ASSINGMENT-4

AVL TREE IMPLEMENTATION

#include <stdio.h>

#include <stdlib.h>

*struct* TreeNode {

*int* key;

*struct* TreeNode\* left;

*struct* TreeNode\* right;

*int* height;

};

*int* max(*int* *a*, *int* *b*) {

    return (*a* > *b*) ? *a* : *b*;

}

*int* getHeight(*struct* TreeNode\* *node*) {

    if (*node* == NULL)

        return 0;

    return *node*->height;

}

*int* getBalanceFactor(*struct* TreeNode\* *node*) {

    if (*node* == NULL)

        return 0;

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

}

*struct* TreeNode\* newNode(*int* *key*) {

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

    node->key = *key*;

    node->left = NULL;

    node->right = NULL;

    node->height = 1;

    return node;

}

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

*struct* TreeNode\* x = *y*->left;

*struct* TreeNode\* T2 = x->right;

    x->right = *y*;

*y*->left = T2;

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

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

    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 = max(getHeight(*x*->left), getHeight(*x*->right)) + 1;

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

    return y;

}

*struct* TreeNode\* insert(*struct* TreeNode\* *node*, *int* *key*) {

    if (*node* == NULL)

        return newNode(*key*);

    if (*key* < *node*->key)

*node*->left = insert(*node*->left, *key*);

    else if (*key* > *node*->key)

*node*->right = insert(*node*->right, *key*);

    else

        return *node*;  // Duplicate keys are not allowed

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

*int* balance = getBalanceFactor(*node*);

    // Left Heavy

    if (balance > 1) {

        if (*key* < *node*->left->key) {

            printf("Right Rotation Applied\n");

            return rightRotate(*node*);

        } else {

            printf("Left-Right Rotation Applied\n");

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

            return rightRotate(*node*);

        }

    }

    // Right Heavy

    if (balance < -1) {

        if (*key* > *node*->right->key) {

            printf("Left Rotation Applied\n");

            return leftRotate(*node*);

        } else {

            printf("Right-Left Rotation Applied\n");

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

            return leftRotate(*node*);

        }

    }

    return *node*;

}

*void* preOrderTraversal(*struct* TreeNode\* *root*) {

    if (*root* != NULL) {

        printf("Key: %d, Balance Factor: %d\n", *root*->key, getBalanceFactor(*root*));

        preOrderTraversal(*root*->left);

        preOrderTraversal(*root*->right);

    }

}

*int* main() {

*struct* TreeNode\* root = NULL;

*int* keys[] = {10, 20, 30, 40, 50, 25};

*int* numKeys = sizeof(keys) / sizeof(keys[0]);

    for (*int* i = 0; i < numKeys; i++) {

        root = insert(root, keys[i]);

        printf("Inserting %d into AVL tree:\n", keys[i]);

        preOrderTraversal(root);

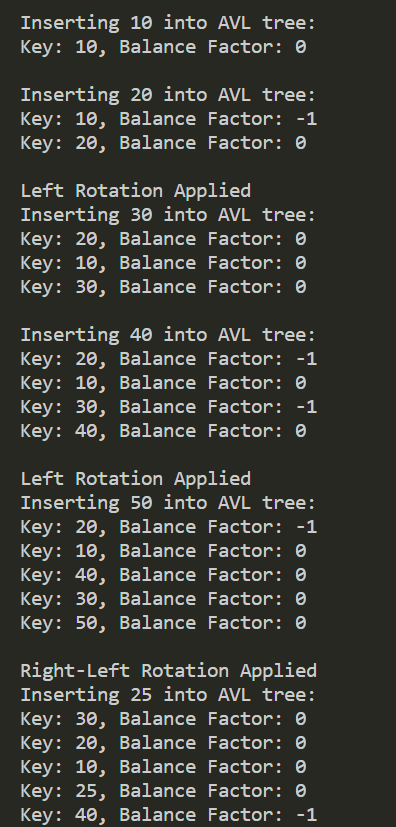
        printf("\n");

    }

    return 0;

}

OUTPUT:-



DELETION IN AVL

*struct* TreeNode \*deleteNode(*struct* TreeNode \**root*, *int* *key*)

{

    if (*root* == NULL)

        return *root*;

    if (*key* < *root*->key)

*root*->left = deleteNode(*root*->left, *key*);

    else if (*key* > *root*->key)

*root*->right = deleteNode(*root*->right, *key*);

    else

    {

        // node with only one child or no child

        if ((*root*->left == NULL) || (*root*->right == NULL))

        {

*struct* TreeNode \*temp = *root*->left ? *root*->left : *root*->right;

            // No child case

            if (temp == NULL)

            {

                temp = *root*;

*root* = NULL;

            }

            else

                \**root* = \*temp;

            free(temp);

        }

        else

        {

*struct* TreeNode \*temp = minValueNode(*root*->right);

*root*->key = temp->key;

*root*->right = deleteNode(*root*->right, temp->key);

        }

    }

    // If the tree had only one node then return

    if (*root* == NULL)

        return *root*;

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

*int* balance = getBalance(*root*);

    // If this node becomes unbalanced, then there are 4 cases

    if (balance > 1 && getBalance(*root*->left) >= 0)

        return rightRotate(*root*);

    if (balance > 1 && getBalance(*root*->left) < 0)

    {

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

        return rightRotate(*root*);

    }

    if (balance < -1 && getBalance(*root*->right) <= 0)

        return leftRotate(*root*);

    if (balance < -1 && getBalance(*root*->right) > 0)

    {

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

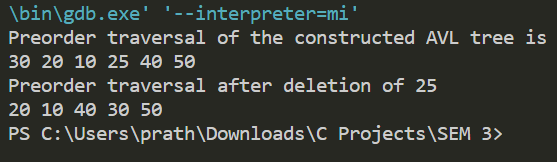
        return leftRotate(*root*);

    }

    return *root*;

}

OUTPUT:-



TO CHECK IF BST TREE IS AVL

*int* isBalanced(

*struct* node \**root*) {

*int* lh;

*int* rh;

    if (*root* == NULL)

        return 1;

    lh = height(*root*->left);

    rh = height(*root*->right);

    if (abs(lh - rh) <= 1 && isBalanced(*root*->left) && isBalanced(*root*->right)){

        return 1;

    }

    return 0;

}

OUTPUTS :-

