Name-Harshita Totala Roll No-SEITA14 Div-IT A

ASSIGNMENT-6
Threaded Binary Tree

AIM : Implement In-order Threaded Binary Tree and traverse it in Inorder and Pre-order.

Objectives :

1) To understand the concept of Threaded Binary Tree as a data structure.
2) Applications of Threaded Binary Tree.

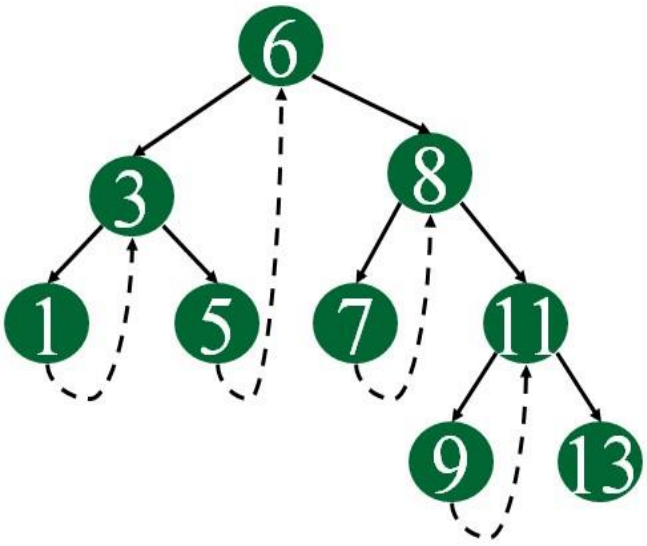
Theory :

Definition of Threaded Binary Tree :

A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node**.**

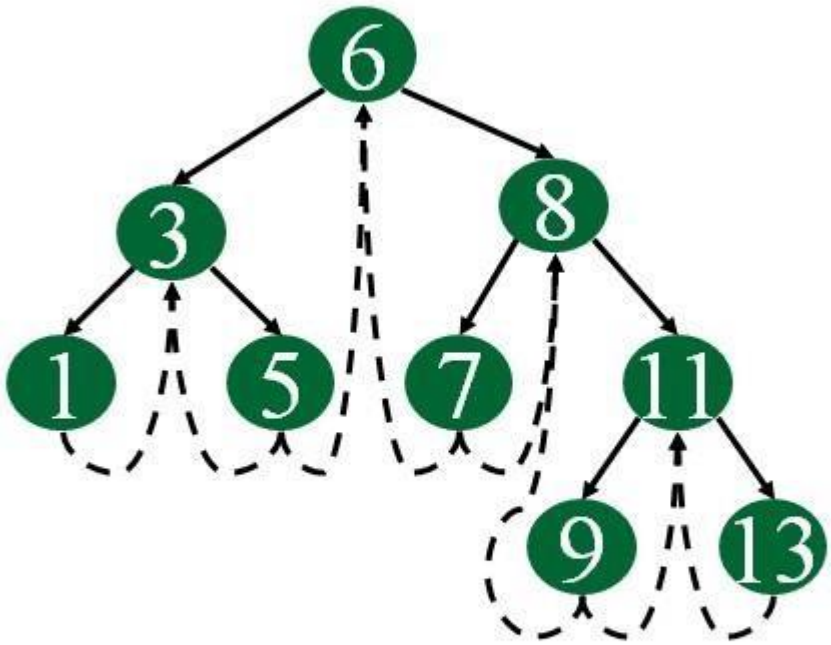
Types of threaded binary trees:

- **Single Threaded**: each node is threaded towards either the in-order predecessor or successor (left **or**right) means all right null pointers will point to inorder successor **OR** all left null pointers will point to inorder predecessor.



Inorder: 1 3 5 6 7 8 9 11 13

- **Double threaded:** each node is threaded towards both the in-order predecessor and successor (left **and**right) means all right null pointers will point to inorder successor **AND** all left null pointers will point to inorder predecessor.



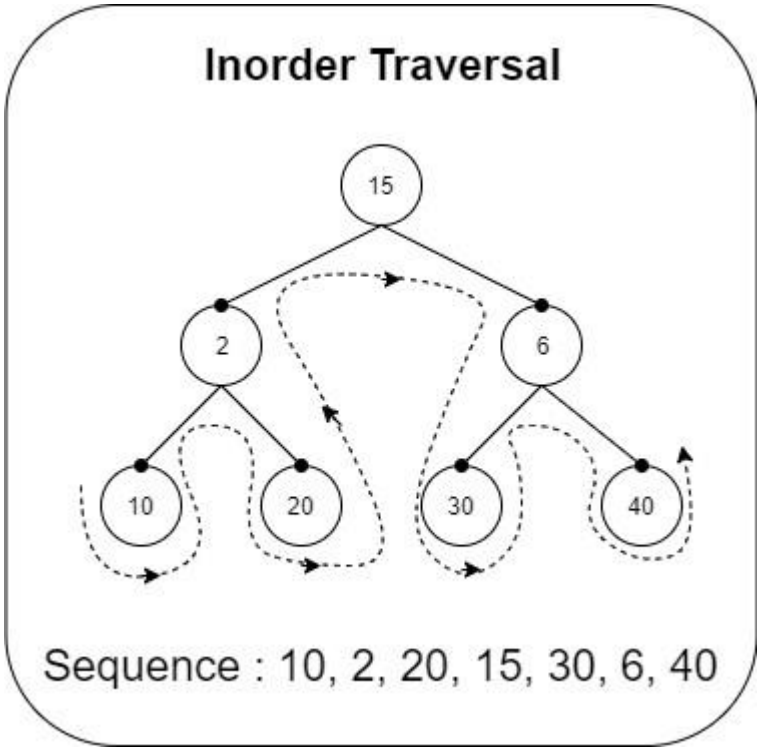
Double Threaded Binary Tree

Inorder: 1 3 5 6 7 8 9 11 13

In-order Traverse :

Name-Harshita Totala Roll No-SEITA14 Div-IT A

Inorder Traversal follows the order LNR (left, node, right).



Steps →

1. Create a stack **inorder** to fit elements of type TreeNode
2. Run a while loop until there are no more elements to process, i.e. the **root** is NULL and the stack is empty
3. In each iteration →

- Keep moving to the left child until it exists and push the elements onto the stack so that we can process them after we process the leftmost element
- Once we reach the left-most element of current subtree, print it and remove it from the stack.
- What's the next priority? Either the right subtree or current node or its parent.

- So we move to its right child if it exists. If it doesn't the parent node is already in the stack. Pop it and process it

Pseudo Code :

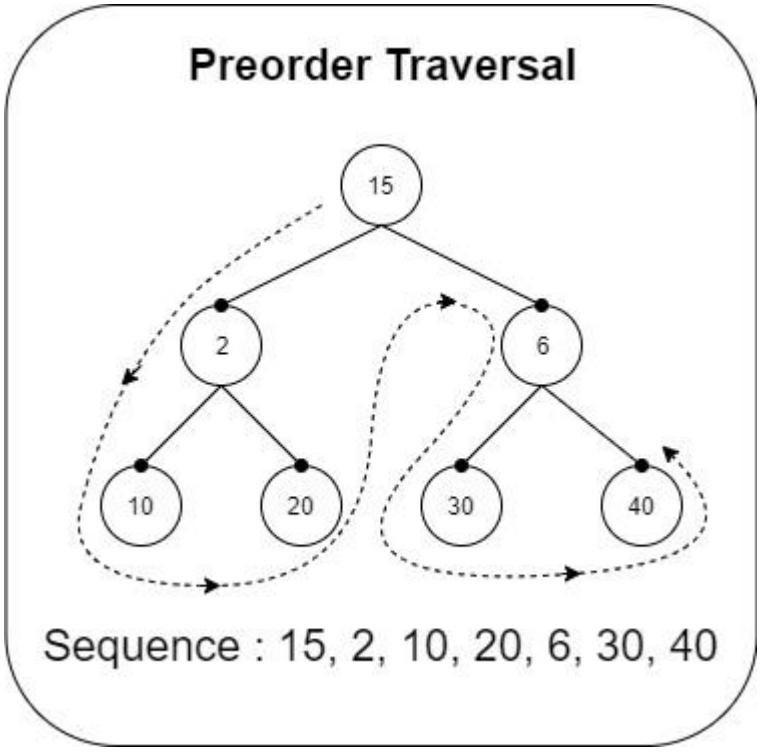
```
     void printInorder(TreeNode root)
{
    Create stack 'inorder' to fit elements of type TreeNode
while ( root != NULL or inorder.empty() != True )
    {
        // Find the leftmost node
while( root != NULL )
        {
            inorder.push(root)
root = root.left
        }

        root = inorder.peek()
        inorder.pop()
        print( root )
        root = root.right
    }                          }
```

Name-Harshita Totala Roll No-SEITA14 Div-IT A

Pre-order Traverse :

Preorder Traversal as we read above, traverse the tree in the order: Node, left child and then the right child.



Steps →

1. Create a stack **preorder** that will temporarily store the elements like the *function stack* in the above recursion.

2. Since priority is given to the root, push the root first to the stack.
3. Now we will run a loop until the stack empties, i.e., there are no more elements left to process.
4. In each iteration, print the node (i.e. the element at top of the stack) then push the right child in the stack and then the left child.

Pseudo Code :

```
     void printPreorder(TreeNode root)

{
    Create stack 'preorder' to fit elements of type TreeNode
preorder.push( root )
    while ( preorder.empty() != True )
    {
        TreeNode node = preorder.peek()
preorder.pop()          if ( node ==
NULL )              continue

        print ( node.val )
        // Note that right child is pushed first
// so that left child is on top
preorder.push( node.right )          preorder.push(
node.left )
    }  }
```

Name-Harshita Totala Roll No-SEITA14 Div-IT A

CODE-

```
#include<bits/stdc++.h>
using namespace std;
class Node{
public:
    int data;
    Node* left;
    Node* right;
    int leftThread; // leftThread=0 -> left pointer points to the inorder
predecessor
    int rightThread; // rightThread=0 -> right pointer points to the inorder
successor
    Node(int val){
        this->data = val;
    }
};
class DoubleThreadedBinaryTree{
private:
    Node* root;
public:
    DoubleThreadedBinaryTree(){
        // dummy Node with value as INT_MAX
        root = new Node(INT_MAX);
        root->left = root->right = root;
        root->leftThread = 0;
        root->rightThread = 1;
    }
    void insert(int data){
        Node* new_node = new Node(data);
        if(root->left == root && root->right == root){
            //Empty Tree
            new_node->left = root;
            root->left = new_node;
            new_node->leftThread = 0;
            new_node->rightThread = 0;
            root->leftThread = 1;
            new_node->right = root;
            return;
        }
        else{
            Node* current = root->left;
```

```
            while(true){
                if(current->data > data){
                    if(current->leftThread == 0 ){
                     // this is the last Node
                     new_node->left = current->left;
                     current->left = new_node;
                     new_node->leftThread = current->leftThread;
                     new_node->rightThread = 0;
                     current->leftThread = 1;
                     new_node->right = current;
                     break;
                    }
                    else{
                        current = current->left;
                    }
                }
                else{
                    if(current->rightThread == 0){
                        // this is the last Node
                        new_node->right = current->right;
                        current->right = new_node;
                        new_node->rightThread = current->rightThread;
                        new_node->leftThread = 0;
                        current->rightThread=1;
                        new_node->left = current;
                        break;
                    }
                    else{
                        current = current->right;
                    }
                }
            }
        }
    }
    Node* findNextInorder(Node* current){
        if(current->rightThread == 0){
            return current->right;
        }
        current = current->right;
        while (current->leftThread != 0)
        {
            current = current->left;
        }
        return current;
    }
```

```
    void inorder(){
        Node* current = root->left;
        while(current->leftThread == 1){
            current = current->left;
        }
        while(current != root){
            cout<<current->data<<" ";
            current = findNextInorder(current);
        }
        cout<<"\n";
    }
    void preorder(){
        Node* current = root->left;
        while(current != root){
            cout<<current->data<<" ";
            if(current->left != root && current->leftThread != 0)
                current= current->left;
            else if(current->rightThread == 1){
                current = current->right;
            }
            else{
                while (current->right != root && current->rightThread == 0)
                {
                    current = current->right;
                }
                if(current->right == root)
                    break;
                else
                {
                    current=current->right;
                }
            }
        }
        cout<<"\n";
    }


};

int main(){
    DoubleThreadedBinaryTree dtbt;
    dtbt.insert(10);
    dtbt.insert(45);
    dtbt.insert(1);
    dtbt.insert(7);
```

```
    dtbt.insert(76);
    dtbt.insert(34);
    dtbt.insert(61);
    dtbt.insert(100);
    cout<<"Inorder -"<<endl;
    dtbt.inorder();
    cout<<endl;
    cout<<"Preorder- "<<endl;
    dtbt.preorder();
   ;
    return 0;
}
```

OUTPUT-

Inorder -

1 7 10 34 45 61 76 100

Preorder-

10 1 7 45 34 76 61 100

Conclusion:

Thus we have implemented In-order Threaded Binary Tree and traversed it in In-order and Pre-order.