Generation of Stack Sequences in Lexicographical Order

ICHIRO SEMBA*

An efficient algorithm is presented which generates all stack sequences (obtained, by using a stack, from input sequence) in lexicographical order. The average time per stack sequence is shown to be bounded by a constant. Our algorithm is derived from properties of a stack.

1. Introduction

We consider the problem to rearrange, using a stack, a sequence $s_1s_2\cdots s_n$ $(s_1<s_2<\cdots<s_n)$ into a sequence $t_1t_2\cdots t_n$. We say as equence $t_1t_2\cdots t_n$ is a stack sequence on $\{s_1, s_2,\cdots, s_n\}$. Particularly, if $s_i=i$ $(1\le i\le n)$ then a sequence $t_1t_2\cdots t_n$ is called a stack permutation. It is an interesting problem to generate all stack sequences on $\{s_1, s_2,\cdots, s_n\}$. Since there is a one-to-one correspondence between stack sequences and binary trees, all binary trees can be generated. A one-to-one correspondence is shown in Trojanowski [1]. He has developed an efficient generating algorithm for stack permutations in lexicographical order, using a different (although equivalent) definition of stack sequences also his derivation is not made by way of a stack. He has shown the average time per stack sequence is bounded by a constant.

In this paper, we establish an efficient generating algorithm for stack sequences in lexicographical order, using the above definition of stack sequences. Our derivation is made by way of a stack. It is based on the generating algorithm [2] for sequences on $\{s_1, s_2,\cdots, s_n\}$ in lexicographical order. The average time per stack sequence is shown to be bounded by a constant.

2. Preliminaries

In this section, we give definitions, fundamental theorems and examples. We denote by $a=a_1a_2\cdots a_n$ a sequence on $\{s_1, s_2,\cdots, s_n\}$ and by $next(a)$ the next sequence succeeding a sequence $a=a_1a_2\cdots a_n$ in lexicographical order. If a sequence $a=a_1a_2\cdots a_n$ is a stack sequence, then we denote by $NEXT(a)$ the next stack sequence succeeding a stack sequence $a=a_1a_2\cdots a_n$ in lexicographical order. Three indices $k(a)$, $l(a)$ and $m(next(a))$ are defined as follows.

$$k(a)=\begin{cases}\max\limits_{1\le i\le n-1}\{i\,|\,a_i<a_{i+1}\}\\ 0 \quad \text{if a sequence } a=a_1a_2\cdots a_n \text{ is the lexically last sequence.}\end{cases}$$

$k(a)$ is the index of the rightmost pair such that $a_i<a_{i+1}$ in $a_1a_2\cdots a_n$ ($k(a)$ is defined to be zero, if $a=a_1a_2\cdots a_n$ is the lexically last sequence).

$$l(a)=\max_{k(a)<i\le n}\{i\,|\,a_{k(a)}<a_i\}$$

$l(a)$ is the index of the rightmost a_i such that $a_{k(a)}<a_i$ in $a_{k(a)+1}\cdots a_n$ ($l(a)$ is not defined, if $a=a_1a_2\cdots a_n$ is the lexically last sequence).

Let $next(a)=b_1b_2\cdots b_n$.

$$m(next(a))=\max_{k(a)<i\le n}\{i\,|\,b_i<b_{k(a)}\}$$

$m(next(a))$ is the index of the rightmost b_i such that $b_i<b_{k(a)}$ in $b_{k(a)+1}\cdots b_n$ ($m(next(a))$ is not defined, if $a=a_1a_2\cdots a_n$ is the lexically last sequence).

We say a sequence $a=a_1a_2\cdots a_n$ contains a pattern $a_ia_ja_k$, if there are three elements a_i, a_j and a_k such that $a_j<a_k<a_i$ and $i<j<k$. We give a fundamental property of stack sequences.

Theorem 1 A sequence $a_1a_2\cdots a_n$ is a stack sequence, if and only if there are no patterns.

The proof is omitted, since a similar property and its proof is found in Knuth [3, §2.2.1, Ex 5].

Example Let $n=4$. We show all sequences $a_1a_2\cdots a_n$ in lexicographical order and three functions, $k(a)$, $l(a)$ and $m(next(a))$. Stack sequences are marked 'O' and patterns are listed.

$a_1a_2a_3a_4$	$k(a)$	$l(a)$	$m(next(a))$	mark	pattern
$s_1s_2s_3s_4$	3	4	4	O	
$s_1s_2s_4s_3$	2	4	3	O	
$s_1s_3s_2s_4$	3	4	4	O	
$s_1s_3s_4s_2$	2	3	4	O	
$s_1s_4s_2s_3$	3	4	4		$s_4s_2s_3$
$s_1s_4s_3s_2$	1	4	2	O	
$s_2s_1s_3s_4$	3	4	4	O	
$s_2s_1s_4s_3$	2	4	3	O	
$s_2s_3s_1s_4$	3	4	4	O	
$s_2s_3s_4s_1$	2	3	4	O	
$s_2s_4s_1s_3$	3	4	4		$s_4s_1s_3$
$s_2s_4s_3s_1$	1	3	3	O	
$s_3s_1s_2s_4$	3	4	4		$s_3s_1s_2$
$s_3s_1s_4s_2$	2	4	3		$s_3s_1s_2$
$s_3s_2s_1s_4$	3	4	4	O	
$s_3s_2s_4s_1$	2	3	4	O	

*Department of Pure and Applied Sciences, College of General Education, University of Tokyo, Komaba, Meguro-ku, Tokyo 153, Japan.

$s_3s_4s_1s_2$	3	4	4		$s_3s_1s_2$, $s_4s_1s_2$
$s_3s_4s_2s_1$	1	2	4	O	
$s_4s_1s_2s_3$	3	4	4		$s_4s_1s_2$, $s_4s_1s_3$, $s_4s_2s_3$
$s_4s_1s_3s_2$	2	4	3		$s_4s_1s_3$, $s_4s_1s_2$
$s_4s_2s_1s_3$	3	4	4		$s_4s_2s_3$, $s_4s_1s_3$
$s_4s_2s_3s_1$	2	3	4		$s_4s_2s_3$
$s_4s_3s_1s_2$	3	4	4		$s_4s_1s_2$, $s_3s_1s_2$
$s_4s_3s_2s_1$	0			O	

3. Generating Algorithm

In this section, we shall establish an efficient algorithm generating all stack sequences in lexicographical order.

By the definition of lexicographical order, we obtain the following relation between the sequence $a=a_1a_2\cdots a_n$ and the sequence $next(a)=b_1b_2\cdots b_n$.

Property 3.1 If the sequence $a=a_1a_2\cdots a_n$ is not the lexically last sequence, then

$$b_i=a_i \quad (1\le i<k(a))$$
$$b_{k(a)}=a_{l(a)}$$
$$\{b_i|k(a)<i\le n\}=\{a_i|k(a)\le i<l(a),\ l(a)<i\le n\}$$
$$b_{k(a)}>b_{k(a)+1}<\cdots<b_n$$

Namely, two elements $a_{k(a)}$ and $a_{l(a)}$ are exchanged. Then, the subsequence string $a_{k(a)+1}\cdots a_n$ is reversed.

For example, let $a=s_2s_1s_5s_7s_6s_4s_3$. Since the index $k(a)=3$ and $l(a)=5$, two elements $a_3=s_5$ and $a_5=s_6$ are exchanged. Then, the subsequence string $s_7s_5s_4s_3$ is reversed. Thus we obtain $next(a)=s_2s_1s_6s_3s_4s_5s_7$.

In what follows, we assume the sequence $a=a_1a_2\cdots a_n$ is a stack sequence and $k(a)\neq 0$. If $next(a)=b_1b_2\cdots b_n$ is not a stack sequence, then by Theorem 1 $next(a)$ contains a pattern $b_ub_vb_w$ such that $b_v<b_w<b_u$ and $u<v<w$. If we fix our attention on the index $k(a)$, then we can see that a pattern $b_ub_vb_w$ satisfies one of the following four cases.

Case 1. $1\le u<v<w<k(a)$.
Case 2. $1\le u<v<k(a)$ and $k(a)\le w\le n$.
Case 3. $1\le u<k(a)$ and $k(a)\le v<w\le n$.
Case 4. $k(a)\le u<v<w\le n$.

Property 3.2 If $next(a)=b_1b_2\cdots b_n$ is not a stack sequence, then a pattern $b_ub_vb_w$ does not satisfy Case 1 and 2.

Proof. If a pattern $b_ub_vb_w$ satisfies Case 1, then by Property 3.1 we have $b_u=a_u$, $b_v=a_v$ and $b_w=a_w$. By Case 1 we obtain a pattern $a_ua_va_w$ such that $a_v<a_w<a_u$ and $u<v<w$. This contradicts the assumption that the sequence $a_1a_2\cdots a_n$ is a stack sequence. Thus a pattern $b_ub_vb_w$ does not satisfy Case 1. If a pattern $b_ub_vb_w$ satisfies Case 2, then by Property 3.1 we have $b_u=a_u$, $b_v=a_v$ and $b_w=a_i$ $(k(a)\le i\le n)$. By Case 2 we obtain a pattern $a_ua_va_i$ such that $a_v<a_i<a_u$ and $u<v<i$. This contradicts the assumption that the sequence $a_1a_2\cdots a_n$ is a stack sequence. Thus a pattern $b_ub_vb_w$ does not satisfy Case 2. This completes the proof.

Now we consider to determine whether or not $next(a)=b_1b_2\cdots b_n$ is a stack sequence.

Property 3.3 If $m(next(a))=k(a)+1$, then $next(a)$ is a stack sequence.

Proof. We assume that $next(a)$ contains a pattern $b_ub_vb_w$ such that $b_v<b_w<b_u$ and $u<v<w$, so we derive a contradiction.

By Property 3.2, a pattern $b_ub_vb_w$ does not satisfy Case 1 and 2. We prove a pattern $b_ub_vb_w$ does not satisfy Case 3 and 4.

If a pattern $b_ub_vb_w$ satisfies Case 3, then there is a pattern $b_ub_{k(a)+1}b_{k(a)+2}$ such that $b_{k(a)+1}<b_{k(a)+2}<b_u$ and $1\le u<k(a)$, because the subsequence $b_{k(a)+1}\cdot b_{k(a)+2}\cdots b_n$ is monotone increasing. Since $m(next(a))=k(a)+1$, we have $b_{k(a)+1}=a_{k(a)}$. By Property 3.1 we have $b_u=a_u$ and $b_{k(a)+2}=a_i$, where $k(a)<i\le n$ and $i\neq l(a)$. Namely, the sequence $a=a_1a_2\cdots a_n$ contains a pattern $a_ua_{k(a)}a_i$ such that $a_{k(a)}<a_i<a_u$ and $1\le u<k(a)<i\le n$, $i\neq l(a)$. This contradicts the assumption the sequence $a_1a_2\cdots a_n$ is a stack sequence. Thus a pattern $b_ub_vb_w$ does not satisfy Case 3.

If a pattern $b_ub_vb_w$ satisfies Case 4, then we have $m(next(a))>k(a)+1$. This contradicts the assumption that $m(next(a))=k(a)+1$. Thus a pattern $b_ub_vb_w$ does not satisfy Case 4.

Consequently, we have proved that a pattern $b_ub_vb_w$ does not satisfy any of the four cases. However, this contradicts the fact that a pattern $b_ub_vb_w$ has to satisfy one of the four cases. Thus $next(a)$ is a stack sequence. This completes the proof.

Property 3.4 If $m(next(a))>k(a)+1$, then $next(a)$ is not a stack sequence.

Proof. Obvious.

When $next(a)=b_1b_2\cdots b_n$ is not a stack sequence, we reverse the subsequence $b_{k(a)+1}\cdots b_{m(next(a))}$ and construct the sequence $c=c_1c_2\cdots c_n$.

Property 3.5

$$c_i=\begin{cases} b_i & (1\le i\le k(a),\ m(next(a))<i\le n) \\ b_{m(next(a))+k(a)+1-i} & (k(a)<i\le m(next(a))) \end{cases}$$

Proof. Obvious.

Property 3.6 The sequence $c=c_1c_2\cdots c_n$ is a stack sequence.

Proof. We assume that $c=c_1c_2\cdots c_n$ contains a pattern $c_uc_vc_w$ such that $c_v<c_w<c_u$ and $u<v<w$ and derive a contradiction. In a similar way as Property 3.2, we can show a pattern $c_uc_vc_w$ does not satisfy Case 1 and 2. We prove a pattern $c_uc_vc_w$ does not satisfy Case 3 and 4.

If a pattern $c_uc_vc_w$ satisfies Case 3, then there is a pattern $c_uc_{k(a)+1}c_{m(next(a))+1}$ such that $c_{k(a)+1}<c_{m(next(a))+1}<c_u$ and $1\le u<k(a)<k(a)+1<m(next(a))+1\le n$. By Property 3.1 and 3.5, we have $c_{k(a)+1}=b_{m(next(a))}=a_{k(a)}$, $c_u=b_u=a_u$, $c_{m(next(a))+1}=b_{m(next(a))+1}$ and $b_{m(next(a))+1}=$

a_i, where $k(a)<i\le n$, $i\ne l(a)$. Namely, the sequence $a=a_1a_2\cdots a_n$ contains a pattern $a_ua_{k(a)}a_i$ such that $a_{k(a)}<a_i<a_u$ and $1\le u<k(a)<i\le n$, $i\ne l(a)$. This contradicts the assumption the sequence $a=a_1a_2\cdots a_n$ is a stack sequence. Thus a pattern $c_uc_vc_w$ does not satisfy Case 3.

Since the subsequence $c_{k(a)}\cdots c_{m(next(a))}$ is monotone decreasing and the subsequence $c_{m(next(a))+1}\cdots c_n$ is monotone increasing and $c_{k(a)}<c_{m(next(a))+1}$, it follows that a pattern $c_uc_vc_w$ does not satisfy Case 4.

Thus we have shown a pattern $c_uc_vc_w$ does not satisfy any of the four cases. This contradicts the fact that a pattern $c_uc_vc_w$ has to satisfy one of the four cases. Therefore the sequence $c=c_1c_2\cdots c_n$ is a stack sequence. This completes the proof.

We denote by $d=d_1d_2\cdots d_n$ the sequence which succeeds $next(a)$ and precedes the sequence $c=c_1c_2\cdots c_n$.

Property 3.7 The sequence $d=d_1d_2\cdots d_n$ is not a stack sequence.

Proof. We note there are two elements d_i and d_j such that $d_i<d_j<d_{k(a)}$ and $k(a)<i<j\le n$. This means there is a pattern $d_{k(a)}d_id_j$. Thus the sequence $d_1d_2\cdots d_n$ is not a stack sequence. This completes the proof.

By Property 3.3, 3.4, 3.6 and 3.7, we can easily obtain the following property.

Property 3.8

$$NEXT(a)=\begin{cases} next(a) & \text{if } m(next(a))=k(a)+1 \\ c & \text{if } m(next(a))>k(a)+1 \end{cases}$$

For example, let $a=s_2s_1s_5s_7s_6s_4s_3$. We have obtained $next(a)=s_2s_1s_6s_3s_4s_5s_7$. The sequence a is a stack sequence, because there are no patterns. The sequence $next(a)$ is not a stack sequence, because there is a pattern $s_6s_3s_4$. Since the index $m(next(a))=6$, the subsequence $s_3s_4s_5$ is reversed. Thus we obtain $NEXT(a)=s_2s_1s_6s_5s_4s_3s_7$.

Now we show three properties which are useful for shortening the running time.

Property 3.9 $m(next(a))=n+1+k(a)-l(a)$

Proof. By Property 3.1 we have

$$\{b_i|k(a)+1\le i\le m(next(a))\}=\{a_{k(a)}\}\cup\{a_i|l(a)+1\le i\le n\}.$$

Thus we obtain the above formula. This completes the proof.

Property 3.10

$$k(NEXT(a))=\begin{cases} n-1 & \text{if } m(next(a))<n \\ k(a)-1 & \text{if } m(next(a))=n \end{cases}$$

Proof. Suppose that $NEXT(a)=next(a)$. If $m(next(a))<n$, then we have $k(next(a))=n-1$ because $b_{m(next(a))}<\cdots<b_n$. Let $m(next(a))=n$. If $b_{k(a)}<b_{k(a)-1}$, then by Property 3.1 we have $b_{k(a)-1}=a_{k(a)-1}$, $b_{k(a)}=a_{l(a)}$ and $a_{k(a)}<a_{l(a)}$. Namely, the sequence $a=a_1a_2\cdots a_n$ contains a pattern $a_{k(a)-1}a_{k(a)}a_{l(a)}$ such that $a_{k(a)}<a_{l(a)}<a_{k(a)-1}$ and $k(a)-1<k(a)<l(a)$. This contradicts the assumption that the sequence $a=a_1a_2\cdots a_n$ is a stack sequence. Thus we have $b_{k(a)-1}<b_{k(a)}$. Since $b_{k(a)}>\cdots>b_n$, we have $k(next(a))=k(a)-1$.

When $NEXT(a)=c$, the proof is similar and therefore is omitted. This completes the proof.

Property 3.11 If $l(a)=k(a)+1$, then $NEXT(a)=a_1\cdots a_{k(a)-1}a_{k(a)+1}a_{k(a)}a_{k(a)+2}\cdots a_n$.

Proof. If $k(a)=n-1$, then by Property 3.1 we have $next(a)=a_1\cdots a_{n-2}a_na_{n-1}$. Since $m(next(a))=n$, we have $m(next(a))=k(a)+1$. Thus by Property 3.8, it follows that $NEXT(a)=next(a)=a_1\cdots a_{n-2}a_na_{n-1}$. If $1\le k(a)<n-1$, then by Property 3.1 two elements $a_{k(a)}$ and $a_{k(a)+1}$ are exchanged and the subsequence $a_{k(a)}a_{k(a)+2}\cdots a_n$ is reversed and $next(a)=a_1\cdots a_{k(a)-1}a_{k(a)+1}a_n\cdots a_{k(a)+2}a_{k(a)}$ is obtained. Since $m(next(a))=n$, we have $m(next(a))>k(a)+1$. Thus by Property 3.8, the subsequence $a_n\cdots a_{k(a)+2}a_{k(a)}$ is reversed and $NEXT(a)=a_1\cdots a_{k(a)-1}\cdot a_{k(a)+1}a_{k(a)}a_{k(a)+2}\cdots a_n$ is obtained. This completes the proof.

Namely, if $l(a)=k(a)+1$, then we have only to exchange two elements $a_{k(a)}$ and $a_{k(a)+1}$.

By Property 3.8, 3.9, 3.10 and 3.11, we can construct an efficient algorithm generating all stack sequences in lexicographical order. It is shown in Fig. 3.1 in a PASCAL-like notation. It uses a procedure reverse $(a_i,\cdots,a_j)$ which reverses the subsequence $a_i\cdots a_j$ and a procedure output $(a_1,\cdots,a_n)$ which prints out the sequence $a_1a_2\cdots a_n$. We write $a_i\Leftrightarrow a_j$ to mean that we exchange a_i and a_j.

In order to gain a better understanding of our algorithm, we shall briefly describe it. Let a sequence $a=a_1a_2\cdots a_n$ be a stack sequence. Since the next sequence $next(a)$ is not always the next stack sequence $NEXT(a)$, we have to examine whether or not $next(a)$ is a stack sequence. By Property 3.3 and 3.4, this problem is solved. If $next(a)$ is not a stack sequence, we have to construct $NEXT(a)$. By Property 3.5, 3.6 and 3.7, this problem is solved. The sequence $s_1s_2\cdots s_n$ is obviously a

```
begin
  for i:=1 to n do a_i:=s_i;
  output (a_1,•••, a_n);
  k:=n-1;
  while k>0 do begin
    {a=a_1a_2•••a_n is not lexically last sequence}
    {determine l(a)}
    l:=n; while a_k>a_l do l:=l-1;
    {exchange a_k(a) and a_l(a)}
    a_k<=>a_l;
    if l=k+1 then
      {we have only to exchange a_k(a) and a_l(a)}
      {determine k(NEXT(a))}
      k:=k-1
    else begin
      reverse (a_k+1,•••, a_n);
      {determine m(next(a))}
      m:=n+1+k-l;
      if m>k+1 then reverse (a_k+1,•••, a_m);
      {determine k(NEXT(a))}
      k:=n-1
    end;
    output (a_1,•••, a_n)
  end
end.
```

Fig. 3.1 Generating algorithm.

stack sequence. Thus our algorithm can generate all stack sequences in lexicographical order.

4. Analysis of Generating Algorithm

In this section, we show the average time per stack sequence is bounded by a constant. We give some preliminary properties.

We denote by $f(n, i)$ the number of stack sequences $a_1 a_2 \cdots a_n$ such that $a_i = s_n$. It is obvious that $f(1, 1) = 1$. We obtain the following recurrence relations.

Property 4.1 For $n \geq 2$,

$$f(n, i) = \sum_{j=1}^{i} f(n-1, j) \quad (1 \leq i \leq n-1)$$

$$f(n, n) = \sum_{j=1}^{n-1} f(n-1, j)$$

Proof. If we remove the element $a_i = s_n$ from the stack sequence $a_1 \cdots a_i \cdots a_n$, then the remaining sequence is a stack sequence. Let $a_j = s_{n-1}$. The index j satisfies $1 \leq j < i$ or $j = i+1$, because the element s_{n-1} is moved from a stack before the element s_n or immediately after the element s_n. When $1 \leq j < i$, the number of remaining sequences is $f(n-1, j)$. When $j = i+1$, the number of remaining sequences is $f(n-1, i)$. Therefore, we obtain $f(n, i) = \sum_{j=1}^{i} f(n-1, j)$ $(1 \leq i \leq n-1)$. We can prove that $f(n, n) = \sum_{j=1}^{n-1} f(n-1, j)$ in a similar way. This completes the proof.

We define a binomial coefficient $\binom{p}{q}$ is zero, if either $q > p$ or $q < 0$.

Property 4.2 For $n \geq 1$ and $1 \leq i \leq n$,

$$f(n, i) = \binom{n-2+i}{i-1} - \binom{n-2+i}{i-2}$$

Proof. Since the solution of the above recurrence relation is uniquely determined, it is sufficient to show our formula satisfies the recurrence relation. It is easily shown. This completes the proof.

Property 4.3 The index $k(a) = i-1$, if and only if $a_i = s_n$.

Proof. (If) When the element s_n is moved from the stack, the contents of the stack is monotone increasing. Thus we have $a_{i+1} > \cdots > a_n$. Since the element $a_i = s_n$ is the largest, we have $a_{i-1} < a_i = s_n$. Therefore, we have $k(a) = i-1$.

(Only if) By the definition of the index $k(a)$, it follows that $a_j \neq s_n$ $(j = i-1, i+1 \leq j \leq n)$. If $a_j = s_n$ $(1 \leq j < i-1)$, then there is a pattern $a_j a_{i-1} a_i$ such that $a_{i-1} < a_i < a_j = s_n$ and $j < i-1 < i$. This contradicts the assumption that the sequence $a = a_1 a_2 \cdots a_n$ is a stack sequence. Therefore we have $a_i = s_n$.

Let $g(n, h)$ be the number of stack sequences $a = a_1 a_2 \cdots a_n$ such that $k(a) = h$. By Property 4.2 and 4.3, we have the following property.

Property 4.4 For $n \geq 1$ and $0 \leq h \leq n-1$,

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

If we implement the generating algorithm in a straightforward manner, then the running time per stack sequence is bounded by a constant times $n-h$. In the worst case this is $O(n)$, but on the average it is on the order of $T(n)$, where

$$T(n) = \frac{\sum_{h=0}^{n-1} (n-h) g(n, h)}{\frac{1}{n+1}\binom{2n}{n}}$$

Property 4.5 For $n \geq 1$,

$$T(n) = \frac{3n}{n+2} < 3$$

Proof. Since

$$\begin{aligned}
&\sum_{h=0}^{n-1} (n-h) g(n, h) \\
&= \frac{n}{n+1}\binom{2n}{n} - \sum_{h=0}^{n-1} h\left(\binom{n-1+h}{h} - \binom{n-1+h}{h-1}\right) \\
&= \frac{n}{n+1}\binom{2n}{n} - (n-1)\binom{2n-1}{n-1} - n\binom{2n-1}{n-2} + \binom{2n-1}{n-3} \\
&= \frac{3n}{(n+1)(n+2)}\binom{2n}{n},
\end{aligned}$$

we have the above formula. This completes the proof.

Theorem 2 The average time per stack sequence is bounded by a constant.

Proof. By Property 4.5, it is easily proved.

Note that we do not count the time needed to print out the stack sequence.

5. Concluding Remarks

We have shown a different approach to stack sequence generation.

We shall examine our algorithm and Trojanowski's algorithm. Both average time per stack sequence are bounded by a constant. His algorithm must save initial elements $s_1, s_2, \cdots, s_n$. On the other hand, our algorithm does not have to save them. Thus it follows that our algorithm works with less memory units.

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. TROJANOWSKI, A. E. Ranking and listing algorithms for k-ary trees, *SIAM J. Computing*, 7 (1978), 492–509.
2. SEDGEWICK, R. Permutation generation methods, *Computing Surveys*, 9, 2 (1977), 137–164.
3. KNUTH, D. E. The Art of Computer Programming, *Addison-Wesley, Reading, Mass.* 1, 1973.

(Received March 3, 1981: revised July 31, 1981)