基于逆序列的字典序改进算法

孔涛* 李文焯†

清华大学计算机科学与技术系，北京 100084

摘要：字典序法是生成全排列的经典算法。本文在对字典序法进行分析的基础上，提出了一种基于逆序列的改进字典序全排列生成算法。通过与传统的四种全排列生成算法进行对比，本文方法可以大大提高全排列的生成效率。

关键词：全排列；字典序；逆序列

1 引言

全排列问题是组合数学中的经典问题。全排列在解决很多问题中作为基础算法出现。目前主要的全排列生成算法有字典序法、递增进位制数法、递减进位制数法和邻位对换法（SJT）[1]。在这四种算法中，字典序法在实际应用中最为普遍。字典序法借鉴字典中字符的排列顺序，在应用中更为直观。C++ 标准程序库中有两个函数 next_permutation, prev_permutation，可以生成字典序排列。因此，如何更快地利用字典序法生成全排列一直是科研工作者的目标。

2 字典序法

2.1 算法流程

字典序法生成全排列的算法如下，设 $P = p_1, p_2, \ldots, p_n$ 是 $1-n$ 的全排列, 下一个排列的生成算法为

- 从排列的右端开始，找出第一个比右边数字小的数字的序号 j（j 从左端开始计算），即 $j = \max\{i | p_i < p_{i+1}\}$。
- 在 p_j 的右边的数字中，找出所有比 p_j 大的数中最小的数字 p_k，即 $k = \max\{i | p_i > p_j\}$（右边的数从右至左是递增的，因此 k 是所有大于 p_j 的数字中序号最大者）。

*通讯作者：kt14@mails.tsinghua.edu.cn
†li-wz14@mails.tsinghua.edu.cn

- 对换 p_j，p_k。
- 再将 $p_{j+1},\ldots,p_{k-1},p_k,p_{k+1},\ldots,p_n$ 倒转得到排列 $P' = p_1p_2\ldots p_{j-1}p_jp_n\ldots p_{k+1}p_kp_{k-1}\ldots p_{j+1}$，这就是排列 P 的下一个下一个排列。

要生成 $1-n$ 的全排列，那么算法的初始序列为 $123\ldots n$，终止序列为 $n(n-1)\ldots 1$。

2.2 算法的正确性

我们首先给出比较两个序列 $P^1 = p_1^1, p_2^1, \ldots, p_n^1$ 和 $P^2 = p_1^2, p_2^2, \ldots, p_n^2$ 大小的定义：假设 P^1 和 P^2 的前 $j(0 < j < n)$ 个数均相同，若 $p_j^1 > p_j^2$，则 $P^1 > P^2$，若 $p_j^1 < p_j^2$，则 $P^1 < P^2$。特别的，若 $p_j^1 = p_j^2$，则 $P^1 = P^2$。

要证明这个算法的正确性，我们只要证明生成的下一个排序是恰好比当前排列大的最小的一个序列即可。

- 第一步找出的序列 $p_{j+1}, p_{j+2}, \ldots, p_n$ 是一个递减序列
- 第二步和第三步得到 $p_1, p_2, \ldots, p_{j-1}, p_k, p_{j+1}, \ldots, p_{k-1}, p_j, p_{k+1}, \ldots, p_n$。因为 $p_k > p_j$，这就保证了新的序列大于原来的序列。
- 第四步通过将 p_k 后边的序列逆转（原序列是递增序列），从而得到比当前序列大的最小的序列。

3 字典序法的改进

首先给序列的逆序列的概念。

定义 1 *序列 P 的逆：设原序列 $P = p_1, p_2, \ldots, p_m$，将 P 按照元素从小到大的顺序排列得到序列 $R = r_1, r_2, \ldots, r_m$，按照元素从大到小的顺序得到序列 $S = s_1, s_2, \ldots, s_m$。则序列 P 的逆序列 Q 的元素 q_i 定义为*

$$q_i = r_j \; where \; s_j = p_i, i = 1, 2, \ldots, m \tag{1}$$

记为 $\bar{P}$。例如序列 1234 的逆序列为 4321，13462 的逆序列为 63214。

定理 3.1 *不同序列的逆序列也不同。*

定理 3.2 *字典序法生成的前 $\frac{n!}{2}$ 个序列的逆序列均大于第 $\frac{n!}{2}$ 个序列。*

定理 3.1的正确性是显然的，这里给出定理 3.2的证明：令 A_P 表示 $1-n$ 的一个排列 P 对应的递增进位制数，显然有 $A_P \leq n!-1$。若 P 位于字典序的前 $\frac{n!}{2}$，则有 $A_P \leq \lfloor \frac{n!-1}{2} \rfloor$，注意到 $A_P + A_{\bar{P}} = n!-1$，所以 $A_{\bar{P}} = n!-1-A_P \geq \lceil \frac{n!-1}{2} \rceil$。定理获证。

定义 2 *序列 P 的 j 后部分逆：设序列 $P = p_1, p_2, \ldots, p_{j-1}, p_j, p_{j+1}, \ldots, p_m$，$Q = p_{j+1}, p_{j+2}, \ldots, p_m$，$R = \bar{Q}$，则序列 P 的 j 后部分逆为 $\bar{P_j} = p_1, p_2, \ldots, p_{j-1}, p_j, R$。*

定理 3.3 *设 $1-n$ 的排列 $P = p_1, p_2, \ldots, p_{j-1}, p_j, p_{j+1}, \ldots, p_n$，易知以 $p_1, p_2, \ldots, p_{j-1}, p_j$ 开头的排列的个数为 $(n-j)!$。则字典序法生成的以 $p_1, p_2, \ldots, p_{j-1}, p_j$ 开头的前 $\frac{(n-j)!}{2}$ 个序列的 j 后部分逆序列均大于以 $p_1, p_2, \ldots, p_{j-1}, p_j$ 开头的第 $\frac{(n-j)!}{2}$ 个序列。*

可以用类似于 3.3的证法证明：将 $p_{j+1}, p_{j+2}, \ldots, p_{n-1}, p_n$ 按照元素的大小顺序映射到 $1-(n-j)$ 的序列，我们只关心序列的大小顺序而不关心具体的数值，那么可以将该问题转化为 j 后的 $1-(n-j)$ 的全排列问题。若只考虑后 $1-(n-j)$ 的排列，根据定理 3.2，前 $\frac{(n-j)!}{2}$ 个序列的逆序列均大于第 $\frac{(n-j)!}{2}$ 个序列。定理获证。

3.1 算法描述

根据定理 3.1和 3.2，我们产生一个字典序的排列时（该排列位于前 $n!/2$），通过对该序列求逆序列，从而调用一次字典序法可以生成两个序列。根据定理 3.3，以 $p_1, p_2, \ldots, p_{j-1}, p_j$ 开头的序列同样可以通过求 j 后逆序列的方式产生新的排列。我们用字典序法产生一个排列 P 时，可以通过产生 $\bar{P}$ 以及 $\bar{P_j}$ 来进一步提高字典序法产生全排列的速度，当 $j=1$ 时，理论上可以达到原来字典序法效率四倍的性能，下面给出改进的字典序法的程序流程。

设 $P = p_1, p_2, \ldots, p_n$ 是 $1-n$ 的一个全排列, 改进的字典序法的流程为

- 产生 P 的逆序列 $\bar{P}$ 输出。
- 依次产生 $Q_j = \bar{P_j}$，$j = 1, 2, \ldots, k$ 输出。
- 依次产生 Q_j 的逆序列 $\bar{Q_j}$ 输出。
- 利用字典序法产生 P 的下一个序列。

下面以举例的形式演示该算法的执行过程。

例 1 *当 $j=0$ 时，该算法只产生 $\bar{P}$。$P = 1243$，则 $\bar{P} = 4312$，这样调用一次字典序法可以产生两个序列。算法初始的序列为 1234，终止序列为 2431，全排列的产生顺序如表 1所示。当 $j=0$ 时，本文的改进算法只需 $n!/2$ 次操作，而传统的字典序法需要 $n!$ 次操作。*

表 1: 当 $j=0, n=4$ 时的改进的字典序法的执行流程

产生顺序	1	2	3	4	5	6
P	1234	1243	1324	1342	1423	1432
$\bar{P}$	4321	4312	4231	4213	4132	4123
产生顺序	7	8	9	10	11	12
P	2134	2143	2314	2341	2413	2431
$\bar{P}$	3421	3412	3241	3214	3142	3124

例 2 当 $j=1$ 时，该算法产生 $\bar{P}$，$Q=\bar{P}_1$ 以及 $\bar{Q}$。$P=1243$，则 $\bar{P}=4312$，$Q=1423$，$\bar{Q}=4132$，这样调用一次字典序法可以产生四个序列。算法初始的序列为 1234，终止序列为 2314，全排列的产生顺序如表 *2*所示。当 $j=1$ 时，本文的改进算法只需 $n!/4$ 次操作，而传统的字典序法需要 $n!$ 次操作。

表 2: 当 $j=1, n=4$ 时的改进的字典序法的执行流程

产生顺序	1	2	3	4	5	6
P	1234	1243	1324	2134	2143	2314
$\bar{P}$	4321	4312	4231	3421	3412	3241
Q	1432	1423	1342	2431	2413	2341
$\bar{Q}$	4123	4132	4213	3124	3142	3214

从这两个例子可以看出，本文提出的改进的字典序法在产生一个字典序排列时可以产生对应的逆序列和 j 后逆序列，从而大大加快字典序法生成全排列的速度。

4 实验与分析

为了检验本文提出的基于逆序列的字典序法的改进算法的效率和正确性，我们将本文方法和传统的字典序法进行对比。实验部分在内存为 1GB 的 PC 机上进行，操作系统为 CentOS，编程语言采用 C 语言。为了得到更准确地算法运行时间，每个实验运行了 10 次，并取运行时间的平均值。计算运行时间时除去了 I/O 开销。

表 3: 改进的字典序法与传统字典序法的效率对比（单位：ms）

	传统字典序法	改进算法 $j=0$	改进算法 $j=0$	SJT 算法
$n=10$	120	60	50	60
$n=11$	1050	530	400	450
$n=12$	11890	5850	5100	5310

从表 3可以看出，本文提出的基于逆序列的字典序改进算法有效地提高了传统字典序法的性能，使得运行时间少于原来的一半[1]。同时我们也看到，当 $j=1$ 时，改进算法

[1]我们将本文方法分别与四种经典的全排列生成方法均进行了对比，但递增进位制数和递减进位制数效率较低，因此比较结果没有在正文中展示，详细见附录。

没有达到理论值（效率是传统方法的四倍），原因是当 $j=1$ 时，在程序中需要加入更多的分支和判断条件，使得算法效率没有期望的高。经典的全排列方法除了字典序法，还有递增进位制数、递减进位制数和邻位对换法。邻位对换法中下一个排列总是上一个排列某相邻两位对换得到的，只需一步，就可以得到一个新的全排列，而且绝不重复，是四种方法中效率最高的算法。因此我们将本文方法与邻位对换法（SJT）进行对比。从表 3 可以看出，当 $j=1$ 时，本文实现的基于逆序列的字典序改进算法在速度上已经优于邻位对换法。这也证明了本文方法的有效性。对于 $j>1$ 的情况，我们没有给出具体的实验结果，但理论上算法效率可已经一步提高。

5 结语

为了有效提高全排列的字典序生成算法的生成效率，本文提出了一种基于逆序列的字典序算法。首先对算法的正确性进行了证明，然后给出具体的实现过程。实验结果表明，本文方法可以有效地提高字典序法的生成速度。

参考文献

[1] Pruesse G, Ruskey F. Generating the linear extensions of certain posets by transpositions[J]. SIAM Journal on Discrete Mathematics, 1991, 4(3): 413-422.

附录

在实验对比部分，我们将本文实现的字典序改进算法与四种传统的全排列生成算法进行了对比。实验结果见表 4和图 1所示。源程序见附件。

表 4: 改进的字典序法与传统方法的效率对比（单位：ms）

	递增进位制数	递减进位制数	邻位对换法	传统字典序法	改进字典序法 j=0	改进字典序法 j=1
n=8	10	10	0	0	0	0
n=9	90	100	0	0	0	0
n=10	720	970	60	120	60	50
n=11	8580	11570	450	1050	530	400
n=12	114650	158220	5310	11890	5850	5100

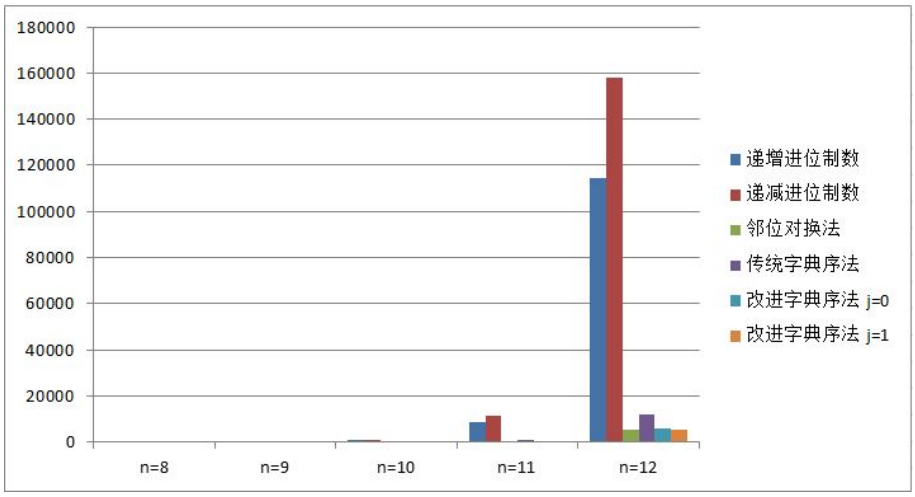

图 1: 改进的字典序法与传统方法的效率对比（单位：ms）