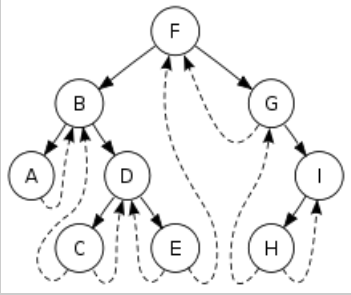
Threaded binary tree

From Wikipedia, the free encyclopedia

In computing, a **threaded binary tree** is a binary tree variant that allows fast traversal: given a pointer to a node in a threaded tree, it is possible to cheaply find its in-order successor (and/or predecessor).



A **threaded tree**, with the special threading links shown by dashed arrows

Motivation [edit]

Binary trees, including (but not limited to) binary search trees and their variants, can be used to store a set of items in a particular order. For example, a binary search tree assumes data items are somehow ordered and maintain this ordering as part of their insertion and deletion algorithms. One useful operation on such a tree is *traversal*: visiting the items in the order in which they are stored (which matches the underlying ordering in the case of BST).

A simple recursive traversal algorithm that visits each node of a BST is the following. Assume t is a pointer to a node, or nil. "Visiting" t can mean performing any action on the node t or its contents.

Algorithm traverse(t):

- Input: a pointer t to a node (or nil)
- If $t = \mathrm{nil}$, return.
- Else:
  - traverse(left-child(t))
  - Visit t
  - traverse(right-child(t))

The problem with this algorithm is that, because of its recursion, it uses stack space proportional to the height of a tree. If the tree is fairly balanced, this amounts to $O(\log n)$ space for a tree containing n elements. In the worst case, when the tree takes the form of a chain, the height of the tree is n so the algorithm takes $O(n)$ space.

In 1968, Donald Knuth asked whether a non-recursive algorithm for in-order traversal exists, that uses no stack and leaves the tree unmodified. One of the solutions to this problem is tree threading, presented by J. M. Morris in 1979.[1][2]

Definition [edit]

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 (**if** it exists), and all left child pointers that would normally be null

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
فارسی
Français
한국어
Português
Српски / srpski
中文
Edit links

point to the inorder predecessor of the node."[3]

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*.

Types of threaded binary trees [edit]

1. Single Threaded: each node is threaded towards **either** the in-order predecessor **or** successor (left **or** right).
2. Double threaded: each node is threaded towards **both** the in-order predecessor **and** successor (left **and** right).

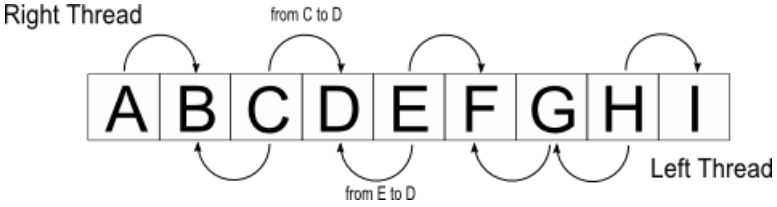
In Python:

```
def parent(node):
    if node is node.tree.root:
        return None
    else:
        x = node
        y = node
        while True:
            if is_thread(y):
                p = y.right
                if p is None or p.left is not node:
                    p = x
                    while not is_thread(p.left):
                        p = p.left
                    p = p.left
                return p
            elif is_thread(x):
                p = x.left
                if p is None or p.right is not node:
                    p = y
                    while not is_thread(p.right):
                        p = p.right
                    p = p.right
                return p
            x = x.left
            y = y.right
```

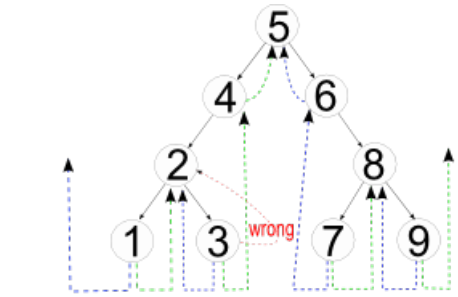
The array of Inorder traversal [edit]

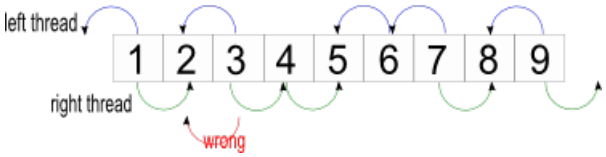
Threads are reference to the predecessors and successors of the node according to an inorder traversal.

Inorder of the threaded tree is ABCDEFGHI, the predecessor of E is D, the successor of E is F.

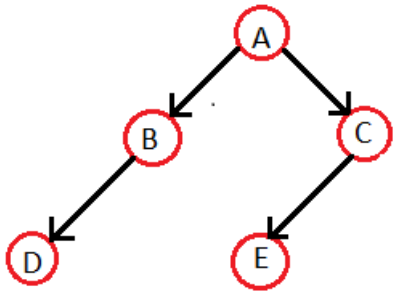


Example [edit]

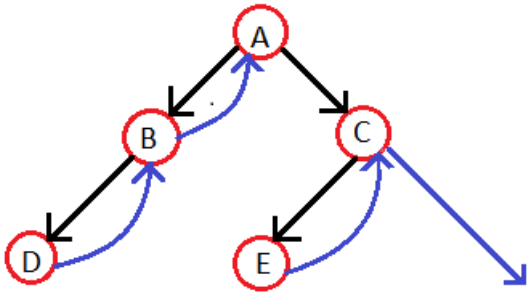




Let's make the Threaded Binary tree out of a normal binary tree



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

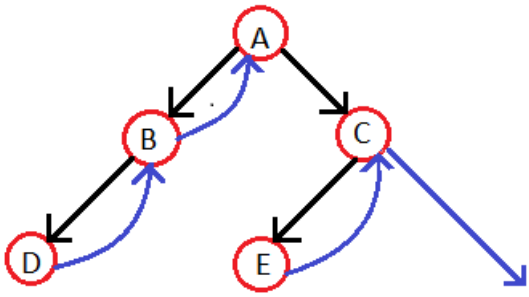


Null link [edit]

In an m-way threaded binary tree with n nodes, there are **n*m - (n-1)** void links.

Non recursive Inorder traversal for a Threaded Binary Tree [edit]

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. We will 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. Please, follow the example given below.



Algorithm [edit]

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

		Li	
step-1	'A' has a left child i.e. B, which has not been visited.So, we put B in our "list of visited nodes" and B becomes our current node in consideration.	B	
step-2	'B' also has a left child, 'D', which is not there in our list of visited nodes. So, we put 'D' in that list and make it our current node in consideration.	B D	
step-3	'D' has no left child, so we print 'D'. Then we check for its right child. 'D' has no right child and thus we check for its thread-link. It has a thread going till node 'B'. So, we make 'B' as our current node in consideration.	B D	D
step-4	'B' certainly has a left child but its already in our list of visited nodes. So, we print 'B'. Then we check for its right child but it doesn't exist. So, we make its threaded node (i.e. 'A') as our current node in consideration.	B D	D B
step-5	'A' has a left child, 'B', but its already there in the list of visited nodes. So, we print 'A'. Then we check for its right child. 'A' has a right child, 'C' and it's not there in our list of visited nodes. So, we add it to that list and we make it our current node in consideration.	B D C	D B A
step-6	'C' has 'E' as the left child and it's not there in our list of visited nodes even. So, we add it to that list and make it our current node in consideration.	B D C E	D B A
step-7		and finally.....	D B A E C

References [edit]

1. ^ Morris, Joseph M. (1979). "Traversing binary trees simply and cheaply". *Information Processing Letters* **9** (5). doi:10.1016/0020-0190(79)90068-1.
2. ^ Mateti and, Prabhaker; Manghirmalani, Ravi (1988). "Morris' tree traversal algorithm reconsidered". *Science of Computer Programming* **11**: 29–43. doi:10.1016/0167-6423(88)90063-9.
3. ^ Van Wyk, Christopher J. Data Structures and C Programs, Addison-Wesley, 1988, p. 175. ISBN 978-0-201-16116-8.

External links [edit]

- Tutorial on threaded binary trees
- GNU libavl 2.0.2, Section on threaded binary search trees

Categories: Binary trees | Search trees