DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

February - May 2023

Report on

"AVL Tree Implementation/Visualisation"

Master of Technology
in
Computer Science & Engineering

UE22CS642A – Experiential Learning (ISA Component)

Submitted by:

SHALINI T (PES1PG22CS045)

1st Semester M.tech

We are using the following operations for the implementation of AVL tree

Balance and Update Height

```
//Balance function computes the balance factor of the node
int Balance(Node key)
{
    if (key == null)
        return 0;

    else
        return ( Height(key.right) - Height(key.left) );
}

//updateHeight function updates the height of the node
void updateHeight(Node key)
{
    int l = Height(key.left);
    int r = Height(key.right);

    key.height = Math.max(l , r) + 1;
}
```

. The balance factor is the difference between the height of the right subtree and the height of the left subtree. The method first checks I the input node is null, indicates an empty subtree, then return o. otherwise return the balance of the tree using height method defined

Update Height: this method takes input and updates its height

Balance Tree

```
// balanceTree function balances the tree using rotations after an insertion or deletion
Node balanceTree(Node root)
{
    updateHeight(root);

    int balance = Balance(root);

    if (balance > 1) //R
    {
        if (Balance(root.right) < 0)//RL
        {
            root.right = rotateRight(root.right);
            return rotateLeft(root);
        }
        else //RR
            return rotateLeft(root);
    }

    if (balance < -1)//L
    {
        if (Balance(root.left) > 0)//LR
        {
            root.left = rotateLeft(root.left);
            return rotateRight(root);
        }
        else//LL
            return rotateRight(root);
    }

    return root;
}

Node Root;
```

In this function, It first updates the height of the current node and then calculates the balance factor.
It checks If the balance is greater, lesser than 1.
If the balance greater than one the node is considered right heavy balance using right rotation. And the code checks if BF of right child less than 0 – RL rotate, right rotate on right child and left rotate on root to balance
And balance factor of right child is greater than 0, indicates right rotation (RR) the code performs left rotate on root node

If balance less than -1 considered as left heavy. Needs to balance using left rotate. And if less than -1 check the BF of left child of root greater than 0 – LR Rotation. Left rotate on left child of root and right rotate on root. If BF less than 0 – LL rotate, right rotate on the root node
If the balance factor are in range 1, -1, 0 the tree is said to be balanced.

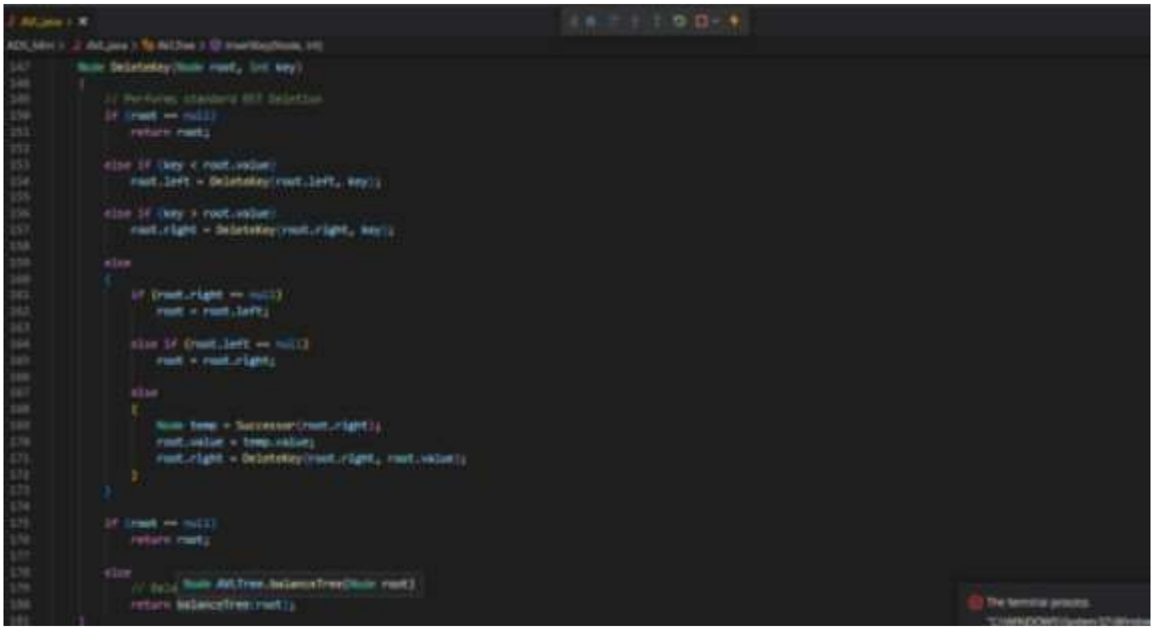
Insert Key

```
AVLjava
ADS_Mini > AVLjava > AVLTree > InsertKey(Node, int)
            }

            return root;
        }

    Node Root;

    Node InsertKey(Node root, int key)
        {
            // Performs normal BST insertion
            if (root == null)
                return new Node(key);

            else if (key < root.value)
                root.left = InsertKey(root.left, key);

            else
                root.right = InsertKey(root.right, key);

            // Balances the tree after BST Insertion
            return balanceTree(root);
        }

        // Successor returns the next largest node
        Node Successor(Node root)
        {
            if (root.left != null)
                return Successor(root.left);

            else
                return root;
        }
```

This is implementation of insert key takes 2 arguments root and key. It compares the value of key to the node. If key less than root.value, code recurses on the left subtree. If key greater than root.value, code recurses on right subtree.
Th successor function returns the next largest node. The code first check if the left subtree of root is non-empty if it is the code recuses on left subtree, if its empty returns the current node which is the successor

The time complexity of insert operation is O(log n)

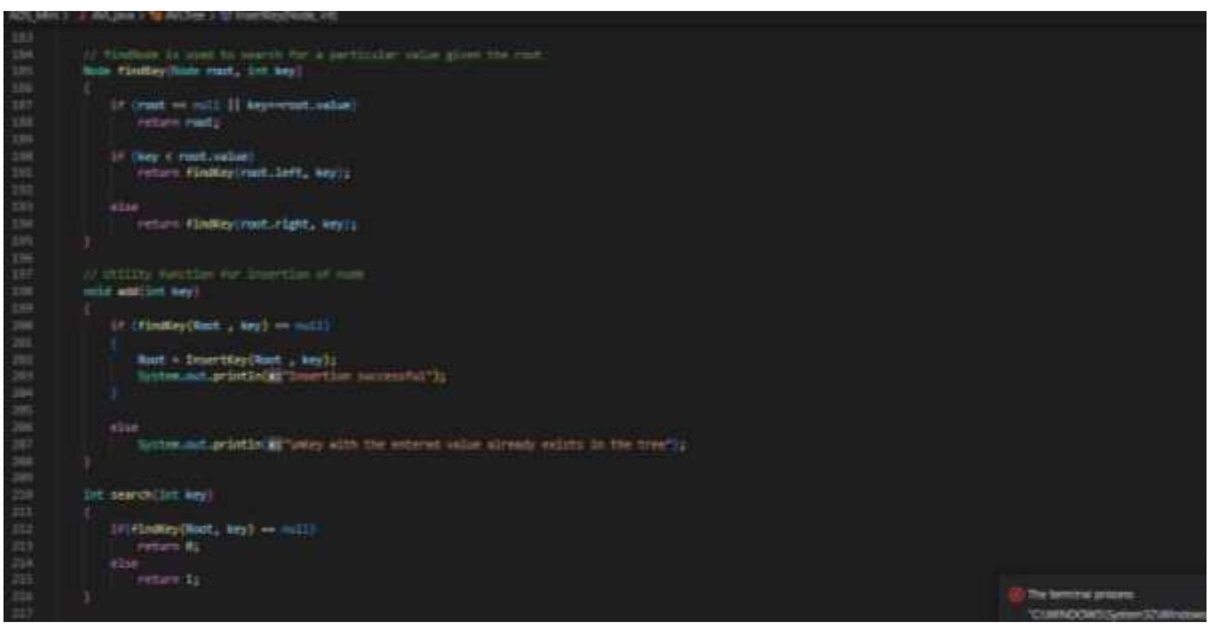
Delete Key



The delete key function compares the key values, if key less than root the key recurses on let subtree by calling delete function and If key greater than root.value code recurses on right subtree.I f key equal to 0 it checks if either left or rightchild is null. If both are not null it finds the successor and sets value of root to successor and recursively calls delete function.

The time complexity of delete key operation is O(log n)

Find Key



The Findkey function first checks if root is null or key equal to value of root. If either is true it returns root node. If value of key less than value of root code recursively searches in left subtree and if key greater than the root the function recursively searches the right subtree. The function continues to recursively search until desired key is found

The time complexity of Findkey operation is O(log n)

Destroy Tree

```
Node destroytree(Node root)
{

    if (root != null)
    {
        destroytree(root.left);
        destroytree(root.right);

        root.left = null;
        root.right = null;
        root = null;

    }

   return root;


}
```

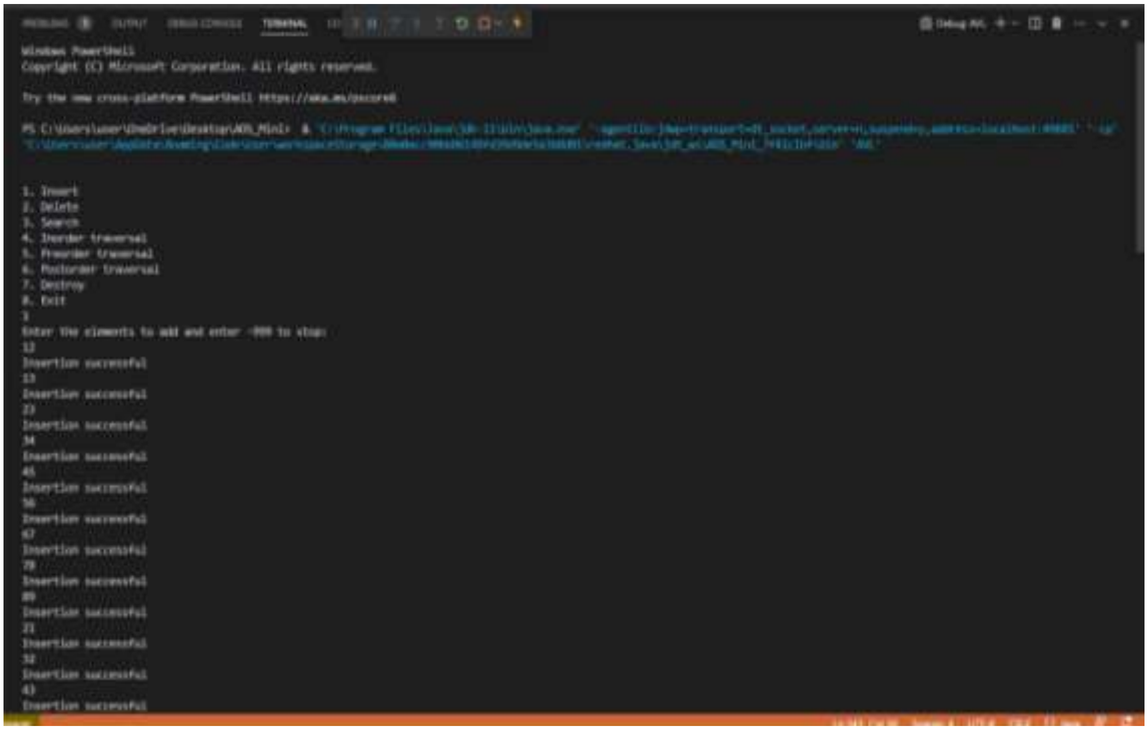
The function starts by checking if root is not null. If root is null it simply returns null. If root is not null function recursively calls destroy tree o left and right subtree. After deleting both subtree the function sets the both left and right pointers of the root node to null and then sets root itself to null. This ensures the entire tree rooted at root has been deleted.

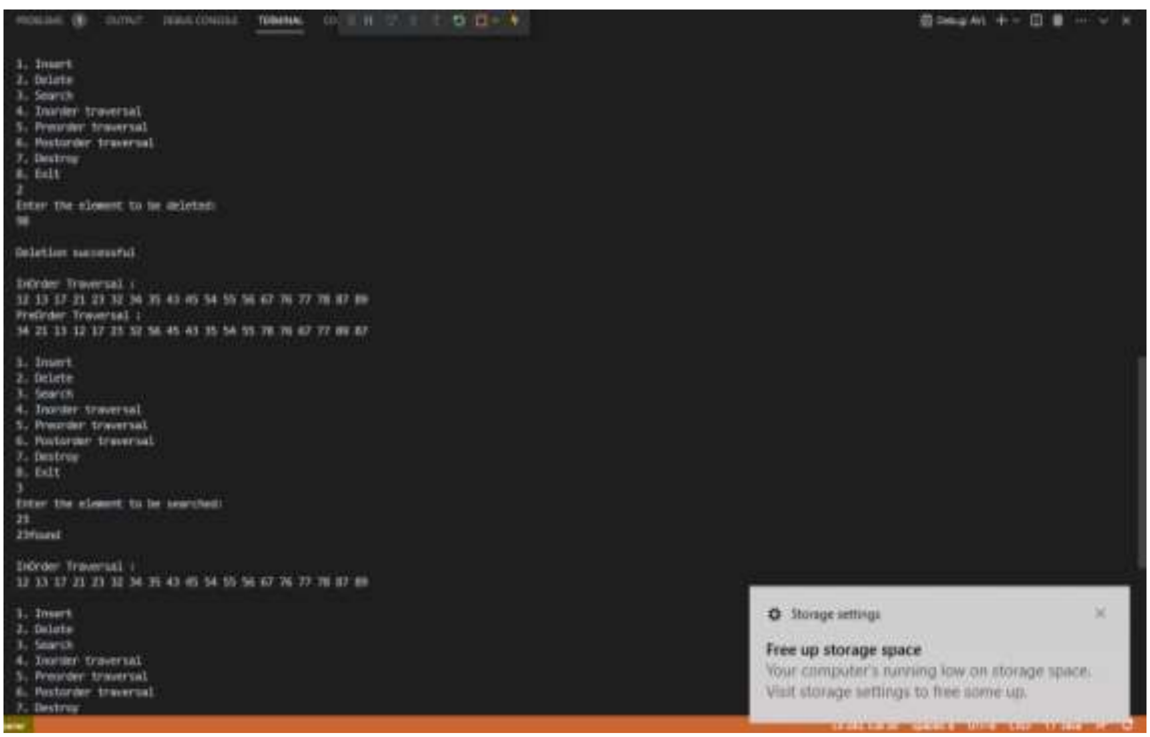
The time complexity of Destroy tree operation is O(n)

Learning Outcomes of Project

Learned abut the concept of balance tree and rotations used to balance the tree.
Concepts of AVL tree properties

OUTPUT


Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
1. Insert
2. Delete
3. Search
4. Inorder traversal
5. Preorder traversal
6. Postorder traversal
7. Destroy
8. Exit
Insertion successful


1. Insert
2. Delete
3. Search
4. Inorder traversal
5. Preorder traversal
6. Postorder traversal
7. Destroy
8. Exit
Deletion successful
Storage settings
Free up storage space
Your computer's running low on storage space.
Visit storage settings to free some up.