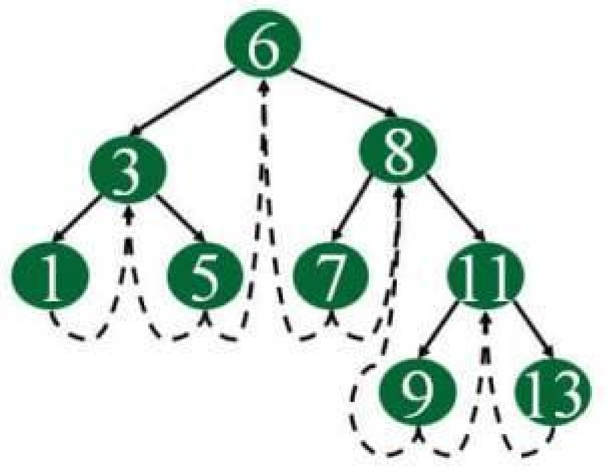
CL-218 Data Structures LAB

Threaded Binary Tree

INSTRUCTOR: MUHAMMAD HAMZA

Threaded Binary Tree:



Double Threaded Binary Tree

In Threaded Binary Tree each node is threaded towards both the in-order predecessor and successor (left and right) respectively, meaning all right null pointers will point to inorder successor AND all left null pointers will point to inorder predecessor.

Implementation:

Let's see how the Node structure will look like

```
class Node {
   int data;
   int leftBit;
   int rightBit;
   Node left;
```

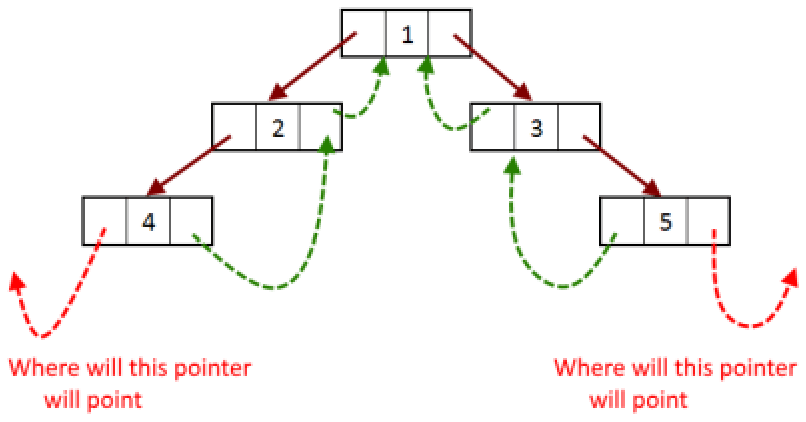
```
    Node right;

    public Node(int data) {
        this.data = data;
    }
}
```

If you notice we have two extra fields in the node than regular binary tree nodes. 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

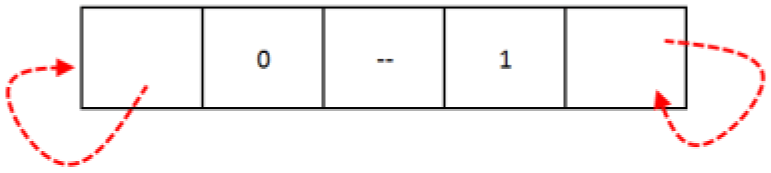

1
2
3
4
5
Where will this pointer will point
Where will this pointer will point

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

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

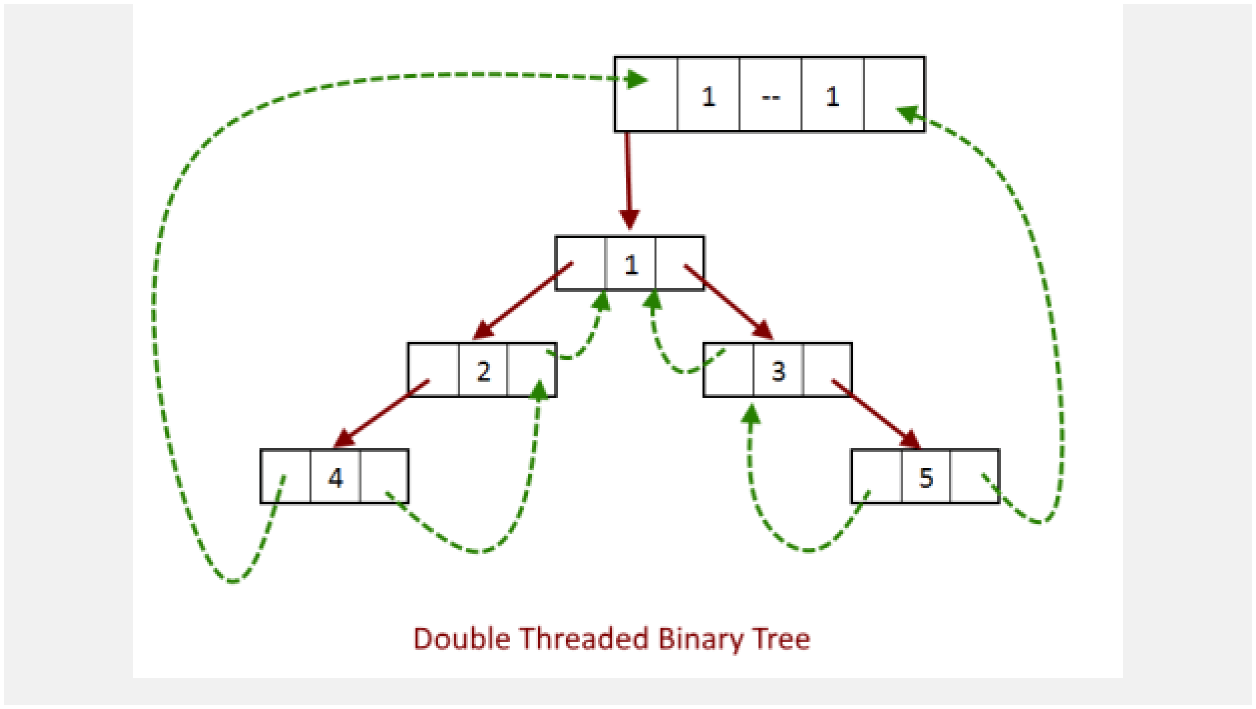
In this dummy node we will put *rightBit* = *1* and its right child will point to itself and *leftBit* = *0*, so we will construct the threaded tree as the left child of the dummy node.

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 have nowhere to point to.



Double Threaded Binary Tree

Now we will see some operations in threaded binary trees.

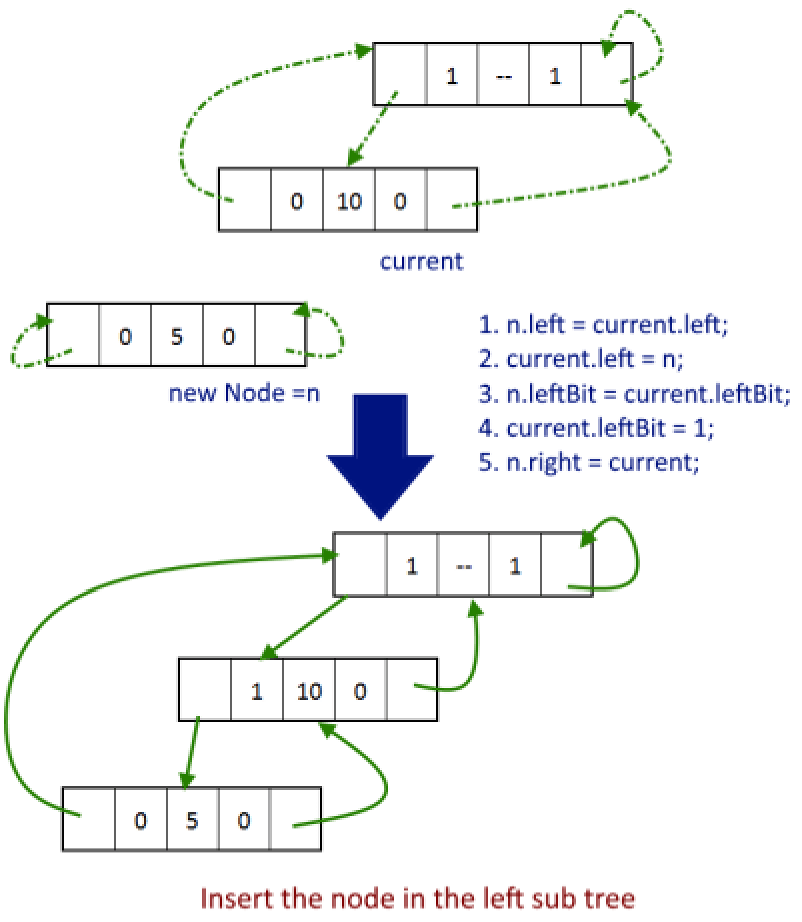
Insert():

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 the tree is empty, meaning the tree has just a dummy node then insert the new node into the left subtree of the dummy node.
3. If the tree is not empty then find the place to insert the node, just like in normal BST.
4. If a new node is smaller than or equal to the 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 a new node is greater than the 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, the next task would be to insert the node. First we will see how the node will be inserted as a left child.

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

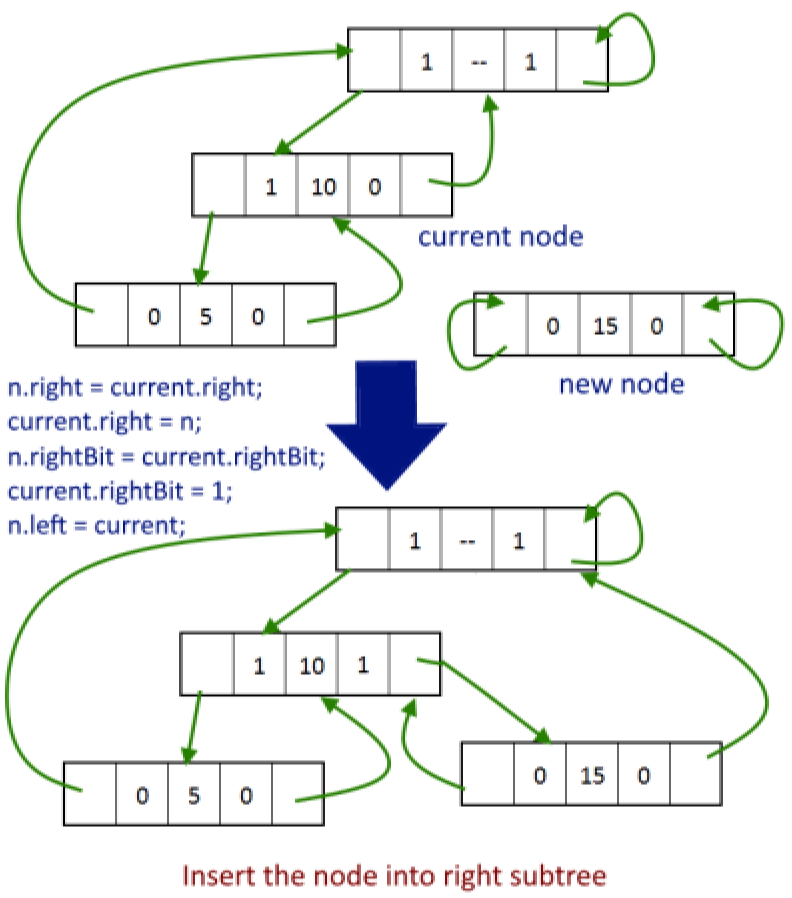
see the image below for better understanding



8. To insert the node as the 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.

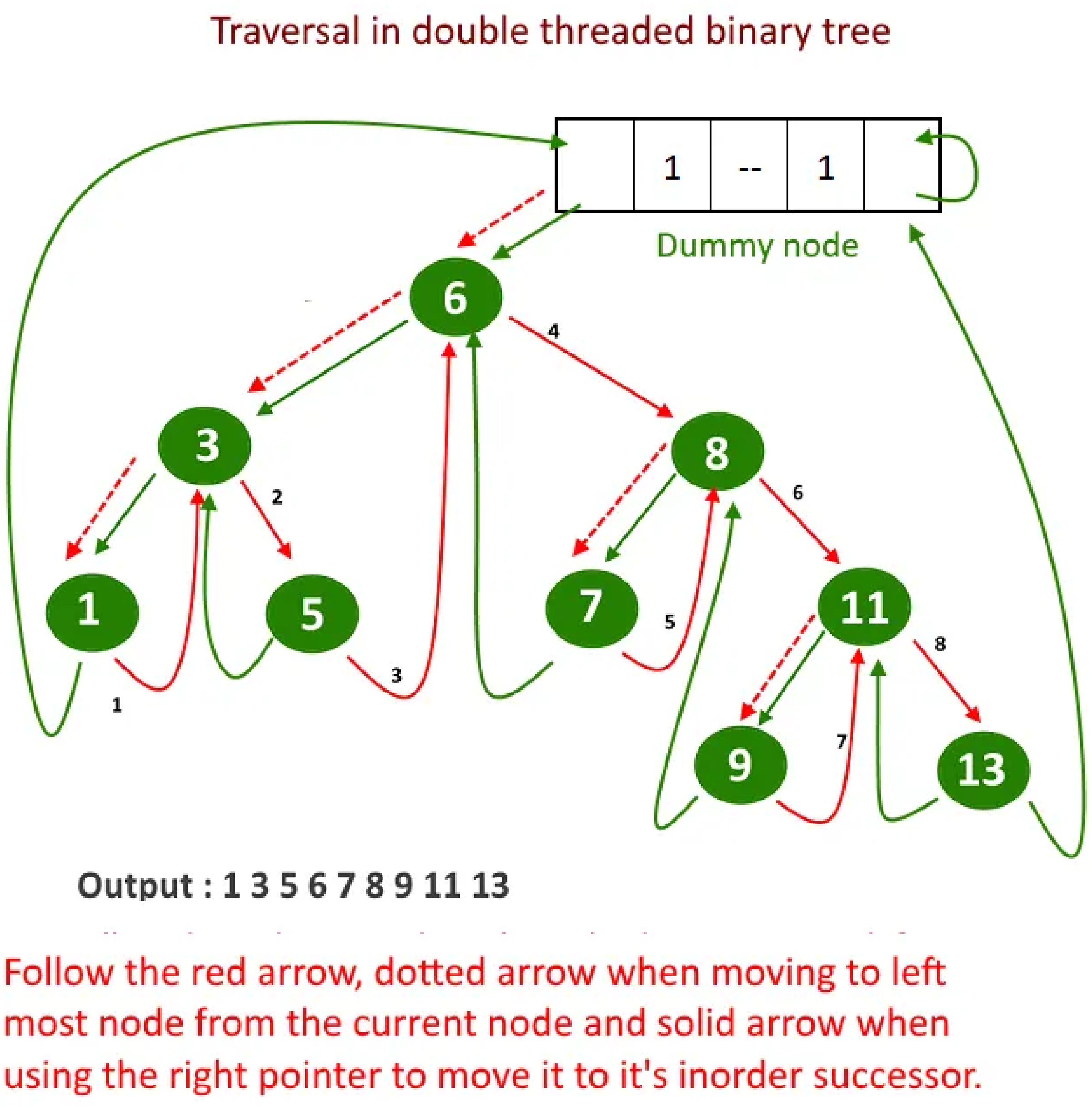


Traverse():

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 in one single traversal in O(n).

Starting from the left most node in the tree, keep traversing the inorder successor and print it.

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.