Assignment No 4:

Convert given binary tree into threaded binary tree. Analyze time and space complexity of an algorithm.

Pre-requisite:

- Knowledge of C++ programming
- Basic knowledge of Threaded Binary Tree, inorder traversal of threaded binary tree
- Conversion of binary search tree into threaded binary tree

Objective:

- To analyze Threaded Binary Tree data structure
- To understand need and advantages of TBT
- To understand practical implementation of TBT

Input:

Node values in integer format.

Outcome:

- A threaded Binary Tree
- Analysis of time and space complexity

Description:

What is 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.

. We have the pointers reference the next node in an inorder traversal; called threads

We need to know if a pointer is an actual link or a thread, so we keep a Boolean for each pointer

Why do we need Threaded Binary Tree?

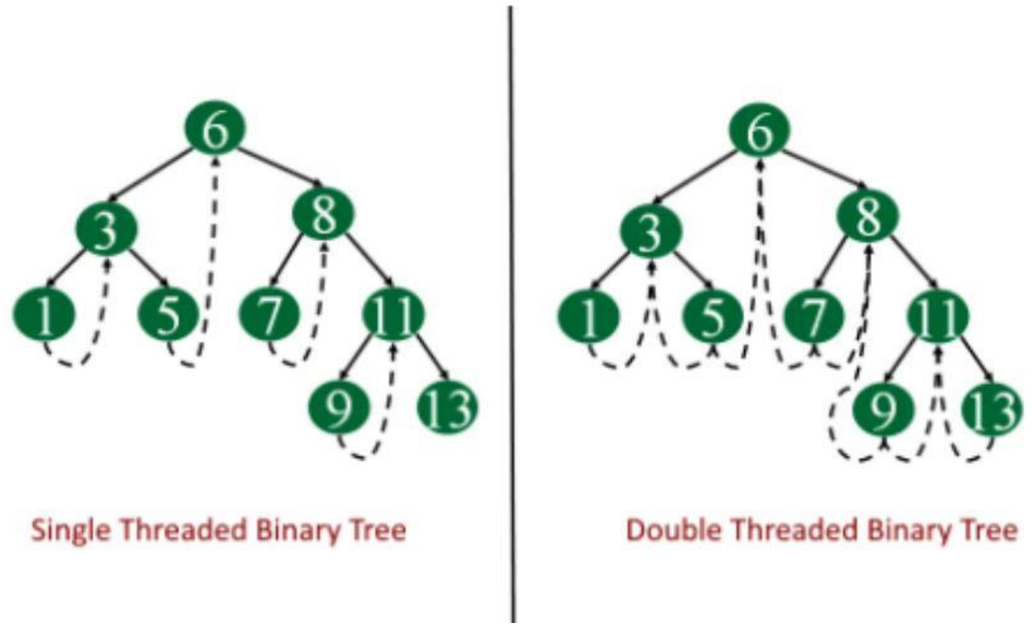
Binary trees have a lot of wasted space, the leaf nodes each have 2 null pointers. We can use these pointers to help us in inorder traversals.

Threaded binary tree makes the tree traversal faster since we do not need stack or recursion for traversal

Types of threaded binary trees:

1) 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.

2) 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.

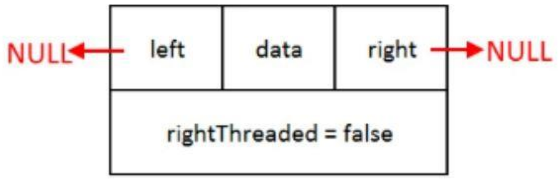


Single Threaded Binary Tree Complete Implementation

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

Implementation:

Let's see how the Node structure will look like



Convert binary tree to threaded binary tree:

Algorithms:

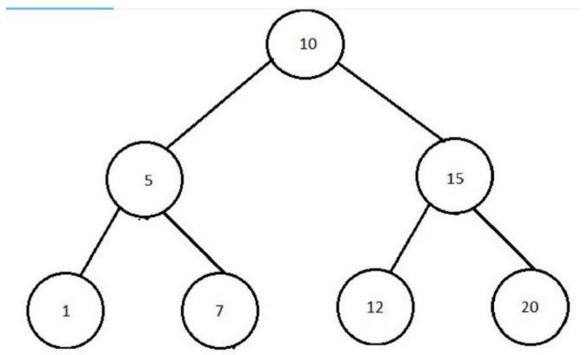
1. Do the reverse inorder traversal, means visit right child first then parent followed by left.

2 In each recursive call also pass the node which you have visited before visiting current node

3 In each recursive call whenever you encounter a node whose right pointer is set to NULL and previous visited node is set to not NULL then make the right pointer of node points to previously visited node and mark the boolean right Thread as true

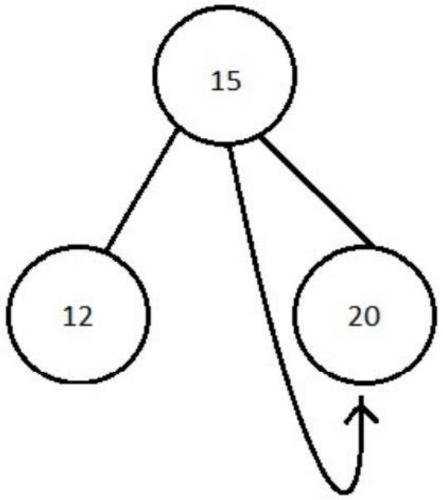
4 Whenever making a new recursive call to right subtree, do not change the previous visited node and when making a recursive call to left subtree then pass the actual previous visited node

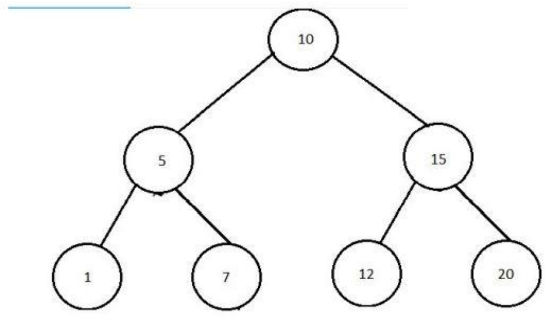
5. Remember that when our right pointer of the current node is pointing to it's children then bool right Thread it set to true and if it is pointing to some of it's ancestor then right thread is set to false.



Step 1 :-
In the given tree we will traverse the tree in reverse inorder traversal which means we will first visit the right subtree then root then followed by the left subtree.

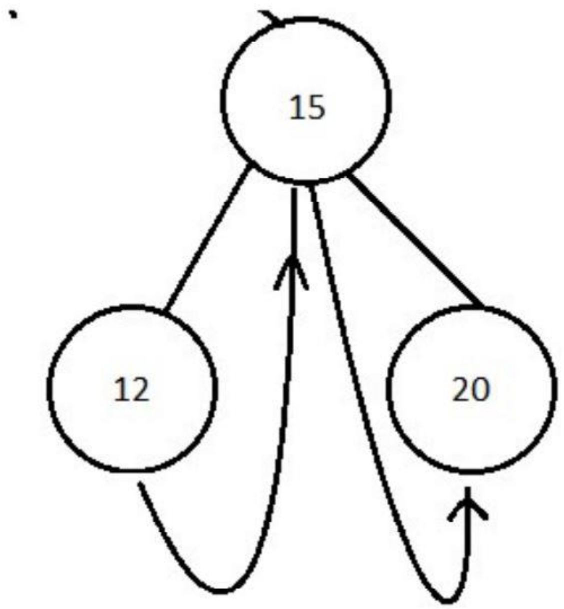
Step 2 :-
As we will follow this algorithm recrusively , so first we will visit the rightmost leaf node 20 , since there is no need which we have visited prior to this node we will make it's rightpointer point to NULL and bool variable as false.

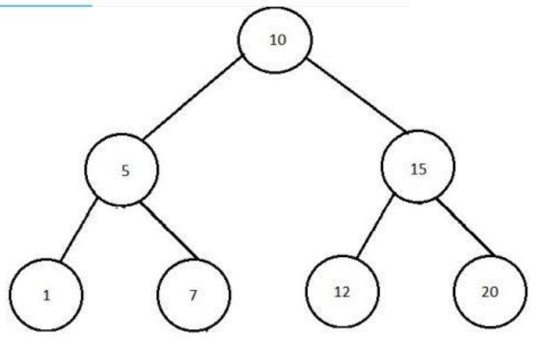




Step 3 :-

Now we will move to root which is node 15 in the given case and since we have already visited node 20 prior to node 15 so we will mark the right pointer of current node (15) to 20 and make bool but currently we are not on the leaf node so we will also make rightThread bool variable as true (indicating it's pointing to it's child node).





step 4 :-

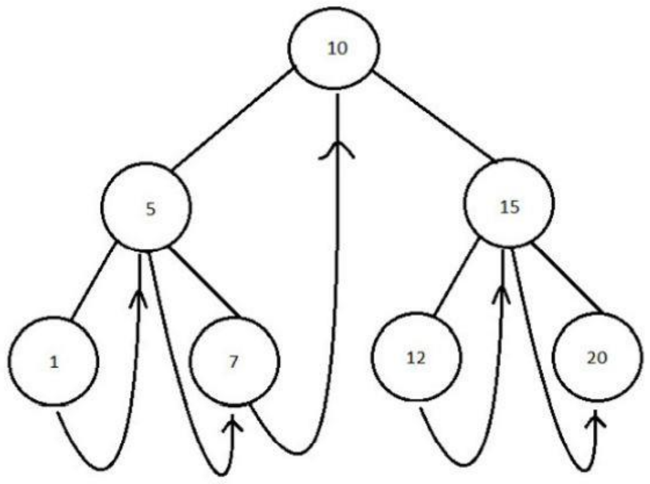
we will again repeat the step three on node 12 but this is a leaf node whose right pointer is pointing to it's ancestor so we will set rightThread bool variable as false.

step 5 :-

We will just keep repeating the steps 2 , 3 and 4 until whole tree is traversed just keeping one thing mind - whenever we make a new recursive call to right subtree, do not change the previous visited node and when we make a recursive call to left subtree then pass the actual previous visited node .

step 6 :-

At the end we will have the whole binary tree converted to threaded binary tree.



Time and space complexity for operations:

Time complexity for

- for insertion : log(n)
- for deletion : log(n)
- for seaching : log(n)

space complexity for insertion is O(1) , for deletion and searching we donot require any extra space.

The time required for finding inorder predecessor or successor for a given node is O(1) provided we are on that node.

Program: Write your own program and attach printouts

Output:

Conclusion:

Thus, we studied the threaded binary tree and converted binary tree into threaded binary tree. Also Analyzed time and space complexity of an algorithm.

Question Bank:

1. Why do we need Threaded Binary Tree?
2. What are the different types of TBT?
3. What are the advantages and disadvantages of TBT?
4. Explain with example insertion and deletion of nodes in In-Order TBT.