Generation of Binary Trees from Stack Permutations

ICHIRO SEMBA*

An efficient algorithm generating all binary trees from stack permutations is presented. The average running time per binary tree is shown to be bounded by a constant.

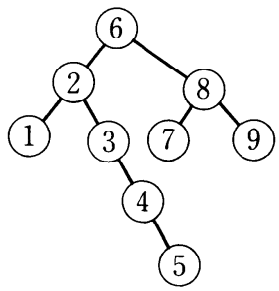
1. Introduction

By binary tree traversal[1, §2.3.1], a binary tree on n nodes is represented by a certain permutation of integers 1, 2,$\cdots$, n. Let BT(n) be the set of binary trees on n nodes.

We consider the following scheme.

Scheme. Label the nodes of a binary tree $T \in BT(n)$ 1 through n in inorder and read the labels in postorder.

We denote by $x(T) = x_1\ x_2 \cdots\ x_n$ the permutation obtained by this scheme and call it inorder-postorder permutation. For a binary tree $T_0 \in BT(9)$, an inorder-postorder permutation $x(T_0)$ is equal to 154327986.



a binary tree To

If $T_1 \neq T_2$, where $T_1, T_2 \in BT(n)$, then $x(T_1) \neq x(T_2)$. Thus the binary tree corresponding to a given inorder-postorder permutation $x = x_1\ x_2 \cdots\ x_n$ is determined uniquely and denoted by $B(x)$. Therefore it follows that $B(x(T_0)) = T_0$.

In this paper, we present an efficient algorithm for converting in a linear time from an inorder-postorder permutation x to a binary tree $B(x)$. Inorder-postorder permutations are shown to be stack permutations. We have already obtained a generating algorithm[2] for all stack permutations. Thus, combining these two algorithms, we can establish an efficient algorithm generating all binary trees. The average running time per binary tree is shown to be bounded by a constant.

*Department of Pure and Applied Sciences, College of General Eduction, University of Tokyo.

2. Preliminaries

In this section, the set of inorder-postorder permutations is shown to be equal to the set of stack permutations. Let $X(n)$ be the set of inorder-postorder permutations $x_1\ x_2 \cdots\ x_n$.

Property 2.1 Let $x_1\ x_2 \cdots\ x_n \in X(n)$ $(n \geq 3)$. There are no elements x_i, x_j and x_k such that $x_j < x_k < x_i$ $(i < j < k)$.

Proof. By the definition of inorder-postorder permutation, if $x_j < x_i$ $(i < j)$, then x_i is contained in the right subtree whose root is x_j. By the same reason, if $x_j < x_k$ $(j < k)$, then x_k is not contained in the subtree whose root is x_j. This means that x_k is not contained in the subtree whose root is x_i and $i < k$. Thus we have $x_i < x_k$. Namely, there are no elements x_i, x_j and x_k such that $x_j < x_k < x_i$ $(i < j < k)$. This completes the proof.

Let $P(n)$ be the set of stack permutations. A stack permutation is obtained, by using a stack, from input permutation $1\ 2 \cdots\ n$. Knuth [1, §2.2.1, Ex 5] has proved the following property.

Property 2.2 A permutation $p_1\ p_2 \cdots\ p_n$ $(n \geq 3)$ is a stack permutation, if and only if there are no elements p_i, p_j and p_k such that $p_j < p_k < p_i$ $(i < j < k)$.

Theorem 1 For $n \geq 1$,

$$X(n) = P(n).$$

Proof. For $n = 1, 2$, the result is immediate. For $n \geq 3$, by Property 2.1 and 2.2, it follows that $X(n) \subseteq P(n)$. On the other hand, by the fact that $|X(n)| = |BT(n)| = \binom{2n}{n}/(n+1)$ (Catalan number) and $|P(n)| = \binom{2n}{n}/(n+1)$, we obtain $X(n) = P(n)$.

Namely, an inorder-postorder permutation is a stack permutation.

3. Converting Algorithm for Converting from an Inorder-postorder Permutation to a Binary Tree

In this section, we give an efficient algorithm for converting, in a lincar time, from an inorder-postorder

permutation $x_1\ x_2 \cdots\ x_n$ to a corresponding binary tree $B(x_1\ x_2 \cdots\ x_n)$. Suppose that a binary tree has $j-i+1$ $(i \le j)$ nodes labeled i through j in inorder. When we read the labels in postorder, we obtain a sequence $t_1 \cdots t_{j-i+1}$. We say it is an inorder-postorder sequence on $\{i, \cdots, j\}$. Namely, an inorder-postorder permutation is an inorder-postorder sequence on $\{1, 2, \cdots, n\}$. We show a basic property of inorder-postorder permutations.

Property 3.1 Let $x_n = i$ $(1 \le i \le n)$. The permutation $x_1\ x_2 \cdots\ x_n$ is inorder-postorder permutation, if and only if the subsequence $x_1 \cdots x_{i-1}$ is inorder-postorder sequence on $\{1, \cdots, i-1\}$ and the subsequence $x_i \cdots x_{n-1}$ is inorder-postorder sequence on $\{i+1, \cdots, n\}$. If either $i=1$ or $i=n$, then the corresponding subsequence is empty.

Proof. From the definition of inorder-postorder permutation, it is easily proved.

By Property 3.1, a binary tree $B(x_1\ x_2 \cdots\ x_n)$ is obtained using the following recursive construction rule.

(I) If $n=1$, then $B(x_1)$ is a binary tree with one node.

(II) If $n>1$, let $x_n=i$ $(1 \le i \le n)$. Then $B(x_1\ x_2 \cdots\ x_n)$ is the binary tree formed by joining the left subtree $B(x_1 \cdots x_{i-1})$ and the right subtree $B(x_i \cdots x_{n-1})$ to the root x_n. If either subsequence is empty, then the corresponding subtree is empty.

The algorithm generating all stack permutations examines the stack permutation from right to left. Thus, in order to combine the generating algorithm and the converting algorithm efficiently, we give a converting algorithm which examines the inorder-postorder permutation (i.e. stack permutation) in a similar way. It is shown in Fig. 3.1 in a PASCAL-like notation. We use the following notations mainly after Knuth [1, §2.2.1]. We write stack to denote its top element. We write stack$\Leftarrow\phi$ to mean that the stack is made empty.

```
 1. begin
 2.   stack⇐φ; stack⇐0; stack⇐x_n; root:=x_n;
 3.   for i:=n-1 downto 1 do begin
 4.     if x_i>stack then
 5.       rson (stack):=x_i
 6.     else begin
 7.       rson (stack):=0; w⇐stack;
 8.       while x_i<stack do begin
 9.         lson(w):=0; w⇐stack
10.       end;
11.       lson(w):=x_i
12.     end;
13.     stack⇐x_i
14.   end;
15.   rson (stack):=0; w⇐stack;
16.   while w>0 do begin lson(w):=0; w⇐stack end
17. end.
```

Fig. 3.1 Converting algorithm.

line number	i	x_i	stack top bottom	w	lson 1 2 3 4 5	rson 1 2 3 4 5	root
2			5 0				5
3	4	2					
7			0	5		0	
11					2	0	
13			2 0				
3	3	4					
5					2	4 0	
13			4 2 0				
3	2	3					
7			2 0	4	2	4 0 0	
11					3 2	4 0 0	
13			3 2 0				
3	1	1					
7			2 0	3	3 2	4 0 0 0	
9			0	2	0 3 2	4 0 0 0	
11					1 0 3 2	4 0 0 0	
13			1 0				
15			0	1	1 0 3 2	0 4 0 0 0	
16				0	0 1 0 3 2	0 4 0 0 0	

Fig. 3.2 The process of the converting algorithm for $x_1x_2x_3x_4x_5=13425$.

We write stack$\Leftarrow w$ to mean that the value w is inserted on the top of the stack. Similarly, the notation $w\Leftarrow$stack is used to mean that the variable w is set equal to the value at the top of the stack and this value is deleted from the stack. This notation is meaningless, when the stack contains no values. The converting algorithm uses integer variables root, lson(i) and rson(i). The integer variable root points to the root of the binary tree $B(x_1\ x_2 \cdots\ x_n)$. The integer variable lson(i) (rson(i)) points to the left (right) subtree of the root i. If the left (right) subtree of the root i is empty, then the integer variable lson(i) (rson(i)) is set to 0. As an example, the process of the converting algorithm for $x_1\ x_2\ x_3\ x_4\ x_5 = 1\ 3\ 4\ 2\ 5$ is shown in Fig. 3.2.

Theorem 2 The converting algorithm constructs the binary tree $B(x_1\ x_2 \cdots\ x_n)$ corresponding to a given inorder-postorder permutation $x_1\ x_2 \cdots\ x_n$ $(n \ge 2)$.

Proof. The proof proceeds by induction on n.

Basis. $n=2$. The converting algorithm constructs the binary tree B(12); ①✓② and the binary tree B(21); ①＼②.

Inductiv estep. $n>2$. Suppose that $x_n=i$ $(1<i<n)$. By line 2, the elements 0 and x_n are put on the stack and root is set to x_n. Then the substring $x_{i-1} \cdots x_{n-1}$ is processed from right to left. Since $x_{n-1} > x_n (=i)$, by line 5 rson(i) is set to x_{n-1} and by line 13 the element x_{n-1} is put on the stack. When the element x_{i-1} is compared with the top of the stack, by line 7, 8, 9, 10 the elements of the stack, without the element 0, are taken off the stack, because $x_{i-1} < x_j (i \le j \le n)$. Since the element $x_n = i$ is taken off the stack lastly, by line 9 w is set to i and by line 11 lson(i) is set to x_{i-1}. By line 13 the element x_{i-1} is put on the stack. By the inductive assumption, the binary tree corresponding to the substring $x_i \cdots\ x_{n-1}$ is constructed. Its root is x_{n-1}. Then the substring $x_1 \cdots$

x_{i-2} is processed from right to left. By the inductive assumption, the binary tree corresponding to the substring $x_1 \cdots x_{i-1}$ is constructed. Its root is x_{i-1}. Since root$=i$, lson$(i)=x_{i-1}$ and rson$(i)=x_{n-1}$, the binary tree corresponding to $x_1 x_2 \cdots x_n$ is constructed. Its root is $x_n=i$. For $x_n=1, n$, the proof can be done in a similar way. This completes the proof.

Property 3.2 The element x_i $(1 \le i \le n)$ is inserted only once on the top of the stack and delected only once from the stack.

Proof. By the converting algorithm, it is easily shown.

Theorem 3 The converting algorithm terminates in a linear time.

Proof. By Property 3.2, it is obvious.

4. Generating Algorithm for all Binary Trees on n nodes

In this section, we establish an efficient algorithm generating all binary trees on n nodes.

Let a permutation $x=x_1\ x_2 \cdots x_n$ be an inorder-postorder permutation and a permutation $y=y_1\ y_2 \cdots y_n$ be lexically the next inorder-postorder permutation. By Theorem 1, both x and y are stack permutations. Thus we can use a generating algorithm [2, Fig. 3.1] for all stack permutations and its properties. The following property is found in Semba [2, Property 3.1]. Let the index $k(z)$ be the rightmost such that $z_i < z_{i+1}$ in a permutation $z=z_1\ z_2 \cdots z_n$ ($k(z)$ is defined to be zero, if z is lexically the last permutation).

$$k(z)= \max_{1 \le i \le n-1} \{i \mid z_i < z_{i+1}\}.$$

Property 4.1 If $x = x_1\ x_2 \cdots x_n$ is not lexically the last inorder-postorder permutation (stack permutation), then $y_i = x_i$ $(1 \le i \le k(x)-1)$ and $\{y_{k(x)}, \cdots, y_n\} = \{x_{k(x)}, \cdots, x_n\}$.

Let STACK(x_i) be the contents of the stack, when the element x_i is put on the stack in the converting algorithm.

Property 4.2 If $k(x) \ge 2$, then

$$\text{STACK}(x_{k(x)-1}) = \{x_i \mid x_i \in \{x_{k(x)-1}, \cdots, x_n\}, x_i \le x_{k(x)-1}\}$$

Proof. Let the right-hand side of the equation be $R(x)$. By the converting algorithm, it follows immediately STACK$(x_{k(x)-1}) \subseteq R(x)$.

If we assume that $x_j \in R(x)$ and $x_j \notin$ STACK$(x_{k(x)-1})$, then there exists an element x_i such that $x_i < x_j < x_{k(x)-1}$ $(k(x)-1 < i < j)$. This contradicts Property 2.1. Thus we have STACK$(x_{k(x)-1}) \supseteq R(x)$. This completes the proof.

Property 4.3 If $k(x) \ge 2$, then

$$\text{STACK}(x_{k(x)-1}) = \text{STACK}(y_{k(x)-1})$$

Proof. By Property 4.1 and 4.2, it follows immediately.

Since the contents of the stack is monotone increasing, by Property 4.3 the output for the subsequence $y_1 \cdots y_{k(x)-2}$ is the same as the output for the subsequence $x_1 \cdots x_{k(x)-2}$. Thus, when $k(x) \ge 2$, we can use the output for $x_1\ x_2 \cdots x_n$ and save running time. When $k(x)=1$, we cannot use the result of $x_1\ x_2 \cdots x_n$. Thus we have to convert $y_1, y_2 \cdots y_n$ to a binary tree B$(y_1\ y_2 \cdots y_n)$ in a linear time. Therefore, combining the generating algorithm [2, Fig. 3.1] and the converting algorithm, we can establish the generating algorithm for all binary trees on n nodes. It is shown in Fig. 4.1. It uses a procedure output (root, lson, rson) which prints out the binary tree. In order to make our algorithm short and clear, we use a procedure nextperm $(x_1, \cdots, x_n, k)$ which generates the next inorder-postorder permutation (stack permutation) and determines its index $k(x)$. The value of the index $k(x)$ is set to the integer variable k. The integer variable km is used to save the value of the index $k(x)$. We note the index $k(x)$ becomes zero, when the inorder-postorder permutation (stack permutation) is lexically last.

```
1.  begin
2.    {initialize}
3.    for i:=1 to n do begin
4.      x_i:=i; lson(i):=i-1; rson(i):=0
5.    end;
6.    root:=x_n;
7.    output (root, lson, rson);
8.    k:=n-1;
9.    while k>0 do begin
10.     {save the index k(x)}
11.     km:=k;
12.     {generate next inorder-postorder permutation
13.     (stack permutation) and determine its index k(x)}
14.     nextperm (x_1, ..., x_n, k);
15.     stack⇐φ; stack⇐0; stack⇐x_n; root:=x_n;
16.     if km=1 then ii:=1 else ii:=km-1;
17.     for i:=n-1 downto ii do begin
18.       if x_i>stack then
19.         rson (stack):=x_i
20.       else begin
21.         rson (stack):=0; w⇐stack;
22.         while x_i<stack do begin
23.             lson(w):=0; w⇐stack
24.         end;
25.         lson(w):=x_i
26.       end;
27.       stack⇐x_i
28.     end;
29.     if km=1 then begin
30.         rson (stack):=0; w⇐stack;
31.       while w>0 do begin lson(w):=0; w⇐stack end
32.     end;
33.     output (root, lson rson)
34.   end
35. end.
```

Fig. 4.1 Generating algorithm for all binary trees on n nodes.

5. Analysis of Generating Algorithm

In this section, the average running time per binary

tree is shown to be bounded by a constant.

Let $g(n, h)$ be the number of inorder-postorder permutations (stack permutations) $x=x_1, x_2, \cdots, x_n$ such that $k(x)=h$. The following property has been shown by Semba [2, Property 4.4].

Property 5.1 For $n \geq 1$ and $0 \leq h \leq n-1$,

$$g(n, h)=\binom{n-1+h}{h}-\binom{n-1+h}{h-1}$$

For the binary tree corresponding to the lexically first inorder-postorder permutation (stack permutation) $1\ 2 \cdots\ n$ and those binary trees corresponding to inorder-postorder permutations (stack permutations) $y=y_1\ y_2 \cdots\ y_n$ such that $k(x)=1$, we have to perform a constant times n operations. For those binary trees corresponding to inorder-postorder permutations (stack permutations) $y=y_1\ y_2 \cdots\ y_n$ such that $k(x) \geq 2$, the number of operations performed is bounded by a constant times $n-k(x)+1$. Thus, on the average the running time per binary tree is on the order of $T(n)$, where

$$T(n)=\frac{n+\sum_{h=1}^{n-1}(n-h+1)\left\{\binom{n+h-1}{h}-\binom{n+h-1}{h-1}\right\}}{\frac{1}{n+1}\binom{2n}{n}}.$$

Property 5.2 For $n \geq 1$,

$$T(n)<4.$$

Proof. Since

$$\sum_{h=1}^{n-1}(n-h+1)\binom{n+h-1}{h}=\binom{2n}{n-1}+\binom{2n-1}{n-1}-(n+1)$$

and

$$\sum_{h=1}^{n-1}(n-h+1)\binom{n+h-1}{h-1}=\binom{2n}{n-2}+\binom{2n-1}{n-2},$$

$$T(n)=$$

$$\frac{n+\left\{\binom{2n}{n-1}+\binom{2n-1}{n-1}-(n+1)\right\}-\left\{\binom{2n}{n-2}+\binom{2n-1}{n-2}\right\}}{\frac{1}{n+1}\binom{2n}{n}}$$

$$=\frac{\left(\frac{n}{n+1}+\frac{1}{2}-\frac{n(n-1)}{(n+2)(n+1)}-\frac{n-1}{2(n+1)}\right)\binom{2n}{n}-1}{\frac{1}{n+1}\binom{2n}{n}}$$

$$<\frac{4n+2}{n+2}$$

$$<4.$$

Thus, we obtain the above formula.

Theorem 4 The average running time per binary tree is bounded by a constant.

Proof. The average running time per inorder-postorder permutation (stack permutation) has been shown to be bounded by a constant by Semba [2, Theorem 2]. Thus, by Property 5.2 it is easily proved.

Note that we do not count the time needed to print out the binary tree.

6. Concluding Remarks

Various algorithms [3, 4, 5, 6] for generating all binary trees have been proposed. However, the average running time per binary tree is not shown to be bounded by a constant. Thus our algorithm is more efficient than these algorithms.

Acknowledgement

The author would like to thank Prof. T. Shimizu and Prof. A. Nozaki for their hearty encouragement. The referee's comments are also acknowledged.

References

1. KNUTH, D. E. The Art of Computer Programming: Vol. 1: Fundamental Algorithms, 2nd ed., Addison-Wesley, Reading, Mass. (1973).
2. SEMBA, I. Generation of stack sequences in lexicographical order, this jurnai, vol. 5, No. 1, (1982) 17–20.
3. KNOTT, G. D. A numbering system for binary trees, Comm. ACM, Vol. 20, No. 2, (1977) 113–115.
4. RUSKEY, F. and HU, T. C. Generating binary trees lexicographically, SIAM J. Comput., Vol. 6, No. 4, (1977) 745–758.
5. ROTEM, D. and VAROL, Y. L. Generation of binary trees from ballot sequences, J. ACM, Vol. 25, No. 3, (1978) 396–404.
6. SOLOMON, M. and FINKEL, R. A. A note on enumerating binary trees, J. ACM, Vol. 27, No. 1, (1980) 3–5.

(Received September 10, 1981: revised January 21, 1981)