**Assignment No :** 06

**Title:** Threaded Binary Tree Data Structure

**Course Outcomes:**

**CO1(**C214447.1**):** Analyze algorithms and to determine algorithm correctness and time efficiency class.

**CO2(**C214447.2**):** Implement abstract data type (ADT) and data structures for given application.

**CO3(**C214447.3**):**Design algorithms based on techniques like brute -force, divide and conquer, greedy, etc.).

**CO5(**C214447.5**):** Analyze of algorithms with respect to time and space complexity.

**Assessment Grade / Marks:**

**Assessor’s Sign with Date:**

**Title:** Threaded Binary Tree Data Structure

**Aim:** To Implement Threaded Binary Tree as an ADT **Problem Statement:**

Implement In-order Threaded Binary Tree 

Traverse it in

* In-order
* Pre-order

**Requirements:** Visual studio code

# Theory / Procedure / Diagram / Circuits:

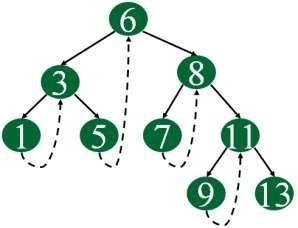
Threaded Binary Tree (TBT):

* 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 pointer to the inorder predecessor of the node (if it exists).
* It is also possible to discover the parent of a node from a threaded binary tree, without explicit use of parent pointer or a stack.
* This can be useful where stack space is limited, or where a stack of parent pointers is unavailable (for finding the parent pointer via DFS)  To see how this is possible, consider a node k that has a right child r.
* Then the left pointer of r must be either a child or a thread back to k.
* In that case that r has a left child, that left child must in turn have either a left child of its own or a thread back to k, and so on for all successive left children.
* So by following the chain of left pointers from r, we will eventually find a thread pointing back to k.
* The situation is symmetrically similar when q is the left child of p we can follow q’s right children to a thread pointing ahead p.

Types of Threaded Binary Tree:

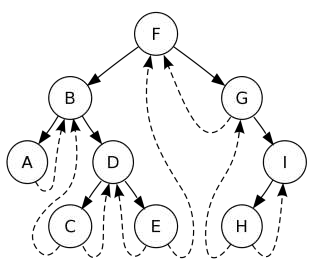
1. Single Threaded:

Each node is threaded towards either the in-order predecessor or successor (left or right).



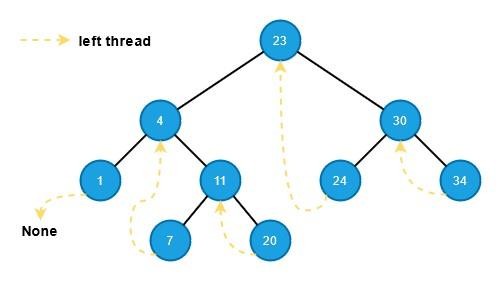
1. Double Threaded:

Each node is threaded towards both the in-order predecessor and successor (left and right)



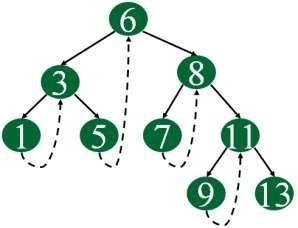
* Left In-Threaded Binary Tree:

If the left pointer of the leaf node points to its in-order predecessor, then such a threaded tree is called left In-Threaded Binary Tree. Ex.

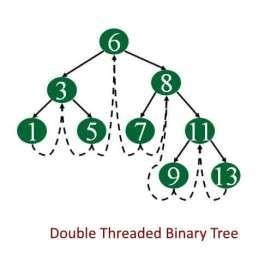


* Right In-Threaded Binary Tree:

If the right pointer of the leaf node points to its in-order successor, then such a threaded tree is referred to as Right In-Threaded Binary Tree. Ex.



* Full In-Threaded Binary Tree:

If both the links, left as well as right are available, then the threaded tree is called full In-Threaded Binary Tree. Ex.

# Algorithms / Methods / Steps:

1. **Algorithm for insert operation on Threaded Binary Tree:** **Case 1: When a new node is inserted in an empty tree**

The new node value becomes the root node, and both the left and right pointers of the value will be set to NULL.

# Case 2: When a new node is inserted as a left child

After inserting the node at its proper place, we will make its left and right child pointers point to in-order predecessor and successor, respectively. So the left and right threads of the new node will be:

value -> left = parent ->left; value -> right = parent;

Before insertion, the left pointer of the parent was a thread, but after insertion, it will be a link pointing to the new node.

parent -> leftThread = false; parent -> left = value;

# Case 3: When a new node is inserted as a right child

The node that was the parent’s in-order successor is now the in-order successor of this new node value. So the left and right threads of the new node will be:

value -> left = parent;

value -> right = parent -> right;

Before insertion, the right pointer of the parent was a thread, but after insertion, it will be a link pointing to the new node.

parent -> rightThread = false; parent

-> right = value;

# Algorithm for delete operation on Threaded Binary Tree: Case 1: When a leaf node needs to be deleted

When deleting a leaf Node in BST, the left or right pointer of the parent node is set to NULL. Whereas in Threaded binary search trees, it is turned into a thread instead of setting the pointer to NULL.

If the leaf Node is the left child of its parent, then after deletion, the parent’s left pointer should become a thread referring to its predecessor.

# Case 2: When the node to be deleted has only one child

After deleting the Node like in a BST the inorder successor and predecessor of the Node are found.

s = inSucc(current); p

= inPred(current); If Node to be deleted has a left subtree, then after deletion,

the right thread of its predecessor should point to its successor.

# Case 3: When the node to be deleted has two children

We find the in-order successor of the current Node (Node to be deleted) and then copy the information of this successor into the current Node. After this, the in-order successor Node is deleted using either Case 1 or Case 2.

# Algorithm for in-Order Traversal on Threaded Binary Tree:

Step 1: Start

Step 2: traverse till leftmost node in tree i.e. smallest node in the tree. Step 3: loop till we reach header node

Step 3.1: display node

Step 3.2: traverse to its right node as it will be its successor Step 4: as we reached the header node the traversal is complete.

Step 5: End

# Algorithm for in-Order Traversal on Threaded Binary Tree:

Step 1: Start

Step 2: make root node as current node Step 3: Until current node is not NULL

Step 3.1: Display current node

Step 3.2: if current node has left child, the make left child as current node.

Step 3.3: Else if current node has right child then make right child as current node.

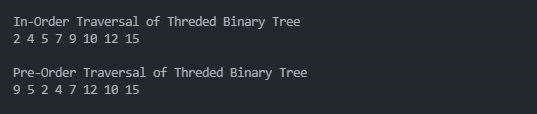
Step 3.4: Else

Until right thread exists for current node Traverse up the right threads and update current if current node = last node STOP.

else make right child of current node as current.

# Input:

Tree is created manually **Output:**



Program For Threaded Binary Tree

1.Create

2.Display

1

Enter The Element 1

Do u want to enter more elements?(y/n)y

Enter The Element 1

Do u want to enter more elements?(y/n)y

Enter The Element 3

Do u want to enter more elements?(y/n)y

Enter The Element 4

Do u want to enter more elements?(y/n)y

Enter The Element 6

Do u want to enter more elements?(y/n)n

Want To See Main Menu?(y/n)y

Program For Threaded Binary Tree

1.Create

2.Display

2

Inorder Traversal... 1 3 4 6

Preorder Traversal... 1 3 4 6

Want To See Main Menu?(y/n)