**Threaded Binary Trees**

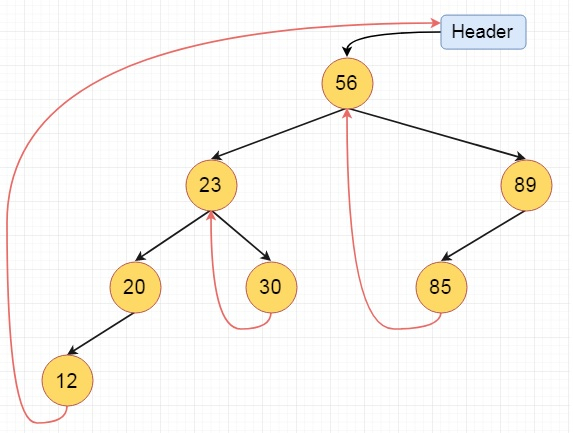
We know that the binary tree nodes may have at most two children. But if they have only one child, or no children, the link part in the linked list representation remains null. Using threaded binary tree representation, we can reuse that empty links by making some threads.

If one node has some vacant left or right child area, that will be used as thread. There are two types of threaded binary tree. The single threaded tree or fully threaded binary tree. In single threaded mode, there are another two variations. Left threaded and right threaded.

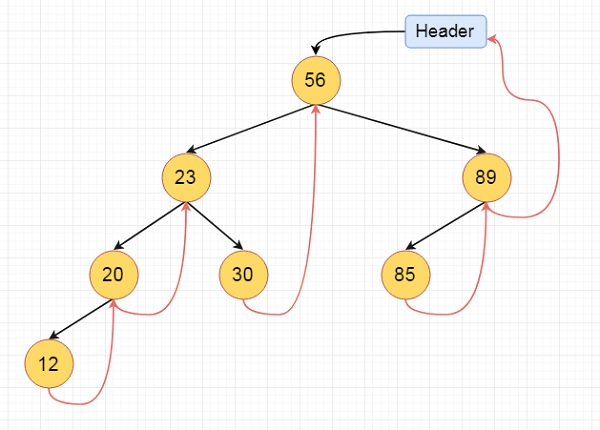
In the left threaded mode if some node has no left child, then the left pointer will point to its inorder predecessor, similarly in the right threaded mode if some node has no right child, then the right pointer will point to its inorder successor. In both cases, if no successor or predecessor is present, then it will point to header node.

For fully threaded binary tree, each node has five fields. Three fields like normal binary tree node, another two fields to store Boolean value to denote whether link of that side is actual link or thread.

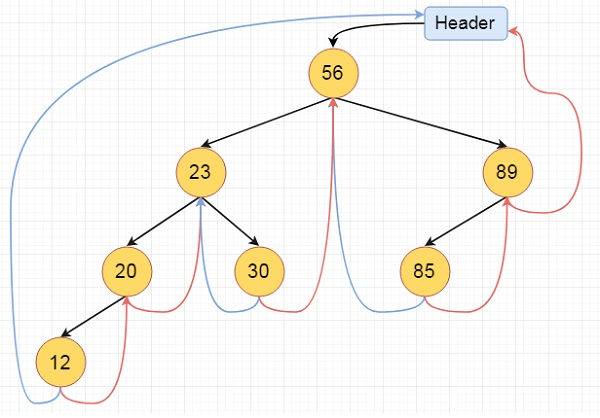
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Left Thread Flag | Left Link | Data | Right Link | Right Thread Flag |



These are the examples of left and right threaded tree



This is the fully threaded binary tree

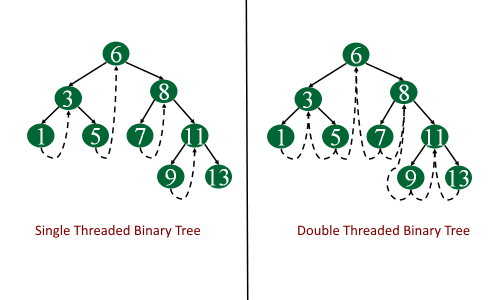


**Why do we need Threaded Binary Tree?**

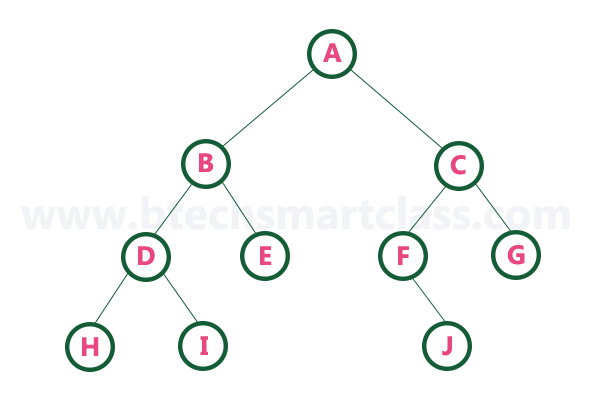
* Binary trees have a lot of wasted space: the leaf nodes each have 2 null pointers. We can use these pointers to help us in inorder traversals.
* Threaded binary tree makes the tree traversal faster since we do not need stack or recursion for traversal

**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 andright) means all right null pointers will point to inorder successor AND all left null pointers will point to inorder predecessor.

[](https://i1.wp.com/algorithms.tutorialhorizon.com/files/2016/03/Single-and-Double-threaded-binary-tree-1.png)

Consider the following binary tree...

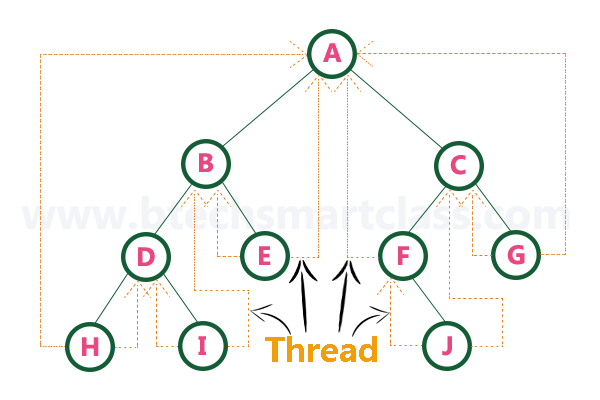


To convert the above example binary tree into a threaded binary tree, first find the in-order traversal of that tree...

**In-order traversal of above binary tree...**

**H - D - I - B - E - A - F - J - C - G**

When we represent the above binary tree using linked list representation, nodes **H, I, E, F, J** and **G** left child pointers are NULL. This NULL is replaced by address of its in-order predecessor respectively (I to D, E to B, F to A, J to F and G to C), but here the node H does not have its in-order predecessor, so it points to the root node A. And nodes **H, I, E, J** and **G** right child pointers are NULL. These NULL pointers are replaced by address of its in-order successor respectively (H to D, I to B, E to A, and J to C), but here the node G does not have its in-order successor, so it points to the root node A.  
  
Above example binary tree is converted into threaded binary tree as follows.



In the above figure, threads are indicated with dotted links.