DATA LAB 9

Binary trees

Name: Ahmed Kasteer

Roll Number: 20F-0336

Section: 3D

**Task 1:**

**Leaf node:** 5

**Siblings of 54:** 77

**Ancestor of 23:** 17,50

**Descendants of 17:** 12,23,9,14,19

**Height/Depth of given tree:** 3

**Is it a complete binary tree? How many nodes will be there?** 15

**In order traversal:** 9 12 14 17 23 19 50 54 67 72 76

**Post order traversal:** 9 14 12 19 23 17 67 54 76 72 50

**Breadth first traversal:** 50 17 72 12 23 5 76 9 14 19 67

**Task 2:**

**Diagram

Description automatically generated**

**Task 3:**

#include <iostream>

using namespace std;

class node {

public:

int data;

node\* left, \* right;

node()

{

data = 0;

left = right = NULL;

}

~node()

{

}

};

class bt {

private:

node\* root;

public:

bt()

{

root = NULL;

}

~bt()

{

}

void insert(int val)

{

node\* newNode = new node();

newNode->data = val;

newNode->left = newNode->right = NULL;

if (root == NULL)

{

root = newNode;

}

else

{

node\* tempNode = root;

while (tempNode->data != val)

{

if (val < tempNode->data)

{

if (tempNode->left != NULL)

{

tempNode = tempNode->left;

}

else

{

tempNode->left = newNode;

}

}

else if (val > tempNode->data)

{

if (tempNode->right != NULL)

{

tempNode = tempNode->right;

}

else

{

tempNode->right = newNode;

}

}

else

{

cout << "Duplicate Value" << endl;

}

}

}

}

node\* search(int key)

{

node\* tempNode = root;

while (tempNode)

{

if (tempNode->data == key)

{

cout << "Key Found" << endl;

return tempNode;

}

else if (key < tempNode->data)

{

tempNode = tempNode->left;

}

else if (key > tempNode->data)

{

tempNode = tempNode->right;

}

}

}

node\* getRoot()

{

return root;

}

void inorder\_traversal(node\* p)

{

if (p != NULL)

{

inorder\_traversal(p->left);

cout << p->data << " ";

inorder\_traversal(p->right);

}

}

void preorder\_traversal(node\* p)

{

if (p != NULL)

{

cout << p->data << " ";

preorder\_traversal(p->left);

preorder\_traversal(p->right);

}

}

void postorder\_traversal(node\* p)

{

if (p != NULL)

{

postorder\_traversal(p->left);

postorder\_traversal(p->right);

cout << p->data << " ";

}

}

int Height(node\* p)

{

if (p == NULL)

{

return -1;

}

else

{

int leftcount = Height(p->left);

int rightcount = Height(p->right);

if (leftcount > rightcount)

{

return(leftcount + 1);

}

else

{

return(rightcount + 1);

}

}

}

int total\_number\_of\_nodes()

{

node\* tempNode = root;

if (tempNode == NULL)

{

return -1;

}

else

{

int nodecount = 0;

count\_nodes(tempNode, nodecount);

return nodecount;

}

}

int total\_number\_of\_leaf\_nodes()

{

node\* tempNode = root;

if (tempNode == NULL)

{

return 0;

}

int nodecount = 0;

count\_nodes(tempNode, nodecount);

int leafnodecount = 0;

leafnodecount = (nodecount + 1) / 2;

return leafnodecount;

}

void count\_nodes(node\* p, int& nc)

{

if (p != NULL)

{

nc++;

count\_nodes(p->left, nc);

count\_nodes(p->right, nc);

}

}

};

int main()

{

bt Tree;

int\* searchresult;

Tree.insert(10);

Tree.insert(12);

Tree.insert(7);

Tree.insert(5);

Tree.insert(11);

Tree.insert(15);

Tree.insert(8);

cout << Tree.search(15) << endl;

cout << Tree.search(5) << endl;

cout << endl;

cout << "Inorder Traversal of tree:" << endl;

Tree.inorder\_traversal(Tree.getRoot());

cout << endl;

cout << "PreOrder Traversal of tree: " << endl;

Tree.preorder\_traversal(Tree.getRoot());

cout << endl;

cout << "Postorder Traversal of tree:" << endl;

Tree.postorder\_traversal(Tree.getRoot());

cout << endl;

cout << endl;

cout << "Height of Tree : ";

cout << Tree.Height(Tree.getRoot()) << endl;

cout << endl;

cout << "Total Number of nodes in Tree : ";

cout << Tree.total\_number\_of\_nodes() << endl;

cout << endl;

cout << "Total Number of leaf nodes in Tree : ";

cout << Tree.total\_number\_of\_leaf\_nodes() << endl;

cout << endl;

}

**Text

Description automatically generated**