PROGRAM NO: 10 DATE: 26-10-2023

**AIM**

To implement AVL Tree with Insertion (Balancing) and Display Operations

**ALGORITHM**

1. START
2. FUNC newNode(key):
3. DECLARE node = new Node()
4. SET node.value = key
5. SET node.left = NULL
6. SET node.right = NULL
7. SET node.height = 1
8. RETURN node.
9. END FUNC
10. FUNC rightRotate(y):
11. DECLARE x = y.left, T2 = x.right
12. SET x.right = y, y.left = T2
13. SET y.height = max(height(y.left), height(y.right)) + 1
14. SET x.height = max(height(x.left), height(x.right)) + 1
15. RETURN x. END FUNC
16. FUNC leftRotate(x):
17. DECLARE y = x.left, T2 = y.left
18. SET y.left = x, x.right= T2
19. SET x.height = max(height(x.left), height(x.right)) + 1
20. SET y.height = max(height(y.left), height(y.right)) + 1
21. RETURN x. END FUNC
22. FUNC getBalance(N):
23. IF N == NULL: RETURN 0
24. RETURN height(N.left) - height(N.right)
25. END FUNC
26. FUNC Insert(node, value):
27. IF node == NULL: RETURN newNode(value). END IF
28. IF value <= node.value: node.left = Insert(node.left, value)
29. ELSE IF value > node.value: node.right = Insert(node.right, value)
30. END IF
31. node.height = 1 + max(height(node.left), height(node.right))
32. DELCARE balance = getBalance(node)
33. IF balance > 1 AND value < node.left.value: RETURN rightRotate(node). END IF
34. IF balance < -1 AND value > node.right.value: RETURN leftRotate(node). END IF
35. IF balance > 1 AND value > node.left.value:
36. node.left = leftRotate(node.left)
37. RETURN rightRotate(node). END IF
38. IF balance < -1 AND value < node.right.value:
39. node.right = rightRotate(node.left)
40. RETURN leftRotate(node). END IF
41. RETURN node
42. END FUNC
43. STOP

**CODE**

#include <bits/stdc++.h>

using namespace std;

typedef struct Node {

    int value;

    Node \*left;

    Node \*right;

    int height;

} Node;

class AVLTree {

    public:

    Node\* rootNode;

    AVLTree() {

        this->rootNode = NULL;

    }

    int height(Node \*N) {

        if (N == NULL) return 0;

        return N->height;

    }

    int max(int a, int b) {

        return (a > b)? a : b;

    }

    Node\* newNode(int key) {

        Node\* node = new Node();

        node->value = key;

        node->left = NULL;

        node->right = NULL;

        node->height = 1;

        return node;

    }

    Node \*rightRotate(Node \*y) {

        Node \*x = y->left;

        Node \*T2 = x->right;

        // Perform rotation

        x->right = y;

        y->left = T2;

        // Update heights

        y->height = this->max(this->height(y->left), this->height(y->right)) + 1;

        x->height = this->max(this->height(x->left), this->height(x->right)) + 1;

        // Return new root

        return x;

    }

    Node \*leftRotate(Node \*x) {

        Node \*y = x->right;

        Node \*T2 = y->left;

        // Perform rotation

        y->left = x;

        x->right = T2;

        // Update heights

        x->height = this->max(this->height(x->left), this->height(x->right)) + 1;

        y->height = this->max(this->height(y->left), this->height(y->right)) + 1;

        // Return new root

        return y;

    }

    // Get Balance factor of node N

    int getBalance(Node \*N) {

        if (N == NULL) return 0;

        return this->height(N->left) - this->height(N->right);

    }

    Node\* insert(Node\* node, int value) {

        if (node == NULL) {

            return this->newNode(value);

        }

        if (value <= node->value)

            node->left = this->insert(node->left, value);

        else if (value > node->value)

            node->right = this->insert(node->right, value);

        node->height = 1 + this->max(this->height(node->left), this->height(node->right));

        int balance = this->getBalance(node);

        // Left Left Case

        if (balance > 1 && value < node->left->value)

            return this->rightRotate(node);

        // Right Right Case

        if (balance < -1 && value > node->right->value) {

            return this->leftRotate(node);

        }

        // Left Right Case

        if (balance > 1 && value > node->left->value) {

            node->left = this->leftRotate(node->left);

            return this->rightRotate(node);

        }

        // Right Left Case

        if (balance < -1 && value < node->right->value) {

            node->right = this->rightRotate(node->right);

            return this->leftRotate(node);

        }

        return node;

    }

    void Insert(int value) {

        this->rootNode = this->insert(this->rootNode, value);

    }

    void DisplaySubTree(Node\* node, int indent, int side) {

        if (this->rootNode == NULL) return;

        for (int i=0; i<indent; i++) {

            cout << "    ";

        }

        if (side == 1) {

            cout << "L";

        } else if (side == 2) {

            cout << "R";

        }

        if (node->left == NULL) {

            cout << "|" << node->value << endl;

        } else {

            cout << "\\" << node->value << endl;

        }

        if (node->left != NULL) {

            this->DisplaySubTree(node->left, indent + 1, 1);

        }

        if (node->right != NULL) {

            this->DisplaySubTree(node->right, indent + 1, 2);

        }

    }

};

int main() {

    int choice, temp;

    AVLTree avl;

    while (true) {

        cout << "\n---- Binary Search Tree (BST) ----" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Display" << endl;

        cout << "3. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        if (choice == 1) {

            cout << "Enter value to insert: ";

            cin >> temp;

            avl.Insert(temp);

        } else if (choice == 2) {

            avl.DisplaySubTree(avl.rootNode, 0, 0);

        } else if (choice == 3) {

            cout << "\n--------- Author ----------------" << endl;

            cout << "Ali Izzath Shazin" << endl;

            cout << "220071601028" << endl;

            cout << "B. Tech CSE A" << endl;

            break;

        } else {

            cout << "Invalid Option" << endl;

        }

    }

    return 0;

}

**OUTPUT**

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 50

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 60

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 70

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 80

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 90

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 40

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 1

Enter value to insert: 30

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 2

\60

L\40

L|30

R|50

R\80

L|70

R|90

---- AVL Tree ----

1. Insert

2. Display

3. Exit

Enter Choice: 3

--------- Author ----------------

Ali Izzath Shazin

220071601028

B. Tech CSE A