# **Bansilal Ramnath Agarwal Charitable Trust’s**

**Vishwakarma Institute of Technology, Pune-37**

*(An Autonomous Institute of Savitribai Phule Pune University)*

|  |  |
| --- | --- |
|  |  |
|  |  |

**Department of Artificial Intelligence and Data Science**

|  |  |
| --- | --- |
| **Division** | AI-A |
| **Batch** | 1 |
| **GR-no** | 12211542 |
| **Roll No.** | 07 |
| **Name** | Anish Naphade |
| **Subject** | ADS |

**ASSIGNMENT 7 –**

**AVL Tree**

**Insert, Delete**

**CODE**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int key;**

**struct Node \*left;**

**struct Node \*right;**

**int height;**

**};**

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

**return (a > b) ? a : b;**

**}**

**int height(struct Node \*node) {**

**if (node == NULL)**

**return 0;**

**return node->height;**

**}**

**int getBalance(struct Node \*node) {**

**if (node == NULL)**

**return 0;**

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

**}**

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

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

**node->key = key;**

**node->left = NULL;**

**node->right = NULL;**

**node->height = 1;**

**return node;**

**}**

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

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

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

**x->right = y;**

**y->left = T2;**

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

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

**return x;**

**}**

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

**struct Node \*y = x->right;**

**struct Node \*T2 = y->left;**

**y->left = x;**

**x->right = T2;**

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

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

**return y;**

**}**

**struct Node \*insert(struct Node \*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;**

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

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

**if (balance > 1 && key < node->left->key)**

**return rightRotate(node);**

**if (balance < -1 && key > node->right->key)**

**return leftRotate(node);**

**if (balance > 1 && key > node->left->key) {**

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

**return rightRotate(node);**

**}**

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

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

**return leftRotate(node);**

**}**

**return node;**

**}**

**struct Node \*minValueNode(struct Node \*node) {**

**struct Node \*current = node;**

**while (current->left != NULL)**

**current = current->left;**

**return current;**

**}**

**struct Node \*deleteNode(struct Node \*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 {**

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

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

**if (temp == NULL) {**

**temp = root;**

**root = NULL;**

**} else**

**\*root = \*temp;**

**free(temp);**

**} else {**

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

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

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

**}**

**}**

**if (root == NULL)**

**return root;**

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

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

**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;**

**}**

**void inOrder(struct Node \*root) {**

**if (root != NULL) {**

**inOrder(root->left);**

**printf("%d ", root->key);**

**inOrder(root->right);**

**}**

**}**

**int main() {**

**struct Node \*root = NULL;**

**int choice, key;**

**do {**

**printf("\nAVL Tree Operations:\n");**

**printf("1. Insert Node\n");**

**printf("2. Delete Node\n");**

**printf("3. Display In-order Traversal\n");**

**printf("4. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter the key to insert: ");**

**scanf("%d", &key);**

**root = insert(root, key);**

**break;**

**case 2:**

**printf("Enter the key to delete: ");**

**scanf("%d", &key);**

**root = deleteNode(root, key);**

**break;**

**case 3:**

**printf("In-order Traversal: ");**

**inOrder(root);**

**printf("\n");**

**break;**

**case 4:**

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

**break;**

**default:**

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

**}**

**} while (choice != 4);**

**return 0;**

**}**