Insertion and Deletion in BST

```
// Aneesh Panchal
// 2K20/MC/21

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

class Node {
public:
    int data=-1;
    Node* right=NULL;
    Node* left=NULL;
};

class BinarySearchTree {
    Node* root;

    public:
        BinarySearchTree(){root = NULL;}

    void insert(int data_){
        Node* temp = root;
        Node* newNode = new Node();
        newNode->data = data_;
        if(root==NULL){
            root = newNode;
            return;
        }
        while(1){
            if(data_<temp->data){
                if(temp->left==NULL){
                    temp->left = newNode;
                    return;
                }
                else
                    temp = temp->left;
            }
            else{
                if(temp->right==NULL){
                    temp->right = newNode;
                    return;
                }
                else
                    temp = temp->right;
            }
        }
    }

    int findmax(Node* temp){
        int max=0;
        do{
```

```
            if(temp->data>max){
                max=temp->data;
            }
            temp = temp->right;
        }
        while(temp!=NULL);
        return max;
    }

    void deletion(int data_){
        int count=-1;
        int max=-1;
        Node* temp = root;
        Node* parent = root;
        Node* newNode = new Node();
        newNode->data = data_;

        if(root==NULL){
            cout<<"Tree Doesn't Exist !!!!!"<<endl;
            return;
        }

        while(1){
            if(data_==temp->data){
                //leaf node
                if(temp->left==NULL && temp->right==NULL){
                    if(temp==root)
                        root=NULL;
                    else if(count==0)
                        parent->left = NULL;
                    else
                        parent->right = NULL;
                    return;
                }

                //both nodes
                if(temp->left!=NULL && temp->right!=NULL){
                    max = findmax(temp->left);
                    deletion(max);
                    temp->data = max;
                    return;
                }

                //left node
                else if(temp->left!=NULL){
                    if(root==temp)
                        root = temp->left;
                    else if(count==0)
                        parent->left = temp->left;
                    else
                        parent->right = temp->left;
                    return;
                }
```

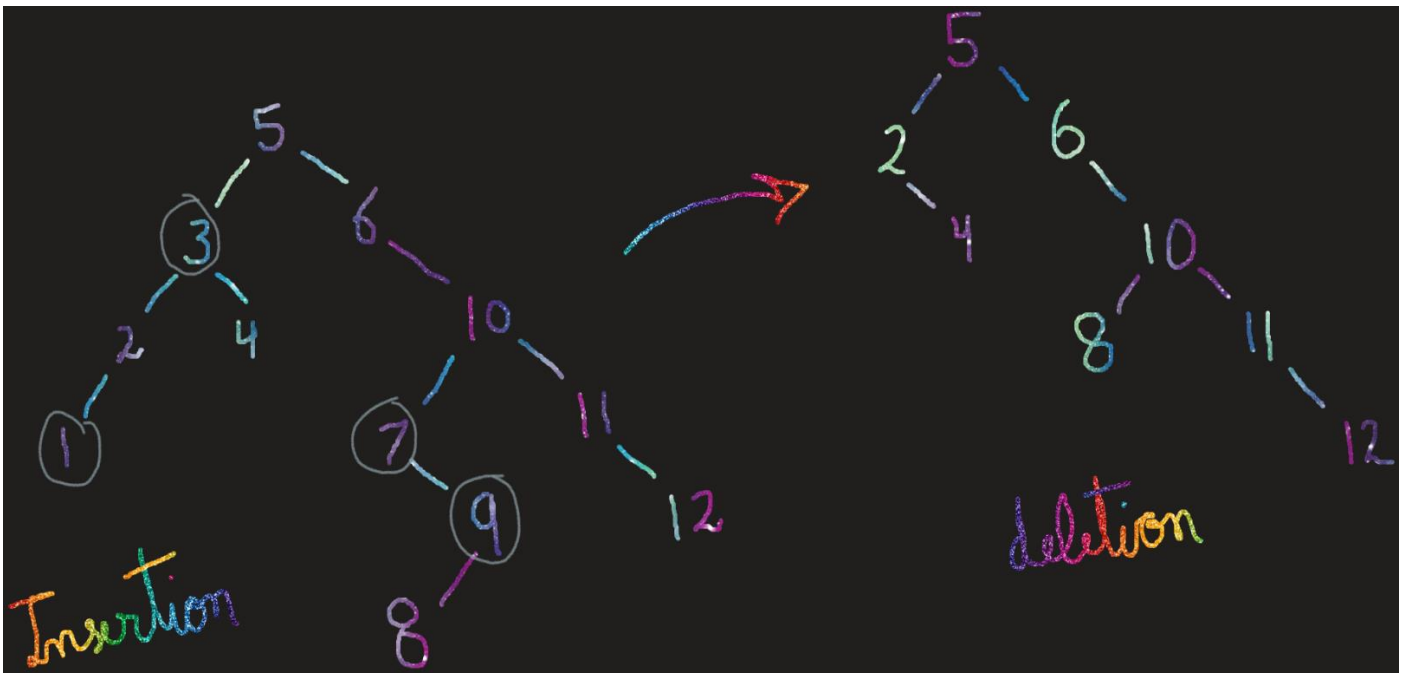
```
            //right node
            else{
                if(root==temp)
                    root = temp->right;
                else if(count==0)
                    parent->left = temp->right;
                else
                    parent->right = temp->right;
                return;
            }
        }

        //traversal to find particular node
        if(data_<temp->data){
            if(temp->left==NULL){
                cout<<"Value Doesn't Exist !!!!!"<<endl;
                return;
            }
            else{
                parent = temp;
                temp = temp->left;
                count = 0;
            }
        }
        else{
            if(temp->right==NULL){
                cout<<"Value Doesn't Exist !!!!!"<<endl;
                return;
            }
            else{
                parent = temp;
                temp = temp->right;
                count = 1;
            }
        }
    }
}

void show(){
    stack<Node*> nodestack;
    Node *curr = root;
    if(root==NULL){
        cout<<"Empty Tree !!!!!"<<endl;
        return;
    }
    while(curr!=NULL || nodestack.empty()==false){
        while(curr!=NULL){
            nodestack.push(curr);
            curr = curr->left;
        }
        curr = nodestack.top();
        nodestack.pop();
        cout<<curr->data<<" ";
        curr = curr->right;
```

```
        }
        cout<<endl;
    }
};

int main(){
    BinarySearchTree BST;

    cout<<endl;
    BST.show(); //Error
    BST.deletion(1); //Error

    BST.insert(5);
    BST.insert(3);
    BST.insert(2);
    BST.insert(6);
    BST.insert(4);
    BST.insert(1);
    BST.insert(10);
    BST.insert(7);
    BST.insert(9);
    BST.insert(8);
    BST.insert(11);
    BST.insert(12);

    cout<<endl<<"Inorder Traversal after all Insertion: ";
    BST.show();
    cout<<endl<<endl;

    BST.deletion(1); //leaf node
    BST.deletion(3); //both node
    BST.deletion(9); //left node
    BST.deletion(7); //right node
    BST.deletion(9); //Error

    cout<<endl<<"Inorder Traversal after all Deletion: ";
    BST.show();
    cout<<endl<<endl;
    return 0;
}
```

File Edit Selection View Go Run Terminal Help — BSTNR.cpp - Codes - Visual Studio Code

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\Codes> cd "e:\Codes\CS 251 DS\6. Trees\" ; if ($?) { g++ BSTNR.cpp -o BSTNR } ; if ($?) { .\BSTNR }

Empty Tree !!!!!
Tree Doesn't Exist !!!!!

Inorder Traversal after all Insertion: 1 2 3 4 5 6 7 8 9 10 11 12


Value Doesn't Exist !!!!!

Inorder Traversal after all Deletion: 2 4 5 6 8 10 11 12


PS E:\Codes\CS 251 DS\6. Trees>
```



Traversal, Insertion and Deletion in Right In-Threaded Binary Tree

```cpp
// Aneesh Panchal
// 2K20/MC/21

#include<iostream>
using namespace std;

class Node {
public:
    int data=-1;
    int thread=0;
    Node* right=NULL;
    Node* left=NULL;
};

class RightInthreadedBinaryTree {
    Node* root;

    public:
        RightInthreadedBinaryTree(){root=NULL;}

    void insert(int data_){
        Node* temp = root;
        Node* tempthread = NULL;
        Node* newNode = new Node();
        newNode->data = data_;
        if(root==NULL){
            root = newNode;
            return;
        }
        while(1){
            if(data_<temp->data){
                if(temp->left==NULL){
                    temp->left = newNode;
                    newNode->right = temp;
                    newNode->thread = 1;
                    return;
                }
                else
                    temp = temp->left;
            }
            else{
                if(temp->thread==1 || temp->right ==NULL){
                    tempthread = temp->right;
                    temp->thread = 0;
                    temp->right = newNode;
                    newNode->right = tempthread;
                    newNode->thread = 1;
                    return;
                }
                else
                    temp = temp->right;
```

```
        }
    }
}

int findmax(Node* temp){
    int max=0;
    while(1){
        if(temp->data>max){
            max=temp->data;
        }
        if(temp->thread==1){
            return max;
        }
        temp = temp->right;
    }
}

void deletion(int data_){
    int count=-1;
    int max=-1;
    Node* temp = root;
    Node* parent = root;
    Node* tempnode = root;
    if(root==NULL){
        cout<<"Tree Doesn't Exist !!!!!"<<endl;
        return;
    }

    while(1){
        if(data_==temp->data){
            //leaf node
            if(temp->left==NULL && (temp->thread==1 || temp->right ==NULL)){
                if(temp==root)
                    root = NULL;
                else if(count==0)
                    parent->left = NULL;
                else
                    parent->right = temp->right;
                return;
            }

            //both nodes
            if(temp->left!=NULL && temp->thread!=1){
                max = findmax(temp->left);
                deletion(max);
                temp->data = max;
                return;
            }

            //left node
            else if(temp->left!=NULL){
                tempnode = temp->left;
                while(1){
                    if(tempnode->thread==1){
```

```
                        break;
                    }
                    else{
                        tempnode = tempnode->right;
                    }
                }
                tempnode->right = temp->right;
                if(root==temp)
                    root = temp->left;
                else if(count==0)
                    parent->left = temp->left;
                else
                    parent->right = temp->left;
                return;
            }

            //right node
            else{
                if(root==temp)
                    root = temp->right;
                else if(count==0)
                    parent->left = temp->right;
                else
                    parent->right = temp->right;
                return;
            }
        }

        //traversal to find particular node
        if(data_<temp->data){
            if(temp->left==NULL){
                cout<<"Value Doesn't Exist !!!!!"<<endl;
                return;
            }
            else{
                parent = temp;
                temp = temp->left;
                count = 0;
            }
        }
        else{
            if(temp->right==NULL || temp->thread==1){
                cout<<"Value Doesn't Exist !!!!!"<<endl;
                return;
            }
            else{
                parent = temp;
                temp = temp->right;
                count = 1;
            }
        }
    }
}
```

```
    void show(){
        if(root==NULL){
            cout<<"Empty Tree !!!!!"<<endl;
            return;
        }

        Node* temp = root;
        while(temp->left != NULL){
            temp = temp->left;
        }

        while(temp != NULL){
            cout<<temp->data<<" ";
            if(temp->thread == 1){
                temp = temp->right;
            }
            else{
                temp = temp->right;
                while(temp->left != NULL){
                   temp = temp->left;
                }
            }
        }
    }
};

int main(){
    RightInthreadedBinaryTree RITBT;

    cout<<endl;
    RITBT.show(); //Error
    RITBT.deletion(1); //Error

    RITBT.insert(5);
    RITBT.insert(3);
    RITBT.insert(2);
    RITBT.insert(6);
    RITBT.insert(4);
    RITBT.insert(1);
    RITBT.insert(10);
    RITBT.insert(7);
    RITBT.insert(9);
    RITBT.insert(8);
    RITBT.insert(11);
    RITBT.insert(12);

    cout<<endl<<"Inorder Traversal after all Insertion: ";
    RITBT.show();
    cout<<endl<<endl;

    RITBT.deletion(1); //leaf node
    RITBT.deletion(3); //both node
    RITBT.deletion(9); //left node
    RITBT.deletion(7); //right node
```

```
    RITBT.deletion(9); //Error

    cout<<endl<<"Inorder Traversal after all Deletion: ";
    RITBT.show();
    cout<<endl<<endl;
    return 0;
}
```

```
File  Edit  Selection  View  Go  Run  Terminal  Help                RITBT.cpp - Codes - Visual Studio Code

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\Codes> cd "e:\Codes\CS 251 DS\6. Trees\" ; if ($?) { g++ RITBT.cpp -o RITBT } ; if ($?) { .\RITBT }

Empty Tree !!!!!
Tree Doesn't Exist !!!!!

Inorder Traversal after all Insertion: 1 2 3 4 5 6 7 8 9 10 11 12

Value Doesn't Exist !!!!!

Inorder Traversal after all Deletion: 2 4 5 6 8 10 11 12

PS E:\Codes\CS 251 DS\6. Trees>
```