Leaning Objective:

In this Module you will be learning the following:

- Threaded Binary Tree

Introduction:

Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor.

Material:

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**.**

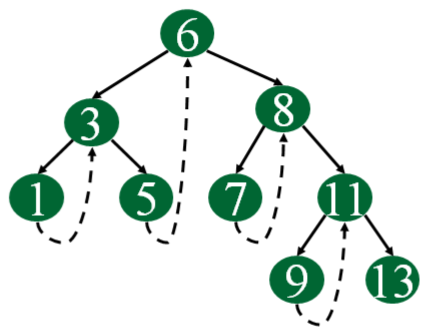
- We have the pointers reference the next node in an inorder traversal; called threads
- We need to know if a pointer is an actual link or a thread, so we keep a boolean for each pointer

Why do we need Threaded Binary Tree?

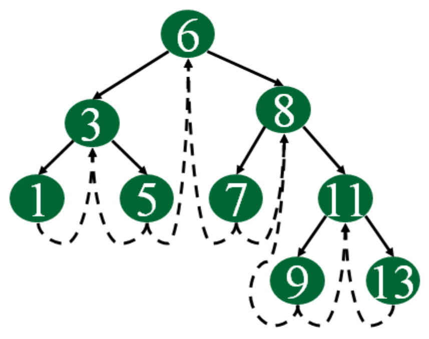
- Binary trees have a lot of wasted space: the leaf nodes each have 2 null pointers. We can use these pointers to help us in inorder traversals.
- Threaded binary tree makes the tree traversal faster since we do not need stack or recursion for traversal

Types of threaded binary trees:

- **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.
- **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.



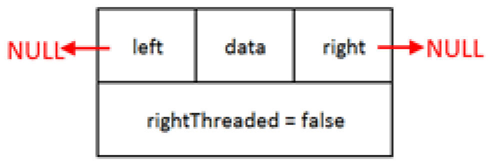
Single Threaded Binary Tree



Double Threaded Binary Tree

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.

Let's see how the Node structure will look like



In normal BST node we have left and right references and data but in threaded binary tree we have boolean another field called "rightThreaded". This field will tell whether node's right pointer is pointing to its inorder successor, but how, we will see it further.

Operations:

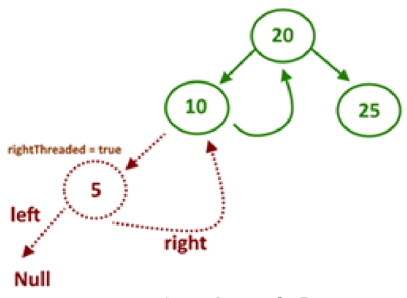
We will discuss two primary operations in single threaded binary tree

1. Insert node into tree
2. Print or traverse the tree.(here we will see the advantage of threaded tree)

Insert():

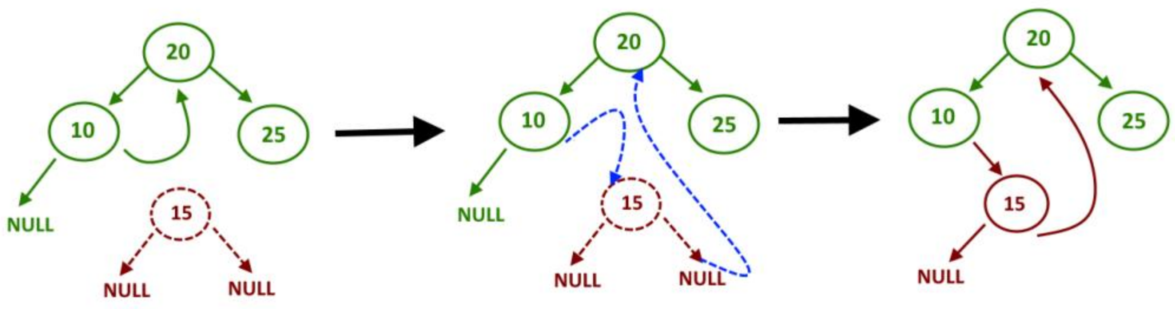
The insert operation will be quite similar to insert operation in Binary search tree with few modifications.

- To insert a node our first task is to find the place to insert the node.
- Take current = root .
- start from the current and compare root.data with n.
- Always keep track of parent node while moving left or right.
- if current.data is greater than n that means we go to the left of the root, if after moving to left, the current = null then we have found the place where we will insert the new node. Add the new node to the left of parent node and make the right pointer points to parent node and rightThread = true for new node.



Inserting node 5

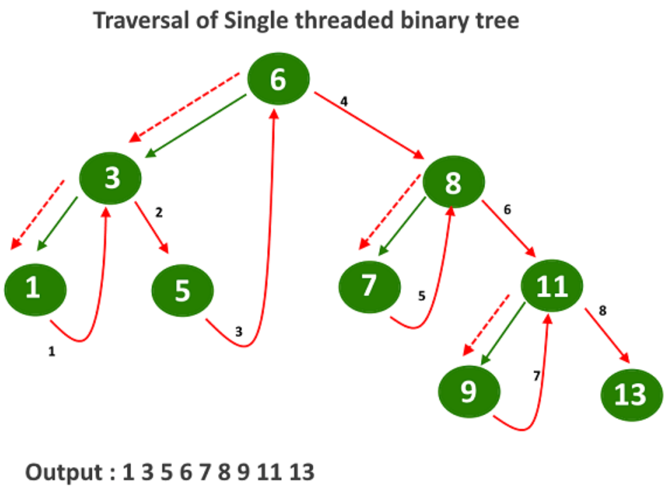
- if current.data is smaller than n that means we need to go to the right of the root, while going into the right subtree, check rightThread for current node, means right thread is provided and points to the in order successor, if rightThread = false then and current reaches to null, just insert the new node else if rightThread = true then we need to detach the right pointer (store the reference, new node right reference will be point to it) of current node and make it point to the new node and make the right reference point to stored reference. (See image and code for better understanding)



Inserting Node 15 into threaded binary tree

Traverse():

traversing the threaded binary tree will be quite easy, no need of any recursion or any stack for storing the node. Just go to the left most node and start traversing the tree using right pointer and whenever rightThread = false again go to the left most node in right subtree. (See image and code for better understanding)


Traversal of Single threaded binary tree
6
3
8
1
5
7
11
9
13
1
2
3
4
5
6
7
8
Output : 1 3 5 6 7 8 9 11 13

Follow the red arrow, dotted arrow when moving to left most node from the current node and solid arrow when using the right pointer to move it to it's inorder successor.

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.

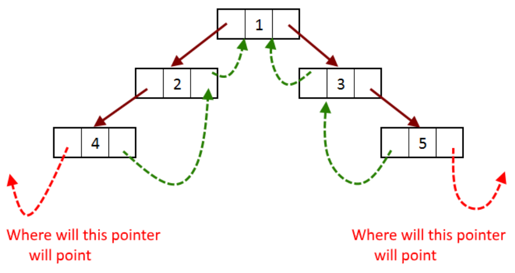
Let's see how the Node structure will look like

```
struct Node {
  int data, leftBit, rightBit;
  Node left, right;
}
```

If you notice we have two extra fields in the node than regular binary tree node. leftBit and rightBit. Let's see what these fields represent.

leftBit=0	left reference points to the inorder predecessor
leftBit=1	left reference points to the left child
rightBit=0	right reference points to the inorder successor
righBit=1	right reference points to the right child

Let's see why do we need these fields and why do we need a dummy node when If we try to convert the normal binary tree to threaded binary

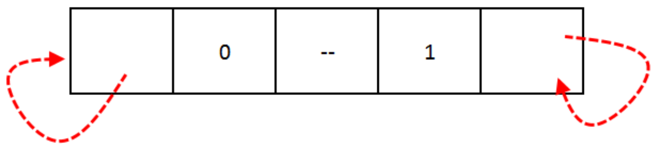

1
2
3
4
5
Where will this pointer will point
Where will this pointer will point

Now if you see the picture above , there are two references left most reference and right most reference pointers has nowhere to point to.

Need of a Dummy Node: As we saw that references left most reference and right most reference pointers has nowhere to point to so we need a dummy node and this node will always present even when tree is empty.

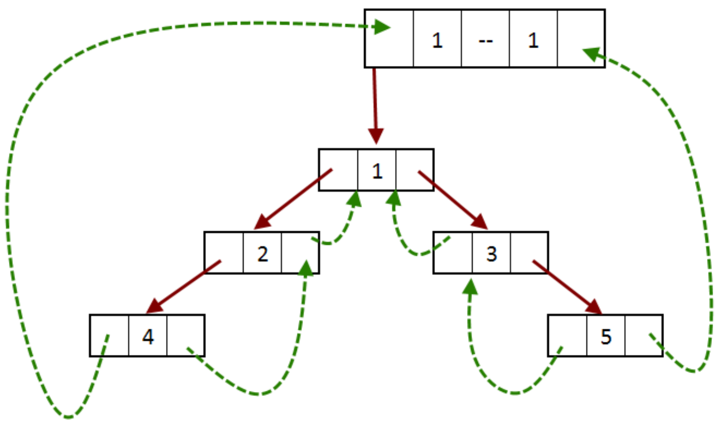
In this dummy node we will put ***rightBit = 1*** and its right child will point to it self and ***leftBit = 0***, so we will construct the threaded tree as the left child of dummy node.

Let's see how the dummy node will look like:



Dummy Node

Now we will see how this dummy node will solve our problem of references left most reference and right most reference pointers has nowhere to point to.



Double Threaded Binary Tree

Double Threaded binary tree with dummy node

Now we will see the some operations in double threaded binary tree.

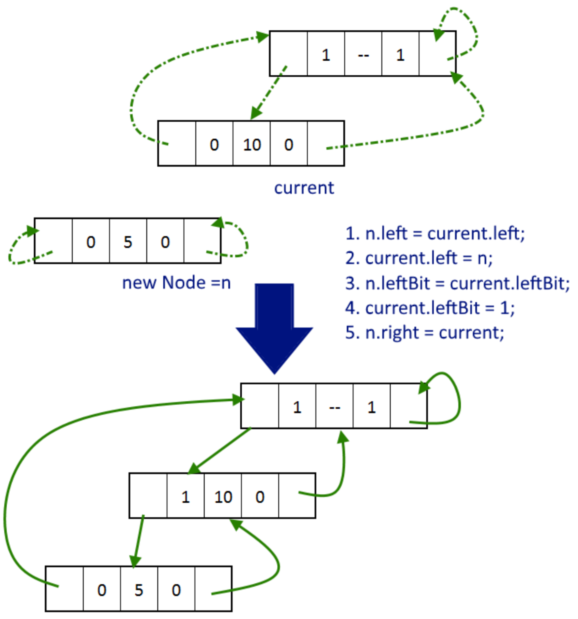
Insert():

The insert operation will be quite similar to Insert operation in Binary search tree with few modifications.

1. To insert a node our first task is to find the place to insert the node.
2. First check if tree is empty, means tree has just dummy node then then insert the new node into left subtree of the dummy node.
3. If tree is not empty then find the place to insert the node, just like in normal BST.
4. If new node is smaller than or equal to current node then check if ***leftBit =0***, if yes then we have found the place to insert the node, it will be in the left of the subtree and if ***leftBit=1*** then go left.
5. If new node is greater than current node then check if ***rightBit =0***, if yes then we have found the place to insert the node, it will be in the right of the subtree and if ***rightBit=1*** then go right.
6. Repeat step 4 and 5 till the place to be inserted is not found.
7. Once decided where the node will be inserted, next task would be to insert the node. first we will see how the node will be inserted as left child.

```
        n.left = current.left;
        current.left = n;
        n.leftBit = current.leftBit;
        current.leftBit = 1;
        n.right = current;
```

see the image below for better understanding

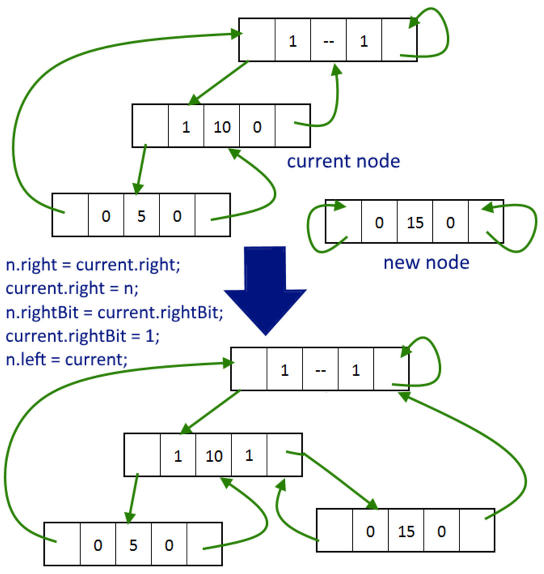


Insert the node in the left sub tree

8. To insert the node as right child.

```
        n.right = current.right;
        current.right = n;
        n.rightBit = current.rightBit;
        current.rightBit = 1;
        n.left = current;
```

see the image below for better understanding.

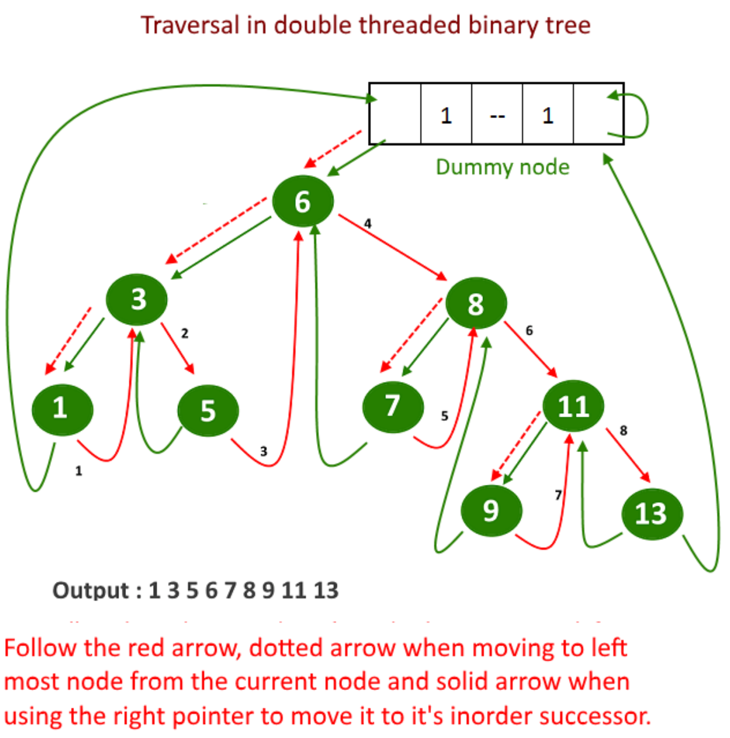


Insert the node into right subtree

Traverse():

Now we will see how to traverse in the double threaded binary tree, we do not need a recursion to do that which means it won’t require stack, it will be done n one single traversal in O(n). Starting from left most node in the tree, keep traversing the inorder successor and print it.(click here to read more about inorder successor in a tree).

See the image below for more understanding.


Traversal in double threaded binary tree
1
--
1
Dummy node
6
3
8
1
5
7
11
9
13
1
2
3
4
5
6
7
8
Output : 1 3 5 6 7 8 9 11 13
Follow the red arrow, dotted arrow when moving to left most node from the current node and solid arrow when using the right pointer to move it to it's inorder successor.

Problem Set:

1. Write a C program to implement single threaded binary tree?
2. Write a C program to implement double threaded binary tree?