Abstract—In a previous work [12], we proposed a method for generating permutations in lexicographic order. In this study, we extend it to generate multiset permutations. A multiset is a collection of items that are not necessarily distinct. 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 combinations generation, since a combination of q items out of n items is a special case of multiset permutations that contain q 1s and (n-q) 0s.

Keywords—multiset permutation, lexicographic order, ranking, unranking, ordinal representation

I. INTRODUCTION

Permutations are one of the most important combinatorial objects in computing. In this study, we focus in multiset permutation. A multiset is a set that each item in the set has a multiplicity which specifies how many times the item repeats. For convenience, without loss of generality, we use notation $S_{m,n}$ to denote the set of all permutations of an n items multiset $\{n_1d_1, n_2d_2, \ldots, n_md_m\}$ that is composed of m distinct, but not necessary successive, integers $d_1,\ldots,d_m$. Here, the multiplicities of the m distinct integers $d_1,\ldots,d_m$ are $n_1, n_2, \ldots,n_m$, respectively, and satisfy the following constraint:

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

For example, the multiset $\{1, 1, 1, 2, 2, 3, 4, 4\}$ can be expressed as $\{3\cdot1, 2\cdot2, 1\cdot3, 2\cdot4\}$. The multiplicities of the items $1, 2, 3,$ and $4$ are $3, 2, 1, 2$ respectively. That is, a permutation $\pi = (\pi_1\pi_2\ldots\pi_n)$ belongs to the $S_{m,n}$ if and only if

$$\pi_i \in \{d_1,\ldots,d_m\}, \text{ for all } i = 1,\ldots,n, \quad (2)$$

and the multiplicities of those items satisfy (1). Moreover, there is no two permutations $\pi_a$ and $\pi_b$ both belong to the $S_{m,n}$ such that $\pi_{a,i} = \pi_{b,i}$ for all $i = 1,\ldots,n$. Clearly, the total number of $S_{m,n}$ is

$$\binom{n}{n_1n_2\ldots n_m} = \frac{n!}{n_1!n_2!\ldots n_m!}. \quad (3)$$

Obviously, a set is a special multiset that the multiplicity of each item is one. Many methods have been proposed on multiset permutation $[2, 3, 6, 10, 11, 17, 19]$. Although these methods have their own characteristics and merits, there is a common feature they share that none of them works in lexicographic order.

The order of a list of permutations is determined by the method used to generate them. However, if such an order without any specific characteristic that can be utilized then the permutation generation method is not good enough. In contrast, there is a nature order of all permutations called lexicographic, or alphabetical, order $[18]$. 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 $\pi_a = (\pi_{a,1}\pi_{a,2}\ldots\pi_{a,n})$ precedes permutation $\pi_b = (\pi_{b,1}\pi_{b,2}\ldots\pi_{b,n})$ if and only if, for some $i \geq 1$, we have $\pi_{a,i} = \pi_{b,i}$ for all $j < i$ and $\pi_{a,i} < \pi_{b,i}$ $[15]$. For example, the lexicographic order of six permutations of three distinct items $\{1, 2, 3\}$ is $(1\ 2\ 3) < (1 \ 3 \ 2) < (2 \ 1 \ 3) < (2 \ 3 \ 1) < (3 \ 1 \ 2) < (3 \ 2 \ 1)$.

Besides, there is a kind of “reverse lexicographic” ordering $[18]$ or called “reverse colex order” $[9]$, the result of reading the lexicographic sequence backwards and the permutations from right to left that is also of some interest. It is worthy to mention that, in passing, our method can be used to generate multiset permutations both in Lexicographic order and Reverse Lexicographic order. In Table 1, we list all 12 permutations of a four items multiset $\{1 \ 2 \ 2 \ 3\}$ in Lexicographic order and Reverse Lexicographic order respectively.
The reason why we concentrate this study on lexicographic order can be seen in the following remark. Furthermore, in the context of a backtrack search for all solutions to some problems, generation of solutions in lexicographic order might be preferred on aesthetic grounds, and has at least two practical advantages, namely (1) When a subset of solutions has been generated, it is immediately clear which permutations have been rejected up to the most recently generated solution. (2) It is easier to verify whether a particular permutation is present in the complete list of solutions if that list is in lexicographic order [7].

Since lexicographic order is a natural and simple order, we believe that it should be easy manipulated by a computer program. Why don’t we design a permutation generation method that can full-utilize this intrinsic order? This drives us to conduct this study.

In the rest of this paper, we will propose a simple and flexible method for generating multiset permutations in lexicographic order. In Section II, we first review a new representation scheme that is conceptually easy to understand and implement. In Section III, the ranking and unranking algorithms are proposed. Example and results are presented in Section IV. Finally, discussion and conclusions are summarized in Section V.

II. 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. Such as: two-line form [9], cycle notation [9], permutation matrix [4], inversion vector [16], inversion table [8], n-ary p-number [14], and a p-sequence [1].

Each one of these representation schemes mentioned above has its own characteristics and operational meaning. However, a good representation scheme of permutation should not only be used for generating all permutations but also should have a property that it can be easily manipulated by simple arithmetic operations directly. Moreover, it should be flexible for different types of permutation problems. From a different operational point of view, we proposed a new representation scheme of a permutation called ordinal representation that meets these goals [12]. Now, let us give a quick review on it.

Definition 1: For a permutation \( \pi \) in the form of ordinal representation, that is \( [D_1D_2\cdots D_n] \), it belongs to \( S_n \) if and only if

\[
1 \leq D_j \leq j, \text{ for all } j = 1, 2, \ldots, n.
\]

Here, we use notation \( S_n \) to denote the set of all permutations of an \( n \) items set and \( [D_1D_2\cdots D_n] \) is called ordinal digits of \( \pi \).

The meaning of ordinal digits is easy to understand, if we imagine that a permutation is the result of a successive withdrawing of items individually, one after the other without replacement, from an ordered item set \( \{1, 2, \ldots, n\} \). At the beginning of withdrawing, there are \( n \) choices we can choose to be the first component of \( \pi \). That is why the inequality \( 1 \leq D_n \leq n \) holds. Once we have chosen an item as the first component of \( \pi \), there are \( n-1 \) choices left in the ordered item set. So, we have \( 1 \leq D_{n-1} \leq n-1 \).

---

Table 1: Lexicographic order and Reverse Lexicographic order of \( S_4(1\cdot1, 2\cdot2, 1\cdot3) \).

<table>
<thead>
<tr>
<th>Lexicographic order</th>
<th>Reverse Lexicographic order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 3</td>
<td>1 2 2 3</td>
</tr>
<tr>
<td>1 2 3 2</td>
<td>2 1 2 3</td>
</tr>
<tr>
<td>1 3 2 2</td>
<td>2 2 1 3</td>
</tr>
<tr>
<td>2 1 2 3</td>
<td>1 2 3 2</td>
</tr>
<tr>
<td>2 1 3 2</td>
<td>2 1 3 2</td>
</tr>
<tr>
<td>2 2 1 3</td>
<td>1 3 2 2</td>
</tr>
<tr>
<td>2 2 3 1</td>
<td>3 1 2 2</td>
</tr>
<tr>
<td>2 3 1 2</td>
<td>2 3 1 2</td>
</tr>
<tr>
<td>2 3 2 1</td>
<td>3 2 1 2</td>
</tr>
<tr>
<td>3 1 2 2</td>
<td>2 2 3 1</td>
</tr>
<tr>
<td>3 2 1 2</td>
<td>2 3 2 1</td>
</tr>
<tr>
<td>3 2 2 1</td>
<td>3 2 2 1</td>
</tr>
</tbody>
</table>
Finally, only one choice is left, so it is $1 \leq D_i \leq 1$. In other words, the component $\pi_{n-j+1}$ of $\pi$ is determined by $D_j$. Intrinsically, the value of $D_j$ is one plus the number of items that are less than $\pi_{n-j+1}$ and to the right of it.

Since each permutation $\pi$ in $S_n$ corresponds uniquely to an integer $q$ 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 $[D_0,D_1,\cdots,D_{j-1}]$ and $(\pi_1,\pi_2,\cdots,\pi_n)$.

**Proof:** Clearly, it is easy to convert any integer $q$ between 0 and $n! - 1$ into its factorial representation [13]. First, we divide $q$ by $(n-1)!$ and set the quotient to be $C_{n-1}$, then the remainder is divided by $(n-2)!$ and the quotient is set to be $C_{n-2}$, and so on. That is, any integer $q$ between 0 and $n! - 1$ can be represented as

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

Here, the following constraints

$$0 \leq C_i \leq i, \text{ for all } i = 0, 1, \ldots, n-1,$$

are imposed to ensure uniqueness. These $C_i$’s are called factorial digits of integer $q$ [13]. Thus, we have

$$1 \leq C_i + 1 \leq i + 1, \text{ for all } i = 0, 1, \ldots, n-1,$$

By Definition 6, we know that

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

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

$$D_j = C_i + 1, \text{ where } j = i + 1, \text{ for all } j = 1, 2, \ldots, n.$$ □

Thus, if we order all permutations of $S_n$ in lexicographic order then we can, for example $n = 7$, use the ordinal digits $[1111111]$ to represent the first (i.e., $0^{th}$) permutation $\pi = (1234567)$, $[1532211]$ to 536$^{th}$ permutation $\pi = (1643527)$, and $[7654321]$ to the last (i.e., 5039$^{th}$) permutation $\pi = (7654321)$, respectively. It is easy to see that $D_n = \pi_1$ for all permutations of $S_n$.

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 uniquely. In next section, we will describe how to generating multiset permutations in lexicographic order by using ordinal representation scheme.

### III. Algorithms

In general, when we mention a method of permutation generation it is inevitable to talking about the ranking and unranking algorithm. A ranking algorithm converts each permutation in $S_{a,n}$ of an $n$ items multiset $(n_1 \cdot d_1, n_2 \cdot d_2, \ldots, n_m \cdot d_m)$ to an integer in the range of uniquely.

$$\left[0, \frac{n!}{n_1!n_2!\cdots n_m!} - 1\right]$$ (4)

In contrast, the corresponding unranking algorithm converts an integer in the range of (4) to one permutation in $S_{a,n}$ uniquely. Knuth mentioned a recurrence formula of ranking for permutations of a multiset. However, there is no unranking formula has been proposed up to date.

Now, let us turn to the main subject of this paper: generate $S_{a,n}$ in lexicographic order. By using ordinal representation scheme, we can easily handle multiset permutation. First of all, we construct a multiset that is an ordered list composed of all $n$ items which are to be arranged. Thus, given an ordinal representation $[D_0,D_1,\ldots,D_{j-1}]$ of a permutation, we can generate the permutation as follows. For each $D_i$, $i = n,n-1,\ldots,1$, output the $D_i$th item of the multiset and immediately delete it from the multiset. For example, if we want to generate the 10$^{th}$ permutation in $S_{1,4}$ of a multiset $(1 \cdot 1, 2 \cdot 2, 1 \cdot 3)$, we initialize the multiset to be $(1 \cdot 2 \cdot 2 \cdot 3)$. Since the ordinal digits of the 10$^{th}$ permutation are $[4111]$, we first output the 4$^{th}$ item, here is 3, of the multiset and delete it from the multiset.
International Journal of Emerging Technology and Advanced Engineering

After the first step, the multiset becomes \{1 2 2\}. Next, we output the 1\textsuperscript{st} item, here is 1, of the multiset and delete it from the multiset. After this second step, the multiset becomes \{2 2\}. By following the same process, the 10\textsuperscript{th} permutation in \(S_{4,1}\) of a multiset \{1 \cdot 1, 2 \cdot 2, 1 \cdot 3\} we finally obtain \(3 1 2 2\). This process is described in Algorithm 1 as follows.

**Algorithm 1: Unranking the ordinal digits** \([D_n D_{n-1} \ldots D_1]\) to a permutation \(\pi = (\pi_1 \pi_2 \ldots \pi_n)\) in the \(S_{m,n}\).

**Input:** \(n; \) a multiset \(M = \{ n_1 \cdot d_1, n_2 \cdot d_2, \ldots, n_m \cdot d_m \} ; \) \(D_n D_{n-1} \ldots D_1\)

**Output:** \(\pi = (\pi_1 \pi_2 \ldots \pi_n)\)

**Begin**

For \(j = n\) To 1

\(\text{Retrieve the } D_j \text{th item of } M\)

\(\text{Let } \pi_{n+1,j} = D_j \text{th item of } M\)

\(\text{Delete } D_j \text{th item of } M\)

Next \(j\)

Output \(\pi = (\pi_1 \pi_2 \ldots \pi_n)\)

**End**

Once we have the **Algorithm 1**, we can easily generate any permutation corresponding to an integer \(k\) in the range of (4). Naturally, we can systematically generate the whole \(S_{m,n}\) in lexicographic order. Since a multiset is a collection of items with repetitions, the major problem of generating the whole \(S_{m,n}\) is how to avoid generating permutations that have been generated. Straightforwardly, we can treat all items of a multiset as “distinct” and generate permutations in the same manner as mentioned in Algorithm 1 but skip those permutations that are generated already. However, the check of duplication is a bottleneck. Fortunately, since we follow the lexicographic order, naturally, the guideline on the check of duplication is to skip those permutations that are less than or equal to the latest generated eligible permutation. This task can be done by **Algorithm 2** as follows.

**Algorithm 2: Generate** \(S_{m,n}\) **in lexicographic order.**

**Input:** \(n; \) a multiset \(M = \{ n_1 \cdot d_1, n_2 \cdot d_2, \ldots, n_m \cdot d_m \}\).

**Output:** All permutations \(\pi = (\pi_1 \pi_2 \ldots \pi_n)\) belong to the \(S_{m,n}\).

**Begin**

For \(j = 1\) To \(n\)

\(\text{Initialize the latest generated eligible permutation } \pi\)

\(\text{Let } \pi_j = -1\)

Next \(j\)

For \(D_n = 1\) To \(n\)

For \(D_{n-1} = 1\) To \(n - 1\)

\(\ldots\)

For \(D_2 = 1\) to 2

For \(D_1 = 1\) to 1

Let multiset \(M = \{ n_1 \cdot d_1, n_2 \cdot d_2, \ldots, n_m \cdot d_m \}\)

\(\text{Ok} = 0\)

For \(j = n\) To 1

\(\text{Retrieve the } D_j \text{th item of } M\)

\(\text{If } \text{Ok} = 1 \text{ then}\)

\(\text{Let } \pi_{n+1,j} = D_j \text{th item of } M\)

\(\text{Delete } D_j \text{th item of } M\)

Goto Next \(j\)

Else if \(D_j \text{th item of } M > \pi_{n+1,j}\) then

\(\text{Let } \pi_{n+1,j} = D_j \text{th item of } M\)

\(\text{Delete } D_j \text{th item of } M\)

\(\text{Ok} = 1\)

Goto Next \(j\)

Else Select Case \(j\)

Case 1

Goto Next \(D_2\)

Case else

\(\text{if } D_j \text{th item of } M = \pi_{n+1,j}\) then

\(\text{Let } \pi_{n+1,j} = D_j \text{th item of } M\)

\(\text{Delete } D_j \text{th item of } M\)

Goto Next \(j\)
Else Select Case \( j \)
   Case \( n \)
      Goto Next \( D_n \)
   Case \( n-1 \)
      Goto Next \( D_{n-1} \)
   \ldots
   Case 3
      Goto Next \( D_3 \)
   Case 2
      Goto Next \( D_2 \)
End Select
Endif
End Select
Endif
Endif
Endif
Next \( j \)

Output \( \pi = (\pi_1, \pi_2, \ldots, \pi_n) \)

Next \( D_1 \)
Next \( D_2 \)
\ldots
Next \( D_{n-1} \)
Next \( D_n \)

End

Originally, in order to generate the whole \( S_{m,n} \) in lexicographic order we need to convert every integer \( k \) in the range of (4) to corresponding permutation in the \( S_{m,n} \) uniquely. With a slightly different, we do not convert integer \( k \) but convert ordinal digits \( D_j \)'s to corresponding permutation. In other words, we omit both operations of converting an integer \( k \) to factorial digits \( C_i \)'s and converting \( C_i \)'s to ordinal digits \( D_j \)'s.

That is, we directly embed the conversion of all integers in the range of (4) 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 2. Consequently, it can effectively be used to handle permutation generation even for a big \( n \). However, those permutation generation methods which have to directly deal with an integer in the range of (4) are unable to handle a big \( n \) because of the limitation of computer hardware.

Now, let us turn to the ranking algorithm. On the other hand, in order to convert each permutation in the \( S_{m,n} \) to its ordinal digits uniquely, we design Algorithm 3 as follows.

**Algorithm 3: Ranking a permutation \( \pi = (\pi_1, \pi_2, \ldots, \pi_n) \) in \( S_{m,n} \) to its ordinal digits \([D_n, D_{n-1}, \ldots, D_1]\).**

\[
\text{Input: } \pi = (\pi_1, \pi_2, \ldots, \pi_n) \\
\text{Output: } [D_n, D_{n-1}, \ldots, D_1]
\]

**Begin**

Let multiset \( M = \{ n_1 \cdot d_1, n_2 \cdot d_2, \ldots, n_m \cdot d_m \} \)

For \( j = n \) To 2
   Let \( D_j = r \), if \( \pi_{n-j+1} = \pi^{th} \) item of \( M \) ‘By using a binary search
   Delete \( \pi^{th} \) item of \( M \)
Next \( j \)

Let \( D_2 = 1 \)

**End**

**IV. Example And Results**

For example, the multiset \( \{0, 0, 1, 1, 1, 1, 1, 1, 1\} \) can be expressed as \( \{2 \cdot 0, 7 \cdot 1\} \). Totally, there is 362280 permutations, but actually only 36 permutations are uniquely. By using an Acer \textregistered\ notebook with an Intel® Core™ 2 CPU T5600 @ 1.83GHz and a VBA program executed under the Microsoft Excel environment, we generate the 36 permutations lexicographically and totally escape 92429 duplicated permutations in one second. Table 2 shows the 36 permutations in Lexicographic order.
**Table 2**

Lexicographic order of $S_{2,5}(20, 7 \cdot 1)$

<table>
<thead>
<tr>
<th>0 0 1 1 1 1 1 1</th>
<th>1 1 1 0 1 1 0 1</th>
<th>1 1 1 0 1 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 1 1 1 1</td>
<td>1 1 1 0 1 1 1 0 1</td>
<td>1 1 1 0 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 0 1 1 1 1 1</td>
<td>1 1 1 1 0 1 1 1 0</td>
<td>1 1 1 1 0 1 1 1 0 1</td>
</tr>
<tr>
<td>0 1 1 1 0 1 1 1 1</td>
<td>1 1 1 1 0 1 1 1 0 1</td>
<td>1 1 1 1 0 1 1 1 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 0 1 1</td>
<td>1 1 1 1 1 0 1 1 1 0</td>
<td>1 1 1 1 1 0 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 0 1</td>
<td>1 1 1 1 1 1 0 1 1 0</td>
<td>1 1 1 1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 0</td>
<td>1 1 1 1 1 1 1 0 0 1</td>
<td>1 1 1 1 1 1 1 1 0 1</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 0 1</td>
<td>1 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 0</td>
<td>1 1 1 1 1 1 1 1 1 0</td>
<td>1 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

**V. DISCUSSION AND CONCLUSIONS**

Multiset permutation can be applied in combinations generation, since a combination of $q$ items out of an $n$ items set is a special case of multiset permutations that contain $q$ 1s and $(n-q)$ 0s. For example, a combination of 4 items out of 8 items can be described as 10011001 to stand for we pick up the first item, the 4th item, the 5th item, and the last item. Obviously, this is a permutation of a multiset $\{4 \cdot 0, 4 \cdot 1\}$.

The algorithms described above are easy to implement by any computer programming language especially those provide data structures and fundamental operations that support us to directly manipulate a set of items.

By using the ordinal representation, our new method is not restricted to number the $n$ items from 1 to $n$, successively.
In other words, without any aid of remapping, we 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 exist a predefined order among these marks.

It is interesting to note that if we reverse all nested “For” loops in the Algorithm 2 from upper bound down to lower bound by step -1 and set \( \pi_j \) be the \( D_j \)th item of the item set, then Algorithm 2 can generate the whole \( S_{n,p} \) 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. Therefore, we intend to continue pursuing this line of study in related topics.

REFERENCES