Journal of Science and Engineering Technology, Vol. 5, No. 4, pp. 21-29 (2009)

A New Method for Generating Permutations in Lexicographic Order

TING KUO
Department of International Trade, Takming University of Science and Technology
No. 56, Sec.1, HuanShan Rd., Neihu, Taipei, Taiwan 11451, R.O.C.

ABSTRACT

First, an ordinal representation scheme for permutations is defined. Next, an "unranking" algorithm that can generate a permutation of n items according to its ordinal representation is designed. By using this algorithm, all permutations can be systematically generated in lexicographic order. Finally, a "ranking" algorithm that can convert a permutation to its ordinal representation is designed. By using these "ranking" and "unranking" algorithms, any permutation that is positioned in lexicographic order, away from a given permutation by any specific distance, can be generated. This significant benefit is the main difference between the proposed method and previously published alternatives. Three other advantages are as follows: First, not being restricted to sequentially numbering the n items from one to n, this method can even handle items with non-numeral marks without the aid of mapping. Second, this approach is well suited to a wide variety of permutation generations such as alternating permutations and derangements. Third, the proposed method can also be extended to a multiset.

***Key Words*:** lexicographic order, ordinal representation, permutation, ranking, unranking

一個依字母大小序產生排列的新方法

郭定
德明財經科技大學國際貿易系
11451 臺北市內湖區環山路一段 56 號

摘　要

首先，我們爲排列問題定義一個順序表示法。第二，我們設計一個定序演算法，它可以根據一個 n 項目的順序表示產生其對應的排列。藉由此演算法，我們可以依字母大小序地系統化產生所有的 n 項目排列。第三，我們設計一個解序演算法，它可以根據一個 n 項目的排列產生其對應的順序表示。藉由此兩個演算法，我們可以產生離一個 n 項目的排列任意距離，依字母大小序而言，的另一個排列；此特點是我們的方法與其它研究最不同的地方。我們的方法還有另三項優點如下：第一，它不限制於必須是 1 至 n 的連續數目，甚至於可以無需藉由轉換而直接處理非數字的排列。第二，它也很適合用在其它的排列問題上，例如產生交替排列與錯位排列。第三，它也可擴展至針對含有重複元素的集合之排列問題上。

關鍵詞：字母大小序，順序表示法，排列，定序，解序

I. INTRODUCTION

Permutation is one of the most important combinatorial objects in computing, and can be applied in various applications, for example, the scheduling problems. There are two reasons that permutation generation is of interest. One is that it can form the basis for a backtracking program to solve any problem involving reordering a set of items. The other one is that there are a number of related procedures for generating other combinatorial objects [33]. It is well known that, for n distinct items, the total number of permutations is $n!$ ($=1\times2\times\ldots\times n$, called n factorial) and this would be enormous when n is large enough. For example, the value of $n!$ is about three and a half million when n=10 and is about 1,300 billion when n=15. However, in real application problems we are usually confronted by dealing with more than fifteen or even hundreds of items. Therefore, to design an algorithm for permutation generation is a fundamental and practical issue. Furthermore, since many algorithms are known by which permutations that are optimum for particular purposes can be found without running through all possibilities [16], it is beneficial to propose a permutation generation method that can embed those constraints into algorithm and generate permutations in the order we desire.

It is the permutation generation method that determines the order of a list of permutations. In fact, there is a natural order of all permutations called lexicographic, or alphabetical, order [32]. In the proper sense of the word, a list of permutations is in lexicographic order if these permutations are sorted as they would appear in a dictionary. Strictly speaking, if the n items going through permutations are ordered by a precedence relation "<", then permutation $\boldsymbol{\pi}_a=(\pi_{a,1}\pi_{a,2}\ldots\pi_{a,n})$ precedes permutation $\boldsymbol{\pi}_b=(\pi_{b,1}\pi_{b,2}\ldots\pi_{b,n})$ if and only if, for some $i \le 1$, we have $\pi_{a,j}=\pi_{b,j}$ for all $j < i$ and $\pi_{a,I} < \pi_{b,i}$ [28]. For example, the lexicographic order of six permutations of three distinct items {1 2 3} is (1 2 3) < (1 3 2) < (2 1 3) < (2 3 1) < (3 1 2) < (3 2 1).

Furthermore, there is a kind of "reverse lexicographic" ordering [31], also called "reverse colex order" [16], which is the result of reading the lexicographic sequence backwards and the permutations from right to left. In Table 1, we list all 24 permutations of four distinct items {1 2 3 4} in lexicographic order and reverse lexicographic order respectively.

Since lexicographic order is a natural and simple order, it should be easily manipulated by a computer program. Why don't we design a permutation generation method that can utilize this intrinsic order? This drives us to find another method for generating permutations in lexicographic order. It is noteworthy that our new method also can be used to generate permutations in reverse lexicographic order. In this paper, we propose a permutation generation method that possesses advantageous attributes of simplicity, speediness, lexicography, bidirectionality, historylessness, and cyclicality [26]. In Section II, we present a brief survey of some published works. In Section III, we define a new representation scheme that is conceptually easy to understand and implement. In Section IV, we propose three algorithms. By using the new representation scheme, we first design a ranking algorithm that can generate any permutation according to the new representation scheme. Then, we design an algorithm that can systematically generate the S_n in lexicographic order. Finally, we design an unranking algorithm that can easily convert each permutation in the S_n to the new representation scheme. In Section V, several advantageous attributes of the new method are discussed. Conclusions are summarized in Section VI.

Table 1. Lexicographic order and reverse lexicographic order of S_4

Lexicographic Order				*Reverse Lexicographic Order*			
1	2	3	4	1	2	3	4
1	2	4	3	2	1	3	4
1	3	2	4	1	3	2	4
1	3	4	2	3	1	2	4
1	4	2	3	2	3	1	4
1	4	3	2	3	2	1	4
2	1	3	4	1	2	4	3
2	1	4	3	2	1	4	3
2	3	1	4	1	4	2	3
2	3	4	1	4	1	2	3
2	4	1	3	2	4	1	3
2	4	3	1	4	2	1	3
3	1	2	4	1	3	4	2
3	1	4	2	3	1	4	2
3	2	1	4	1	4	3	2
3	2	4	1	4	1	3	2
3	4	1	2	3	4	1	2
3	4	2	1	4	3	1	2
4	1	2	3	2	3	4	1
4	1	3	2	3	2	4	1
4	2	1	3	2	4	3	1
4	2	3	1	4	2	3	1
4	3	1	2	3	4	2	1
4	3	2	1	4	3	2	1

II. SURVEY

Before we discuss the proposed method, let us present a brief survey of some works related to permutations. Sedgewick published a well-known survey paper on the problem of permutation generation [31]. In that paper, he provides extensive discussion of numerous methods including those based on exchange and those based on some other elementary operations, for example, nested cycling or arithmetic addition and others. Recently, Knuth has published his fourth volume in the series of influential books entitled "*The Art of Computer Programming*." In this volume, he discusses several methods that have been proposed for permutation generation [16]. Doubtlessly, both are valuable for those who are devoted to this subject today.

Although various methods have been proposed to generate permutations in lexicographic order, they can be classified into two categories [25, 28, 29, 33]. Some of these studies require the generation of the next permutation from the beginning while others produce it by a small modification of the predecessor permutation [24]. The former enables us to generate permutations in any order we desire, while the latter restricts us to generate permutations one by one sequentially [13]. The method we proposed belongs to the former.

In addition to the problem of generating all permutations of n distinct items, several related topics have been studied, such as derangements [1, 3, 20], alternating permutations [4], random permutations [2, 8], cyclic permutations [10], permutations with inversions [9, 16], permutations with k-difference [30], permutations with multiset [5, 12, 13, 17, 35], alphametics [18], and parallel permutation generation [22], etc.

However, our concern in this study can be seen in the following quotation.

> "By an *orderly listing* of permutations we mean a generation for which it is possible to obtain the k^{th} permutation directly from the number k, and conversely, given a permutation, it is possible to determine at once its *rank*, or *serial* number, in the list without generating any of the other permutations." [21, p. 19]

In general, when we discuss a method of permutation generation it is inevitable to consider the ranking and unranking algorithm for permutations. A ranking algorithm converts each permutation in the S_n to an integer in the range of [0, n!-1] uniquely. In contrast, the corresponding unranking algorithm converts an integer in the range of [0, n!-1] to one permutation in the S_n uniquely. Myrvold and Ruskey proposed a ranking and unranking algorithm for random permutations [23]. In other words, the list of permutations generated by their algorithm is not in lexicographic order. According to their study, the ranking algorithm in lexicographic order seems inseparable from the problem of computing the number of inversions in a permutation. Let $\boldsymbol{\pi}=(\pi_1\pi_2\ldots\pi_n)$ be a permutation of n distinct items, and a pair of (π_i, π_j) is called an inversion of $\boldsymbol{\pi}$ if $i < j$ and $\pi_i > \pi_j$. Compbell also proposed a ranking and unranking algorithm, but that is not for generating permutations in lexicographic order [6].

On the other hand, there are some other studies that approach this problem from different angles. For example, Lehmer's method is based on the idea that each permutation can be thought of as a base k integer [20]. Howell expands Lehmer's method by adding more than one to each successive integer with a multiple of (k-1) radix k [11]. Then, Spoletini proved two formulas, using an arithmetic method, that can be used to generate the next permutation with respect to a given one and to generate the permutation which corresponds to an integer in the range of [0, n!-1], both in lexicographic order [34]. However, these two formulas are too complicated. On the other hand, the new method we proposed in this paper is conceptually easy to understand and implement and is well-suited to a wide variety of permutation problems.

III. REPRESENTATION SCHEMES

Representation schemes are of central interest in scientific research. Not only because that they provide us a way to realize the concept discussed, but also because that they enable us to manipulate the objects which they represent. In combinatorics and mathematics, several representation schemes have been used for permutation, including the two-line form [14], cycle notation [14], permutation matrix [7], inversion vector [27], inversion table [15]. We begin our discussion by introducing these representation schemes, and then present our proposal.

For convenience, we use the notation S_n to denote the set of all permutations of n distinct items $\{1,\ldots, n\}$. That is, a permutation $\boldsymbol{\pi}=(\pi_1\pi_2\ldots\pi_n)$ belongs to the S_n if and only if

$\pi_i \in \{1, 2,\ldots, n\}$, for all i=1, 2,…, n,

and

$\pi_i \neq \pi_j$, for all $i \neq j$.

A permutation $\boldsymbol{\pi}=(\pi_1\pi_2\ldots\pi_n)$ of n distinct items, for example (3 6 1 2 5 4), can be expressed, in the two-line form, as

$$\pi = \begin{pmatrix} 123456 \\ 361254 \end{pmatrix}$$

or, in the cycle notation, as

π=(13)(264)

with the meaning that $\boldsymbol{\pi}$ takes 1→3, 2→6, 3→1, 4→2, 5→5, and 6→4; a cycle like '(5)' need not be indicated.

A permutation matrix is an $n \times n$ matrix such that each row and each column has exactly one entry of 1, and 0's elsewhere. For example, the permutation matrix of the permutation (3 6 1 2 5 4) is

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Such a matrix is called a permutation matrix because of multiplying a vector $\boldsymbol{x}$ by it has the effect of permuting the elements of $\boldsymbol{x}$.

In contrast to those described above, in the following two kinds of presentations, a sequence of positive integers

$d_1\, d_2 \ldots d_n$

is associated with a permutation. However, these integers have different operational meanings in different cases. In the case of the inversion vector, d_j is the number of items greater than π_j and to its left in the permutation. Therefore, it satisfies the following constraints,

$$0 \le d_j \le j-1, \text{ for all } j=1, 2,\ldots, \text{n.} \quad (1)$$

For example, the inversion vector of the permutation (3 6 1 2 5 4) is 0 0 2 2 1 2. On the other hand, in the case of the inversion table, d_j is the number of items to the left of j that is greater than j. Therefore, it satisfies the following constraints,

$$0 \le d_j \le n-j, \text{ for all } j=1, 2,\ldots, \text{n.} \quad (2)$$

For example, the inversion table of the permutation (3 6 1 2 5 4) is 2 2 0 2 1 0. It should be noted that there are several other definitions of inversion table [15].

Each one of these representation schemes has its own characteristics and operational meaning. However, a good representation scheme for a permutation should not only be used for generating all permutations but also should have the property that it can be easily manipulated by simple arithmetic operations directly. Moreover, it should be flexible for different types of permutation problems. Now, from a different operational point of view, we propose a new representation scheme for a permutation that meets these goals.

Definition 1: For a permutation $\boldsymbol{\pi}$ in the form of ordinal representation, that is $[D_n D_{n-1} \ldots D_1]$, $\boldsymbol{\pi}$ belongs to S_n if and only if

$$1 \le D_j \le j, \text{ for all } j=1, 2,\ldots, \text{n.}$$

We call $[D_n\, D_{n-1} \ldots D_1]$ the ordinal digits of a permutation $\boldsymbol{\pi}$.

The meaning of ordinal digits is easy to understand if we imagine a permutation to be the result of a successive withdrawal of items, one after the other without replacement, from the item set $\{1, 2,\ldots, n\}$. At the beginning, there are n choices we can choose as the first component of $\boldsymbol{\pi}$. That is why we let $1 \le D_n \le n$. Once we choose an item as the first component of $\boldsymbol{\pi}$, there are n-1 choices left in the item set. So, we let $1 \le D_{n-1} \le n-1$. Finally only one choice left, so we let $1 \le D_1 \le 1$.

In other words, the component π_{n-j+1} of $\boldsymbol{\pi}$ is determined by D_j, and the value of D_j is one plus the number of items that are less than π_{n-j+1} and to the right of it. Since each permutation $\boldsymbol{\pi}$ in S_n corresponds uniquely to an integer k in the range of $[0, n!-1]$, we have the following theorem.

Theorem 1: In S_n, there is a one-to-one correspondence between $(\pi_1 \pi_2 \ldots \pi_n)$ and $[D_n D_{n-1} \ldots D_1]$.

Proof. Clearly, it is easy to convert an integer q to its factorial representation. First, we divide integer q by (n-1)! and the quotient is set to C_{n-1}, then the remainder is divided by (n-2)! and the quotient is set to C_{n-2}, and so on. That is, any integer q between 0 and n!-1 can be represented as

$$q = C_{n-1} \times (n-1)! + C_{n-2} \times (n-2)! + \cdots + C_1 \times 1! + C_0 \times 0! \quad (3)$$

Here, the following constraints

$$0 \le C_i \le i, \text{ for all } i=0, 1,\ldots, n-1, \quad (4)$$

are imposed to ensure uniqueness. These C_i's are called the factorial digits of integer q [21].

By Definition 1, we know that

$$1 \le D_j \le j, \text{ for all } j=1, 2,\ldots, n.$$

And from operational point of view, both factorial digits and ordinal digits are lexicographic. That is, $(C_{a,n-1}\ C_{a,n-2} \ldots C_{a,0})$ precedes $(C_{b,n-1}\ C_{b,n-2} \ldots C_{b,0})$ if and only if, for some $k \ge 0$, we have $C_{a,j}=C_{b,j}$ for all $j > k$ and $C_{a,k} < C_{b,k}$ Similarly, $(D_{a,n}$

$D_{a,n-1}$ … $D_{a,1}$) precedes ($D_{b,n}$ $C_{b,n-1}$ … $D_{b,1}$) if and only if, for some $k \geq 1$, we have $D_{a,j}=D_{b,j}$ for all $j > k$ and $D_{a,k} > D_{b,k}$ Hence, we have a *one-to-one* correspondence between D_j and C_i as follows:

$D_j=C_i+1$, where $j=i+1$, for all $j=1, 2,\ldots, n$. ■ (5)

Thus, if we order all permutations of S_n in lexicographic order then we can, for example n=4, use the ordinal digits [1 1 1 1] to represent the first permutation π=(1 2 3 4), [2 3 1 1] to 11th permutation π=(2 4 1 3), and [4 3 2 1] to the last permutation π=(4 3 2 1), respectively. It is easy to see that $D_n=\pi_1$ for all permutations of S_n. Table 2 illustrates, for example n=4, integers q and their corresponding factorial digits, ordinal digits, and permutation.

IV. ALGORITHMS

By using the ordinal representation, we can easily handle the ranking and unranking of permutation. In this paper we use the term "ranking" to refer to converting each permutation in the S_n to its ordinal digits uniquely, and "unranking" means to convert ordinal digits to its corresponding permutation

Table 2. Integers q and their corresponding factorial digits, ordinal digits, and permutation

q	C_3	C_2	C_1	C_0	D_4	D_3	D_2	D_1	π_1	π_2	π_3	π_4
0	0	0	0	0	1	1	1	1	1	2	3	4
1	0	0	1	0	1	1	2	1	1	2	4	3
2	0	1	0	0	1	2	1	1	1	3	2	4
3	0	1	1	0	1	2	2	1	1	3	4	2
4	0	2	0	0	1	3	1	1	1	4	2	3
5	0	2	1	0	1	3	2	1	1	4	3	2
6	1	0	0	0	2	1	1	1	2	1	3	4
7	1	0	1	0	2	1	2	1	2	1	4	3
8	1	1	0	0	2	2	1	1	2	3	1	4
9	1	1	1	0	2	2	2	1	2	3	4	1
10	1	2	0	0	2	3	1	1	2	4	1	3
11	1	2	1	0	2	3	2	1	2	4	3	1
12	2	0	0	0	3	1	1	1	3	1	2	4
13	2	0	1	0	3	1	2	1	3	1	4	2
14	2	1	0	0	3	2	1	1	3	2	1	4
15	2	1	1	0	3	2	2	1	3	2	4	1
16	2	2	0	0	3	3	1	1	3	4	1	2
17	2	2	1	0	3	3	2	1	3	4	2	1
18	3	0	0	0	4	1	1	1	4	1	2	3
19	3	0	1	0	4	1	2	1	4	1	3	2
20	3	1	0	0	4	2	1	1	4	2	1	3
21	3	1	1	0	4	2	2	1	4	2	3	1
22	3	2	0	0	4	3	1	1	4	3	1	2
23	3	2	1	0	4	3	2	1	4	3	2	1

uniquely. Let us begin with the unranking algorithm. First of all, we construct an item set that is an ordered list composed of all n items which are to be arranged. Usually, although it is not necessary, these n distinct items are numbered from 1 to n. In other words, the item set is initialized as {1, …, n} at the beginning of the process of generating a permutation.

Given an ordinal representation [$D_nD_{n-1}\ldots D_1$] of a permutation, we can generate the permutation as follows: for each D_j, $j=n, n-1, \ldots, 1$, output the D_j^{th} item of the item set and immediately delete it from the item set. This process is described in Algorithm 1 as follows.

Algorithm 1: Unranking the ordinal digits [$D_nD_{n-1}\ldots D_1$] to a permutation $\pi=(\pi_1\pi_2\ldots\pi_n)$ in S_n.

```
Input: n; [Dn Dn-1 ... D1]
Output: π=(π1 π2 ... πn)
Begin
  itemset A = {1, ..., n}
    For j=n To 1
          Retrieve the Dj th item of the itemset A
          Let πn+1-j = Dj th item of the itemset A
          Delete Dj th item of the itemset A
    Next j
End
```

Once we have the Algorithm 1, we can easily generate any permutation corresponding to an integer in the range of [0, n!-1]. Naturally, we can systematically generate the S_n in lexicographic order. This task can be done by Algorithm 2 as follows.

Algorithm 2: Generate S_n in lexicographic order.

```
Input: n
Output: all permutations π=(π1 π2 ... πn) in Sn.
Begin
For Dn=1 To n
    For Dn-1=1 To n-1
                  ...
      For D2=1 to 2
          For D1=1 to 1
             embed Algorithm 1 here
          Next D1
      Next D2
            ...
    Next Dn-1
Next Dn
End
```

By Theorem 1, we know that there is a one-to-one correspondence between D_j and C_i. Thus, in Algorithm 2, we can skip the operation of converting an integer q to its factorial digits C_i, but explicitly embed the conversion of all integers between 0 and n!−1 to D_j's. Consequently, it can be used

effectively to handle the problem of permutation generation even for a large number of n. However, those permutation generation methods which need to directly deal with an integer in the range of [0, n!-1] are unable to handle a large number of n because of computer hardware limitations.

Now, let us turn to consider the ranking algorithm. In order to convert a permutation in the S_n to its ordinal digits uniquely we design Algorithm 3 as follows.

Algorithm 3: Ranking a permutation $\boldsymbol{\pi}=(\pi_1\pi_2\ldots\pi_n)$ in S_n to its ordinal digits $[D_nD_{n-1}\ldots D_1]$.

```
Input: π=(π1 π2 ... πn)
Output: [DnDn-1...D1]
Begin
Itemset={1, ..., n}
For j=n To 2
   'By using a binary search, we let Dj=s, if πn-j+1=sth item of the itemset
      first=1
      last=j
      While first ≤ last Do
         s=⌊(first+last)/2⌋
         '⌊x⌋ means the greatest integer less than or equal to real number x
         If πn-j+1=sth item of the itemset then Dj=s; exit
         If πn-j+1 < sth item of the itemset then last=s–1 else first=s+1
      Wend
      Delete sth item of the itemset
Next j
Let D1=1
End
```

In summary, we show the conceptual framework of the proposed method in Figure 1. To illustrate the proposed method, we will give three examples as follows.

Example 1: How to generate the 11th permutation in S_4.

In case of n=4, the factorial digits $[C_3\ C_2\ C_1\ C_0]$ of integer 10 are [1 2 0 0]. Thus, by Theorem 1, we know that the ordinal digits $[D_4\ D_3\ D_2\ D_1]$ of the 11th permutation are [2 3 1 1]. Therefore, by using Algorithm 1, we first initialize the item set $\boldsymbol{A}$ to be {1 2 3 4}, then output the 2nd item, which here is 2, of $\boldsymbol{A}$ and delete it from $\boldsymbol{A}$. After this step, $\boldsymbol{A}$ becomes {1 3 4}. Next, we output the 3rd item, which here is 4, of $\boldsymbol{A}$ and delete it from $\boldsymbol{A}$. After this step, $\boldsymbol{A}$ becomes {1 3}. By following the same process, the 11th permutation we finally obtain is (2 4 1 3).

Example 2: How to generate the permutation immediately follows the 11th permutation in S_4.

That is, we want to generate the permutation in the S_4 immediately follows (2 4 1 3). This can be done by the following three steps: ranking (Algorithm 3), adding, and unranking (Algorithm 1). First of all, by using Algorithm 3, we rank the permutation (2 4 1 3) to its ordinal digits [2 3 1 1]. By Table 2, we know that the factorial digits of one are [0 0 1 0].

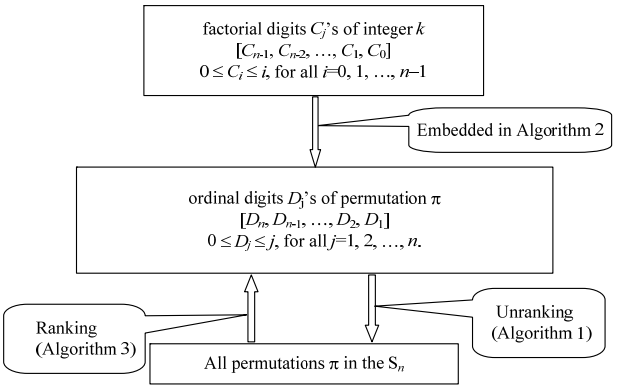

Fig. 1. Conceptual framework of the proposed method for generating permutations in lexicographic order

Thus, we add [2 3 1 1] by [0 0 1 0] to get the new ordinal digits [2 3 2 1]. It is noted that the operations of addition are under amixed-radix system that is based on 2, 3, and 4 from right to left, since $1 \le D_j \le j$, for j=2, 3, 4. Finally, as what we have done in Example 1**,** we unrank the new ordinal digits [2 3 2 1] to obtain the permutation (2 4 3 1) that is immediately following permutation (2 4 1 3).

More noteworthy is the fact that the three steps (ranking, adding, and unranking) in Example 2 also can be applied to generate a permutation that is positioned away from a given permutation by any specific distance.

Example 3: How to generate the permutation in S_4 that is positioned follows the 11th permutation by distance 8.

We can follow the three steps mentioned above except for adding 8 instead of adding one in the second step. That is, we add [2 3 1 1] by [1 1 0 0], the factorial digits of 8, to get the new ordinal digits [4 1 1 1]. Finally, we unrank the new ordinal digits [4 1 1 1] to obtain the permutation (4 1 2 3) that is positioned, in lexicographic order, away from permutation (2 4 1 3) by distance 8, i. e., the 19th permutation in S_4.

These examples show that the ordinal representation of a permutation has the property that it can be easily manipulated by simple arithmetic operations. By using the "ranking" and "unranking" algorithms, we can generate any permutation that is positioned, in lexicographic order, away from a given permutation by any specific distance. This significant merit makes our method different from other sophisticated algorithms for ranking and unranking permutations in lexicographic order that need an intermediate step, such as an inversion vector, for conversion between ranks and permutations.

V. DISCUSSION

Permutations are one of the most important combinatorial objects in computing. Evaluations of combination generators have been reported that the advantages of these generators include the following: simplicity, speed, distance-two, bidirectional, cyclic, historyless, and lexicographical [26]. Distance-two means that a single exchange of two items is required to generate the next combination. Bidirectional means that the algorithm can easily be coded to traverse the sequence in either forward or reverse order. Cyclic means that the first combination can be derived from the last combination by using the same rules for generating the next combination from the previous one. Historyless means that rules for generating the next combination depend only on knowledge of the current combination.

The new method we proposed possesses all the attributes mentioned above except for the distance-two. In fact, distance-two and lexicographical are two mutually exclusive attributes.

1. Simplicity

The algorithms described above are easy to implement by any computer programming language, especially those languages that provide data structures and fundamental operations that support the direct manipulation of a set of items.

2. Speediness

The time complexities of Algorithm 1 and Algorithm 3 are $O(lgn!)$, and of Algorithm 2 is $O(lgn!)$ per each permutation generation. What Algorithm 1 performs is the so called "unranking" function. The operations required for "unranking" are only retrieve and delete. Under an order-statistic tree, an augmented red-black tree which is a kind of balanced binary tree, the time "cost", in the worst-case hereafter, of the retrieve operation and the delete operation is $O(lgn)$ [7]. Thus, it is clear that the time complexity of Algorithm 1 is $O(nlgn)$. Precisely, it is $O(lgn!)$ since each time when the "For" loop is executed the size of item set is reduced by one.

Because of that all the nested "For" loops in Algorithm 2 are used for generating combinations of those factorial digits D_j of permutations and that the time complexity of Algorithm 1 is $O(lgn!)$, it is clear that the time complexity of Algorithm 2 is $O(lgn!)$ per each permutation generation. By using the same data structures used in Algorithm 1, the time complexity of Algorithm 3 is $O(lgn!)$. These results are as good as the best to our knowledge. Myrvold and Ruskey provide some discussion about the time complexity of ranking and unranking algorithms in lexicographic order [23].

3. Lexicography

Originally, in order to generate the S_n in lexicographic order we need to convert every integer k in the range of [0, n!-1] to its corresponding permutation in the S_n uniquely. With a slightly different approach, we do not convert integer k but convert ordinal digits D_j's to its corresponding permutation. In other words, we skip the operations both of converting an integer k to factorial digits C_i's and of converting C_i's to ordinal digits D_j's. That is, we directly embed the conversion of all integers in the range of [0, n!-1] to D_j's in the algorithm. This is why we use the lower and upper bounds of D_j's in each nested "For" loop statement of Algorithm 3.

4. Bidirectionality

It is noteworthy that if we reverse all nested "For" loops from the upper bound down to the lower bound by step-1 in Algorithm 2, and set π_j is the D_j^{th} item of the item set in Algorithm 1, then Algorithm 2 can generate the S_n in reverse lexicographic order.

5. Historylessness

It also should be emphasized that, by using Algorithms 1 and 3, we can generate the lexicographic successor of a given permutation depend only on knowledge of the given permutation. This task can be done, as Example 2 illustrated above, by following three steps: ranking (Algorithm 3), adding, and unranking (Algorithm 1). Since the time complexity of adding operation is $O(n)$, the total time complexity of these three steps is $O(lgn!)+O(n)+O(lgn!)=O(lgn!)$.

6. Cyclicality

Naturally, it is clear that if we add one to the ordinal digits [4 3 2 1] of the last permutation (4 3 2 1) then we will obtain the ordinal digits [1 1 1 1] of the first permutation (1 2 3 4).

In summary, based on the discussion mentioned above, the new method possesses attributes of simplicity, speediness, lexicography, bidirectionality, historylessness, and cyclicality. In terms of attribute speed, our algorithms may be further improved by using an advanced data structure.

Finally, there are three other important aspects in which the proposed method differs from previous studies. First, by using the ordinal representation, the proposed method is not restricted to number the n distinct items from one to n, sequentially. In other words, without the aid of mapping, this new method can directly generate the permutations of distinct items that are numbered, for example, by 3, 5, 8, 12, 18, and 31, or even with non-numeral marks, provided there exists a predefined order among these marks. The reason is that the meaning of ordinal digits is just the order of a successive

withdrawal of items, one after the other without replacement, from an item set. Second, the proposed method is well-suited to generate several special kinds of permutations that satisfy some predefined constraints, provided the constraints are embedded into the algorithm. For example, we can generate all alternating permutations or derangements in lexicographic order. Alternating permutations of {1, 2, …, *n*} are those permutations $\boldsymbol{\pi}=(\pi_1\ \pi_2\ \ldots\ \pi_n)$ such that $\pi_1 < \pi_2 > \pi_3 < \pi_4 > \ldots$. Derangements of {1, 2, …, *n*} are those permutations $\boldsymbol{\pi}=(\pi_1\ \pi_2\ \ldots\ \pi_n)$ such that $\pi_1 \neq 1, \pi_2 \neq 2, \ldots, \pi_n \neq n$. The reason is that we can embed those constraints into algorithm and generate permutations in the order we desire. In Table 3, we list alternating permutations and derangements of four distinct items {1 2 3 4} in lexicographic order respectively. Third, the proposed method also can be extended to a multiset. A multiset is a set that each item in the set has a multiplicity which specifies how many times the item repeats. The guideline of the extension is to skip, as soon as possible, those partially-formed permutations that are less than or equal to the latest generated eligible permutation. Multiset permutation can be applied in application of scheduling problems. Suppose that we have jobs to be processed through *n* machines and that each job must pass through each machine once and once only. These *n* machines can be classify into *m* types and the multiplicities of the *m* types of machine are $n_1, n_2, \ldots, n_m$ respectively and satisfy

$$n = \sum_{i=1}^{m} n_i \text{ , for all } n_i \geq 1. \qquad (6)$$

We shall suppose that the jobs can be processed through the *m* types of machine in an arbitrarily order. However, for a certain type *j* of machine, each job should follow a sequentially but not necessary successively order between those n_j machines. The questions are, for a job, how many feasible flows exist and how to generate them? The answer to the first question is

Table 3. *Alternating* permutations and *derangements* of {1, 2, 3, 4}

Alternating permutations				*Derangements*			
1	3	2	4	2	1	4	3
1	4	2	3	2	3	4	1
2	3	1	4	2	4	1	3
2	4	1	3	3	1	4	2
3	4	1	2	3	4	1	2
-	-	-	-	3	4	2	1
-	-	-	-	4	1	2	3
-	-	-	-	4	3	1	2
-	-	-	-	4	3	2	1

$$\binom{n}{n_1 n_2 \ldots n_m} = \frac{n!}{n_1!n_2!\ldots n_m!} \qquad (7)$$

and Algorithm 2 can be extended to the second question.

VI. CONCLUSIONS

The order of a list of all permutations of *n* items is determined by the method used to generate these permutations. However, if such an order has no specific characteristics that can be utilized then the permutation generation method is rather insufficient. In contrast, the lexicographic order is a natural and simple order, and we propose a permutation generation method that can utilize this intrinsic order. In this study, a new representation scheme of a permutation called ordinal representation is presented. By using this representation scheme, we first design an "unranking" algorithm that can generate a permutation of *n* distinct items according to its ordinal representation. Then, we design an algorithm that can easily and systematically generate all permutations of *n* distinct items in lexicographic order. We also design a "ranking" algorithm that can convert a permutation to its *ordinal* representation. It should be emphasized that our new method can be used to generate a permutation that is positioned, in lexicographic order, away from a given permutation by any specific distance. This significant merit is the main difference between our method and other previous studies. Particularly, our new method also can be used to generate permutations in Reverse Lexicographic order. In conclusion, the new method is conceptually easy to understand and implement and is well-suited to a wide variety of permutation problems. It is noteworthy that our new method has been applied to design an optimal algorithm for generating permutations with a fixed number of inversions in lexicographic order [19]. Therefore, we intend to continue pursuing this line of study in related topics.

REFERENCES

1. Akl, S. G. (1980) A new algorithm for generating derangements. *BIT Numerical Mathematics*, 20(1), 2-7.
2. Balbine, G. D. (1967) Note on random permutations. *Mathematics of Computation*, 21(100), 710-712.
3. Baril, J. and V. Vajnovszki (2004) Gray code for derangements. *Discrete Applied Mathematics*, 140(1-3), 207-221.
4. Bauslaugh, B. and F. Ruskey (1990) Generating alternating permutations lexicographically. *BIT Numerical Mathematics*, 30(1), 17-26.
5. Bratley, P. (1967) Permutations with repetitions (Algorithm 306). *Communications of the ACM*, 10(7),

450-451.

6. Campbell, W. H. (2004) Indexing permutations. *Journal of Computing Sciences in Colleges*, 19(3), 296-300.
7. Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein (2001) *Introduction to Algorithms*, 2nd Ed., The MIT Press, MA.
8. Durstenfeld, R. (1964) Random permutation (Algorithm 235). *Communications of the ACM*, 7(7), 420.
9. Effler, S. and F. Ruskey (2003) A CAT algorithm for generating permutations with a fixed number of inversions. *Information Processing Letters*, 86(2), 107-112.
10. Er, M. C. (1989) Efficient enumeration of cyclic permutations in situ. *International Journal of Computer Mathmttics*, 29(2-4), 121-129.
11. Howell, J. R. (1962) Generation of permutations by addition. *Mathematics of Computation*, 16(78), 243-244.
12. Hu, T. C. and B. N. Tien (1976) Generating permutations with nondistinct items. *The American Mathematical Monthly*, 83, 629-631.
13. Johnson, S. M. (1963) Generation of permutations by adjacent transposition. *Mathematics of Computation*, 17(83), 282-285.
14. Knuth, D. E. (1973) *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 2nd Ed., Addison-Wesley, Reading, MA.
15. Knuth, D. E. (1998) *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd Ed., Addison-Wesley, Reading, MA.
16. Knuth, D. E. (2005) *The Art of Computer Programming, Volume 4, Fascicle 2: Generating all Tuples and Permutations*, Pearson Education, Upper Saddle River, NJ.
17. Korsh, J. F. and P. S. LaFollette (2004) Constant time generation of deran gements. *Information Processing Letters*, 90(4), 181-186.
18. Korsh, J. F. and S. Lipschutz (1997) Generating multiset permutations in constant time. *Journal of Algorithms*, 25(2), 321-335.
19. Kuo, T. (2009) Using ordinal representation for generating permutations with a fixed number of inversions in lexicographic order. *Journal of Computers*, 19(4), 1-7.
20. Lehmer, D. H. (1960) Teaching combinatorial tricks to a computer. In: *Combinatorial Analysis*. Proceedings of Symposia in Applied Mathematics, 10, 179-193. R. Bellman and M. Hall, Jr., Ed. American Mathematical Society, Providence, RI.
21. Lehmer, D. H. (1964) The machine tools of combinatorics. In: *Applied Combinatorial Mathematics*, 5-31. E. F. Beckenbach, Ed. John Wiley & Sons, New York, NY.
22. Lin, C. J. (1991) Parallel permutation generation on linear array. *International Journal of Computer Mathematies*, 38, 113-121.
23. Myrvold, W. and F. Ruskey (2001) Ranking and unranking permutations in linear time. *Information Processing Letters*, 79(6), 281-284.
24. Nijenhuis, A. and H. S. Wilf (1978) *Combinatorial Algorithms: For Computers and Calculators*, 2nd Ed., Academic Press, New York, NY.
25. Ord-Smith, R. J. (1968) Generation of permutations in lexicographic order (algorithm 323). *Communications of the ACM*, 11(2), 117.
26. Payne, W. H. and F. M. Ives (1979) Combination generators. *ACM Transactions on Mathematical Software*, 5(2), 163-172.
27. Reingold, E. M., J. Nievergelt and N. Deo (1977) *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ.
28. Rosen, K. H., J. G. Michaels, J. L. Gross, J. W. Grossman, and D. R. Shier (2000) *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, Boca Raton, FL.
29. Savage, C. D. (1990) Generating permutations with k-differences. *SIAM Journal Discrete Mathematics*, 3(4), 561-573.
30. Schrack, G. F. and M. Shimrat (1962) Permutation in lexicographical order (algorithm 102). *Communications of the ACM*, 5(6), 346.
31. Sedgewick, R. (1977) Permutation generation methods. *ACM Computing Surveys*, 9(2), 137-163.
32. Sedgewick, R. (1983) *Algorithms*. Addison-Wesley, Reading, MA.
33. Shen, M. K. (1963) Generation of permutations in lexicographical order (algorithm 202). *Communications of the ACM*, 6(9), 517.
34. Spoletini, E. (1984) Generation of permutations following Lehmer and Howell. *Mathematics of Computation*, 43(168), 565-572.
35. Yen, L. (1994) A note on multiset permutations. *SIAM Journal Discrete Mathematics*, 7(1), 152-155.

收件：98.02.09　修正：98.04.21　接受：98.06.18