NONRECURSIVE ALGORITHMS FOR RECONSTRUCTING A BINARY TREE FROM ITS TRAVERSALS

G. H. Chen, M. S. Yu, and L. T. Liu

Department of Computer Science and Information Engineering,
National Taiwan University,
Taipei, Taiwan, China

Abstract

Given the inorder traversal of a binary tree, along with one of its preorder or postorder traversals, the original binary tree can be uniquely identified. Previous algorithms, recursive or nonrecursive, need $O(N^2)$ time to reconstruct the original binary tree. In this paper, we present two nonrecursive reconstruction algorithms; one, which requires $O(N)$ time, is time optimal but space inefficient and the other requires $O(N\log N)$ time and $O(N)$ space.

1. Introduction

It is well-known that given the inorder traversal of a binary tree, along with one of its preorder or postorder traversals, the original binary tree can be uniquely identified. It is not difficult to write a recursive algorithm to reconstruct the binary tree [4]. The computation time required is $O(N^2)$ where N is the number of nodes in the tree. Recently, Burgdorff et al. [1] presented a nonrecursive algorithm for solving this problem. They reconstructed the binary tree from its inorder-preorder sequence (i-p sequence for short) [2] and their algorithm needs also $O(N^2)$ computation time. In this paper, we propose two nonrecursive reconstruction algorithms; one, which requires $O(N)$ time, is time optimal but space inefficient and the other requires $O(N\log N)$ time and $O(N)$ space.

2. Nonrecursive reconstruction algorithms

In this section we present two nonrecursive algorithms for reconstructing the original binary tree from its inorder and preorder traversals. The algorithms proceed in two stages. First, the i-p sequence is constructed from the inorder and preorder traversals. Then, the original binary tree is reconstructed from the i-p sequence.

Definition. [2] The *i-p sequence* of a binary tree with N nodes is the numeric sequence output by the following algorithm:
(1) Label the nodes of the tree as accessed in inorder by consecutive integers 1,2, ... ,N;
(2) Output these numeric labels as the nodes are accessed in preorder.

For example, let us refer to Figure 1(a) where a binary tree is shown. The numeric labels of nodes and the i-p sequence are shown in Figure 1(b) and Figure 1(c) respectively. The i-p sequence is also known as tree-permutation [3]. Further, there is a one-to-one correspondence between the set of all (ordered) binary trees with N nodes and the set of all i-p sequences of length N [1].

Let I[1..N] and P[1..N] be the two sequences that represent the inorder traversal and the preorder traversal of the given binary tree, respectively. Without loss of generality, assume that nodes are represented by distinct alphanumeric labels. Also, let IP[1..N] represent the corresponding i-p sequence. In fact, IP[i]=I^{-1}[P[i]], $1\le i\le N$. We have two approaches to implement the mapping I^{-1}. The first approach is to sort I and then construct IP by searching for each P[j] in I, $1\le j\le N$. If binary search is used, the computation time and space required are $O(N\log N)$ and $O(N)$ respectively. The second approach uses the technique of hashing [5] to accelerate the searching. That is, we first store I[i]'s into hashing table according to their contents and then directly search for each P[j] in the hashing table by its value. The computation time required is $O(N)$, which is optimal within a constant factor. But, the space required is as large as the size of the hashing table.

Before presenting the reconstruction algorithm, let us look at Figure 1(c) again. For each node, all its descendants immediately follow it in the preorder sequence, and moreover, its left descendants are prior to its right descendants. Also, its numeric label is less (greater) than the numeric labels of its left (right) descendants. This property does not just hold for Figure 1; it is a property of the i-p sequence. The following lemma is an immediate result from the definition of the i-p sequence.

Lemma 1. Let P[i] be a nonleaf node and s (s>0) be the number of its descendants. Then, IP[i]>IP[j] for $i<j<k$ and IP[i]<IP[j] for $k\le j\le i+s$ iff P[i+1], ... ,P[k-1] are the left descendants of P[i] and P[k], ... ,P[i+s] are the right descendants of P[i]. Moreover, P[i+1] is the left child of P[i] and P[k] is the right child of P[i].

According to lemma 1, we can reconstruct the binary tree by sequentially examining array P. For each nonleaf node P[i], if s is known, we can find its left child and right child (if they exist) as stated by lemma 1. Let S[i], $1\le i\le N$, denote the cardinality of the set $\{j \mid j<i$ and IP[j]<IP[i]$\}$. S[i] is the number of nodes that precede P[i] in both the inorder and the preorder sequences. Array S will be used in the reconstruction algorithm and can be computed according to the following lemma, with S[1]=0 initially.

Lemma 2. Let P[j] be the parent of P[i]. If P[i] is the left child of P[j], then S[i]=S[j]. If P[i] is the right child of P[j], then S[i]=IP[j].
Proof. We prove this lemma by counting the number of nodes that precede P[i] in both the inorder and the preorder

sequences. Two cases are discussed in the following.
case 1. P[*i*] is the left child of P[*j*]. Since P[*i*] is the left child of P[*j*], then a node *v* precedes P[*j*] in both the inorder and the preorder sequences iff *v* precedes P[*i*] in both the inorder and the preorder sequences. Thus, S[*i*]=S[*j*] is implied.
case 2. P[*i*] is the right child of P[*j*]. Since P[*i*] is the right child of P[*j*], the nodes that precede P[*i*] in the inorder sequence fall into three classes: left descendants of P[*i*], P[*j*], and the nodes that precede P[*j*] in the inorder sequence. The nodes belonging to the first class follow P[*i*] in the preorder sequence, while the others still precede P[*i*] in the preorder sequence. Therefore, S[*i*] equals the number of nodes belonging to the second and the third classes, which is exactly IP[*j*]. **Q.E.D.**

Let [L1[*i*]..L2[*i*]] be the range of array P where the left descendants of P[*i*] are located and [R1[*i*]..R2[*i*]] be the range of array P where the right descendants of P[*i*] are located. If L1[*i*]>L2[*i*] (R1[*i*]>R2[*i*]), then P[*i*] has no left (right) descendants. Initially, L1[1]=2, L2[1]=IP[1], and R2[1]=*N*. Also, R1[1]=L2[1]+1 if L1[1]≤L2[1], R1[1]=2 otherwise. Let P[*j*] (*j*<*i*) be the parent of P[*i*]. L1[*i*], L2[*i*], R1[*i*], and R2[*i*] are computed as follows:

```
L1[i] := i+1;
L2[i] := i+IP[i]-S[i]-1;
R1[i] := if L1[i]≤L2[i] then L2[i]+1 else i+1;
R2[i] := if P[i] is the right child of P[j] then R2[j]
                                            else L2[j].
```

The computations of L1[*i*], R1[*i*], and R2[*i*] are straightforward; the computation of L2[*i*] is based on the following lemma.

Lemma 3. The number of left descendants of P[*i*] equals IP[*i*]-S[*i*]-1.
Proof. The nodes that precede P[*i*] in the inorder sequence fall into two classes. The left descendants of P[*i*] belong to the first class. The others belong to the second class. Each node in the first class follows P[*i*] in the preorder sequence; each node in the second class precedes P[*i*] in the preorder sequence. Thus, the number of left descendants of P[*i*] equals the number of nodes preceding P[*i*] in the inorder sequence minus the number of nodes preceding P[*i*] in both the inorder and the preorder sequences, which is IP[*i*]-S[*i*]-1. **Q.E.D.**

In the following, we present the reconstruction algorithm.

```
L1[1] := 2;  L2[1] := IP[1];
R1[1] := if L1[1]≤L2[1] then L2[1]+1 else 2;
R2[1] := N;  S[1] := 0;
if L1[1]≤L2[1] then P[L1[1]] is the left child of P[1];
if R1[1]≤R2[1] then P[R1[1]] is the right child of P[1];
for i := 2 to N do
    begin {Assume P[j] is the parent of P[i]}
      S[i] := if P[i] is the left child of P[j] then S[j]
                                                 else IP[j];
      L1[i] := i+1;
      L2[i] := i+IP[i]-S[i]-1;
      R1[i] := if L1[i]≤L2[i] then L2[i]+1 else i+1;
      R2[i] := if P[i] is the right child of P[j] then R2[j]
                                                 else L2[j];
      if L1[i]≤L2[i] then P[L1[i]] is the left child of P[i];
      if R1[i]≤R2[i] then P[R1[i]] is the right child of P[i]
    end
```

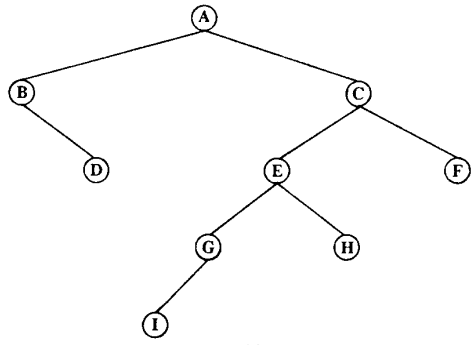
The computation time and space requied for executing the reconstruction algorithm are $O(N)$. Figure 2 shows the contents of S, L1, L2, R1, and R2 when Figure 1 is taken as the input problem instance.

3. Concluding remarks

In this paper, we proposed two nonrecursive algorithms for reconstructing the original binary tree from its inorder and preorder traversals. The proposed algorithms proceed in two stages; the first stage establishes the i-p sequence and the second stage reconstructs the binary tree from the i-p sequence. If sorting and binary search are used, then the space required is optimal within a constant factor. If, instead, hashing is used, the computation time required is optimal within a constant factor. Besides, simple modification of the proposed algorithm can be used to reconstruct the original binary tree from its inorder and postorder traversals. An interested reader will work out the details without much effort.

References

[1] H.A. Burgdorff, S. Jajodia, F. N. Springsteel, and Y. Zalcstein, "Alternative methods for the reconstruction of trees from their traversals," *BIT*, vol. **27,** no. 2, p. 134, 1987.
[2] T. Hikita, "Listing and counting subtrees of equal size of a binary tree," *Inform. Process. Lett.*, vol. 17, no. 4, p. 225, 1983.
[3] G.D. Knott, "A numbering system for binary trees," *Comm. ACM*, vol. 20, no. 2, p. 113, 1977.
[4] D.E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1973.
[5] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.



(a)

inorder:	B	D	A	I	G	E	H	C	F
labels:	1	2	3	4	5	6	7	8	9

(b)

preorder:	A	B	D	C	E	G	I	H	F
i-p sequence:	3	1	2	8	6	5	4	7	9

(c)

Figure 1. An example: (a) a binary tree; (b) the numeric labels of the nodes; (c) the i-p sequence.

i:	1	2	3	4	5	6	7	8	9
P[i]:	**A**	**B**	**D**	**C**	**E**	**G**	**I**	**H**	**F**
S[i]:	0	0	1	3	3	3	3	6	8
L1[i]:	2	3	4	5	6	7	8	9	10
L2[i]:	3	2	3	8	7	7	7	8	9
R1[i]:	4	3	4	9	8	8	8	9	10
R2[i]:	9	3	3	9	8	7	7	8	9

Figure 2. The contents of the arrays S, L1, L2, R1, and R2 if the input is the problem instance shown in Figure 1.