DSA Lab
Name: Siddhi Parekh
Reg No.: 221071047
Batch: C
SY Comps

Experiment No.: 7

AIM:

To learn what a binary search tree(BST) is and to perform insertion and checking if a binary tree is a BST or not. To do preorder traversal using an iterative way using stack. To do inorder traversal recursively

THEORY:

Definition :

i) It is a binary tree i.e each parent node can have maximally two children nodes.

ii) At each node, the left child data is less than the parent node data and right child data is greater than the parent node data.

Note : A printing of inorder traversal for BST results in a sorted (ascending) printing of data.

iii) Preorder traversal is defined as a type of tree traversal that follows the Root-Left-Right policy where:

The root node of the subtree is visited first.

- Then the left subtree is traversed.
- At last, the right subtree is traversed.

iv)Inorder traversal is defined as a type of tree traversal technique which

follows the Left-Root-Right pattern, such that:

The left subtree is traversed first

- Then the root node for that subtree is traversed
- Finally, the right subtree is traversed

ALGORITHM:

i) To do preorder traversal using stack

1) Initialize a stack and push the root node.
2) Enter the while loop having condition !st.isEmpty()
3) Pop the node, print the data and then push the right child of the temporary node and then push the left child of the temporary node respectively.

Example : Consider a BST,

root = 20
root->left = 10
root->left->left = 5
root->left->right = 15
root->right = 30
root->right->left = 25
root->right->right = 35

i) We will start from the root, and push the root into the stack.
ii) Popping the top element from stack i.e 20 and pushing the root and left child respectively. Stack={30,10}
iii) Again popping the stack top element i.e 10, print and push the right and left child respectively. Stack={30,15,5}
iv) Print 5 , Stack = {30,15}
v) Print 15 , stack = {30}
vi) Print 30, Stack = {35,25}

vii) Print 25, Stack = {35}
viii) Print 35, Stack = {}
Since, stack has now become empty, therefore exit this while loop.

ii) To do inorder traversal recursively

a) Traverse the left subtree
b) Perform the action on the current node
c) Traverse the right subtree

iii) To make a bst

a)Create a structure that contains value, address of left and right child of the Tree node.

```
b)create(q)
    node*r,*p
    r=getnode()
    if(root==NULL)
       root = r
    else
      p=root
      while(1)
          a=p
          if(a->data<q)  traverse in right subtree and insert at suitable position.
          else                traverse in left subtree and insert at suitable position
```

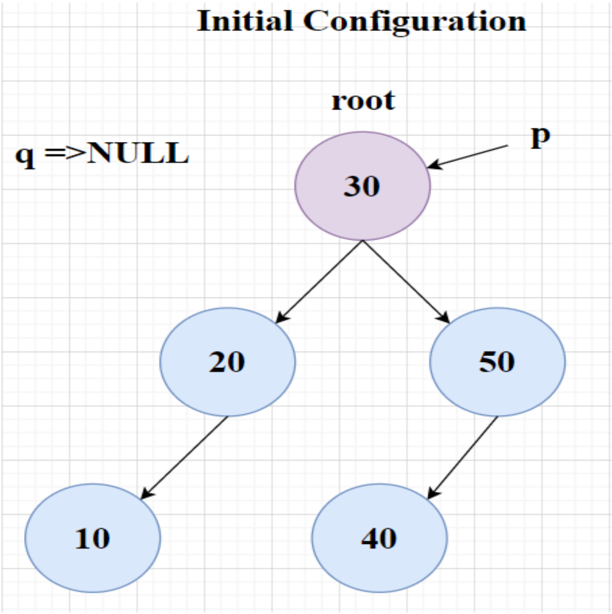
iv) Validation of BST

a)Find the max of all the nodes in the left subtree, and check if the max is less than the current node.
b)Find the min of all the nodes in the right subtree, and check if the min is greater than the current node.
c)Validate if both left and right subtrees satisfy the bst conditions.

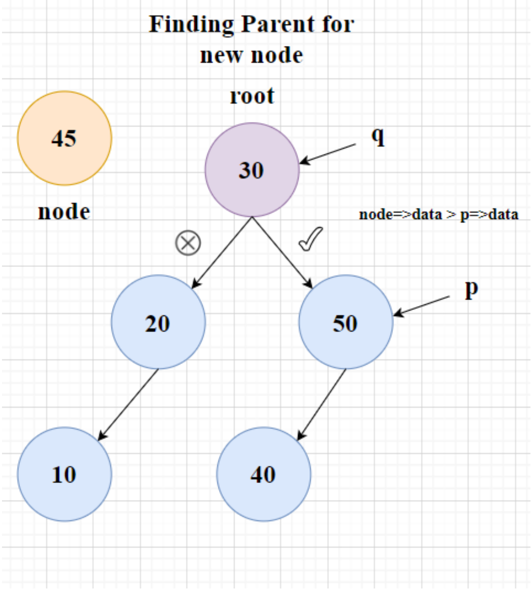
EXAMPLE(I):

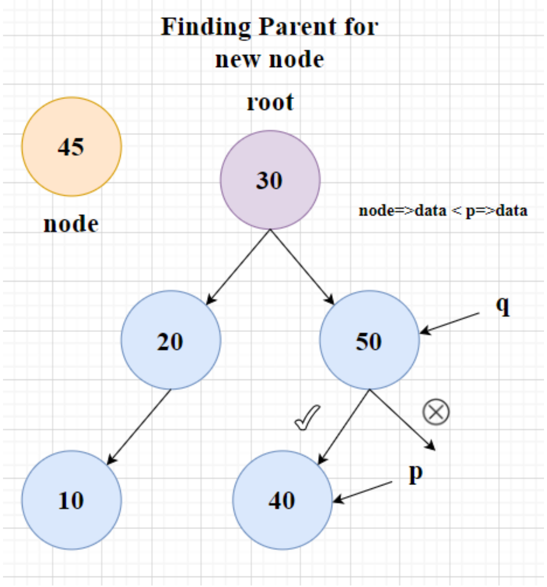
i) Iterative way: We want to add a node with data = 45.

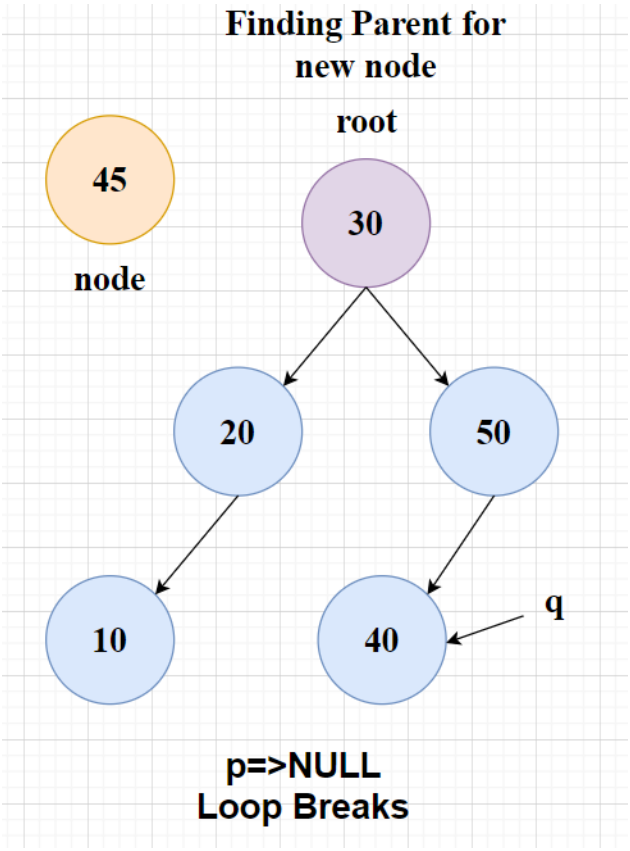
Step 1 : Initialization



Step 2 : Looking for the parent node for the given input data







Step 3 : Looking for its position according to data of its parent node.
Since the node data = 45 and it is greater than q->data i.e 40.
Therefore it will become a right child node 40.

Note : When the first node is added to a BST then we directly add the node to our bst and return from there.

Example(II) :

1)

```
// Construct a sample BST
root->value =2
root->left = 1
root->right =3

// Check if it's a BST
result = isBST(root)
```

```
print(result)  // Output: true
```

2)

```
// Construct a sample BST
root->value =2
root->left = 3
root->right =1

// Check if it's a BST
result = isBST(root)
print(result)  // Output: false
```

Example : Consider a bst as given in the above example.
Therefore, printing will occur in the following fashion 5,10,15,20,25,30,35.

CONCLUSION:

Thus we have studied how a BST is created, the manner in which preorder traversal is performed (i.e iterative approach), and how to validate whether the tree given is BST or not.

CODE:

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


struct node
{
        int data;
        struct node *left, *right;
};
struct node *root=NULL;
```

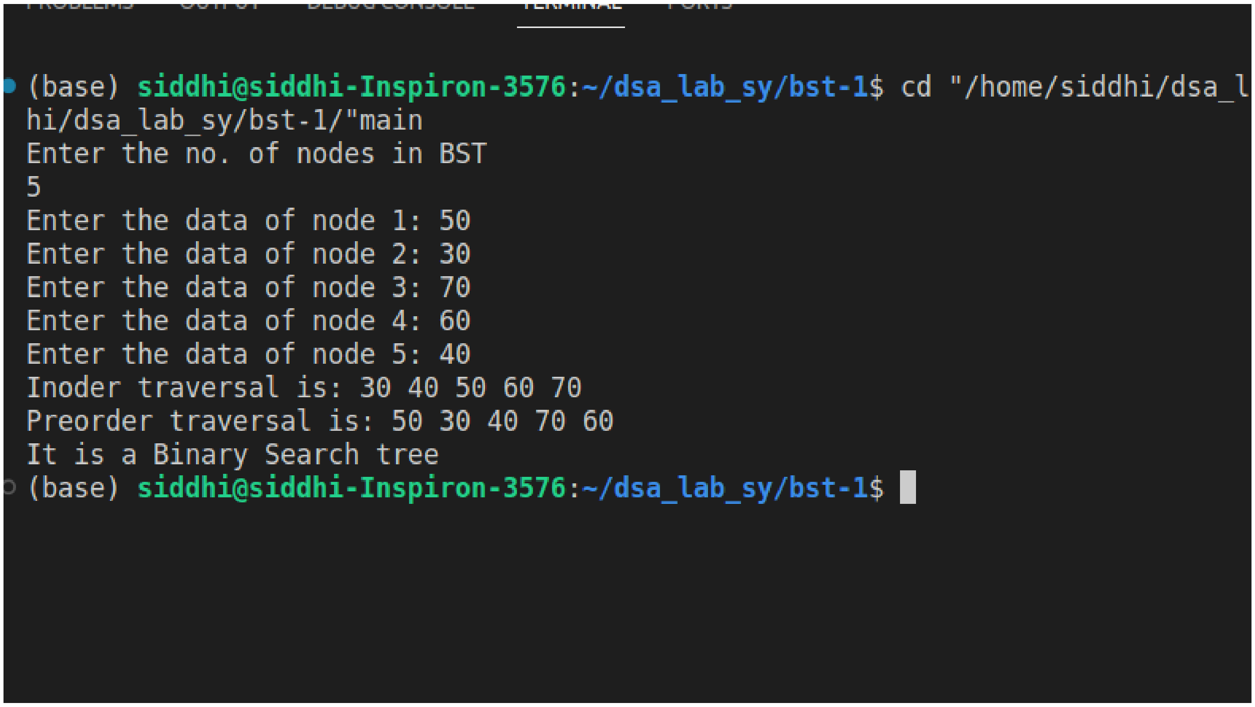
```
void create(int n)
{
        for(int i=0; i<n; i++){
        int q;
        cout<<"Enter the data of node "<<i+1<<": ";
        cin>>q;
        node *r=(struct node *)malloc(sizeof(struct node));
        r->data=q;
        r->right=NULL;
        r->left=NULL;
        node *p,*a;

        if(root==NULL)
        {
        root=r;
        //save=root;
        }



        else
        {
        p=root;

        while(true)
        {
        a=p;

        if(a->data<q)
        {
                p=p->right;

                if(p==NULL)
                {
                a->right=r;
                //return;
                break;
                }
        }

        else
        {
                p=p->left;

                if(p==NULL)
                {
                a->left=r;
                //return;
                break;
                }
        }
        }

        }


        }
}

void inorder(struct node* m)
{
        if(m!=NULL){
        inorder(m->left);
        cout<<m->data<<" ";
        inorder(m->right);
        }

        //cout<<endl;
}

void preorder(struct node *p)
{
        stack<node*>st;
```

```
        //node *p = (struct node *)malloc(sizeof(struct node));
        node *a=NULL;
        p=root;
        //a=p;
        //st.push(p->data);

        cout<<"Preorder traversal is: ";

        while(!st.empty() || p!=NULL)
        {

        if(p!=NULL)
        {
        cout<<p->data<<" ";
        st.push(p);
        p=p->left;
        }

        else
        {
        a=st.top();
        st.pop();
        p=a->right;
        }

        }
}

// void search(int key)
// {
//      node *p=(struct node *)malloc(sizeof(struct node));
//      p=root;

//      if(root==NULL)
//      {
//      cout<<"\nTree is empty"<<endl;
//      }
```

```
//	else{
//	while(p!=NULL)
//	{
//	if(p->data<key)
//	{
//		p=p->right;

//		if(p==NULL)
//		{
//		cout<<"\nNo such value in the tree"<<endl;
//		}
//	}

//	else if(p->data>key)
//	{
//		p=p->left;

//		if(p==NULL)
//		{
//		cout<<"\nNo such value in the tree"<<endl;
//		}
//	}

//	else if(p->data==key)
//	{
//		cout<<"\nValue exists"<<endl;
//		break;
//	}
//	}

//	//return p;
//	}

//	//return p;
// }
```

```
bool isValidBST1(struct node* root, struct node* minNode, struct node*
maxNode) {
        if(!root) return true;
        if(minNode && root->data <= minNode->data || maxNode && root->data >=
maxNode->data)
        return false;
        return isValidBST1(root->left, minNode, root) && isValidBST1(root->right,
root, maxNode);
}

bool isValidBST(struct node* root) {
        return isValidBST1(root, NULL, NULL);
}

int main()
{
        // create(10);
        // create(20);
        // create(5);
        int size;
        cout<<"Enter the no. of nodes in BST"<<endl;
        cin>>size;
        create(size);
        cout<<"Inoder traversal is: ";
        inorder(root);
        //search (10);
        cout<<endl;
        preorder(root);
        bool is_bst=isValidBST(root);
        if(is_bst){
        cout<<"\nIt is a Binary Search tree"<<endl;
        }
        else{
        cout<<"\nIt is not a Binary Search tree"<<endl;
        }
        return 0;
}
```

OUTPUT:



```
(base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/bst-1$ cd "/home/siddhi/dsa_l
hi/dsa_lab_sy/bst-1/"main
Enter the no. of nodes in BST
5
Enter the data of node 1: 50
Enter the data of node 2: 30
Enter the data of node 3: 70
Enter the data of node 4: 60
Enter the data of node 5: 40
Inoder traversal is: 30 40 50 60 70
Preorder traversal is: 50 30 40 70 60
It is a Binary Search tree
(base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/bst-1$
```