Generation of Permutations by Using a Stack or a Queue

ICHIRO SEMBA*

We consider the problems of generating permutations on $\{1, 2, \cdots, n\}$, by using a stack or a queue. Efficient generating algorithms are presented. The average time per permutation is shown to be bounded by a constant. Several combinatorial properties are proved.

1. Introduction

We consider the problems of generating permutations on $\{1, 2, \cdots, n\}$, by using a stack or a queue. These problems are closely related together and interesting examples of the use of computers in combinatorial mathematics.

A stack is a list for which insertions and deletions are always made at one end, called the top. We consider two elementary operations, S and X. S stands for "insert the next element of a permutation $12\cdots n$ into the stack" and X stands for "delete the element from the top and move it to the output". Operation X is meaningless when the stack is empty.

A stack realizable permutation on $\{1, 2, \cdots, n\}$ is defined to be a permutation which is constructed from a permutation $12\cdots n$, using a stack. A stack sortable permutation on $\{1, 2, \cdots, n\}$ is defined to be a permutation from which a permutation $12\cdots n$ is constructed, using a stack. In Rotem[1], a class of stack realizable permutations on $\{1, 2, \cdots, n\}$ is denoted by SR_n and a class of stack sortable permutations on $\{1, 2, \cdots, n\}$ is denoted by SS_n.

For example, a stack realizable permutation 2431 is constructed from a permutation 1234 by performing the sequence of operations, SSXSSXXX and a permutation 1234 is constructed from a stack sortable permutation 2143 by performing the sequence of operations, SSXXSSXX.

A queue is a list for which all insertions are done at one end, called the rear, and all deletions are done at the other end, called the front. We consider three elementary operations, Q, X and Y. Q stands for "insert the next element of a permutation $12\cdots n$ into the queue", and X stands for "delete the element from the front and move it to the output", and Y stands for "move the next element of a permutation $12\cdots n$ to the output." Operation X is meaningless when the queue is empty.

A queue realizable permutation on $\{1, 2, \cdots, n\}$ is defined to be a permutation which is constructed from a permutation $12\cdots n$, using a queue. A queue sortable permutation on $\{1, 2, \cdots, n\}$ is defined to be a permutation from which a permutation $12\cdots n$ is constructed, using a queue. We denote by QR_n a class of queue realizable permutations on $\{1, 2, \cdots, n\}$ and by QS_n a class of queue sortable permutations on $\{1, 2, \cdots, n\}$.

For example, a queue realizable permutation 2413 is constructed from a permutation 1234 by performing the sequence of operations, QYQYXX and a permutation 1234 is constructed from a queue sortable permutation 2413 by performing the sequence of operations, QQYXYX.

Many generating algorithms[2] for all permutations on $\{1, 2, \cdots, n\}$ have been published. Trojanowski[3] has proposed the direct insertion order on the set of all permutations on $\{1, 2, \cdots, n\}$. This order is simple and natural. He has applied this order to generating all stack realizable permutations on $\{1, 2, \cdots, n\}$ and established an efficient generating algorithm. He has shown the average time per stack realizable permutation is bounded by a constant.

Our generating algorithm for all stack realizable permutations on $\{1, 2, \cdots, n\}$ is also based on the direct insertion order, however, different from Trojanowski's generating algorithm. Our idea can be applied to generating all queue realizable permutations on $\{1, 2, \cdots, n\}$. The average time per stack(queue) realizable permutation is proved to be bounded by a constant.

We propose another direct insertion order and establish a generating algorithm for all stack(queue) sortable permutations on $\{1, 2, \cdots, n\}$. The average time per stack(queue) sortable permutation is proved to be bounded by a constant.

Since these generating algorithms are efficient, they can be employed for systematic generation of combinatorial objects which are one-to-one correspondence with stack (queue) realizable permutations or stack(queue) sortable permutations.

2. Generation of All Permutations on $\{1, 2, \cdots, n\}$ in Direct Insertion Order

The purpose of this section is to establish a generating algorithm for all permutations on $\{1, 2, \cdots, n\}$ in direct insertion order proposed by Trojanowski. The average time per permutation is shown to be bounded by a constant. This algorithm forms the foundation of a

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

generating algorithm for all stack(queue) realizable permutations.

Let $n \geq 2$. For each permutation on $\{1, 2, \cdots, n-1\}$, n permutations on $\{1, 2, \cdots, n\}$ are generated by inserting the new element n into all possible positions from right to left. Thus, we may obtain all permutations on $\{1, 2, \cdots, n\}$, as level n in a tree(permutation tree) in Fig. 2.1. The root 1 is defined to be level 1.

The direct insertion order on the set of all permutations on $\{1, 2, \cdots, n\}$ is defined by moving from left to right along level n of this permutation tree. Thus, given a permutation $a_1a_2\cdots a_n$ on $\{1, 2, \cdots, n\}$, where $a_1a_2\cdots a_n \neq n\cdots 21$, we can define its immediate successor $b_1b_2\cdots b_n$ as follows.

(1) If $a_i = n$ $(1 < i \leq n)$, then we exchange a_{i-1} and a_i. Thus, we have $b_j = a_j$ $(1 \leq j < i-1, i < j \leq n)$, $b_{i-1} = a_i$ and $b_i = a_{i-1}$.

(2) If $a_1 = n$, then let a permutation $c_1c_2\cdots c_{n-1}$ on $\{1, 2, \cdots, n-1\}$ be the immediate successor of a permutation $a_2a_3\cdots a_n$ on $\{1, 2, \cdots, n-1\}$. Then, $b_j = c_j$ $(1 \leq j \leq n-1)$ and $b_n = n$.

The first permutation is defined to be $12\cdots n$.

Our fundamental idea is to traverse the permutation tree in inorder. A generating algorithm is described in a PASCAL-like notation and shown in Fig. 2.2. A permutation $a_1a_2\cdots a_n$ on $\{1, 2, \cdots, n\}$ is represented by an integer array $a[1\cdots n]$. We denote by x_i $(1 \leq i \leq n)$ the position in which the element i is located at level i of the permutation tree. Since x_1 is always equal to 1, we use an integer array $x[2\cdots n]$ to represent x_i $(2 \leq i \leq n)$. The procedure output(a) prints out the permutation $a_1a_2\cdots a_n$. For $n=4$, the result of this algorithm is shown in Fig. 2.3.

Fig. 2.1 A permutation tree.

```
1.  begin
2.    a[1]:=1;
3.    for j:=2 to n do begin a[j]:=j; x[j]:=j end;
4.  1:
5.    output (a);
6.    i:=n;
7.    repeat
8.      i:=i-1; a[i+1]:=a[i]; a[i]:=n;
9.      output (a);
10.   until i=1;
11.   k:=n;
12.   repeat
13.     for j:=i+1 to k do a[j-1]:=a[j];
14.     a[k]:=k; x[k]:=k;
15.     k:=k-1;
16.     if k=1 then goto 2;
17.     i:=x[k]
18.   until i>1;
19.   i:=i-1; a[i+1]:=a[i]; a[i]:=k; x[k]:=i;
20.   goto 1;
21. 2:
22. end.
```

Fig. 2.2 The generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

Now we show the average time per permutation is bounded by a constant. We denote by $f(n)$ the number of times the instruction in the for loop(line 13) may be executed and by $g(n)$ the number of times other instructions in the repeat-until loop(line 12–18) may be executed, in order to generate all permutations on $\{1, 2, \cdots, n\}$. Since other instructions are executed at most once per permutation on $\{1, 2, \cdots, n\}$, we have only to show that two functions $f(n)/n!$ and $g(n)/n!$ are bounded by a constant.

Property 2.1 Let $n \geq 2$.

(1) $f(n) = n! - 1$

(2) $g(n) = (n-1)! + (n-2)! + \cdots + 2! + 1!$

Proof. Let $t_1t_2\cdots t_{i-1}$ be a node of the permutation tree at level $i-1$. The instruction in the for loop is executed, when an immediate successor of $t_1t_2\cdots t_{i-1}$ is constructed. The number of execution times is equal to the number of children of a node $t_1t_2\cdots t_{i-1}$ minus one. Since the number of nodes of level i is $i!$, total number of execution times between level i and level $i-1$ is $i! - (i-1)!$. By the definition of $f(i)$, we have the following recurrence relation.

$f(i) = f(i-1) + i! - (i-1)!$ $\quad (2 < i \leq n)$

$f(2) = 1$

In a similar way, we have the following recurrence

	$a_1a_2a_3a_4$	$x_1x_2x_3x_4$
1:	1 2 3 4	1 2 3 4
2:	1 2 4 3	1 2 3 3
3:	1 4 2 3	1 2 3 2
4:	4 1 2 3	1 2 3 1
5:	1 3 2 4	1 2 2 4
6:	1 3 4 2	1 2 2 3
7:	1 4 3 2	1 2 2 2
8:	4 1 3 2	1 2 2 1
9:	3 1 2 4	1 2 1 4
10:	3 1 4 2	1 2 1 3
11:	3 4 1 2	1 2 1 2
12:	4 3 1 2	1 2 1 1
13:	2 1 3 4	1 1 3 4
14:	2 1 4 3	1 1 3 3
15:	2 4 1 3	1 1 3 2
16:	4 2 1 3	1 1 3 1
17:	2 3 1 4	1 1 2 4
18:	2 3 4 1	1 1 2 3
19:	2 4 3 1	1 1 2 2
20:	4 2 3 1	1 1 2 1
21:	3 2 1 4	1 1 1 4
22:	3 2 4 1	1 1 1 3
23:	3 4 2 1	1 1 1 2
24:	4 3 2 1	1 1 1 1

Fig. 2.3 The result of a generating algorithm for all permutations on $\{1, 2, 3, 4\}$.

relation, with respect to the quantity $g(n)$.

$g(i)=g(i-1)+(i-1)!$ $(2<i\leq n)$

$g(2)=1$

Solving these recurrence relations, we have the above results.

Theorem 2.1 Let $n\geq 2$.

The average time per permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.

Proof. By Property 2.1, it is proved.

Note that we do not consider the time needed to print out the permutations.

3. Generation of All Stack Realizable Permutations on $\{1, 2, \cdots, n\}$

In this section, we will establish a generating algorithm for all stack realizable permutations on $\{1, 2, \cdots, n\}$ and show the average time per stack realizable permutation is bounded by a constant.

We give a basic property of stack realizable permutations on $\{1, 2, \cdots, n\}$.

Property 3.1 Let $n\geq 3$.

A permutation on $\{1, 2, \cdots, n\}$ $a_1a_2\cdots a_n$ is a stack realizable permutation, if and onl0 if there are no subsequences $a_ia_ja_k$ such that $a_j<a_k<a_i$ $(i<j<k)$.

The proof is omitted, since a similar property and its proof is found in Knuth[4, Sec. 2.2.1, Ex5]. Considering this property well, we notice that it suggests the method of generating all stack realizable permutations on $\{1, 2, \cdots, n\}$.

Property 3.2 Let $n\geq 2$.

(1) If a permutation $a_1a_2\cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ is not a stack realizable permutation, then no stack realizable permutations on $\{1, 2, \cdots, n\}$ can be constructed by inserting the new element n into possible positions.

(2) If a permutation $a_1a_2\cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ is a stack realizable permutation, then stack realizable permutations on $\{1, 2, \cdots, n\}$ can be constructed by inserting the new element n at the right of a_{n-1} and at the left of a_{n-1} and a_i $(1\leq i\leq n-2)$, where $a_j>a_{j+1}$ $(i\leq j\leq n-2)$.

Proof. (1) Obvious. (2) By Property 3.1, the new element n has to be inserted so that subsequences na_ua_v $(a_u<a_v<n,\ u<v)$ may not be constructed. Thus, the new element n can be inserted at the right (left) of a_{n-1}. Suppose that the new element n can be inserted at the left of a_i $(1\leq i\leq n-2)$, then two elements a_u and $a_v(a_u<a_v,\ i\leq u<v)$ must not exist, that is, $a_j>a_{j+1}$ for all j, $i\leq j\leq n-2$.

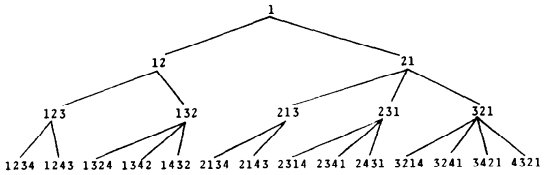

Fig. 3.1 A stack realizable permutation tree.

By Property 3.2, we can obtain all stack realizable permutations on $\{1, 2, \cdots, n\}$, as level n in a tree(stack realizable permutation tree) in Fig. 3.1. The root 1 is defined to be level 1. The new element n is assumed to be inserted from right to left.

To generate successively all stack realizable permutations on $\{1, 2, \cdots, n\}$, we consider to move from left to right along level n of the stack realizable permutation tree. Thus, given a stack realizable permutation $a_1a_2\cdots a_n$ of $\{1, 2, \cdots, n\}$, where $a_1a_2\cdots a_n\neq n\cdots 21$, we can define its immediate successor $b_1b_2\cdots b_n$ as follows.

(1) If $a_n=n$, then we exchange a_{n-1} and a_n.

(2) If $a_i=n$ and $a_{i-1}>a_{i+1}$ $(1\leq i<n)$, then we exchange a_{i-1} and a_i.

(3) If $a_i=n$ and $a_{i-1}<a_{i+1}$ $(1\leq i<n)$, then let a permutation $c_1c_2\cdots c_{n-1}$ on $\{1, 2, \cdots, n-1\}$ be the immediate successor of a stack realizable permutation $a_1\cdots a_{i-1}a_{i+1}\cdots a_n$ on $\{1, 2, \cdots, n-1\}$. Then, $b_j=c_j$ $(1\leq j\leq n-1)$ and $b_n=n$.

The first stack realizable permutation on $\{1, 2, \cdots, n\}$ is defined to be $12\cdots n$. The element a_0 is set to zero to detect that the index i is equal to 1.

Our fundamental idea is to traverse the stack realizable permutation tree in inorder. Thus, a generating algorithm for all stack realizable permutation on $\{1, 2, \cdots, n\}$ is established by making the following partial revisions for the generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

(1) line 2 $a[1]:=1;$ → $a[0]:=0;$ $a[1]:=1;$
(2) line 10 until $i=1;$ → until $a[i-1]<a[i+1];$
(3) line 18 until $i>1;$ → until $(i=k)$ or $(a[i-1]>a[i+1]);$

The element a_0 is represented by $a[0]$. Thus, we use an integer array $a[0\cdots n]$. The result of a generating algorithm for all stack realizable permutations on $\{1, 2, 3, 4\}$ is shown in Fig. 3.2.

Now we show the average time per stack realizable permutation is bounded by a constant. Two functions $f(n)$ and $g(n)$ are defined for stack realizable permutation on $\{1, 2, \cdots, n\}$ as they are for permutations on $\{1, 2, \cdots, n\}$. Let $C_n=\binom{2n}{n}/(n+1)$ be Catalan number.

Property 3.3 Let $n\geq 2$.

(1) $f(n)=C_n-1$

	$a_1a_2a_3a_4$	$x_1x_2x_3x_4$
1:	1 2 3 4	1 2 3 4
2:	1 2 4 3	1 2 3 3
3:	1 3 2 4	1 2 2 4
4:	1 3 4 2	1 2 2 3
5:	1 4 3 2	1 2 2 2
6:	2 1 3 4	1 1 3 4
7:	2 1 4 3	1 1 3 3
8:	2 3 1 4	1 1 2 4
9:	2 3 4 1	1 1 2 3
10:	2 4 3 1	1 1 2 2
11:	3 2 1 4	1 1 1 4
12:	3 2 4 1	1 1 1 3
13:	3 4 2 1	1 1 1 2
14:	4 3 2 1	1 1 1 1

Fig. 3.2 The result of a generating algorithm for all stack realizable permutations on {1, 2, 3, 4}.

(2) $g(n)=C_{n-1}+C_{n-2}+\cdots+C_1$

Proof. We note that the number of nodes of level i in the stack realizable permutation tree is C_i [4, Sec. 2.2.1, Ex4]. In a similar way as Property 2.1, we obtain the following recurence relations.

$$f(i)=f(i-1)+C_i-C_{i-1} \quad (2<i\leq n)$$
$$f(2)=1$$
$$g(i)=g(i-1)+C_{i-1} \quad (2<i\leq n)$$
$$g(2)=1$$

Solving these recurrence relations, we obtain the above results.

Property 3.4 Let $n\geq 5$.

$$1/3+2/(3n-4)<(C_1+\cdots+C_{n-1})/C_n<1/3+2/(3n-5)$$

Proof. The proof proceeds by induction on n.

Basis. $n=5, 6$. By direct computation, the above inequalities are obtained.

Inductive step. $n>6$. We can evaluate $(C_1+\cdots+C_{n-1})/C_n$ as follows.

$$\frac{C_1+\cdots+C_{n-1}}{C_n}=\frac{n+1}{4n-2}\left(\frac{C_1+\cdots+C_{n-2}}{C_{n-1}}+1\right)$$
$$>\frac{n+1}{4n-2}\left(\frac{4}{3}+\frac{2}{3n-7}\right)$$
$$=\frac{(n+1)(6n-11)}{3(2n-1)(3n-7)}$$
$$>\frac{1}{3}+\frac{2}{3n-4}$$

$$\frac{C_1+\cdots+C_{n-1}}{C_n}=\frac{n+1}{4n-2}\left(\frac{C_1+\cdots+C_{n-2}}{C_{n-1}}+1\right)$$
$$<\frac{n+1}{4n-2}\left(\frac{4}{3}+\frac{2}{3n-8}\right)$$
$$=\frac{(n+1)(6n-13)}{3(2n-1)(3n-8)}$$
$$<\frac{1}{3}+\frac{2}{3n-5}$$

Thus, we have proved the above inequalities.

Theorem 3.1 Let $n\geq 2$.
The average time per stack realizable permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.

Proof. For a similar reason as the average time per permutation on $\{1, 2, \cdots, n\}$, it is proved by Property 3.3 and 3.4.

4. Generation of All Queue Realizable Permutations on $\{1, 2, \cdots, n\}$

In this section, we will establish a generating algorithm for all queue realizable permutations on $\{1, 2, \cdots, n\}$ and show the average time per queue realizable permutation is bounded by a constant.

We derive a basic property of queue realizable permutations on $\{1, 2, \cdots, n\}$.

Property 4.1 Let $n\geq 3$.
A permutation on $\{1, 2, \cdots, n\}$ $a_1a_2\cdots a_n$ is a queue realizable permutation, if and only if there are no subsequences $a_ia_ja_k$ such that $a_i>a_j>a_k$ $(i<j<k)$.

Proof. (Only if) Suppose that a queue realizable permutation $a_1a_2\cdots a_n$ contains a subsequence $a_ia_ja_k$ such that $a_i>a_j>a_k$ $(i<j<k)$. Since $a_i>a_j$ $(i<j)$, we see a_j is inserted into the queue. Since $a_i>a_k$ $(i<k)$, we see a_k is inserted into the queue. On the other hand, since $a_j>a_k$ $(j<k)$, a_k is inserted into the queue before a_j and moved to the output before a_j. This means that a subsequence $a_ia_ka_j$ such that $a_i>a_j>a_k$ $(i<j<k)$ is obtained. This contradicts the assumption. Thus, a queue realizable permutation $a_1a_2\cdots a_n$ contains no subsequences $a_ia_ja_k$ such that $a_i>a_j>a_k$ $(i<j<k)$.

(If) The desired permutation $a_1a_2\cdots a_n$ can be obtained by performing the following algorithm.

Let the input permutation be $12\cdots n$. For $j=1, 2, \cdots, n$ move zero or more elements (as many as necessary) from the input permutation to the rear of the queue, until a_j first appears in the input permutation or in the front of the queue, then output a_j.

This algorithm can fail only if we reach a j for which the input permutation becomes empty and a_j is not at the front of the queue. Then, it is covered by some element a_k $(j<k)$. Since the elements of the queue is always monotone increasing, we have $a_j>a_k$ $(j<k)$. This element a_j exists there only if there is an element $a_i>a_j$ $(i<j)$. Namely, we have obtained a subsequence $a_ia_ja_k$ such that $a_i>a_j>a_k$ $(i<j<k)$. Thus, if there are no subsequences $a_ia_ja_k$ such that $a_i>a_j>a_k$ $(i<j<k)$, a permutation $a_1a_2\cdots a_n$ is a queue realizable permutation.

This property also suggests the method of generating all queue realizable permutations on $\{1, 2, \cdots, n\}$.

Property 4.2 Let $n\geq 2$.
(1) If a permutation $a_1a_2\cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ is a queue realizable permutation, then no queue

realizable permutations on $\{1, 2, \cdots, n\}$ can be constructed by inserting the new element n into possible positions.

(2) If a permutation $a_1a_2\cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ is a queue realizable permutation, then queue realizable permutations on $\{1, 2, \cdots, n\}$ can be constructed by inserting the new element n at the right of a_{n-1} and at the left of a_{n-1} and a_i $(1 \le i \le n-2)$, where $a_j < a_{j+1}$ $(i \le j \le n-2)$.

Proof. In a similar way as Property 3.2, it is proved by Property 4.1.

By Property 4.2, we can obtain all queue realizable permutations on $\{1, 2, \cdots, n\}$, as level n in a tree(queue realizable permutation tree) in Fig. 4.1. The root 1 is defined to be level 1. The new element n is assumed to be inserted from right to left.

To generate successively all queue realizable permutations on $\{1, 2, \cdots, n\}$, we consider to move from left to right along level n of the queue realizable permutation tree. Thus, given a queue realizable permutation $a_1a_2\cdots a_n$ on $\{1, 2, \cdots, n\}$, where $a_1a_2\cdots a_n \neq 23\cdots n1$, we can define its immediate successor $b_1b_2\cdots b_n$ as follows.

(1) If $a_n = n$, then we exchange a_{n-1} and a_n.

(2) If $a_i = n$ and $a_{i-1} < a_{i+1}$ $(1 \le i < n)$, then we exchange a_{i-1} and a_i.

(3) If $a_i = n$ and $a_{i-1} > a_{i+1}$ $(1 \le i < n)$, then let a permutation $c_1c_2\cdots c_{n-1}$ on $\{1, 2, \cdots, n-1\}$ be the immediate successor of a queue permutation $a_1\cdots a_{i-1}a_{i+1}\cdots a_n$ on $\{1, 2, \cdots, n-1\}$. Then, $b_j = c_j$ $(1 \le j \le n-1)$ and $b_n = n$.

The first queue realizable permutation on $\{1, 2, \cdots, n\}$ is defined to be $12\cdots n$. The element a_0 is set to $n+1$ to detect that the index i is equal to 1.

Our fundamental idea is to traverse the queue realizable permutation tree in inorder. Thus, a generating algorithm for all queue realizable permutation on $\{1, 2, \cdots, n\}$ is established by making the following partial revisions for the generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

(1) line 2 $a[1] := 1$; → $a[0] := n+1$; $a[1] := 1$;

(2) line 10 until $i = 1$; → until $a[i-1] > a[i+1]$;

(3) line 18 until $i > 1$; → until $(i = k)$ or $(a[i-1] < a[i+1])$;

The element a_0 is represented by $a[0]$. Thus, we use an integer array $a[0\cdot\cdot n]$. The result of a generating algorithm for all queue realizable permutations on $\{1, 2, 3, 4\}$ is shown in Fig. 4.2.

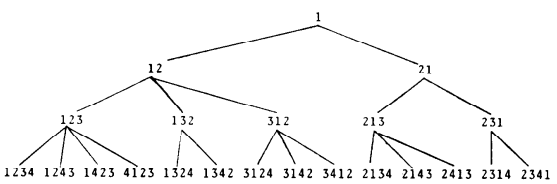

Fig. 4.1 A queue realizable permutation tree.

	$a_1a_2a_3a_4$	$x_1x_2x_3x_4$
1:	1 2 3 4	1 2 3 4
2:	1 2 4 3	1 2 3 3
3:	1 4 2 3	1 2 3 2
4:	4 1 2 3	1 2 3 1
5:	1 3 2 4	1 2 2 4
6:	1 3 4 2	1 2 2 3
7:	3 1 2 4	1 2 1 4
8:	3 1 4 2	1 2 1 3
9:	3 4 1 2	1 2 1 2
10:	2 1 3 4	1 1 3 4
11:	2 1 4 3	1 1 3 3
12:	2 4 1 3	1 1 3 2
13:	2 3 1 4	1 1 2 4
14:	2 3 4 1	1 1 2 3

Fig. 4.2 The result of a generating algorithm for all queue realizable permutations on {1, 2, 3, 4}.

Now we show that the average time per queue realizable permutation is bounded by a constant. Two functions $f(n)$ and $g(n)$ are defined for queue realizable permutations on $\{1, 2, \cdots, n\}$ as they are for permutations on $\{1, 2, \cdots, n\}$.

Property 4.3 Let $n \ge 2$.

(1) $f(n) = C_n - 1$

(2) $g(n) = C_{n-1} + C_{n-2} + \cdots + C_1$

Proof. We note that the number of nodes of level i in the queue realizable permutation tree is C_i [5]. We can prove in the same way as Property 3.3.

Theorem 4.1 Let $n \ge 2$.
The average time per queue realizable permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.

Proof. For the same reason as the average time per permutation on $\{1, 2, \cdots, n\}$, it is proved by Property 4.3 and 3.4.

5. Generation of All Permutations on $\{1, 2, \cdots, n\}$ in Another Direct Insertion Order

The purpose of this section is to propose another direct insertion order on the set of all permutations on $\{1, 2, \cdots, n\}$ and establish a generating algorithm for all permutations on $\{1, 2, \cdots, n\}$. The average time per permutation is shown to be bounded by a constant. This algorithm forms the foundation of generating algorithm for all stack (queue) sortable permutations.

Let $1 \le m \le n-1$. For each permutation on $\{m+1, \cdots, n\}$, $n-m+1$ permutations on $\{m, m+1, \cdots, n\}$ are generated by inserting the new element m into all possible positions from left to right. Thus, we may obtain all permutations on $\{m, \cdots, n\}$, as level $n-m+1$ in a tree (permutation tree) in Fig. 5.1. The root n is defined to be level 1. Our direct insertion order on the set of all

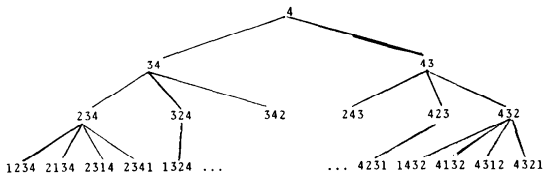

Fig. 5.1 A permutation tree.

permutations $a_m \cdots a_n$ on $\{m, \cdots, n\}$ is defined by moving from left to right along level $n-m+1$ of this permutation tree. Thus, given a permutation $a_m \cdots a_n$ on $\{m, \cdots, n\}$, where $a_m \cdots a_n \neq n(n-1) \cdots m$, we can define its immediate successor $b_m \cdots b_n$ as follows.

(1) If $a_i = m$ $(m \leq i < n)$, then we exchange a_i and a_{i+1}. Thus, we have $b_j = a_j$ $(m \leq j < i,\ i+1 < j \leq n)$, $b_i = a_{i+1}$ and $b_{i+1} = a_i$.

(2) If $a_n = m$, then let a permutation $c_{m+1} \cdots c_n$ on $\{m+1, \cdots, n\}$ be the immediate successor of a permutation $a_m \cdots a_{n-1}$ on $\{m+1, \cdots, n\}$. Then $b_j = c_j$ $(m < j \leq n)$ and $b_m = m$.

The first permutation is defined to be $m \cdots n$.

Our fundamental idea is to traverse the permutation tree in inorder. A generating algorithm is described in PASCAL-like notation and shown in Fig. 5.2. A permutation $a_1 a_2 \cdots a_n$ on $\{1, 2, \cdots, n\}$ is represented by an integer array $a[1 \cdots n]$.

We denote by y_i $(1 \leq i \leq n)$ the position in which the element i is located at level $n-i+1$ in the permutation tree. Since y_n is always equal to n, we use an integer array $y[1 \cdots n-1]$ to represent y_i $(1 \leq i \leq n-1)$. For $n=4$, the result of this algorithm is shown in Fig. 5.3.

Now we show the average time per permutation is bounded by a constant. We denote by $f(n)$ the number of times the instruction in the for loop(line 13) may be executed and by $g(n)$ the number of times other instructions in the repeat-until loop(line 12–18) may be executed, in order to generate all permutations on $\{1, 2, \cdots, n\}$.

```
1.  begin
2.    a[1]:=1;
3.    for j:=2 to n do begin a[j]:=j; y[j]:=j end;
4.  1:
5.    output (a);
6.    i:=1;
7.    repeat
8.      i:=i+1; a[i-1]:=a[i]; a[i]:=1;
9.      output (a)
10.   until i=n;
11.   k:=1;
12.   repeat
13.     for j:=i-1 downto k do a[j+1]:=a[j];
14.     a[k]:=k; y[k]:=k;
15.     k:=k+1;
16.     if k=n then goto 2;
17.     i:=y[k]
18.   until i<n;
19.   i:=i+1; a[i-1]:=a[i]; a[i]:=k; y[k]:=i;
20.   goto 1;
21. 2:
22. end.
```

Fig. 5.2 The generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

	$a_1a_2a_3a_4$	$x_1x_2x_3x_4$
1:	1 2 3 4	1 2 3 4
2:	2 1 3 4	2 2 3 4
3:	2 3 1 4	3 2 3 4
4:	2 3 4 1	4 2 3 4
5:	1 3 2 4	1 3 3 4
6:	3 1 2 4	2 3 3 4
7:	3 2 1 4	3 3 3 4
8:	3 2 4 1	4 3 3 4
9:	1 3 4 2	1 4 3 4
10:	3 1 4 2	2 4 3 4
11:	3 4 1 2	3 4 3 4
12:	3 4 2 1	4 4 3 4
13:	1 2 4 3	1 2 4 4
14:	2 1 4 3	2 2 4 4
15:	2 4 1 3	3 2 4 4
16:	2 4 3 1	4 2 4 4
17:	1 4 2 3	1 3 4 4
18:	4 1 2 3	2 3 4 4
19:	4 2 1 3	3 3 4 4
20:	4 2 3 1	4 3 4 4
21:	1 4 3 2	1 4 4 4
22:	4 1 3 2	2 4 4 4
23:	4 3 1 2	3 4 4 4
24:	4 3 2 1	4 4 4 4

Fig. 5.3 The result of a generating algorithm for all permutations on $\{1, 2, 3, 4\}$.

Property 5.1 Let $n \geq 2$.
(1) $f(n) = n! - 1$
(2) $g(n) = (n-1)! + (n-2)! + \cdots + 2! + 1!$

Proof. In a similar way as Property 2.1, it is proved.

Theorem 5.1 Let $n \geq 2$.
The average time per permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.

Proof. By Property 5.1, it is proved.

Note that we do not consider the time needed to print out the permutations.

6. Generation of All Stack Sortable Permutation on $\{1, 2, \cdots, n\}$

In this section, we will establish a generating algorithm for all stack sortable permutations on $\{1, 2, \cdots, n\}$ and show the average time per stack sortable permutation is bounded by a constant.

We give a basic property of stack sortable permutations on $\{1, 2, \cdots, n\}$.

Property 6.1 Let $n \geq 3$.
A permutation on $\{1, 2, \cdots, n\}$ $a_1 a_2 \cdots a_n$ is a stack sortable permutation, if and only if there are no

subsequences $a_i a_j a_k$ such that $a_k < a_i < a_j$ $(i<j<k)$.

The proof is omitted, since a similar property and its proof is found in Rotem[1, Theorem 2].

Property 6.2 Let $1 \leq m \leq n-1$.

(1) If a permutation $a_{m+1} \cdots a_n$ on $\{m+1, \cdots, n\}$ is not a stack sortable permutation, then no stack sortable permutations on $\{m, m+1, \cdots, n\}$ can be constructed by inserting the new element m into a possible positions.

(2) If a permutation $a_{m+1} \cdots a_n$ on $\{m+1, \cdots, n\}$ is a stack sortable permutation, then a stack sortable permutations on $\{m, m+1, \cdots, n\}$ can be constructed by inserting the new element m at the left of a_{m+1} and at the right of a_{m+1} and a_i $(m+2 \leq i \leq n)$, where $a_{j-1} > a_j$ $(m+2 \leq j \leq i)$.

Proof. (1) Obvious, (2) By Property 6.1, the new element m has to be inserted so that subsequences $a_u a_v m$ $(m < a_u < a_v, u < v)$ may not be constructed.

Thus, the new element m can be inserted at the left(right) of a_{m+1}. Suppose that the new element m can be inserted at the right of a_i $(m+2 \leq i \leq n)$, then two element a_u and a_v $(a_u < a_v, u < v \leq i)$ must not exist, that is, $a_{j-1} > a_j$ for all j, $m+2 \leq j \leq i$.

By Property 6.2, we can obtain all stack sortable permutations on $\{m, \cdots, n\}$, as level $n-m+1$ in a tree (stack sortable permutation tree) in Fig. 6.1. The root n is defined to be level 1. The new element m is assumed to be inserted from left to right.

To generate successively all stack sortable permutations on $\{m, \cdots, n\}$, we consider to move from left to right along level $n-m+1$ in the stack sortable permutation tree. Thus, given a stack sortable permutation $a_m \cdots a_n$ on $\{m, \cdots, n\}$, where $a_m \cdots a_n \neq n(n-1) \cdots m$, we can define its immediate successor $b_m \cdots b_n$ as follows.

(1) If $a_m = m$, then we exchange a_m and a_{m+1}.

(2) If $a_i = m$ and $a_{i-1} > a_{i+1}$ $(m < i \leq n)$, then we exchange a_i and a_{i+1}.

(3) If $a_i = m$ and $a_{i-1} < a_{i+1}$ $(m < i \leq n)$, then let a permutation $c_{m+1} \cdots c_n$ on $\{m+1, \cdots, n\}$ be the immediate successor of a stack sortable permutation $a_m \cdots a_{i-1} a_{i+1} \cdots a_n$ on $\{m+1, \cdots, n\}$. Then, $b_j = c_j$ $(m+1 \leq j \leq n)$ and $b_m = m$.

The first stack sortable permutation on $\{m, \cdots, n\}$ is defined to be $m \cdots n$. The element a_{n+1} is set to $n+1$ to detect that the index i is equal to n.

Our fundamental idea is to traverse the stack sortable permutation tree in inorder. Thus, a generating algorithm for all stack sortable permutation on $\{1, 2, \cdots, n\}$ is established by making the following partial revisions for the generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

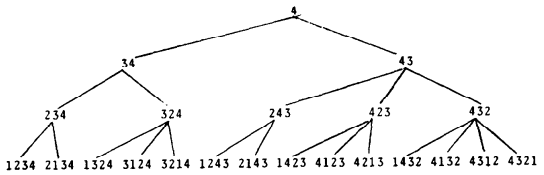

Fig. 6.1 A stack sortable permutation tree.

	$a_1a_2a_3a_4$	$x_1x_2x_3x_4$
1:	1 2 3 4	1 2 3 4
2:	2 1 3 4	2 2 3 4
3:	1 3 2 4	1 3 3 4
4:	3 1 2 4	2 3 3 4
5:	3 2 1 4	3 3 3 4
6:	1 2 4 3	1 2 4 4
7:	2 1 4 3	2 2 4 4
8:	1 4 2 3	1 3 4 4
9:	4 1 2 3	2 3 4 4
10:	4 2 1 3	3 3 4 4
11:	1 4 3 2	1 4 4 4
12:	4 1 3 2	2 4 4 4
13:	4 3 1 2	3 4 4 4
14:	4 3 2 1	4 4 4 4

Fig. 6.2 The result of a generating algorithm for all stack sortable permutations on {1, 2, 3, 4}.

(1) line 2 $a[1]:=1$; → $a[1]:=1$; $a[n+1]:=n+1$;
(2) line 10 until $i=n$; → until $a[i-1]<a[i+1]$;
(3) line 18 until $i<n$; → until $(i=k)$ or $(a[i-1]>a[i+1])$;

The element a_{n+1} is represented by $a[n+1]$. Thus, we use an integer array $a[1 \cdots n+1]$. The result of a generating algorithm for all stack sortable permutations on {1, 2, 3, 4} is shown in Fig. 6.2.

Now we show the average time per stack sortable permutation is bounded by a constant. Two functions $f(n)$ and $g(n)$ are defined for stack sortable permutations on $\{1, 2, \cdots, n\}$ as they are for permutations on $\{1, 2, \cdots, n\}$.

Property 6.3 Let $n \geq 2$.

(1) $f(n) = C_n - 1$

(2) $g(n) = C_{n-1} + C_{n-2} + \cdots + C_1$

Proof. We note that the number of nodes of level i in the stack sortable permutation tree is $C_i[1]$. In a similar way as Property 3.3, we obtain the above results.

Theorem 6.1 Let $n \geq 2$.

The average time per stack sortable permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.

Proof. For the same reason as the average time per permutation on $\{1, 2, \cdots, n\}$, it is proved by Property 6.3 and 3.4.

7. Generation of All Queue Sortable Permutations on $\{1, 2, \cdots, n\}$

In this section, we will establish a generating algorithm for all queue sortable permutations on $\{1, 2, \cdots, n\}$ and show the average time per queue sortable permutation

is bounded by a constant.

We give basic properties of queue sortable permutations on $\{1, 2, \cdots, n\}$.

Property 7.1 Let $n \geq 3$.
A permutation on $\{1, 2, \cdots, n\}$ $a_1 a_2 \cdots a_n$ is a queue sortable permutation, if and only if there are no subsequences $a_i a_j a_k$ such that $a_i > a_j > a_k$ $(i<j<k)$.

Proof. In a similar way as Property 4.1, it is proved.

Property 7.2 Let $n \geq 1$.
$QR_n = QS_n$

Proof. By Property 4.1 and 7.1, it is proved.

Therefore, a generating algorithm for all queue realizable permutations on $\{1, 2, \cdots, n\}$ can be employed for the generation for all queue sortable permutations on $\{1, 2, \cdots, n\}$. However, we will establish another generating algorithm based on the generating algorithm in Fig. 5.2.

Property 7.3 Let $1 \leq m \leq n-1$.

(1) If a permutation $a_{m+1} \cdots a_n$ on $\{m+1, \cdots, n\}$ is not a queue sortable permutation, then no queue sortable permutations on $\{m, m+1, \cdots, n\}$ can be constructed by inserting the new element m into a possible positions.

(2) If a permutation $a_{m+1} \cdots a_n$ on $\{m+1, \cdots, n\}$ is a queue sortable permutation, then queue sortable permutations on $\{m, m+1, \cdots, n\}$ can be constructed by inserting the new element m at the left of a_{m+1} and at the right of a_{m+1} and a_i $(m+2 \leq i \leq n)$, where $a_{j-1} < a_j$ $(m+2 \leq j \leq i)$.

Proof. In a similar way as Property 6.2, it is proved.

By Property 7.3, we can obtain all queue sortable permutations on $\{m, \cdots, n\}$, as level $n-m+1$ in a tree (queue sortable permutation tree) in Fig. 7.1. The root n is defined to be level 1. The new element m is assumed to be inserted from left to right.

To generate successively all queue sortable permutations on $\{m, \cdots, n\}$, we consider to move from left to right along level $n-m+1$ in the queue sortable permutation tree. Thus, given a queue sortable permutation $a_m \cdots a_n$ on $\{m, \cdots, n\}$, where $a_m \cdots a_n \neq n(n-1) \cdots m$, we can define its immediate successor $b_m \cdots b_n$ as follows.

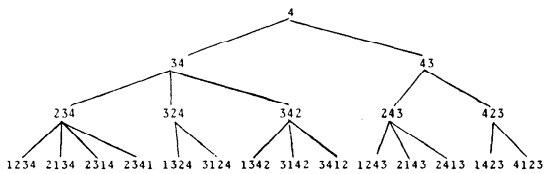

Fig. 7.1 A queue sortable permutation tree.

(1) If $a_m = m$, then we exchange a_m and a_{m+1}.

(2) If $a_i = m$ and $a_{i-1} > a_{i+1}$ $(m < i \leq n)$, then we exchange a_i and a_{i+1}.

(3) If $a_i = m$ and $a_{i-1} < a_{i+1}$ $(m < i \leq n)$, then let a permutation $c_{m+1} \cdots c_n$ on $\{m+1, \cdots, n\}$ be the immediate successor of a queue sortable permutation $a_m \cdots a_{i-1} a_{i+1} \cdots a_n$ on $\{m+1, \cdots, n\}$. Then, $b_j = c_j$ $(m+1 \leq j < n)$ and $b_m = m$.

The first queue sortable permutation on $\{m, \cdots, n\}$ is defined to be $m \cdots n$. The element a_{n+1} is set to zero to detect that the index i is equal to n.

Our fundamental idea is to traverse the queue sortable permutation tree in inorder. Thus, a generating algorithm for all queue sortable permutation on $\{1, 2, \cdots, n\}$ is established by making the following partial revisions for the generating algorithm for all permutations on $\{1, 2, \cdots, n\}$.

(1) line 2 $a[1]:=1;$ → $a[1]:=1;$ $a[n+1]:=0;$
(2) line 10 until $i=n;$ → until $a[i-1] > a[i+1];$
(3) line 18 until $i<n;$ → until $(i=k)$ or $(a[i-1] < a[i+1]);$

The element a_{n+1} is represented by $a[n+1]$. Thus, we use an integer array $a[1 \cdots n+1]$. The result of a generating algorithm for all queue sortable permutations on $\{1, 2, 3, 4\}$ is shown in Fig. 7.2.

Now we show the average time per queue sortable permutation is bounded by a constant. Two functions $f(n)$ and $g(n)$ are defined for queue sortable permutations on $\{1, 2, \cdots, n\}$ as they are for permutations on $\{1, 2, \cdots, n\}$.

Property 7.4 Let $n \geq 2$.
(1) $f(n) = C_n - 1$
(2) $g(n) = C_{n-1} + C_{n-2} + \cdots + C_1$

Proof. We note that the number of nodes of level i in the queue sortable permutation tree is C_i by Property 7.2. In a similar way as Property 3.3, we obtain the above results.

	$a_1 a_2 a_3 a_4$	$x_1 x_2 x_3 x_4$
1:	1 2 3 4	1 2 3 4
2:	2 1 3 4	2 2 3 4
3:	2 3 1 4	3 2 3 4
4:	2 3 4 1	4 2 3 4
5:	1 3 2 4	1 3 3 4
6:	3 1 2 4	2 3 3 4
7:	1 3 4 2	1 4 3 4
8:	3 1 4 2	2 4 3 4
9:	3 4 1 2	3 4 3 4
10:	1 2 4 3	1 2 4 4
11:	2 1 4 3	2 2 4 4
12:	2 4 1 3	3 2 4 4
13:	1 4 2 3	1 3 4 4
14:	4 1 2 3	2 3 4 4

Fig. 7.2 The result of a generating algorithm for all queue sortable permutations on $\{1, 2, 3, 4\}$.

Theorem 7.1 Let $n \geq 2$.
The average time per queue sortable permutation on $\{1, 2, \cdots, n\}$ is bounded by a constant.
Proof. For the same reason as the average time per permutation on $\{1, 2, \cdots, n\}$, it is proved by Property 7.4 and 3.4.

8. Combinatorial Properties

In this section, several interesting combinatorial properties are proved. Four classes SR_n, SS_n, QR_n, QS_n have the following relations. We note $QR_n = QS_n$.

Property 8.1 Let $n \geq 1$ and F_n be Fibonacci number such that $F_{n+2} = F_{n+1} + F_n$, $F_2 = 1$, $F_1 = 1$.

(1) $|SS_n \cap SR_n| = 2^{n-1}$ (2) $|SR_n \cap QR_n| = 2^{n-1}$
(3) $|SS_n \cap QS_n| = 2^{n-1}$ (4) $|SS_n \cap SR_n \cap QR_n| = F_{n+1}$

Proof.
(1) The proof is found in Rotem [1].
(2) The proof proceeds by induction on n.
Basis. $n = 1$. Obvious.
Inductive step. $n > 1$. By Properties 3.2 and 4.2, it follows that the elements of $SR_n \cap QR_n$ can be constructed by inserting the new element n into the elements of $SR_{n-1} \cap QR_{n-1}$. Let a permutation $a_1 a_2 \cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ be the element of $SR_{n-1} \cap QR_{n-1}$. Only two permutations $a_1 \cdots a_{n-1} n$ and $a_1 \cdots a_{n-2} n a_{n-1}$ can be constructed as the element of $SR_n \cap QR_n$. Therefore, we have

$$|SR_n \cap QR_n| = 2|SR_{n-1} \cap QR_{n-1}| = 2^{n-1}.$$

(3) In a similar way as (2), it is proved.
(4) The proof proceeds by induction on n.
Basis. $n = 1$. Obvious.
Inductive step. $n > 1$. By Properties 3.2, 4.2 and 6.2, it follows that the elements of $SS_n \cap SR_n \cap QR_n$ can be constructed by inserting the new element n into the elements of $SS_{n-1} \cap SR_{n-1} \cap QR_{n-1}$. Let a permutation $a_1 a_2 \cdots a_{n-1}$ on $\{1, 2, \cdots, n-1\}$ be the element of $SS_{n-1} \cap SR_{n-1} \cap QR_{n-1}$. Only two permutations $a_1 \cdots a_{n-1} n$ and $a_1 \cdots a_{n-2} n(n-1)$ can be constructed as the element of $SS_n \cap SR_n \cap QR_n$. Therefore, we have

$$|SS_n \cap SR_n \cap QR_n| = |SS_{n-1} \cap SR_{n-1} \cap QR_{n-1}| + |SS_{n-2} \cap SR_{n-2} \cap QR_{n-2}| = F_{n+1}.$$

Property 8.2 Let $n \geq 1$.
(1) A permutation $a = a_1 a_2 \cdots a_n$ on $\{1, 2, \cdots, n\}$ is a stack sortable permutation, if and only if a^{-1} is a stack realizable permutation.
(2) If a permutation $a = a_1 a_2 \cdots a_n$ on $\{1, 2, \cdots, n\}$ is a queue realizable (sortable) permutation, then a^{-1} is a queue realizable (sortable) permutation.
Proof.
(1) This property is shown in Rotem [1].
(2) Let $a^{-1} = b_1 b_2 \cdots b_n$. Suppose that there is a subsequence $b_i > b_j > b_k$ $(i < j < k)$. If $b_i = w$, $b_j = v$ and $b_k = u$, then $a_w = i$, $a_v = j$ and $a_u = k$. This means that there is a subsequence $a_u > a_v > a_w$ $(u < v < w)$. This contradicts the assumption that $a = a_1 a_2 \cdots a_n$ is a queue realizable (sortable) permutation. Thus, a^{-1} is a queue realizable (sortable) permutation.

Let $a_1 a_2 \cdots a_n$ be a permutation on $\{1, 2, \cdots, n\}$. We denote by $\alpha(a_1 a_2 \cdots a_n)$ the order in direct insertion order proposed by Trojanowski and by $\beta(a_1 a_2 \cdots a_n)$ the order in direct insertion order proposed by us. $\alpha(12 \cdots n)(\beta(12 \cdots n))$ is defined to be 1.

Property 8.3 Let $n \geq 2$.
(1) $\alpha(a_n a_{n-1} \cdots a_1) = n! - \alpha(a_1 a_2 \cdots a_n) + 1$
(2) $\beta(a_n a_{n-1} \cdots a_1) = n! - \beta(a_1 a_2 \cdots a_n) + 1$

Proof.
(1) The proof proceeds by induction on n.
Basis. $n = 2$. Obvious.
Inductive step. $n > 2$. Let $a_i = n$. $(1 \leq i \leq n)$. By the definition of direct insertion order, it follows that

$$\alpha(a_1 a_2 \cdots a_n) = (\alpha(a_1 \cdots a_{i-1} a_{i+1} \cdots a_n) - 1)n + n - i + 1 = \alpha(a_1 \cdots a_{i-1} a_{i+1} \cdots a_n)n - i + 1$$

$$\alpha(a_n a_{n-1} \cdots a_1) = (\alpha(a_n \cdots a_{i+1} a_{i-1} \cdots a_1) - 1)n + i = \alpha(a_n \cdots a_{i+1} a_{i-1} \cdots a_1)n - n + i.$$

Therefore, we have

$$\begin{aligned}\alpha(a_n a_{n-1} \cdots a_1) &= ((n-1)! - \alpha(a_1 \cdots a_{i-1} a_{i+1} \cdots a_n) + 1) \times n - n + i \\ &= n! - (\alpha(a_1 \cdots a_{i-1} a_{i+1} \cdots a_n)n - i) \\ &= n! - \alpha(a_1 a_2 \cdots a_n) + 1\end{aligned}$$

(2) In a similar way as (1), it is proved.

In order to generate all permutations $a_1 a_2 \cdots a_n$ on $\{1, 2, \cdots, n\}$, we have only to generate permutations $a_1 a_2 \cdots a_n$, where $1 \leq \alpha(a_1 a_2 \cdots a_n) \leq n!/2$, because permutations $a_n a_{n-1} \cdots a_1$, where $n!/2 < \alpha(a_n a_{n-1} \cdots a_1) \leq n!$, are obtained from permutation $a_1 a_2 \cdots a_n$. This property will effect a saving of 50 per cent in the amount of time needed to generate all permutations on $\{1, 2, \cdots, n\}$.

Property 8.4 Let $n \geq 2$.
(1) The stack realizable permutations $a_1 a_2 \cdots a_n$ are generated in lexicographical order.
(2) The stack sortable permutations $a_1 a_2 \cdots a_n$ are generated in reverse lexicographical order.

Proof.
(1) The proof proceeds by induction on n.
Basis $n = 2$. Obvious.
Inductive step. $n > 2$. Let $b_1 b_2 \cdots b_{n-1}$ be an immediate successor of the stack realizable permutation $a_1 a_2 \cdots a_{n-1}$. We denote by $p(a_1 a_2 \cdots a_{n-1}, b_1 b_2 \cdots b_{n-1})$ the

leftmost index where the element of $a_1a_2\cdots a_{n-1}$ is smaller than the element of $b_1b_2\cdots b_{n-1}$.

$$p(a_1a_2\cdots a_{n-1}, b_1b_2\cdots b_{n-1}) = \min_{1\le k\le n-1}\{k \mid a_i=b_i (1\le i<k), a_k<b_k\}$$

The children of $a_1a_2\cdots a_{n-1}$ ($b_1b_2\cdots b_{n-1}$) in the stack realizable permutation tree are generated in lexicographical order. Thus, it is sufficient to prove that $b_1b_2\cdots b_{n-1}n$, which is lexically first among the children of $b_1b_2\cdots b_{n-1}$, is the successor of $a_1\cdots a_ina_{i+1}\cdots a_{n-1}$ ($a_i<a_{i+1}>a_{i+2}>\cdots>a_{n-1}$), which is lexically last among the children of $a_1\cdots a_ia_{i+1}\cdots a_{n-1}$. In order to prove the above fact, we show that $p(a_1a_2\cdots a_{n-1}, b_1b_2\cdots b_{n-1})\le i$. Let $m=p(a_1a_2\cdots a_{n-1}, b_1b_2\cdots b_{n-1})$. If $m>i$, then we have $a_m<b_m$ and $\{a_m,\cdots, a_{n-1}\}=\{b_m,\cdots, b_{n-1}\}$. However, this contradicts the fact that $a_m>a_{m+1}>\cdots>a_{n-1}$. Thus we obtain $m\le i$.

(2) In a similar way as (1), it is proved.

Remark

In Knott[6], stack realizable permutations are called stack permutations and stack sortable permutations are called tree permutations. In Semba[7], another generating algorithm for all stack realizable permutations is proposed and the average time per stack realizable permutation is shown to be bounded by a constant.

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. Rotem, D. Stack sortable permutations, Discrete Mathematics, 33 (1981), 185–196.
2. Sedgewick, R. Permutation generation methods, Computing Surveys, 9, 2 (1977), 137–164.
3. Trojanowski, A. E. Ranking and listing algorithms for k-ary trees SIAM J. Computing, 7, 4 (1978), 492–509.
4. Knuth, D. E. The Art of Computer Programming, Fundamental Algorithms, 1, 2nd ed., Reading, Mass., Addison-Wesley (1973).
5. Semba, I. On the numbers of permutations obtainable by a stack or by a queue (in Japanese), Trans. IPSJ, 20, 6 (1979), 522–523.
6. Knott, G. D. A numbering system for binary trees, Comm. ACM, 20, 2 (1977), 113–115.
7. Semba, I. Generation of stack sequences in lexicographical order, this journal, 5, 1 (1982), 7–10.

(Received November 11, 1981: revised March 12, 1982)