Data Structures (UCS301) Assignment

NAME-Suneet Arora **ROLL NO-1024030174**

```cpp
#include<iostream>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
root=NULL;
}

};
void preorder(Node* root){
    if(root==NULL) return;
    cout<<root->data<<" ";
    preorder(root->left);
    preorder(root->right);
}
void inorder(Node* root){
    if(root==NULL) return;
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);
}
void postorder(Node* root){
    if(root==NULL) return;
    postorder(root->left);
    postorder(root->right);
cout<<root->data<<" ";
```

```
}
int main(){
BST a;
a.root = new Node(1);
a.root->left = new Node(2);
a.root->right = new Node(3);
a.root->left->left = new Node(4);
a.root->left->right = new
Node(5);
cout<<"Preorder ";preorder(a.root);
cout<<endl;
cout<<"Inorder ";inorder(a.root);
cout<<endl;
cout<<"postorder ";postorder(a.root);

return 0;
}
```

```
#include<iostream>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
    left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
    root=NULL;
}

};
//Doing recursively

// bool search(Node* root,int x){
//    if(root==NULL) return false;
```

```
//    else if(root->data == x){
//         return true;
//     }
//     else if(root->data<x) return search(root->right,x);
//     else{
//        return search(root->left,x);
//     }

// }

//Doing non recursively

bool search(Node* root,int x){
    while(root!=NULL){
     if(root->data==x) return true;
     else if(root->data<x){
        root = root->right;
     }
     else{
        root = root->left;
     }
    }
return false;
}
int main(){
BST tree;
tree.root = new Node(8);
tree.root->left = new Node(3);
tree.root->right = new
Node(10);
tree.root->left->left = new Node(1);
tree.root->left->right = new Node(6);
tree.root->left->right->left = new Node(4);
tree.root->left->right->right = new Node(7);
tree.root->right->right = new Node(14);
tree.root->right->right->left = new
Node(13); if(search(tree.root,10)){
    cout<<"Element found."<<endl;
}
else{
    cout<<"Element is not present."<<endl;
}

    return 0;
}
#include<iostream>
```

```
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
    data = a;
    left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
    root=NULL;
}

};
int max(Node* root){
    if(root==NULL) return-1;
    while(root->right!=NULL)
    {
        root = root->right;
    }
    return root->data;
}
int main(){
BST tree;
tree.root = new Node(8);
tree.root->left = new Node(3);
tree.root->right = new
Node(10);
tree.root->left->left = new Node(1);
tree.root->left->right = new Node(6);
tree.root->left->right->left = new Node(4);
tree.root->left->right->right = new Node(7);
tree.root->right->right = new Node(14);
tree.root->right->right->left = new
Node(13);
if(max(tree.root)==-1) cout<<"The tree is empty."<<endl;
else{
    cout<<"The maximum element is: "<<max(tree.root)<<endl;
}
```

```
return 0;
}
```

```
#include<iostream>
using namespace std;

class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
root=NULL;
}

};
int min(Node* root){
    if(root==NULL) return-1;
    while(root->left!=NULL){
root = root->left;
    }
return root->data;
}
int main(){
BST tree;
tree.root = new Node(8);
tree.root->left = new Node(3);
tree.root->right = new Node(10);
tree.root->left->left = new Node(1);
tree.root->left->right = new Node(6);
tree.root->left->right->left = new Node(4);
```

```
tree.root->left->right->right = new Node(7);
tree.root->right->right = new Node(14);
tree.root->right->right->left = new Node(13);
if(min(tree.root)==-1) cout<<"The tree is empty."<<endl;
else{
cout<<"The minimum element is: "<<min(tree.root)<<endl;
}









return 0;
}
```

```
#include<iostream>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
root=NULL;
}

};
int mini(Node* root){
    if(root==NULL) return-1;
    while(root->left!=NULL){
root = root->left;
    }
return root->data;
```

```
}
int Successor(Node* root,Node* x){
      if(x->right!=NULL){
       return mini(x->right);
    }
     else{
    Node* successor = NULL;
    while(root!=NULL){
        if(root->data>x->data){
            successor = root;
            root = root->left;
        }
        else if(root->data<x->data){
            root = root->right;
        }
     else{
        break;
     }
    }

   if(successor!=NULL) return successor->data;
        else{
            return -1;
        }
    }

}
int main(){
BST tree;
tree.root = new Node(8);
tree.root->left = new Node(3);
tree.root->right = new
Node(10);
tree.root->left->left = new Node(1);
tree.root->left->right = new Node(6);
tree.root->left->right->left = new Node(4);
tree.root->left->right->right = new Node(7);
tree.root->right->right = new Node(14);
tree.root->right->right->left = new
Node(13);
cout<<Successor(tree.root,tree.root);
    return 0;
}
```

```
#include<iostream>
```

```
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
    data = a;
    left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
    root=NULL;
}

};
int maxi(Node* root){
    if(root==NULL) return-1;
    while(root->right!=NULL)
    {
        root = root->right;
    }
    return root->data;
}
int Predecessor(Node* root, Node* x){
    if(x->left!=NULL){
        return maxi(x->left);
    }
    else{
        Node* predecessor = NULL;
        while(root!=NULL){
            if(root->data<x->data){
                predecessor = root;
                root = root->right;
            }
            else if(root->data> x->data){
                root = root->left;
            }
            else{
                break;
            }
        }
```

```
    if(predecessor!=NULL){ return
        predecessor->data;
    }
else {
return -1;
    }
}
}
int main(){
BST tree;
tree.root = new Node(8);
tree.root->left = new Node(3);
tree.root->right = new Node(10);
tree.root->left->left = new Node(1);
tree.root->left->right = new Node(6);
tree.root->left->right->left = new Node(4);
tree.root->left->right->right = new Node(7);
tree.root->right->right = new Node(14);
tree.root->right->right->left = new Node(13);
cout<<Predecessor(tree.root,tree.root);
return 0;
}
```

```
#include<iostream>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
root=NULL;
}
```

```
};
void insert(Node* &root,Node* x){
    Node* temp = root;
if(root==NULL){ root = x;
return;
}
else if(temp->left==NULL && temp->right==NULL){
    if(temp->data<x->data){
        temp->right = x;
    }
    else{
        temp->left = x;
    }
}
else{
    while(temp!=NULL){
        if(temp->data<x->data){
            if(temp->right==NULL){
                temp->right = x;
                break;
            }
            else{
                temp = temp->right;
            }
        }
        else{
            if(temp->left==NULL){
                temp->left = x;
                break;
            }
            else{
                temp = temp->left;
            }
        }
    }
}
}
int main(){
BST tree;
insert(tree.root,new Node(5));
insert(tree.root,new Node(7));
insert(tree.root,new Node(9));
insert(tree.root,new Node(2));
insert(tree.root,new Node(3));
```

```
insert(tree.root,new Node(4));
insert(tree.root,new Node(11));
insert(tree.root,new Node(13));
insert(tree.root,new Node(10));

return 0;
}
```

```
#include <iostream>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
Node(int a){
data = a;
left = right =NULL;
}
};

class BST{
public:
Node* root;
BST(){
root=NULL;
}

};
Node* mini(Node* root){
    if(root==NULL) return NULL;
    while(root->left!=NULL){
root = root->left;
    }
return root;
}
Node* Successor(Node* root,Node* x){
      if(x->right!=NULL){
return mini(x->right);
    }
else{
Node* successor = NULL;
    while(root!=NULL){
```

```
        if(root->data>x->data){
            successor = root;
            root = root->left;
        }
        else if(root->data<x->data){
            root = root->right;
        }
     else{
        break;
     }
    }

   if(successor!=NULL) return successor;
        else{
            return NULL;
        }
    }

}
void remove (Node *&root, int x)
{
    Node *temp = root;
    if(temp->right==NULL && temp->left==NULL){ delete root;root = NULL; return;}
    Node *prev = NULL;
    while (temp != NULL)
    {
        if (temp->data < x)
        {
            prev = temp;
            temp = temp->right;
        }
        else if (temp->data > x)
        {
            prev = temp;
            temp = temp->left;
        }
        else
        {
            break;
        }
    }
    if (temp->left == NULL && temp->right == NULL)
    {
        if (temp->data > prev->data)
        {
```

```
            prev->right = NULL;
            delete temp;
        }
        else
        {
            prev->left = NULL;
            delete temp;
        }
    }
    else if(temp->left!=NULL && temp->right==NULL){
        if(temp->data > prev->data){
            prev->right = temp->left;
            delete temp;
        }
        else{
            prev->left = temp->left;
            delete temp;
        }
    }
    else if(temp->left==NULL && temp->right!=NULL){
        if(temp->data >prev->data){
            prev->right = temp->right;
            delete temp;
        }
        else{
            prev->left = temp->right;
            delete temp;
        }
    }
    else{
        Node* temp2 = root;
        Node* g = temp;
        Node *cur_Successor = Successor(temp2,g);
        temp->data = cur_Successor->data;
        remove(temp->right,cur_Successor->data);
    }
}

int main()
{
    return 0;
}
```

```cpp
#include<iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int x) {
        data = x;
        left = right = NULL;
    }
};

class BST {
public:
    Node* root;
    BST() {
        root = NULL;
    }
};

int find_max(int a, int b) {
    return (a > b) ? a : b;
}

int max_depth(Node* root) {
    if (root == NULL)
        return 0;

    int left_depth = max_depth(root->left);
    int right_depth =
    max_depth(root->right);

    return find_max(left_depth, right_depth) + 1;
}

int main() {
    BST tree;
    tree.root = new Node(5);
    tree.root->left = new Node(2);
    tree.root->right = new
    Node(12);
    tree.root->left->left = new Node(1);
    tree.root->left->right = new Node(3);
    tree.root->right->left = new Node(9);
```

```
tree.root->right->right = new Node(21);
    tree.root->right->right->left = new Node(19);
    tree.root->right->right->right = new Node(25);

cout << "The maximum depth of the BST is: " << max_depth(tree.root); return
    0;
}
```

```
#include<iostream>
using namespace std;

class Node {
public:
int data; Node*
    left; Node*
    right;

    Node(int x) {
        data = x;
left = right = NULL;
    }
};

class BST {
public:
Node* root;
    BST() {
root = NULL;
    }
};

int find_min(int a, int b) {
    return (a < b) ? a : b;
}

int min_depth(Node* root) {
    if (root == NULL)
return 0;

if (root->left == NULL)
        return min_depth(root->right) + 1;

if (root->right == NULL)
```

```
        return min_depth(root->left) + 1;

    return find_min(min_depth(root->left), min_depth(root->right)) + 1;
}

int main() {
BST tree;
    tree.root = new Node(5);
    tree.root->left = new Node(2);
    tree.root->right = new Node(12);
    tree.root->left->left = new Node(1);
    tree.root->left->right = new Node(3);
    tree.root->right->left = new Node(9);
    tree.root->right->right = new Node(21);
    tree.root->right->right->left = new Node(19);
tree.root->right->right->right = new Node(25);

cout << "The minimum depth of the BST is: " << min_depth(tree.root); return
    0;
}
```

```
#include<iostream>
#include<vector>
using namespace std;
class Node{
public:
int data; Node*
    left; Node*
    right;
    Node(int x){
data = x;
left = right = NULL;
    }
};
class BST{
public: Node*
    root;
    BST(){
root = NULL;
    }
};
vector<int> v;
```

```
void inorder(Node* root){
if(root==NULL){
    return;
}
inorder(root->left);
v.push_back(root->data);
inorder(root->right);
}
bool is_BST(vector<int> &v){
    if((v[0]>v[1]) || (v[v.size()-2]>v[v.size()-1])) return false;

    for(int i=1;i<v.size()-1;i++){
        if(v[i]>=v[i+1]) return false;
    }
    return true;
}
int main(){
      BST tree;
    tree.root = new Node(5);
    tree.root->left = new Node(2);
    tree.root->right = new Node(4);
    tree.root->left->left = new Node(1);
    tree.root->left->right = new Node(3);
    tree.root->right->left = new Node(9);
    tree.root->right->right = new Node(21);
    tree.root->right->right->left = new
    Node(19);
    tree.root->right->right->right = new Node(25);
inorder( tree.root);
if(is_BST(v)) cout<<"The given binary tree is BST"<<endl;
else cout<<"The given binary tree is not BST"<<endl;
    return 0;
}
```