Information Processing in Japan Vol. 17, 1977

Generating and Sorting Permutations Using Restricted-Deques

Atsumi IMAMIYA* and Akihiro NOZAKI*

Abstract

This paper is concerned with the problem of generating and sorting permutations using restricted-deques. Characterizations of permutations, that are genetable and sortable using a restricted-deque, are given and also two sorting algorithms, using networks of output restricted-deques, are presented. The number of output restricted-deques is also given in a cascade network which can sort any permutations by the algorithm.

1. INTRODUCTION

For a given class of devices or equipments for information processing, their ability has been studied by many researchers, e.g., the correspondences between a switching gate and Boolean functions, and between an automaton and formal languages.

A memory may be regarded as an information processing device with input, output, and storage facilities, but with no explicit functional capability unlike switching gates or automata. The output from the memory may be regarded as the permutations which are multiset[2] on the input. Therefore, it is interesting to study what permutations can be generated by a given class of memory.

On the other hand, linear lists are frequently used in the study of computer algorithms[1,2,5]. Knuth[1] has shown the capability of the stack to generate permutations and the correspondence between the permutations and Young Tableau[2].

Inspired by Knuth's work, Even and Itai[3] have shown how to generate the permutations in the network of queues or stacks, and the correspondence between the permutations and graphs. Sorting, using the networks of queues or stacks, has been given by Tarjan[4].

In this paper we wish to consider the problems of generating and sorting permutations using restricted-deques.

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 17, No. 12 (1976), pp. 1128～1134.

* Department of Computer Science, Yamanashi University

A deque(double-ended queue) is a linear list for which all loadings and unloadings (and usually all access) are made at the ends of the list. A deque is, therefore, more general than a stack(all loadings and unloadings are made at one end of the list) or a queue(all loadings are made at one end of the list, all unloadings are made at the other end). We also distinguish the output restricted-deque(ORDQ) or input restricted-deque(IRDQ). ORDQ and IRDQ are generally called RDQ(see the definitions in Section 2).

In this paper we wish to consider the following problems by using RDQ('s):

(a) Generating permutations; suppose there are symbols 1, 2,$\cdots$, n in a source queue, in their natural order. We want to rearrange 1, 2,$\cdots$, n by moving symbols through an RDQ or the network of RDQs, and by putting them from the system of RDQ into a sink queue. After a suitable number of such moves, the permutation of 1, 2,$\cdots$, n will be in a sink queue[1,3].

(b) Sorting permutations; suppose there is a permutation p(1)p(2)$\cdots$p(n) in a source queue. We want to arrange the permutation by moving symbols through an RDQ or the network of RDQs, and by putting them from the system into a sink queue. If they are in order, smallest to largest, the permutation has been sorted by the system[2].

2. GENERATING AND SORTING PERMUTATIONS USING AN RDQ

In this section, the RDQ's capabilities of generating and sorting permutations are shown.

[DEFINITION] An output restricted-deque(ORDQ) or an input restricted-deque(IRDQ), is one in which unloadings or loadings, respectively, are allowed to take place at only one end[1]. ■

ORDQ and IRDQ are also called RDQ. Let S, X, and Q denote respectively, the operations of loading an element at one end, unloading an element from the same end, and loading at the other end of an ORDQ.

Let S, X, and Y denote respectively, the operations of loading an

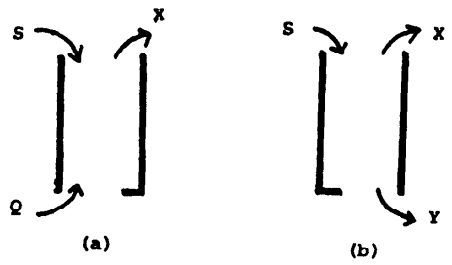


Fig. 1 Restricted Deque
(a) Output-Restricted Deque (ORDQ),
(b) Input-Restricted Deque (IRDQ).

element at one end, unloading an element from the same end, and unloading at the other end of an IRDQ(see Fig. 1).

Let assume that the RDQ gets stuck if at some time a symbol p is not accessible in the some state of loaded symbols. The symbols unloaded from the RDQ must not be loaded in it once again.

Each RDQ's capability of generating permutations is shown in Theorem 1 and Theorem 2.

[THEOREM 1] It is possible to generate the permutation $P = p_1 p_2 \cdots p_n$ from $12\cdots n$ using an ORDQ if and only if it contains no subsequence $p_{i1}p_{i2}p_{i3}p_{i4}$ such that

(T1) $p_{i1} > p_{i3} > p_{i4} > p_{i2}$,

or/and

(T2) $p_{i1} > p_{i3} > p_{i2} > p_{i4}$,

for all i1, i2, i3, and i4, where $1 \leq i1 < i2 < i3 < i4 \leq n$.

Proof.

ONLY IF : Suppose that an output permutation $P = p_1 p_2 \cdots p_n$ contains a subsequence $p_{i1}p_{i2}p_{i3}p_{i4}$ with the stated order (T1). We must keep p_{i2}, p_{i3}, and p_{i4} on the ORDQ until p_{i1} was put on and then unloaded from the ORDQ by the condition (T1). We also must keep these four symbols in the state shown in Fig.2(a) for generating the permutation P.

First, symbol p_{i2} must be loaded by an operation S or Q within the four symbols, because p_{i2} precedes the others in the input. Next, symbol p_{i4} must be loaded by an operation Q, because the output order is p_{i1}, p_{i2}, p_{i3}, and then p_{i4}(see Fig.2(b)). Then the symbol p_{i3} must be inserted between $p_{i2}$and p_{i4} because of the subsequece $p_{i1}p_{i2}p_{i3}p_{i4}$. But we find that it is impossible from the state shown in Fig.2(b).

IF : Suppose at some point the ORDQ gets stuck, because of the state loaded on it. If the ORDQ gets stuck, it will not generate the permutation $P = p_1 p_2 \cdots p_n$, and vice versa. Therefore, if we can obtain properties making the ORDQ get stuck, it must be a sufficient condition for this theorem that the permutation to be generated does not satisfy these properties.

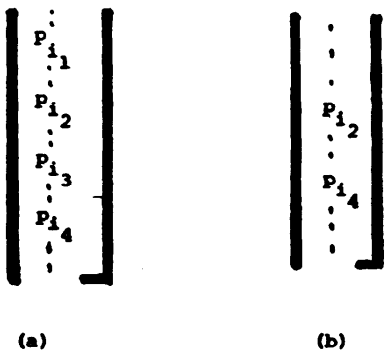


Fig. 2 Aspect of Symbols Loaded in ORDQ

The desired permutation can be generated by using the following algorithm;

"Assume that at some point $p_1p_2\cdots p_k$ has been unloaded. Zero or more input symbols must be loaded(as many as necessary) until p_{k+1} is at the top of the ORDQ as follows. Let the symbol u be at the top of the ORDQ at the time the symbol i is loaded on it. If symbol i precedes u in the permutation P, symbol i must be loaded by an operation S. Otherwise, it must be loaded by an operation Q. If the symbol $p_{k+1}$is at the top of the ORDQ, then unload p_{k+1}. This process continue until all of the symbols have been unloaded, or until the ORDQ gets stuck."

If the algorithm can fail, what conditions does the permutation P has? Suppose at some point the ORDQ gets stuck and the symbol a, which must be unloaded next, is inaccessible below the top of the ORDQ. That is, at least one symbol b exists above a in the ORDQ, and symbol a precedes symbol b in the output permutation P. We must obtain the conditions for such a symbol b:

(1.1) $a > b$; The top symbol u ($a > u$, $u \not< b$[@]) precedes symbol a in the output permutation P at the time symbol a is loaded. Therefore, the symbol a must be loaded on the ORDQ by an operation Q. The ORDQ does not get stuck at the time symbol a is unloaded unless all a, b, and u stay on the ORDQ until symbol c is loaded on by an operation S and unloaded. The symbol c satisfies the following inequalities; $c > a, b, u$.

Let $p_{i1} = c$, $p_{i2} = u$, $p_{i3} = a$, and $p_{i4} = b$. In this case, the permutation contains the subsequence $p_{i1}p_{i2}p_{i3}p_{i4}$ such that

(T1) $p_{i1} > p_{i3} > p_{i4} > p_{i2}$,

or

(T2) $p_{i1} > p_{i3} > p_{i2} > p_{i4}$, for $1 \leq i1 < i2 < i3 < i4 \leq n$.

(1.2) $a < b$; Suppose that there exists symbol v at the top of the ORDQ at the time symbol b is loaded, where $b > v$, and $v \not< a$[@@]. By the hypothesis, if symbol b does not precede v in the permutation P, the algorithm can not load symbol b above symbol a. Similarly for (1.1), if there exists symbol c above a, b, and v, they can stay together on the ORDQ at this time. A symbol c satisfies the following inequalities; $c > a, b, v$. Let $p_{i1} = c$, $p_{i2} = a$, $p_{i3} = b$, and $p_{i4} = v$.

@ If not so, because the algorithm can load the symbol a by an operation S, it is contrary to the assumption that the symbol b is above the symbol a in the ORDQ.

@@ If no so, it is contrary to the assumption that the ORDQ gets stuck at the time the symbol a is unloaded.

In this case, the permutation contains the subsequence $p_{i1}p_{i2}p_{i3}p_{i4}$ such that

(T1) $p_{i1} > p_{i3} > p_{i4} > p_{i2}$,

or

(T2) $p_{i1} > p_{i3} > p_{i2} > p_{i4}$, for $1 \leq i1 < i2 < i3 < i4 \leq n$. ■

We can have the following similar theorem for an IRDQ.

[THEOREM 2] It is possible to generate the permutation $P = p_1p_2\cdots p_n$ from $12\cdots n$ using an IRDQ if and only if it contains no subsequence $p_{i1}p_{i2}p_{i3}p_{i4}$ such that

(T2) $p_{i1} > p_{i3} > p_{i2} > p_{i4}$,

or/and

(T3) $p_{i1} > p_{i4} > p_{i2} > p_{i3}$,

for all i1, i2, i3, and i4, where $1 \leq i1 < i2 < i3 < i4 \leq n$. ■

Each RDQ's capability of sorting permutations is shown in Theorem 3.

[THEOREM 3] A permutation $P = q_1q_2\cdots q_n$ is sortable by using an ORDQ(an IRDQ) if and only if the permutation contains no subsequence $q_{i1}q_{i2}q_{i3}q_{i4}$ that satisfies(S1) or/and (S2)((S2) or/and (S3)) such that

(S1) $q_{i2} > q_{i3} > q_{i1} > q_{i4}$,

(S2) $q_{i1} > q_{i3} > q_{i2} > q_{i4}$,

(S3) $q_{i3} > q_{i1} > q_{i2} > q_{i4}$,

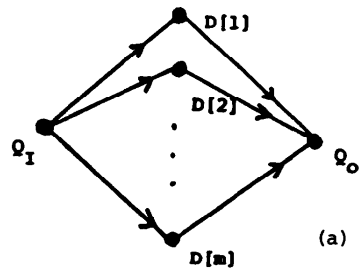
for all i1, i2, i3, and i4, where $1 \leq i1 < i2 < i3 < i4 \leq n$. ■

3. SORTING ALGORITHMs USING NETWORKs OF ORDQs

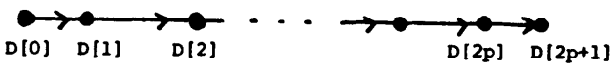
In this section we shall give two sorting algorithms which use the parallel network and the cascade network of ORDQs.

3.1. Sorting Using Parallel Network of ORDQs

We give the sorting algorithm for any permutation $P = p_1p_2\cdots p_n$ stored in a source Q_I into a sink Q_O through ORDQs in a parallel network of ORDQs shown in Fig.3(a).



(a)



(b)

(a) ORDQ Parallel Net

(b) ORDQ Cascade Net

Fig.3 Sorting Networks

Let $C[j,q_{i4}]$ be a set of the ordered sequences $q_{i1}q_{i2}q_{i3}$ as follows;

$C[j,q_{i4}] = \{ q_{i1}q_{i2}q_{i3} \mid$ subsequence $q_{i1}q_{i2}q_{i3}q_{i4}$ satisfies the condition (Sj) in Theorem 3, where j= 1, 2$\}$.

Let $C[j] = \bigcup_{q_{i4}} C[j,q_{i4}]$, and $C = C[1] \cup C[2]$.

ALGORITHM PS : Given a set $C = \{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_t\}$ of a permutation $q_1q_2\cdots q_n$, where $\bar{c}_k = q_{i1}q_{i2}q_{i3}$ for $1 \leq k \leq t$. If a symbol a is in D[i], it denotes that $a \in D[i]$.

Let i be the number of the ORDQs and j be the number of input symbols at the stage.

PS 1 : Set j $\longleftarrow$ 1.

PS 2 : Set i $\longleftarrow$ 1.

PS 3 : [Check the set C] If $\bar{c}_k$ exists in C such that $\bar{c}_k = abq_j$ for $\forall a,b \in D[i]$, set i $\longleftarrow$ i+1 and continue step PS 3; otherwise go to step PS 4.

PS 4 : [Load the symbol q_j into D[i]] Set D[i] $\Leftarrow$ D[i] $\oplus$ q_j @@@, i.e., load q_j into D[i].

PS 5 : [Next symbol] Set j $\longleftarrow$ j+1, and if j $\leq$ n, go to step PS 2; otherwise terminate the loadings, then unload all symbols from the smallest to the largest by comparing the top symbols of each ORDQ. If all ORDQs are empty, the algorithm terminates. ∎

[EXAMPLE 1] Sort the permutation 2746531 by using ALGORITHM PS. The set C is given as follows ; $C = \{274,276,275,273,243,265,263,253,465,746\}$. The aspect of loadings is shown in Fig.4. ∎

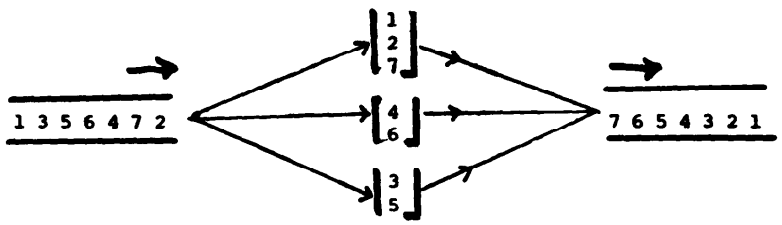


Fig.4 **Sorting 2746531 Using ORDQ Parallel Net**

3.2. Sorting Using Cascade Network of ORDQs

We shall now give a sorting algorithm for a cascade network of ORDQs as shown in Fig.3(b). We shall also give the number of ORDQs for the sorting.

Suppose that all n elements of the given permutation are loaded before unloading begins at any ORDQ.

@@@ The content of D[i] is an ordered set of q_j's. $\oplus$ means that when q_j is going to be loaded, for the top symbol u in D[i] if $u < q_j$ then q_j should be loaded into D[i] by the operation Q; otherwise q_j should be loaded into D[i] by the operation S.

ALGORITHM CS : In Fig.3(b), D[k] can be a queue for k = 0, $2\cdot\lceil \lg n \rceil + 1$. Each symbol p(i) of the permutation P = p(1)p(2)$\cdots$p(n) is denoted in binary form, $b_p b_{p-1} \cdots b_2 b_1$, as follows : $p(i) = b_p \cdot 2^{p-1} + b_{p-1} \cdot 2^{p-2} + \cdots + b_2 \cdot 2 + b_1$, where $p = \lceil \lg n \rceil$ and $b_j = 0, 1$.

CS 1 : Set k ⟵ 0(k is the number of ORDQs), and set j ⟵ 0($0 \leq j \leq p$).

CS 2 : [Loading] Set k ⟵ k+1, and set j ⟵ j+1. The following operations should be done for increasing order of i(i = 1, 2, $\cdots$, n);

"If $b_j = 0$, load p(i) into D[k] by the operation S; otherwise load p(i) into D[k] by the operation Q."

CS 3 : [Unloading and next loading] Set k ⟵ k+1. If $j \leq p$, the following operations should be done for increasing order of i:

"Unload p(i) from D[k-1] and then if $b_j = 0$, load p(i) into D[k] by the operation S; otherwise load p(i) into D[k] by the operation Q."

Then go to step CS 2. If $j > p$, go to step CS 4.

CS 4 : [Terminating] $k = \lceil \lg n \rceil$, all symbols can be loaded from D[2p] to D[2p+1] by the operation Q. Then the algorithm terminates. ■

[EXAMPLE 2] Sort the permutation 7465132 by using ALGORITHM CS.

The aspect of loading in ORDQs is shown in Fig.5. ■

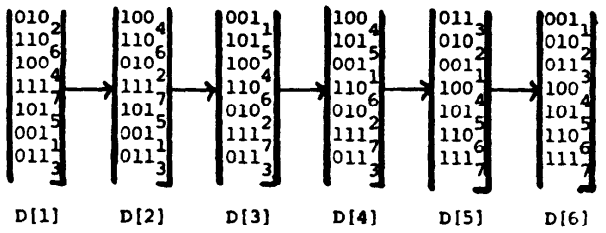


Fig.5 **Sorting Net of ORDQs in cascade for 7465132**

The proof of Theorem 4 follows from algorithm CS.

[THEOREM 4] Algorithm CS can sort any permutations of n elements by using cascade network of $2\cdot\lceil \lg n \rceil$ ORDQs. ■

REFERENCES

[1] Knuth, D.E. Fundamental Algorithms, Addison Wesley, Ch.2, 2-nd Ed.,1973.

[2] Knuth, D.E. Sorting and Searching, Addison Wesley, Ch.5, 1973.

[3] Even, S. and Itai,A. Queues, Stacks, and Graphs, in Theory of Machines and Computations, Academic Press, pp.71-86, 1971.

[4] Tarjan, R., Sorting Using Networks of Queues and Stacks, J.ACM, Vol.19 No.2, pp.341-346, 1972.

[5] Aho, A.V., Hopcroft, J.E. and Ullman, J.D. The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.