DAY 18

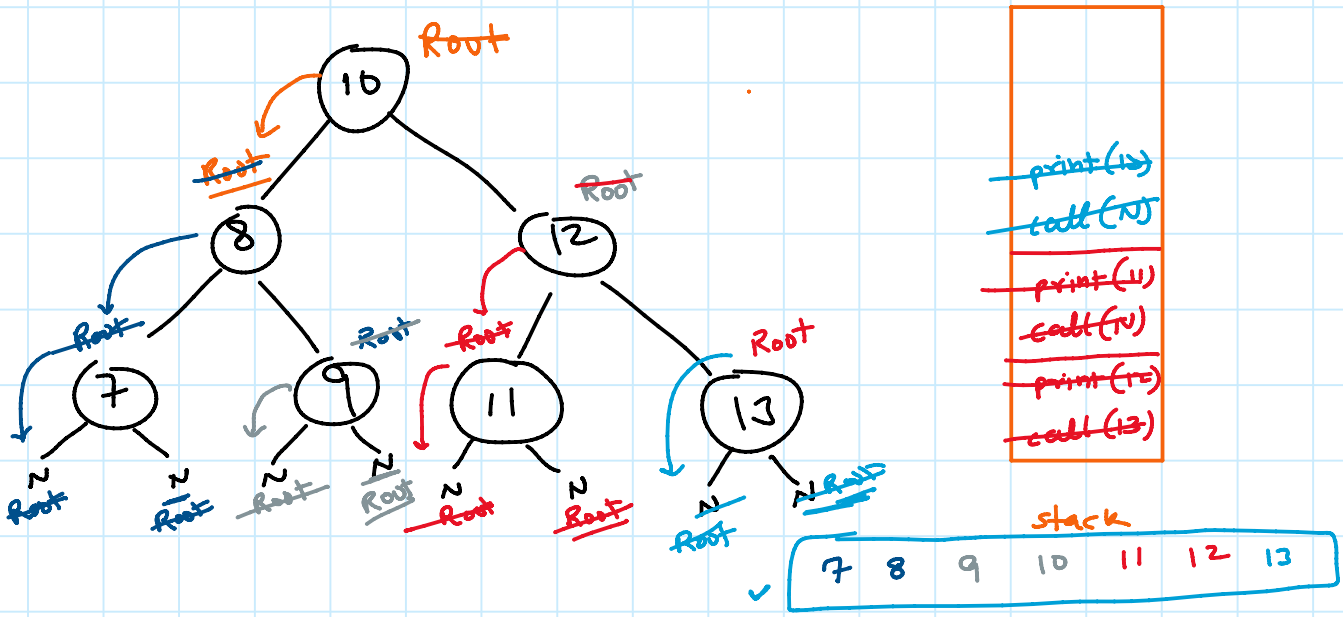
14 May 2024 01:03 PM

Tree :-

LST Root RST

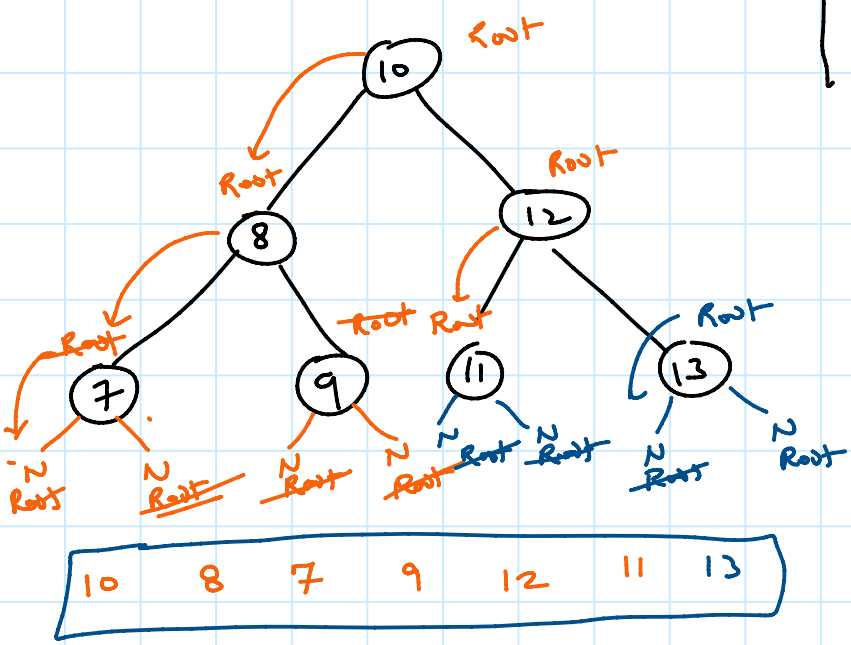
BST

```
void Inorder(root)
{
    if (root == nullptr)
        return;
    Inorder(root->left)
    print(root->val)
    Inorder(root->right)
}
```



Pre Order Traverse → (Root, LST, RST)

```
void Pre order (root)
{
    if (root == nullptr)
        return;
    print (root->val)
```

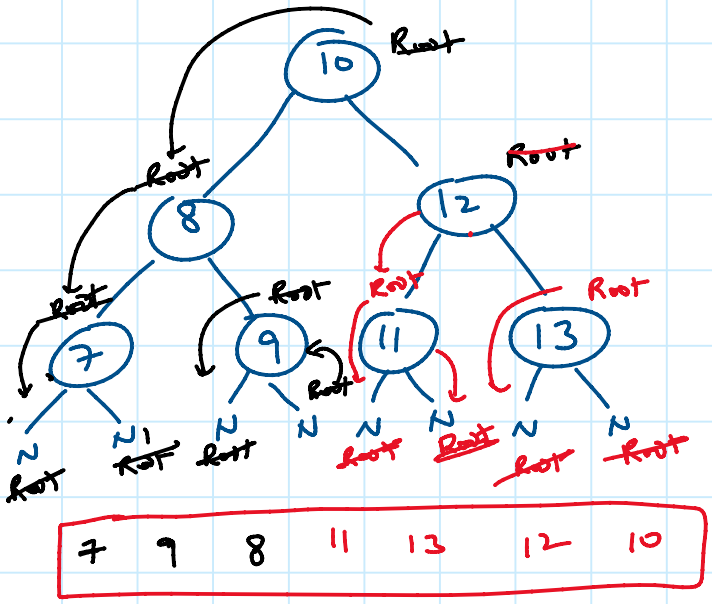



```
        return;
    print (root → val)
    preorder (root → left)
    preorder (root → right)
}
```

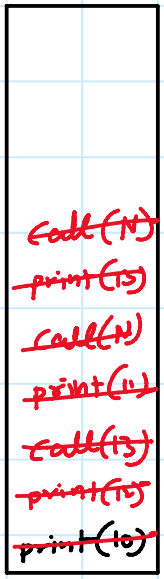
stack

Post Order → (LST, RST, Root)

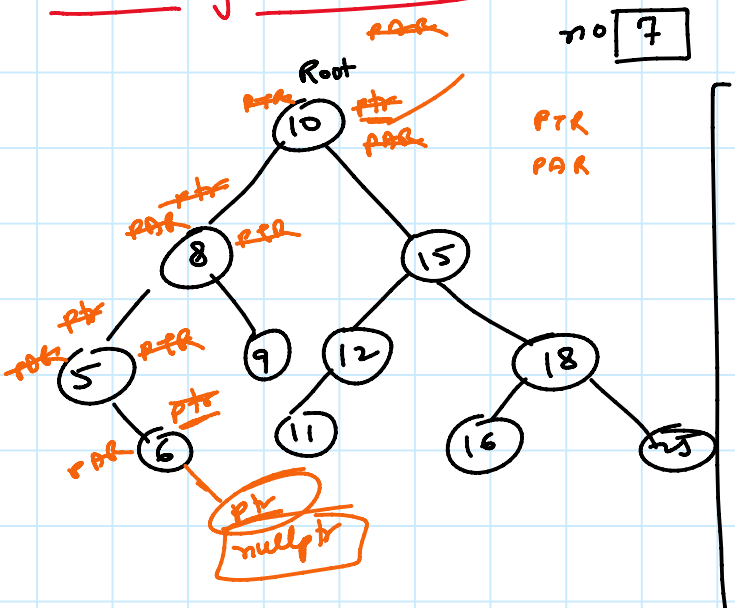
```
void postorder (Root)
{
    if (Root == nullptr)
        return;
    postorder (Root → left);
    postorder (Root → right);
    print (root → val);
}
```



7 9 8 11 13 12 10



Searching in BST :-

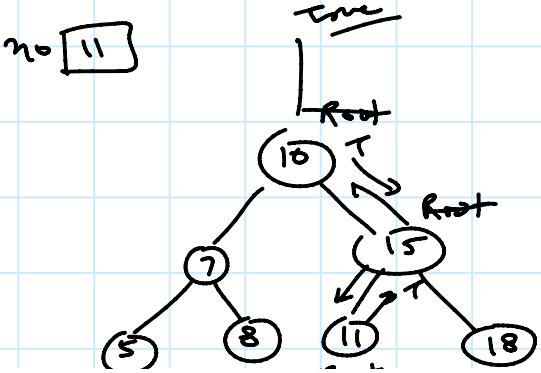


Non Recursive method

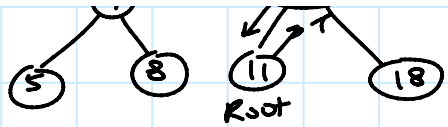
```
bool Search (Root , no)
PAR = nullptr
{
    node * ptr = root
    while (ptr != nullptr)
    {
        if (ptr -> val == no)
            return true;
        PAR = ptr
        if (no < ptr -> val)
            ptr = ptr -> left;
        else
            ptr = ptr -> right
    }
    return false;
}
```

Recursive method

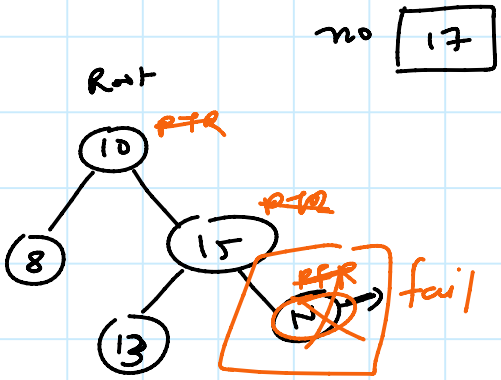
```
bool Search (root, no)
{
    if (root == nullptr)
        return false;
```

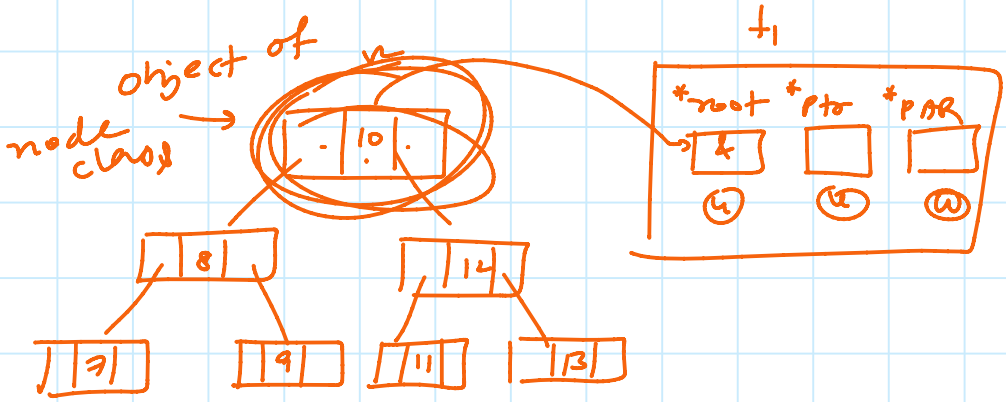


```
        return false;
if (root -> val == no)
        return true;
if (no < root -> val)
  return search (root -> left, no)
else
  return search (root -> right, no)
}
```

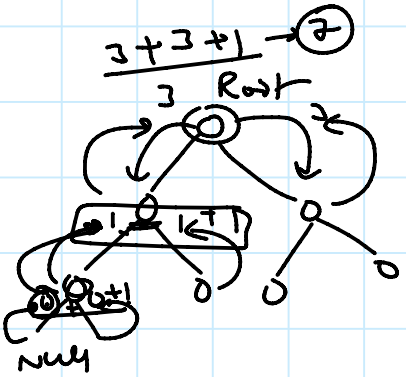


Add Bst :-





Count No of nodes in Bst :-



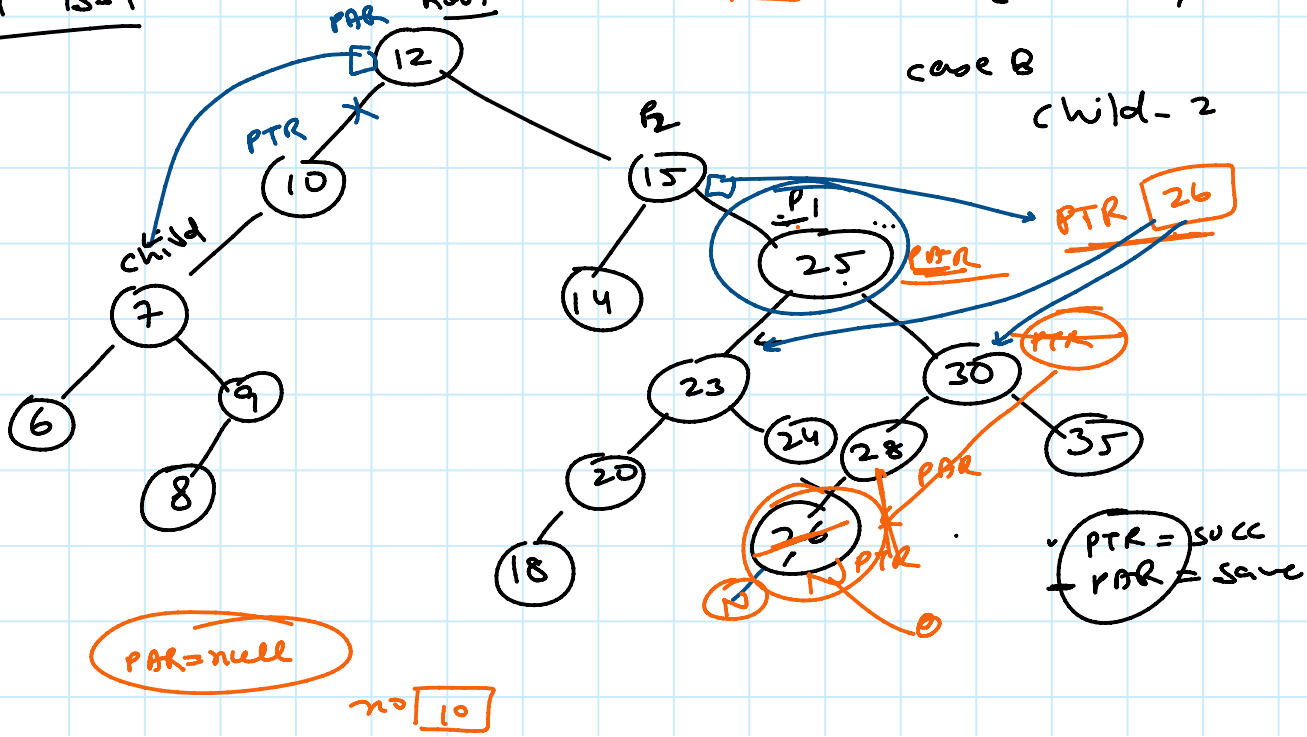

```
int Count (root)
{
    if (root == nullptr)
        return 0
```

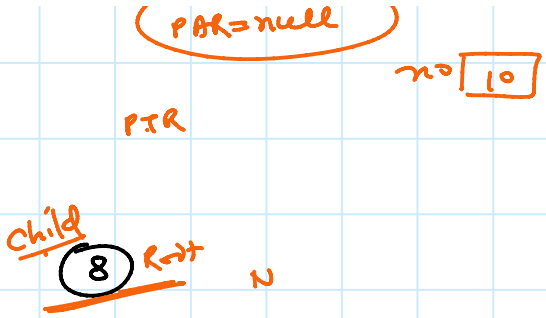
```
    return 0
return Count(root->left) + Count(root->right) + 1;
}
```

height of tree :-

```
int height (root)
{
    if (root == nullptr)
        return 0;
return max (height (root->left) , height (root->right)) +1
}
```

Del BST





.

```
#include<iostream>
using namespace std;
class node
{
    public:
        int val;
        node *left, *right;
        node(int no)
        {
            val=no;
            right = left = nullptr;
        }
};
class BST
{
    node *root;
    node *ptr;
    node *par;
    public:
        BST()
        {
            root=nullptr;
        }
        bool search(int no)
        {
            ptr=root;
            par=nullptr;
            while (ptr != nullptr)
            {
                if(ptr->val == no)
                    return true;
                par = ptr;
```

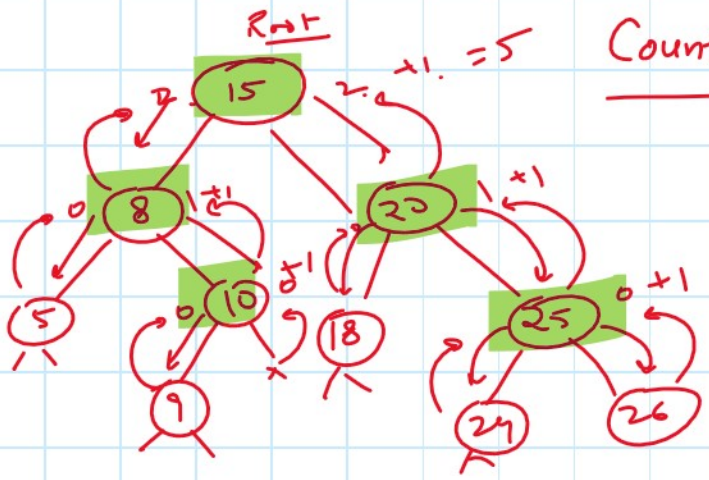
```
            if(no < ptr->val)
                ptr=ptr->left;
            else
                ptr = ptr->right;
        }
        return false;
    }
    void addBST(int no)
    {
        if(search(no))
        {
            cout<<no<<" is Dulpicate element\n";
            return;
        }
        if(par==nullptr)
        {
            root = new node(no);
        }
        else
        {
            if(no < par->val)
                par->left = new node(no);
            else
                par->right=new node(no);
        }
    }
private:
    void inorder(node *root)
    {
        if(root == nullptr)
            return;
        inorder(root->left);
        cout<<root->val<<" ";
        inorder(root->right);
    }
    void preorder(node *root)
    {
        if(root == nullptr)
            return;
        cout<<root->val<<" ";
        preorder(root->left);
        preorder(root->right);
    }
    void postorder(node *root)
```

```
        {
            if(root == nullptr)
                return;
            postorder(root->left);
            postorder(root->right);
            cout<<root->val<<" ";
        }
        int Count(node *root)
        {
            if(root == nullptr)
                return 0;
            return Count(root->left) + Count(root->right) +
1;
        }
        int Height(node *root)
        {
            if(root == nullptr)
                return 0;
            return max(Height(root->left),Height(root->
right)) + 1;
        }
    public:
        void traverse()
        {
            cout<<"Inorder ";
            inorder(root);
            cout<<endl;
            cout<<"post Order ";
            postorder(root);
            cout<<endl;
            cout<<"Pre Order ";
            preorder(root);
            cout<<endl;
        }
        int CountNodes()
        {
            return Count(root);
        }
        int HeightOfTree()
        {
            return Height(root);
        }
    private:
        void caseA()
```

```
{
    node *child;
    if(ptr->left != nullptr)
        child=ptr->left;
    else if(ptr->right != nullptr)
        child = ptr->right;
    else
        child=nullptr;
    if(par ==  nullptr)
    {
        root=child;
    }
    else
    {
        if(ptr == par->left)
            par->left = child;
        else
            par->right = child;
    }
}
void caseB()
{
    node *p1=ptr,*p2=par;
    par=p1;
    ptr=p1->right;
    while(ptr->left != nullptr)
    {
        par=ptr;
        ptr=ptr->left;
    }
    caseA();
    if(p2 == nullptr)
    {
        root=ptr;
    }
    else
    {
        if(p1=p2->left)
            p2->left = ptr;
        else
            p2->right=ptr;
    }
    ptr->left = p1->left;
    ptr->right = p1->right;
```

```
                ptr=p1;
            }
        public:
            void delBST(int no)
            {
                if(!search(no))
                {
                    cout<<"Element Not Found\n";
                    return;
                }
                if(ptr->left !=nullptr and ptr->right != nullptr)
                {
                    caseB();
                }
                else
                {
                    caseA();
                }
                cout<<ptr->val<<" deleted\n";
                delete ptr;
                ptr=nullptr;
            }
    };
    int main()
    {
        BST t1;
        t1.addBST(10);
        t1.addBST(8);
        t1.addBST(12);
        t1.addBST(7);
        t1.addBST(9);
        t1.addBST(11);
        t1.addBST(13);
        t1.addBST(8);
        t1.traverse();
        if(t1.search(70))
            cout<<"Found\n";
        else
            cout<<"Not found\n";
        cout<<t1.CountNodes()<<endl;
        cout<<"Height = "<<t1.HeightOfTree()<<endl;
        t1.delBST(7);
        t1.traverse();
        t1.delBST(10);
```

```
        t1.traverse();
    }
```



Count all the internal nodes ?