**Assignment-6**

**Name of Student: Ayush Sanjay Dhangar**

**Batch: 02 Class: SY\_IT-A Roll No: 42**

**PRN: 12210406**

**Subject: IT2265 Advanced Data Structures**

**Problem Statement**: Write C/C++ program to check whether the tree is balanced or not and tree is AVL or not. Also, your code should tell which rotation case is required if tree is imbalanced. Show all rotations cases. Input should be user choice**.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

    int data;

    struct TreeNode\* left;

    struct TreeNode\* right;

    int height;

};

struct TreeNode\* createNode(int data) {

    struct TreeNode\* newNode = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

    if (!newNode) {

        printf("Memory allocation failed!\n");

        exit(1);

    }

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    newNode->height = 1;

    return newNode;

}

int getHeight(struct TreeNode\* node) {

    if (node == NULL)

        return 0;

    return node->height;

}

struct TreeNode\* rightRotate(struct TreeNode\* y) {

    struct TreeNode\* x = y->left;

    struct TreeNode\* T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = 1 + (getHeight(y->left) > getHeight(y->right) ? getHeight(y->left) : getHeight(y->right));

    x->height = 1 + (getHeight(x->left) > getHeight(x->right) ? getHeight(x->left) : getHeight(x->right));

    return x;

}

struct TreeNode\* leftRotate(struct TreeNode\* x) {

    struct TreeNode\* y = x->right;

    struct TreeNode\* T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = 1 + (getHeight(x->left) > getHeight(x->right) ? getHeight(x->left) : getHeight(x->right));

    y->height = 1 + (getHeight(y->left) > getHeight(y->right) ? getHeight(y->left) : getHeight(y->right));

    return y;

}

int getBalanceFactor(struct TreeNode\* node) {

    if (node == NULL)

        return 0;

    return getHeight(node->left) - getHeight(node->right);

}

struct TreeNode\* insertNode(struct TreeNode\* root, int data) {

    if (root == NULL)

        return createNode(data);

    if (data < root->data)

        root->left = insertNode(root->left, data);

    else if (data > root->data)

        root->right = insertNode(root->right, data);

    else

        return root;

    root->height = 1 + (getHeight(root->left) > getHeight(root->right) ? getHeight(root->left) : getHeight(root->right));

    int balance = getBalanceFactor(root);

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

        return rightRotate(root);

    if (balance < -1 && data > root->right->data)

        return leftRotate(root);

    if (balance > 1 && data > root->left->data) {

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

        return rightRotate(root);

    }

    if (balance < -1 && data < root->right->data) {

        root->right = rightRotate(root->right);

        return leftRotate(root);

    }

    return root;

}

int isBalanced(struct TreeNode\* root) {

    if (root == NULL)

        return 1;

    int leftHeight = getHeight(root->left);

    int rightHeight = getHeight(root->right);

    if (abs(leftHeight - rightHeight) <= 1 && isBalanced(root->left) && isBalanced(root->right))

        return 1;

    return 0;

}

int isAVL(struct TreeNode\* root) {

    if (root == NULL)

        return 1;

    int balance = getBalanceFactor(root);

    if (abs(balance) <= 1 && isAVL(root->left) && isAVL(root->right))

        return 1;

    return 0;

}

void printRotationCases(struct TreeNode\* root, int data) {

    struct TreeNode\* newRoot = insertNode(root, data);

    if (isBalanced(newRoot))

        printf("The tree is balanced.\n");

    else {

        printf("The tree is imbalanced.\n");

        int balance = getBalanceFactor(newRoot);

        printf("Balance factor: %d\n", balance);

        if (balance > 1 && data < newRoot->left->data)

            printf("Left-Left rotation needed.\n");

        else if (balance < -1 && data > newRoot->right->data)

            printf("Right-Right rotation needed.\n");

        else if (balance > 1 && data > newRoot->left->data) {

            printf("Left-Right rotation needed.\n");

            printf("Performing Left-Right rotation...\n");

            newRoot->left = leftRotate(newRoot->left);

            newRoot = rightRotate(newRoot);

            printf("Rotation performed.\n");

        } else if (balance < -1 && data < newRoot->right->data) {

            printf("Right-Left rotation needed.\n");

            printf("Performing Right-Left rotation...\n");

            newRoot->right = rightRotate(newRoot->right);

            newRoot = leftRotate(newRoot);

            printf("Rotation performed.\n");

        }

    }

}

void freeTree(struct TreeNode\* root) {

    if (root == NULL)

        return;

    freeTree(root->left);

    freeTree(root->right);

    free(root);

}

int main() {

    struct TreeNode\* root = NULL;

    int choice, data;

    do {

        printf("\n----- AVL Tree Operations -----\n");

        printf("1. Insert Node in AVL tree\n");

        printf("2. Check if AVL tree is Balanced\n");

        printf("3. Check if it is AVL tree\n");

        printf("4. Check Rotation Cases\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter data to insert: ");

                scanf("%d", &data);

                root = insertNode(root, data);

                break;

            case 2:

                if (isBalanced(root))

                    printf("The tree is balanced.\n");

                else

                    printf("The tree is not balanced.\n");

                break;

            case 3:

                if (isAVL(root))

                    printf("The tree is an AVL tree.\n");

                else

                    printf("The tree is not an AVL tree.\n");

                break;

            case 4:

                printf("Enter data to check rotation cases: ");

                scanf("%d", &data);

                printRotationCases(root, data);

                break;

            case 5:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice! Please enter a valid option.\n");

        }

    } while (choice != 5);

    freeTree(root);

    return 0;

}

**Result:**

**Actual Output:**

   