BST PROPERTIES

SUBMITTED BY:

JASKIRAT SINGH

2020CSC1008

Question

Add functions to your BST to compute:

a) leaf count

b) non leaf count

c) height

d) Mirrorify

e) Check if identical

f) Morris Inorder traversal

g) Successor

h) Predecessor

Upload PDF containing code and output screenshot.

Code

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
class Node
{
public:
    int data;
    Node *left, *right;
    Node() // Default constructor.
    {
        data = 0;
        left = right = NULL;
    }
    Node(int value) // Parameterized constructor.
    {
        data = value;
        left = right = NULL;
    }
};
```

```
class BST
{
public:
    Node *root;
    int count;

    BST()
    {
        root = NULL;
        count = 0;
    }
    Node *createNode(int data);
    void Insert(int data);
    void inorder(Node *troot);
    int getLeafCount(Node *node);
    int height(Node *node);
    int countNonleaf(Node *root);
    Node *mirrorify(Node *node);
    bool isIdentical(Node *root1, Node *root2);
    Node *predecessor(Node *x);
    Node *successor(Node *x);
    void MorrisInorder();

};
```

```
Node *BST::createNode(int data)
{
    Node *newNode = new Node;
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void BST ::Insert(int value)
{
    Node *newNode = createNode(value);
    if (!root)
    {
        root = newNode; // Insert the first node, if root is NULL.
        count++;
        return;
    }
    Node *temp = root, *prev = NULL;
    while (temp != NULL)
    { // find a place for inserting new node;
        prev = temp;
        if (value < temp->data)
            temp = temp->left;
```

```
        else
            temp = temp->right;
    }
    if (value < prev->data)
        prev->left = newNode;
    if (value > prev->data)
        prev->right = newNode;
    count++;
}

void BST::inorder(Node *troot)
{
    if (troot != NULL)
    {
        inorder(troot->left);
        cout << troot->data << " ";
        inorder(troot->right);
    }
}

int BST::getLeafCount(Node *node)
{
    if (node == NULL)
        return 0;
    if (node->left == NULL && node->right == NULL)
```

```
        return 1;
    else
        return getLeafCount(node->left) + getLeafCount(node->right);
}

int BST::countNonleaf(Node *root)
{
    if (root == NULL || (root->left == NULL && root->right == NULL))
        return 0;
    // If root is Not NULL and its one of its child is also not NULL
    return 1 + countNonleaf(root->left) + countNonleaf(root->right);
}

int BST::height(Node *node)
{
    if (node == NULL)
        return 0;
    else
    {
        /* compute the depth of each subtree */
        int lDepth = height(node->left);
        int rDepth = height(node->right);

        /* use the larger one */
        if (lDepth > rDepth)
```

```
            return (lDepth + 1);
        else
            return (rDepth + 1);
    }
}

Node *BST::mirrorify(Node *node)
{
    if (node == NULL)
        return NULL;
    else
    {
        Node *mirror = createNode(node->data);
        mirror->right = mirrorify(node->left);
        mirror->left = mirrorify(node->right);
        return mirror;
    }
}

bool BST::isIdentical(Node *root1, Node *root2)
{
    if (root1 == NULL && root2 == NULL) // Check if both the trees are empty
        return true;
    else if (root1 != NULL && root2 == NULL) // If any one of the tree is non-empty and other is empty, return false
```

```
        return false;
    else if (root1 == NULL && root2 != NULL)
        return false;
    else // Check if current data of both trees equal and recursively check for left and right subtrees
    {
        if (root1->data == root2->data && isIdentical(root1->left, root2->left) && isIdentical(root1->right, root2->right))
            return 1;
        else
            return 0;
    }
}

Node *BST::predecessor(Node *x)
{
    Node *pred = NULL;
    if (x == root && x->left == NULL)
    {
        return NULL;
    }
    else if (x->left)
    {
        pred = x->left;
        while (pred->right)
            pred = pred->right;
```

```
    }
    else
    {
        Node *temp = root;
        while (1)
        {
            if (temp->data < x->data)
            {
                pred = temp;
                temp = temp->right;
            }
            else if (temp->data > x->data)
                temp = temp->left;
            else
                break;
        }
    }
    return pred;
}

Node *BST::successor(Node *x)
{
    Node *succ = NULL;
    if (x == root && x->right == NULL)
    {
```

```
    return NULL;
}
else if (x->right)
{
    succ = x->right;
    while (succ->left)
        succ = succ->left;
}
else
{
    Node *temp = root;
    while (1)
    {
        if (temp->data > x->data)
        {
            succ = temp;
            temp = temp->left;
        }
        else if (temp->data < x->data)
            temp = temp->right;
        else
            break;
    }
}
return succ;
```

```
}


void BST::MorrisInorder()
{
    Node *p = root, *tmp;
    while (p != NULL)
    {
        if (p->left == NULL)
        {
            cout << p->data << " ";
            p = p->right;
        }
        else
        {
            tmp = p->left;
            while (tmp->right != NULL && tmp->right != p) // go to the rightmost node of
the left subtree
                tmp = tmp->right;                    // or to the temporary parent of p;
            if (tmp->right == NULL)
            {                   // if 'true' rightmost node was reached,
                tmp->right = p; //make it a temporary parent of the current root
                p = p->left;
            }
            else
            {                           // else a temporary parent has been found
```

```
            cout << p->data << " "; //visit node p
            tmp->right = NULL;      // and then cut the right pointer of the current parent
            p = p->right;
        }
    }
  }
}

int main()
{
    BST bst;

    int choice;
    char ans;

    cout << "\n Enter the number of nodes... ";
    int n;
    cin >> n;

    cout << "\n Inserting " << n << " random numbers...\n";
    srand(time(0));
    for (int i = 0; i < n; i++)
    {
        int random = rand() % 50;
        cout << "\nInserting " << random << "...";
```

```
    bst.Insert(random);
}
cout << "\n"
     << n << " random numbers inserted successfully!";

Node *head = new Node;
head = bst.root;
cout << "\nElements in the tree : ";
bst.inorder(head);

do
{

    cout << "\n*********Menu**********"
         << "\n1. Get Leaf Count"
         << "\n2. Count Non Leaf"
         << "\n3. Height"
         << "\n4. Mirrorify"
         << "\n5. Identical"
         << "\n6. Predecessor"
         << "\n7. Successor"
         << "\n8. Morris Inorder"
         << "\n0. exit \n\n"
         << "Enter your choice... ";
    cin >> choice;
```

```
switch (choice)
{
case 1:
{
    cout << "\nNumber of leaves : " << bst.getLeafCount(head);
    break;
}
case 2:
{
    cout << "\nNumber of non-leaves : " << bst.countNonleaf(head);
    break;
}
case 3:
{
    cout << "\nHeight of the tree : " << bst.height(head);
    break;
}
case 4:
{
    cout<<"\nMirrorify tree... ";
    Node *New = bst.mirrorify(head);
    bst.inorder(New);
    break;
}
```

```
case 5:
{
    BST bst2;
    cout << "\nEnter number of nodes for tree 2 : ";
    cin >> n;
    cout << "\nEnter elements in tree 2... ";
    int ele;
    for (int i = 0; i < n; i++)
    {
        cin >> ele;
        bst2.Insert(ele);
    }
    Node *head2 = new Node;
    head2 = bst2.root;
    cout << "\nElements in the tree 2 : ";
    bst2.inorder(head2);

    cout << "\n\nisIdentical : " << boolalpha << bst.isIdentical(head, head2);
    break;
}
case 6:
{
    Node *New = new Node;
    New = bst.root->left->right;
    Node *New2 = bst.predecessor(New);
```

```
    cout<<"\nPredecessor of "<<New->data<<" : "<<New2->data;
    New = bst.root->right->left;
    New2 = bst.predecessor(New);
    cout<<"\nPredecessor of "<<New->data<<" : "<<New2->data;
    break;
}
case 7:
{
     Node *New = new Node;
    New = bst.root->left->right;
    Node *New2 = bst.successor(New);
    cout<<"\nSuccessor of "<<New->data<<" : "<<New2->data;
    New = bst.root->right->left;
    New2 = bst.successor(New);
    cout<<"\nSuccessor of "<<New->data<<" : "<<New2->data;
    break;
}
case 8:
{
    cout<<"\nElements in the tree... ";
    bst.MorrisInorder();
    break;
}
default:
{
```

```
            exit(101);
        }
        }

        cout << "\n\nDo you want to continue?"
             << "\nPress y to continue"
             << "\nelse press n... ";
        cin >> ans;

    } while (ans == 'y');

    return 0;
}
```

Output

Leaf count

Non-leaf count

```
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 2

Number of non-leaves : 4

Do you want to continue?
Press y to continue
else press n...
```

Height

```
*********Menu************
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 3

Height of the tree : 4

Do you want to continue?
Press y to continue
else press n...
```


Mirrorify

```
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 4

Mirrorify tree... 34 24 14 10 7 3

Do you want to continue?
Press y to continue
else press n...
```

Check if identical

```
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 5

Enter number of nodes for tree 2 : 6

Enter elements in tree 2... 3
7
10
14
24
34

Elements in the tree 2 : 3 7 10 14 24 34

isIdentical : false

Do you want to continue?
```

Morris Inorder traversal

```
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 8

Elements in the tree... 7 8 36 43 49

Do you want to continue?
Press y to continue
else press n...
```

Successor

```
 Enter the number of nodes... 4

 Inserting 4 random numbers...

Inserting 17...
Inserting 11...
Inserting 16...
Inserting 3...
4 random numbers inserted successfully!
Elements in the tree : 3 11 16 17
*********Menu************
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 7
```

Predecessor

```
1. Get Leaf Count
2. Count Non Leaf
3. Height
4. Mirrorify
5. Identical
6. Predecessor
7. Successor
8. Morris Inorder
0. exit

Enter your choice... 6

Predecessor of 27 : 26
```