DSP LAB

03-11-2021

Implement Height balanced binary tree(AVL tree) in the following way(all nodes in the tree should be unique)

provide an insert function to insert the node in the tree

Provide a delete function to delete the node from the tree(by value) provide a function to display pre-order,in-order,post-order Level,level-order tree traversal

Provide a function to calculate the sum of all nodes in the left subtree of the root node.

Source Code:

#include <stdio.h>

#include <stdlib.h>

// Create Node

struct Node {

int data;

struct Node \*left;

struct Node \*right;

int height;

};

int max(int a, int b);

// Calculate height

int height(struct Node \*root) {

if (root == NULL)

return 0;

return root->height;

}

int max(int a, int b) {

return (a > b) ? a : b;

}

// Create a node

struct Node \*newNode(int data) {

struct Node \*node = (struct Node \*)

malloc(sizeof(struct Node));

node->data = data;

node->left = NULL;

node->right = NULL;

node->height = 1;

return (node);

}

// Left rotate

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;

}

// Right rotate

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 \*minValueNode(struct Node \*node) {

struct Node \*curr = node;

while (curr->left != NULL)

curr = curr->left;

return curr;

}

// Get the balance factor

int Balance(struct Node \*N) {

if (N == NULL)

return 0;

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

}

// Insert node

struct Node \*insertNode(struct Node \*node, int data) {

if (node == NULL)

return (newNode(data));

if (data < node->data)

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

else if (data > node->data)

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

else

return node;

// Update the balance factor of each node and

// Balance the tree

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

height(node->right));

int balance = Balance(node);

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

return rightRotate(node);

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

return leftRotate(node);

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

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

return rightRotate(node);

}

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

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

return leftRotate(node);

}

return node;

}

// Delete a nodes

struct Node \*deleteNode(struct Node \*root, int data) {

// Find the node and delete it

if (root == NULL)

return root;

if (data < root->data)

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

else if (data > root->data)

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

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->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

}

if (root == NULL)

return root;

// Update the balance factor of each node and

// balance the tree

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

height(root->right));

int balance = Balance(root);

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

return rightRotate(root);

if (balance > 1 && Balance(root->left) < 0) {

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

return rightRotate(root);

}

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

return leftRotate(root);

if (balance < -1 && Balance(root->right) > 0) {

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

return leftRotate(root);

}

return root;

}

// Print the tree

void inorderTraversal(struct Node\* root)

{

if (root == NULL) {

return;

}

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

void preorderTraversal(struct Node\* root)

{

if (root == NULL) {

return;

}

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct Node\* root)

{

if (root == NULL)

return;

postorderTraversal(root->left);

postorderTraversal

(root->right);

printf("%d ", root->data);

}

int tree\_height(struct Node\* root) {

if (!root)

return 0;

else {

int left\_height = tree\_height(root->left);

int right\_height = tree\_height(root->right);

if (left\_height >= right\_height)

return left\_height + 1;

else

return right\_height + 1;

}

}

void print\_level(struct Node\* root, int level\_no) {

if (!root)

return;

if (level\_no == 0) {

printf("%d -> ", root->data);

}

else {

print\_level(root->left, level\_no - 1);

print\_level(root->right, level\_no - 1);

}

}

void print\_tree\_level\_order(struct Node\* root) {

if (!root)

return;

int height = tree\_height(root);

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

printf("Level %d: ", i);

print\_level(root, i);

printf("\n");

}

printf("\nThe Level-order traversal is ");

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

print\_level(root, i);

}

printf("\n");

}

int sumvalue=0;

void Sumofleftnodes(struct Node\* root){

if(root==NULL)

return;

else{

sumvalue+=root->data;

Sumofleftnodes(root->left);

Sumofleftnodes(root->right);

}

}

int main()

{

int n,value;

int choice=0;

struct Node \*root = NULL;

while (choice!=8)

{

printf("\n 1.Insert a value to the AVL tree\n 2.Delete the value from AVL tree\n 3.Display Preorder traversal\n 4.Display Inorder traversal\n 5.Display Postorder traversal\n 6.Display Level-order traversal\n 7.Sum of all nodes in the left subtree of the root node\n 8.Exit\n Enter your choice : ");

if(scanf("%d", &choice)==1){

if(choice==1){

printf("\nEnter Node to AVL Tree:");

scanf("%d",&value);

root = insertNode(root, value);

}

else if(choice== 2){

printf("\nEnter an element to delete from the AVL Tree:");

scanf("%d",&value);

root = deleteNode(root, value);

}

else if(choice== 3)

{

printf("\nThe preorder traversal is : ");

preorderTraversal(root);

}

else if(choice==4) {

printf("\nThe Inorder traversal is : ");

inorderTraversal(root);

}

else if(choice==5) {

printf("\nThe postorder traversal is : ");

postorderTraversal(root);

}

else if(choice==6) {

print\_tree\_level\_order(root);

}

else if(choice==7) {

Sumofleftnodes(root->left);

printf("\nSum of all nodes in the left subtree of the root node is : %d",sumvalue);

}

else if(choice==8)

break;

else

printf("Enter valid choice\n");

}

else{

printf("Invalid input. Only integers allowed\n");

exit(0);

}

}

return 0;

}

Output:







