6. Threaded Binary Tree

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

OBJECTIVE:

1. Understand the concept of Threaded binary tree and binary tree.
2. Understand the different type of traversals (recursive & non-recursive)

THEORY:

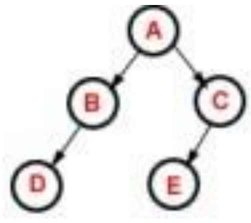
Explain an inorder threaded binary tree.

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

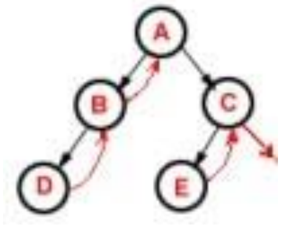
The node structure for a threaded binary tree varies a bit and its like this-

```
struct NODE
{
struct NODE *leftchild;
int node_value;
struct NODE *rightchild;
struct NODE *thread;
}
```

Let's make the Threaded Binary tree out of a normal binary tree...



The INORDER traversal for the above tree is -- D B A E C. So, the respective Threaded Binary tree will be –



B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread.

A **threaded binary tree** defined as follows:

"A binary tree is *threaded* by making all right child pointers that would normally be null point to the inorder successor of the node, and all left child pointers that would normally be null point to the inorder predecessor of the node."[1]

A threaded binary tree makes it possible to traverse the values in the binary tree via a linear traversal that is more rapid than a recursive in-order traversal. It is also possible to discover the parent of a node from a threaded binary tree, without explicit use of parent pointers or a stack, albeit slowly. 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 the 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 to *p*.

Non recursive Inorder traversal for a Threaded Binary Tree

As this is a non-recursive method for traversal, it has to be an iterative procedure; meaning, all the steps for the traversal of a node have to be under a loop so that the same can be applied to all the nodes in the tree. I'll consider the INORDER traversal again. Here, for every node, we'll visit the left sub-tree (if it exists) first (if and only if we haven't visited it earlier); then we visit (i.e print its value, in our case) the node itself and then the right sub-tree (if it exists). If the right sub tree is not there, we check for the threaded link and make the threaded node the current node in consideration.

Show the different type of traversals with example

To traverse a non-empty binary tree in **preorder**, Visit the root.

Traverse the left subtree.

Traverse the right subtree.

To traverse a non-empty binary tree in **inorder**:

Traverse the left subtree. Visit the root.

Traverse the right subtree.

To traverse a non-empty binary tree in **postorder**,

Traverse the left subtree.

Traverse the right subtree.

22Visit the root.

• **Pre-order (prefix)**

+ · 2 3 / 8 4

• **In-order (infix)**

2 · 3 + 8 / 4

• **Post-order (postfix)**

2 3 · 8 4 / +

ALGORITHM:

Step-1: For the current node check whether it has a left child which is not there in the visited list. If it has then go to step-2 or else step-3.

Step-2: Put that left child in the list of visited nodes and make it your current node in consideration. Go to step-6.

Step-3: For the current node check whether it has a right child. If it has then go to step-4 else go to step-5.

Step-4: Make that right child as your current node in consideration. Go to step-6.

Step-5: Check for the threaded node and if its there make it your current node.

Step-6: Go to step-1 if all the nodes are not over otherwise quit.

INPUT: Normal binary tree

.

OUTPUT:

Display result of each operation with error checking.

CONCLUSION: In this way, we have studied implementation of In-order Threaded Binary Tree (TBT) and traversing of it in In-order and Pre-order.

FAQ:

1. What is tree?
2. What are properties of trees?
3. What is Binary tree, Binary search tree, Expression tree & General tree, Threaded Binary Tree?