101



Note: While performing any rotation always remember the median element would be the root.





CODE:

```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
```

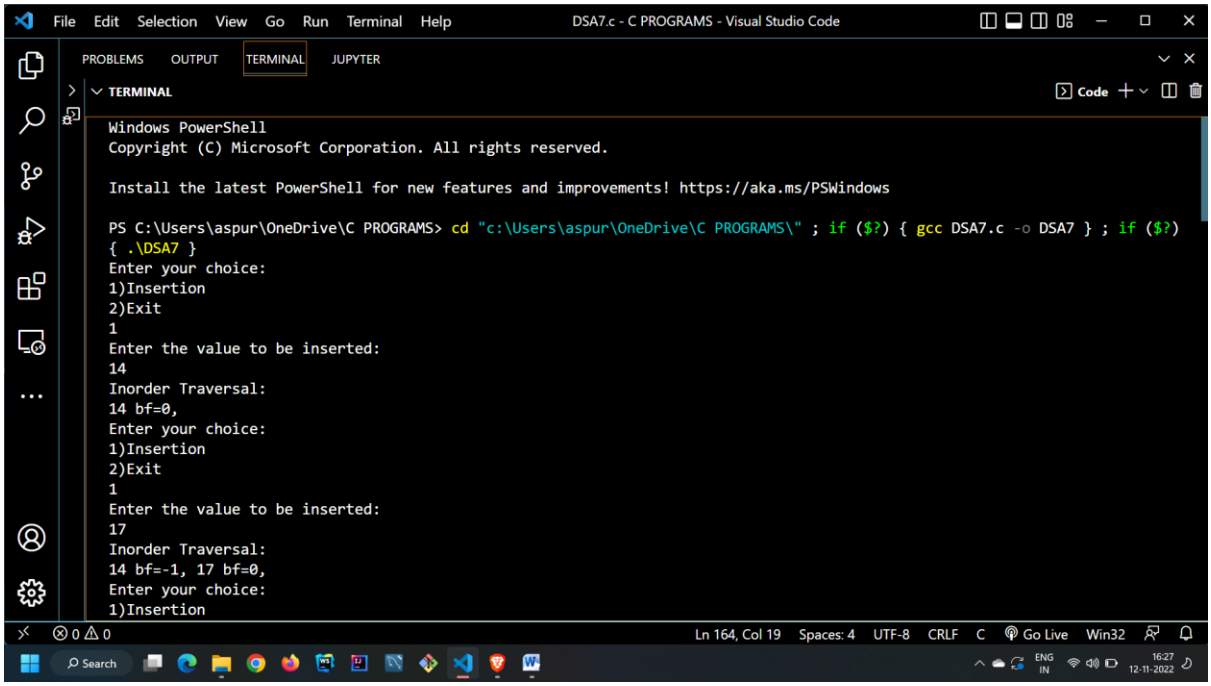
```
    int height;
};

int height(struct node *new_node)
{
    if (new_node == NULL)
    {
        return 0;
    }
    return new_node->height;
}

struct node *create(int data)
{
    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    ptr->data = data;
    ptr->left = NULL;
    ptr->right = NULL;
    ptr->height = 1;

    return ptr;
}

int findMax(int n1, int n2)
{
    if (n1 > n2)
    {
        return n1;
    }
    else{
        return n2;
    }
}

int balanceFact(struct node *new)
{
    if (new == NULL)
    {
        return 0;
    }
    return height(new->left) - height(new->right);
}

struct node *leftRotation(struct node *root)
{
    struct node *rtright = root->right;
    struct node *rtrl = rtright->left;
```

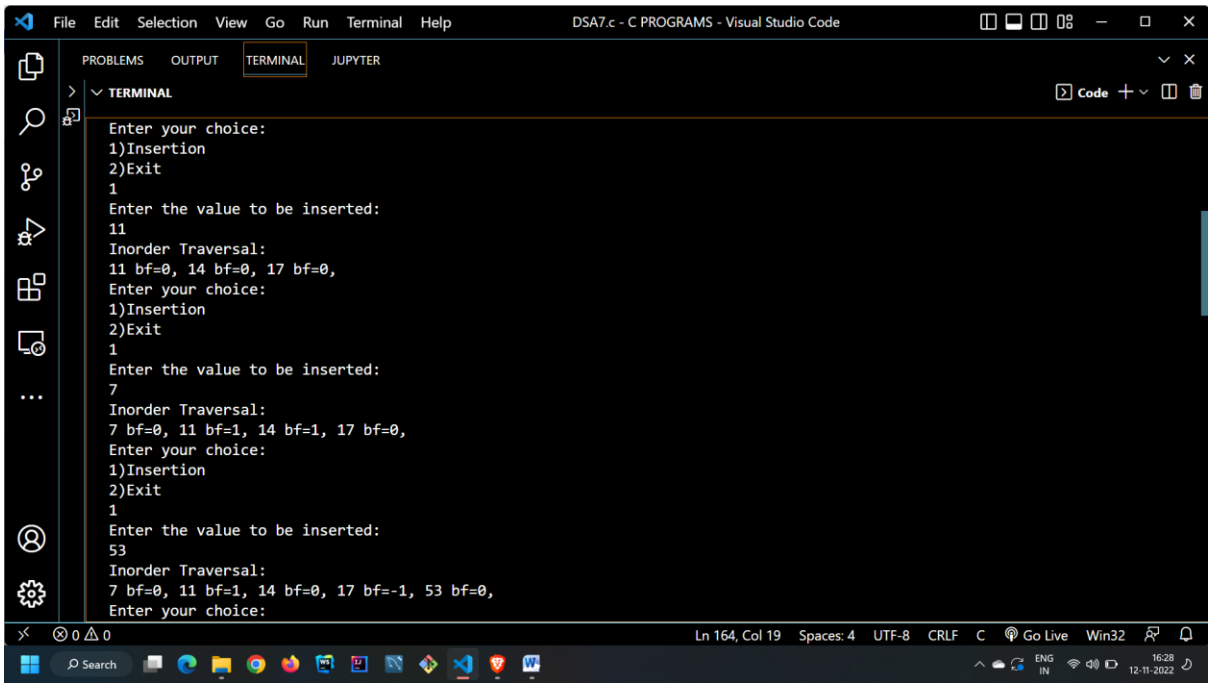
```
    rtright->left = root;
    root->right = rtrl;

    root->height = 1 + findMax(height(root->left), height(root->right));
    rtright->height = 1 + findMax(height(rtright->left), height(rtright-
>right));

    return rtright;
}

struct node *rightRotation(struct node *root)
{
    struct node *rtleft = root->left;
    struct node *rtlr = rtleft->right;

    rtleft->right = root;
    root->left = rtlr;

    root->height = 1 + findMax(height(root->left), height(root->right));
    rtleft->height = 1 + findMax(height(rtleft->left), height(rtleft->right));

    return rtleft;
}

struct node *Insertion(struct node *root, int data)
{
    if (root == NULL)
    {
        return (create(data));
    }
    else if (root->data > data)
    {
        root->left = Insertion(root->left, data);
    }
    else if (root->data < data)
    {
        root->right = Insertion(root->right, data);
    }
    else
    {
        return root;
    }

    root->height = 1 + findMax(height(root->left), height(root->right));

    int bal = balanceFact(root);

    //LL Rotation
```

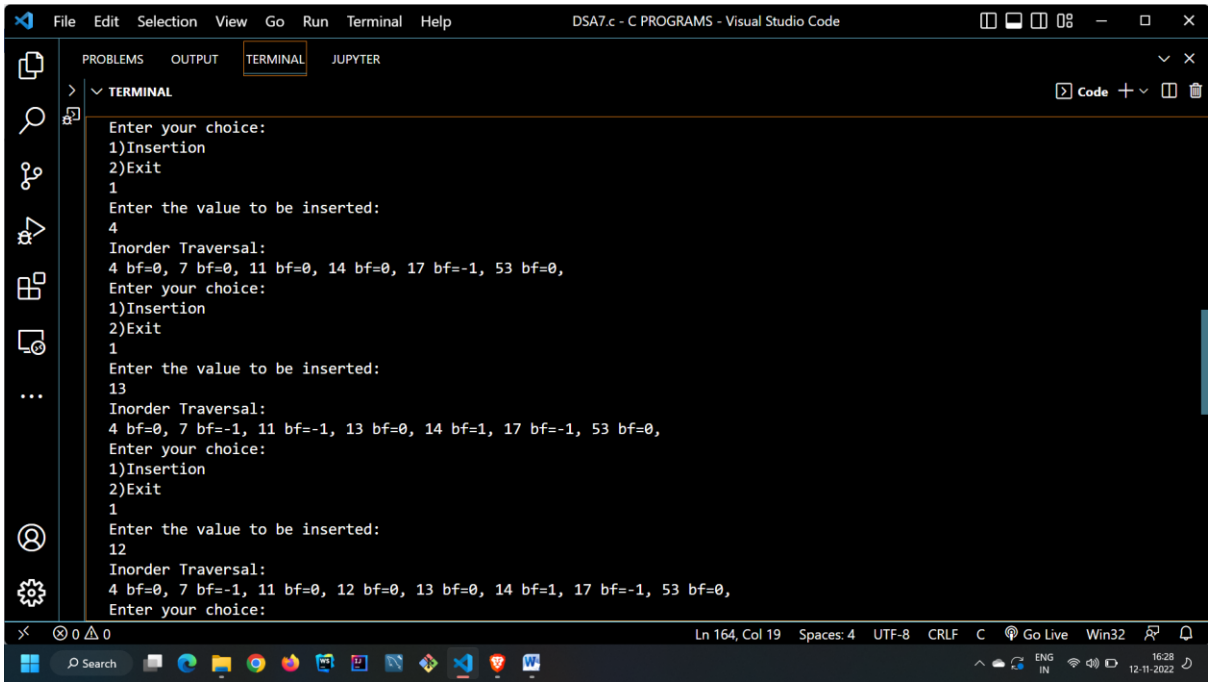
```
    if (bal > 1 && root->left->data > data)
    {
        return rightRotation(root);
    }

    //RR Rotation
    if (bal < -1 && root->right->data < data)
    {
        return leftRotation(root);
    }

    //RL Rotation
    if (bal > 1 && root->left->data < data)
    {
        root->left = leftRotation(root->left);
        return rightRotation(root);
    }

    //LR Rotation
    if (bal < -1 && root->right->data > data)
    {
        root->right = rightRotation(root->right);
        return leftRotation(root);
    }

    return root;
}

void Inorder(struct node *root)
{

    if (root != NULL)
    {
        Inorder(root->left);
        printf("%d bf=%d, ", root->data,balanceFact(root));
        Inorder(root->right);
    }
}
int main()
{

    struct node *root = NULL;
    int val, ch;

    int flag = 0;
    do
    {
        printf("Enter your choice:\n1)Insertion\n2)Exit\n");
```

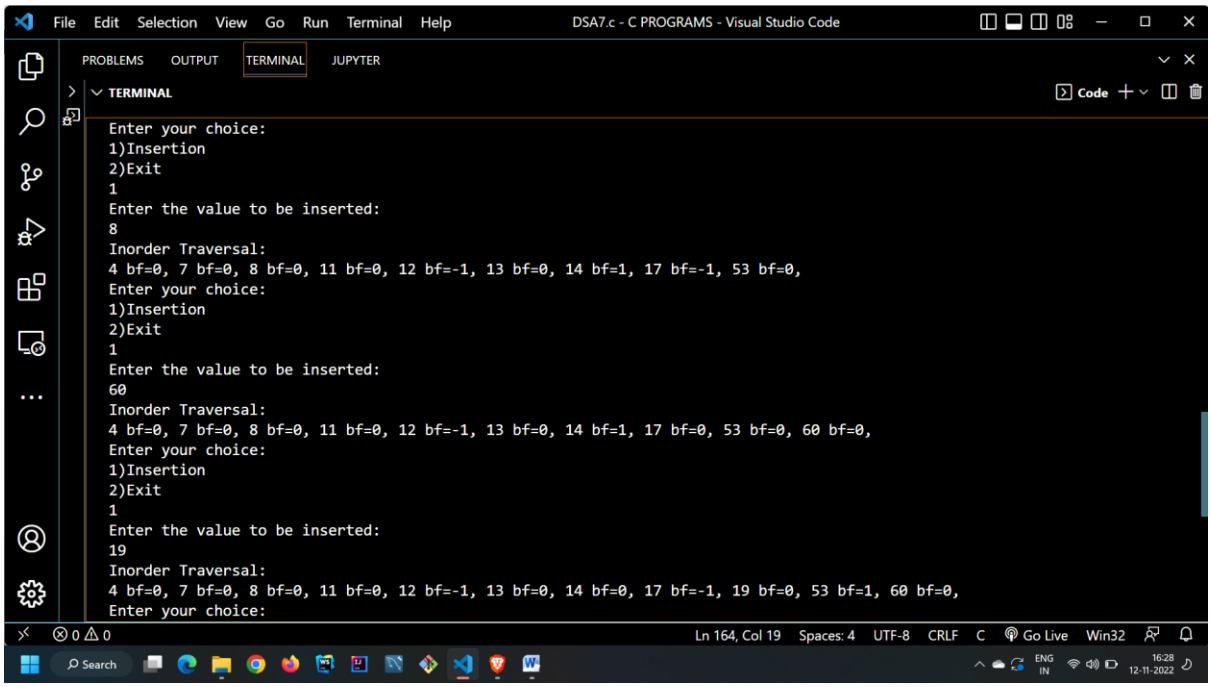
```
        scanf("%d",&ch);

        switch (ch)
        {
        case 1:
        {
            printf("Enter the value to be inserted:\n");
            scanf("%d",&val);
            root = Insertion(root, val);
            printf("Inorder Traversal:\n");
            Inorder(root);
            printf("\n");
            break;
        }
        case 2:
        {
            flag = 1;
            printf("Program finished!\n");
            break;
        }
        default:
        {
            printf("Invalid choice!\n");
            break;
        }

        }

    } while (flag != 1);

    return 0;
}
```

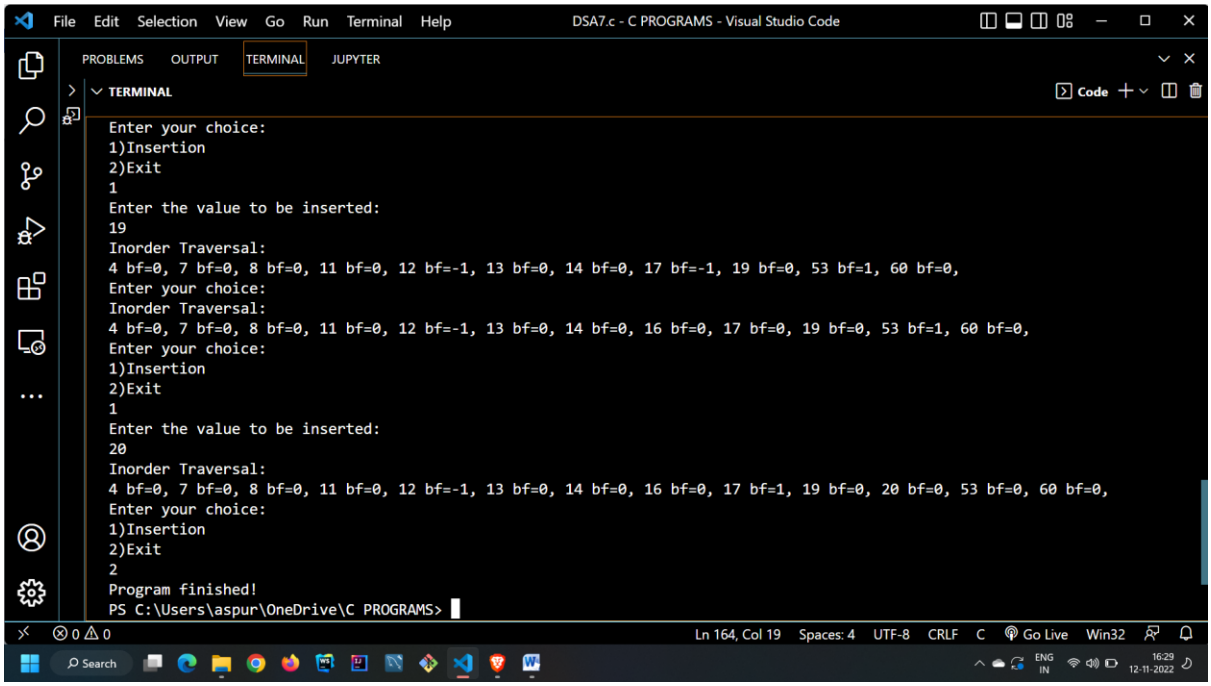
OUTPUT SCREENSHOT:


File Edit Selection View Go Run Terminal Help
DSA7.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\aspur\OneDrive\C PROGRAMS> cd "c:\Users\aspur\OneDrive\C PROGRAMS\" ; if ($?) { gcc DSA7.c -o DSA7 } ; if ($?)
{ .\DSA7 }
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
14
Inorder Traversal:
14 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
17
Inorder Traversal:
14 bf=-1, 17 bf=0,
Enter your choice:
1)Insertion
Ln 164, Col 19 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
16:27
12-11-2022


File Edit Selection View Go Run Terminal Help
DSA7.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
11
Inorder Traversal:
11 bf=0, 14 bf=0, 17 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
7
Inorder Traversal:
7 bf=0, 11 bf=1, 14 bf=1, 17 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
53
Inorder Traversal:
7 bf=0, 11 bf=1, 14 bf=0, 17 bf=-1, 53 bf=0,
Enter your choice:
Ln 164, Col 19 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search
16:28
12-11-2022


File Edit Selection View Go Run Terminal Help
DSA7.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
4
Inorder Traversal:
4 bf=0, 7 bf=0, 11 bf=0, 14 bf=0, 17 bf=-1, 53 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
13
Inorder Traversal:
4 bf=0, 7 bf=-1, 11 bf=-1, 13 bf=0, 14 bf=1, 17 bf=-1, 53 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
12
Inorder Traversal:
4 bf=0, 7 bf=-1, 11 bf=0, 12 bf=0, 13 bf=0, 14 bf=1, 17 bf=-1, 53 bf=0,
Enter your choice:
Ln 164, Col 19 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search


File Edit Selection View Go Run Terminal Help
DSA7.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
8
Inorder Traversal:
4 bf=0, 7 bf=0, 8 bf=0, 11 bf=0, 12 bf=-1, 13 bf=0, 14 bf=1, 17 bf=-1, 53 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
60
Inorder Traversal:
4 bf=0, 7 bf=0, 8 bf=0, 11 bf=0, 12 bf=-1, 13 bf=0, 14 bf=1, 17 bf=0, 53 bf=0, 60 bf=0,
Enter your choice:
1)Insertion
2)Exit
1
Enter the value to be inserted:
19
Inorder Traversal:
4 bf=0, 7 bf=0, 8 bf=0, 11 bf=0, 12 bf=-1, 13 bf=0, 14 bf=0, 17 bf=-1, 19 bf=0, 53 bf=1, 60 bf=0,
Enter your choice:
Ln 164, Col 19 Spaces: 4 UTF-8 CRLF C Go Live Win32
Search



CONCLUSION:

In this experiment we learnt about AVL Trees and how they overcome the disadvantages over a skewed binary search trees. We learnt about the various terminologies in AVL Trees and the concept of balance factor. We learnt about the internal structure of the node of an AVL Tree. In the end we implemented the concept of AVL Trees using C Programming language.