

Data structures

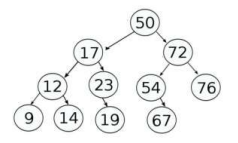
Lab-09



Name: Talib Husain

Roll# 21F-9070

**Task-1**



1. Total leaf nodes : 4

2. Siblings of 54: 54,76

3. Ancestors of 23:  17,50

4. Descendent of 17: 12,9,14,23,19

5. Height and depth of given tree: 3 and 4

6. Is it a complete binary tree? Total number of nodes in a complete binary tree with depth 4No it is not complete binary tree because it’s height is not satisfy the formaula. 2^4-1 =

7. In-order traversal will be: 9,12,14,17,19,23,50,54,67,72,76

8. Post-order traversal will be: 9,14,12,19,23,17,67,54,76,72,50

9. **Bonus Point** Breadth-first traversal of given tree will be?

50,17,72,12,23,54,76,9,14,19,67

**Task-2**

#include<iostream>

using namespace std;

//Creating a tree

struct Tree{

    int \*arr;

    int size;

    //Making a constructor

    Tree(int s =10 ) {

        size =s;

        arr = new int[size];

        for (int i=0;i<size;i++)

        arr[i] = -1;

    }

    void insert(int data, int parent){

        parent = (parent\*2);

        if(parent!=0){

            parent++;

        }

    if(arr[parent] == -1){

       arr[parent] = data;

        return;

    }

    else if( arr[parent+1] == -1){

        arr[parent+1] = data;

    }

   else if( arr[parent+2] == -1){

        arr[parent+2] = data;

    }

    else{

        cout<<"Tree is already filled\n";

    }

    }

    int GetRoot(){

        return arr[0];

    }

    bool Full(){

        if(arr[size-1]!= -1)

        return true;

        return false;

    }

    bool isEmpty(){

        for (int i=0;i<size;i++){

        if(arr[0] != -1)

            return false;

            break;

        }

        return true;

    }

    //Displaying A Node

    void display(){

        cout << "\n";

     for (int i = 0; i < size; i++) {

        if (arr[i] != -1)

        cout << arr[i];

    else

      cout << "-";

      }

    }

//Deleting a NOde

    void deleteNode(int data){

        int parent = 0;

        while (parent<size){

            if(arr[parent]== data){

                arr[parent] = -1;

            }

            parent++;

        }

    }

};

// Driver Code

int main(){

    Tree t(10);

    t.insert(1,0);

    t.insert(2,2);

    t.insert(3,0);

    t.insert(4,1);

    t.insert(5,3);

    t.insert(6,3);

    //Before Deleting a Node

    cout<<"Before Deleting a Node";

    t.display();

    t.deleteNode(6);

    //After Deleting a Node

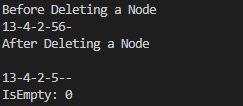
    cout<<"\nAfter Deleting a Node\n";

    t.display();

    cout<<"\nIsEmpty: "<<t.isEmpty()<<endl;

}

**Output**

****

**Task-3**

**Code**

#include<iostream>

#include<math.h>

using namespace std;

//creating a struct for tree node

struct Node{

    Node \*left,\*right;

    int data;

    Node(int d){

        left = NULL;

        right = NULL;

        data = d;

    }

};

//craeting a  Binary Tree ADT

class BT{

    Node \*root;

public:

//Constructor

    BT(Node \*r = NULL){

        root = r;

    }

    //Method to insert

    void insert(int data){

        Node \*newNode = new Node(data);

        //if tree does't exist craete a root

        if(root==NULL){

            root = newNode;

            return;

        }

        Node \* temp = root;

        //Finding the place of inseering according to rules of BST

        while (temp!=NULL){

            //if data less than root move to left

            if(temp->data>data){

                if(temp->left)

                temp = temp->left;

            else {

               temp->left =  newNode;

               break;

            }

            }

            //if data greate than root move to right

            else if(temp->data<data){

                if(temp->right){

                        temp = temp->right;

                }

                    else{

                        temp->right = newNode;

                       break;

                    }

                }

            //if duplicate found don't insert

            else{

                cout<<"Duplicate Value\n";

                               break;

            }

       }

    }

    // to search any node

    Node\* search(int data){

        //calling a finding node method

        return find(root,data);

    }

    //Method to display

    void Display\_Inorder(){

        inorderTravers(root);

        cout<<endl;

    }

    void Display\_PreOrder(){

        preorderTraverse(root);

        cout<<endl;

    }

    void Display\_PostOrder(){

        postorderTraverse(root);

        cout<<endl;

    }

    // method to find height

    int height(){

        int h = Find\_Height(root);

        return h;

    }

    //Fndiing the no of

    int No\_of\_Nodes(){

        //Gettting the height of tree abnd applying formula on it

        int level = height();

        int nodes = pow(2,level);

        nodes--;

        return nodes;

    }

    //Finding the no of NOdes

     int No\_of\_LeafNodes(){

        //Gettting the height of tree abnd applying formula on it

        int level = height();

        level--;

        int leafnodes = pow(2,level);

        return leafnodes;

    }

    Node \* GetRoot(){

        return root;

    }

private:

//recursily find the desired node if not found simpple return

    Node\* find(Node\*n,int data){

         if(n == NULL)

        return n;

        else if(n->data == data)

        return n;

        else if(n->data<data){

         return find(n->right,data);

        }

        else if(n->data>data){

         return find(n->left,data);

        }

    }

    //To recusily traves inoder

    void inorderTravers(Node \*n){

         if (n){

            inorderTravers(n->left);

            cout << n->data << " ";

            inorderTravers(n->right);

         }

         return;

    }

    //To recusily travese preorder

     void preorderTraverse(Node \*n){

         if (n){

            cout << n->data << " ";

            preorderTraverse(n->left);

            preorderTraverse(n->right);

         }

         return;

    }

    //To recursivly Traverse preorder

     void postorderTraverse(Node \*n){

         if (n){

            postorderTraverse(n->left);

            postorderTraverse(n->right);

            cout << n->data << " ";

         }

         return;

    }

    int Find\_Height(Node\* n){

        if (!n)

        return 0;

    else {

        int left\_height = Find\_Height(n->left);

        int right\_height = Find\_Height(n->right);

        if (left\_height >= right\_height)

            return left\_height + 1;

        else

            return right\_height + 1;

    }

    }

};

int main(){

    BT t;

    bool exit = false;

    int choice;

    int data;

    while (!exit)  {

        cout<<"1-Insert\n2-Search\n3-Display Inorder\n4-Display PreOrder\n5-Display PostOrder\n";

        cout<<"6-Get Height\n7-Total No of NOdes\n0-1Eixt\nChosse option: ";

        cin>>choice;

        switch (choice)

        {

        case 1:

        cout<<"Enter Data: ";

        cin>>data;

        t.insert(data);

            break;

        case 2:

        cout<<"Enter Data to search: ";

        cin>>data;

        cout<<t.search(data)->data<<endl;

        break;

        case 3:

        t.Display\_Inorder();

        break;

        case 4:

        t.Display\_PreOrder();

        break;

        case 5:

        t.Display\_PostOrder();

        break;

        case 6:

        cout<<"Height: "<<t.height()<<endl;

        break;

        case 7:

        default:

        exit = true;

            break;

        }

    }

}

Output

Text

Description automatically generated Text

Description automatically generated