1. Write a code to find total number of nodes exist in a BST .

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

using namespace std;

int n=1;

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

//printing the whole tree

void PrintTree(struct node \*root)

{

if (root == NULL)

return;

cout<<root->key<<" ";

PrintTree(root->left);

PrintTree(root->right);

}

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

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

return node;

}

struct node \* minValueNode(struct node\* node)

{

struct node\* current = node;

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

current = current->left;

return current;

}

int CountNode( node \*root)

{

if(root == NULL)

return 0;

if(root -> left != NULL)

{

n = n+1;

n = CountNode(root -> left);

}

if(root -> right != NULL)

{

n = n+1;

n = CountNode(root -> right);

}

return n;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

root = insert(root, 91);

cout<<"Printing Tree: ";

PrintTree(root);

cout<<"\nTotal Node: "<<CountNode(root)<<endl;

return 0;

}

# 2. Find the most minimum and maximum value from the BST

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

using namespace std;

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

//printing the whole tree

void PrintTree(struct node \*root)

{

if (root == NULL)

return;

cout<<root->key<<" ";

PrintTree(root->left);

PrintTree(root->right);

}

//printing the Minimum value of whole tree

void minValueNode(struct node\* node)

{

struct node\* current = node;

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

current = current->left;

cout<<current->key;

}

//printing the Maximum value of whole tree

void maxValueNode(struct node\* node)

{

struct node\* current = node;

while (current && current->right != NULL)

current = current->right;

cout<<current->key;

}

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

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

return node;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

root = insert(root, 91);

cout<<"Printing Tree: ";

PrintTree(root);

cout<<"\nMinimum value of whole node: ";

minValueNode(root);

cout<<"\nMaximum value of whole node: ";

maxValueNode(root);

cout<<endl;

return 0;

}

# 3. Find the distance of two node from BST (calculate the distance based on counting the number of edges in between)

// CPP program to find distance between two nodes in BST

#include <iostream>

using namespace std;

struct Node

{

struct Node\* left, \*right;

int key;

};

struct Node\* newNode(int key)

{

struct Node\* ptr = new Node;

ptr->key = key;

ptr->left = ptr->right = NULL;

return ptr;

}

// Standard BST insert function

struct Node\* insert(struct Node\* root, int key)

{

if (!root)

root = newNode(key);

else if (root->key > key)

root->left = insert(root->left, key);

else if (root->key < key)

root->right = insert(root->right, key);

return root;

}

// This function returns distance of x from

// root. This function assumes that x exists

// in BST and BST is not NULL.

int distanceFromRoot(struct Node\* root, int x)

{

if (root->key == x)

return 0;

else if (root->key > x)

return 1 + distanceFromRoot(root->left, x);

return 1 + distanceFromRoot(root->right, x);

}

// Returns minimum distance between a and b.

// This function assumes that a and b exist

// in BST.

int distanceBetween2(struct Node\* root, int a, int b)

{

if (root == NULL)

return 0;

// Both keys lie in left

if (root->key > a && root->key > b)

return distanceBetween2(root->left, a, b);

// Both keys lie in right

if (root->key < a && root->key < b) // same path

return distanceBetween2(root->right, a, b);

// Lie in opposite directions (Root is

// LCA of two nodes)

if (root->key >= a && root->key <= b)

return distanceFromRoot(root, a) + distanceFromRoot(root, b);

}

// This function make sure that a is smaller

// than b before making a call to findDistWrapper()

int findDistWrapper(Node \*root, int a, int b)

{

if (a > b)

{

swap(a, b);

}

return distanceBetween2(root, a, b);

}

// Driver code

int main()

{

struct Node\* root = NULL;

root = insert(root, 5);

insert(root, 54);

insert(root, 12);

insert(root, 45);

insert(root, 21);

insert(root, 19);

insert(root, 25);

insert(root, 31);

insert(root, 56);

int a = 12, b = 31;

int r = findDistWrapper(root, a, b);

cout <<r;

return 0;

}

# 4. Traverse the whole BST and only print the left and right nodes.

#include <iostream>

using namespace std;

void leftView(struct Node \*root);

void rightView(struct Node \*root);

struct Node

{

struct Node\* left, \*right;

int key;

};

struct Node\* newNode(int key)

{

struct Node\* ptr = new Node;

ptr->key = key;

ptr->left = ptr->right = NULL;

return ptr;

}

// Standard BST insert function

struct Node\* insert(struct Node\* root, int key)

{

if (!root)

root = newNode(key);

else if (root->key > key)

root->left = insert(root->left, key);

else if (root->key < key)

root->right = insert(root->right, key);

return root;

}

// Left View node in BST

void checkRight(struct Node \*node)

{

if(node==NULL)

return;

leftView(node->left);

checkRight(node->right);

}

void leftView(struct Node \*root)

{

if(root==NULL)

return;

cout<<root->key<<" ";

leftView(root->left);

checkRight(root->right);

}

// Right View node in BST

void checkLeft(struct Node \*node)

{

if(node==NULL)

return;

rightView(node->right);

checkLeft(node->left);

}

void rightView(struct Node \*root)

{

if(root==NULL)

return;

cout<<root->key<<" ";

rightView(root->right);

checkLeft(root->left);

}

// Driver code

int main()

{

struct Node\* root = NULL;

root = insert(root, 20);

insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 30);

insert(root, 25);

insert(root, 35);

insert(root, 2);

insert(root, 57);

cout<<"Left view node: ";

leftView(root);

cout<<"\nRight view node: ";

rightView(root);

return 0;

}

# 5. Sort the BST nodes and print all nodes.

#include<iostream>

using namespace std;

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

struct node \*temp = new node;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

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

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

return node;

}

//printing the whole tree

void PrintTree(struct node \*root)

{

if (root == NULL)

return;

cout<<root->key<<" ";

PrintTree(root->left);

PrintTree(root->right);

}

// Sorted Tree

void sortedTree(struct node \*root)

{

if (root == NULL)

return;

sortedTree(root->left);

cout<<root->key<<" ";

sortedTree(root->right);

}

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 69);

root = insert(root, 60);

root = insert(root, 97);

root = insert(root, 41);

root = insert(root, 100);

root = insert(root, 7);

root = insert(root, 2);

cout<<"Printing Tree: ";

PrintTree(root);

cout<<"\n\nSorted tree: ";

sortedTree(root);

return 0;

}

# 6. Insert all the values of BST to an array and print the sum of all values.

#include<iostream>

using namespace std;

int arr[100];

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

struct node \*temp = new node;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

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

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

return node;

}

//printing the whole tree

void PrintTree(struct node \*root)

{

if (root == NULL)

return;

cout<<root->key<<" ";

PrintTree(root->left);

PrintTree(root->right);

}

// Converting a BST into an Array

void BSTtoArray(node \*root, int A[])

{

static int pos = 0;

if(root == NULL) return;

BSTtoArray(root->left, A);

A[pos++] = root->key;

BSTtoArray(root->right, A);

}

int treeSize(node \*root) {

if(root == NULL) return 0;

else

return treeSize(root->left) + treeSize(root->right) + 1;

}

// Drive Code

int main()

{

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 13);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 3);

root = insert(root, 9);

root = insert(root, 80);

root = insert(root, 4);

root = insert(root, 60);

root = insert(root, 111);

cout<<"Printing Tree: ";

PrintTree(root);

int treeSZ = treeSize(root);

int arrySum = 0;

int \*A = new int[treeSZ];

BSTtoArray(root, A);

cout << "\nNew Array: ";

for(int i = 0; i < treeSZ; i++)

{

cout << A[i] << " ";

arrySum = A[i] + arrySum;

}

cout<<"\nAll Value Summation: "<<arrySum<<endl;

return 0;

}