PREPRINT. Information Processing Letters, 34(1):21-25, 1990.

Construction of a Tree from its Traversals in Optimal Time and Space *

Arne Andersson Svante Carlsson
Department of Computer Science
Lund University
Lund, Sweden

Abstract

Given the preorder traversal of a tree together with some additional structure information, the tree can be constructed in linear time with a simple algorithm. The additional information may be the inorder or postorder traversal. In one case, the binary search tree, no additional information is required. We also show how to transform the construction algorithm into a non-recursive algorithm requiring only constant space.

Keywords: Analysis of algorithms, data structures, binary tree, multiway tree, binary search tree, tree traversal, tree construction

1 Introduction

The problem of constructing a binary tree from its preorder and inorder traversals is well known. Lately Burgdorff et. al. [2] gave a non-recursive algorithm that solves the problem in $O(n^2)$ time for a tree with n nodes. Chen, Yu, and Liu [3] improved this result to $O(n \log n)$ with linear extra space. By using hashing they also obtained a space-inefficient algorithm with linear time bounds in the average case.

In this paper we present a faster and more general solution. We show that we can construct a tree in linear time from its preorder traversal together with some additional structure information like, for instance, its inorder traversal. With the postorder traversal as the additional information,

*This work was supported under a NFR Grant No. F-FU 8992-100 (Sweden)

a binary tree can not be perfectly constructed, since there is no way to determine whether a single child is a left- or right-child. However, in some applications we are only interested in the order between the children and do not differ between a single left- and right-child. In this case we can use the preorder and postorder traversals for construction. Moreover, we can build a tree of arbitrary degree given these two traversals.

Thus our result implies that a tree may be constructed in linear time from any two of its standard traversals. As a special case we show how to build a binary search tree from only its preorder traversal in linear time.

Our algorithms require no extra space apart from a stack to handle recursion. In the case when the preorder and inorder traversals are given, this extra space can be eliminated.

The previous algorithms for construction of a tree from its traversals have time bounds that make them of low practical interest. The linearity obtained in this paper makes this method competitive with other methods for temporary storage of trees in secondary memory.

2 Construction Algorithms

A tree is said to be *ordered* if the children of a node are ordered from left to right. A tree is *directed* if each child has a given position, e.g. when each child in a binary tree is either a left-child or a right-child. The binary search tree is an example of a directed tree. Clearly, a directed tree is ordered.

We introduce our main lemma showing that it is possible to construct a tree from a preorder listing together with some information on when a subtree is completely constructed.

Lemma 1 *An ordered tree may be constructed from its preorder listing if for each node it can be determined when all its subtrees are completely constructed. If it is possible to determine also the position of each subtree, a directed tree can be constructed.*

Proof: This is easily proved by induction. The lemma is trivially true for a tree consisting of a single node. For larger trees we proceed as follows:

1. Create a node p containing the first element in the preorder list that is not being used.

2. Construct all p's subtrees from left to right until they are completely constructed.

3. The tree rooted at p is now constructed

The root is correctly placed, since it is the first element in the preorder traversal. In Step 2 each subtree is correctly constructed due to the induction hypothesis. The order between the subtrees is also correct since they are correctly ordered and non-overlapping in the preorder list.

From this follows that an ordered tree may be properly constructed from the given information. It is also clear that if the position of each subtree can be determined, the corresponding directed tree can be constructed. □

In Lemma 1, we have a powerful tool for constructing a tree from its preorder traversal together with some extra information. Using this algorithm, we only have to show how to determine when a subtree is completely constructed. If we can test this in constant time, the algorithm will run in $\Theta(n)$ time. One way to obtain this information is by putting explicit marks in the preorder sequence or by keeping a separate list telling where in the preorder sequence a tree ends. In fact, it is possible to make such a list as short as $2n$ bits. In the following, we give some cases where the necessary structural information is given by traversals or ordering instead of by marks.

Theorem 1 *An ordered tree may be constructed in linear time given its preorder and postorder traversals.*

Proof: The tree may be constructed using the algorithm given in the proof of Lemma 1. As a test for completion of a subtree we use the fact that when all subtrees of a node p are completed, p is the next element in the postorder list. We can do this since, according to the algorithm, the subtrees of p are constructed from left to right recursively before the entire subtree rooted at p is completed. Since this order of completion is the same order as the postorder traversal of the tree we can just check the nodes in the order as they appear in the postorder list while they are completed. The algorithm is linear, since each test for completion takes constant time. The algorithm is illustrated in Figure 2. □

In Theorem 1 we showed that our algorithm is valid for trees of arbitrary degree when the extra information is the postorder traversal. If instead we have the inorder traversal, the algorithm works only for binary trees. This is natural, since the inorder traversal together with the preorder traversal does not define a unique multiway tree. On the other hand we can use the inorder traversal to construct not only ordered trees but also directed trees.

```
procedure PrePost (var T: Tree; var PRE, POST: NodeList);
var temp: Tree;
begin
    { The first element in preorder list PRE is the root of T }
    T := CreateNode (First(PRE));
    Delete first element of PRE;

    { Construct subtrees until the root is found in the postorder list }
    while First(POST) ≠ T↑.value do begin
        PrePost (temp, PRE, POST);
        AddNewChild (T, temp)
        { temp becomes the next subtree of T }
    end;

    { All subtrees of T are constructed }
    Delete first element of POST;
end;
```

Figure 1: *Pascal procedure PrePost constructs the nonempty multiway tree T, whose preorder and postorder traversals are prefixes of the lists PRE and POST, respectively. The elements in T are deleted from the lists.*

Theorem 2 *A directed binary tree may be constructed in linear time given its preorder and inorder traversals.*

Proof: We use the same proof technique as in Theorem 1. Here we note that if the nodes are encountered in the order in which their left subtrees are completed, they are ordered as in the inorder traversal. From this we conclude that the subtree rooted at a node p is completed when p's nearest ancestor to the right is the next element in the inorder list. References to the nearest right ancestors may be kept in a stack or by recursion. Thus, we may compare p with its nearest right ancestor as a test for completion. Since we know when a left subtree is completed we can determine the position of each subtree. The time bound can be shown to be linear in the same way as in Theorem 1. The algorithm is illustrated in Figure 2. □

An interesting special case where we do not need any extra information at all, apart from the preorder list, is when we construct a binary search tree. It is well known that if we insert the nodes into the tree in the order in which they occur in the preorder list, we get the same tree as before. However, this construction method requires $\Theta(n^2)$ time in the worst case and $\Theta(n \log n)$ time in the best case. In Theorem 3 we show that this construction can be made in linear time.

Theorem 3 *A binary search tree may be constructed in linear time given its preorder traversal.*

Proof: In this case we know that the subtree rooted at p is completed when the next element in the preorder list has a larger value than p's nearest right ancestor. The rest of the proof follows the proof of Theorem 2. □

Theorem 3 may be applied for example on the optimum search tree [4], which is a binary search tree where the exact structure is important. With our algorithm, such a tree can easily be temporarily stored and reconstructed in linear time. The method may also be applied on search tree-like structures such as quad trees [1].

Finally, we show how to construct a directed binary tree in optimal space given its preorder and postorder traversals. The algorithms presented above requires extra storage proportional to the height of the tree in order to handle the recursion. This extra space may be reduced to a constant by using empty pointers in the tree, instead of a stack. Following those pointers we

```
PROCEDURE PreIn (var  T: Tree; var  PRE, IN: Itemlist; right-anc: data);
begin
   if  Empty (PRE) or  First(IN) = right-anc then
       { the construction of the right ancestor's left subtree is completed }
       T := nil
   else  begin
       { The first element in the preorder list is the root of T }
       T := CreateNode (PRE[pr]);
       Delete first element of PRE;

       { Construct the left subtree }
       PreIn (T↑.left, PRE, IN, T↑.value);

       { The root of T is the first element in the inorder list }
       Delete first element of IN;

       { Construct the right subtree. }
       PreIn (T↑.right, PRE, IN, right-anc);
   end
end;
```

Figure 2: *The Pacal procedure PreIn constructs the nonempty binary tree T, whose preorder and inorder traversals are prefixes of the lists PRE and IN, respectively.* **right-anc** *contains the value of the nearest ancestor to the right of the root of T. If no such ancestor exists,* **right-anc** *has the value* **void**. *The elements in T are deleted from the lists.*

can traverse the tree during its construction. The rest of the changes in the algorithm is a standard iterative implementation of the recursive algorithm.

Theorem 4 *There is an algorithm using constant extra space that constructs a directed binary tree in linear time from its preorder and inorder traversals.*

Proof: The only extra information we need while constructing a binary tree from its preorder and inorder traversals is a stack of references to the nearest right ancestors. However, the only nodes from which we need such references are nodes where the right subtree is not yet constructed. Thus we can use their empty right-child pointers to refer to the nearest right ancestor. Following those pointers backwards, we can eliminate the recursion as well as all extra storage except for the tree itself. □

The space-optimal algorithm is given in Figure 3. A similar algorithm may be used for the construction of a binary search tree.

Example:
We give an example of optimal tree construction. The tree is constructed during a preorder traversal and the current visited node is marked in the figures.

(a) The current node is not first in the inorder list. Remove the first element from the preorder list and place it as the left-child.

(b) The current node is first in the inorder list, thus its left subtree is constructed. Remove the current node from the inorder list. Since the new first element in the inorder list is its right ancestor, the current node has an empty right subtree.

(c) The current node
the inorder list, but t
ement is not the righ
Thus we start const
right subtree.

3 Conclusion

We have presented optimal algorithms for rebuilding trees from their traversals. The optimality together with the simplicity of the algorithms make this method a competitive alternative to other methods for temporary storage of a tree in secondary memory. It may for instance be used on tree structures that are expensive to construct, such as optimum search trees and multidimensional trees.

The results for the preorder traversal are easily shown to be valid also for postorder traversal. This implies, for instance, that a binary search tree may be constructed from its postorder traversal (by scanning the list backwards) and that a directed binary tree may be constructed from its postorder and inorder traversals. Thus, an ordered binary tree may be constructed in linear time from any two of its standard traversals.

```
procedure NonRecursive (var T: Tree; var PRE, IN: NodeList);
var current, right-anc: Tree;
begin
    T := CreateNode (First(PRE));
    Delete first element of PRE;
    current := T;
    while not Empty (IN) do
        if current↑.data ≠ First (IN) then begin
        { current has a non-empty left subtree }
            right-anc := current;
            current↑.left := CreateNode (First(PRE));
            Delete first element of PRE;
            current := current↑.left;
            current↑.right := right-anc
        end
        else begin
        { current's left subtree has been constructed }
            Delete first element of IN;
            if not Empty (IN) and
            (current↑.right = nil cor First(IN) ≠ current↑.right.data) then begin
            { current has a right subtree }
                right-anc := current↑.right;
                current↑.right := CreateNode (First(PRE));
                Delete first element of PRE;
                current := current↑.right;
                current↑.right := right-anc
            end
            else begin
            { current's right subtree is empty }
                right-anc := current↑.right;
                current↑.right := nil;
                current := right-anc
            end
        end
    end
end
```

Figure 3: *The space-optimal algorithm for constructing a binary tree from its preorder and inorder traversals.* **right-anc** *is used for temporary reference to the nearest right ancestor.*

Acknowledgements

We would like to thank the referees for their valuable comments on the presentation of the paper. We would also like to thank Christer Mattsson and the rest of the algorithm theory group in Lund.

References

[1] J.L. Bentley: Multidimensional Binary Search Trees Used for Associative Searching, CACM 18, 9 (1975), 509 - 517

[2] H.A. Burgdorff, S. Jajodia, F.N. Springsteel and Y. Zalcstein: alternative Methods for the Reconstruction of Trees from their Traversals, BIT 27, 2 (1987), 134 - 140

[3] G-H. Chen, M.S. Yu and L.T. Liu: Two Algorithms for Constructing a Binary Tree from its Traversals, IPL 28, 6 (1988), 297 - 299

[4] D. E. Knuth: Optimum Binary Search Trees, Acta Informatica 1, 1 (1971), 14 - 25