元素可重复的全排列数的中介数法

贾子洲，黄晓昱

November 11, 2018

1 摘要

本文提出了以一种针对于元素可重复的全排列数的中介数方法。该方法建立了包含重复元素的排列数和特定进制的中介数之间的一一映射关系，同时使得任意的带有重复元素的排列数都存在对应的中介数表示以及使得任意中介数都存在唯一的排列数与之对应，从而实现了通过任意数值索引在固定时间内获得该索引所对应的带有重复元素的排列数的方法。

关键词：元素可重复的全排列数，中介数法

2 介绍

全排列生成算法广泛应用于计算机领域，目前，历史上已经有多种不重复元素的全排列数的中介数法被提出和实现，但针对于可重复元素全排列数的中介数方法则相对较少。本文针对于生成存在可重复元素的全排列数提出了一种基于的中介数方法，该方法在可重复元素的全排列数和特定进制的中介数之间建立了一一映射关系，使得特定的带有重复元素的排列数能够映射为唯一对应的中介数，同时对于特定中介数也能够映射为唯一对应的排列数，同时对于任意排列数存在对应的中介数表示。

3 相关工作

基于中介数的全排列生成算法常见的有四种，分别是字典序法、递增进位制法、递减进位制法和邻位对换法。对于排列P=$P_1P_2,,P_n$,字典序法的中介数是递增进位制数，用$(k_1,k_2,,k_{n-1})\uparrow$表示，对应的每一位的进制为(n+1,n,⋯,2),记录当前数字P_i右边比当前数字小的数字数，较为麻烦。递增进位制数法的中介数也是递增进位制数，用$(a_n,a_{n-1},,a_2)\uparrow$表示，对应的每一位的进制为(n,n-1,⋯,2),记录数字i右边比i小的数字数，建立位置和数字间的对应关系,但进位频繁。递减进位制数法的中介数是递减进位制数，用$(a_2,a_3,,a_n)\downarrow$表示，对应的每一位的进制为(2,3,⋯,n),记录数字i右边比i小的数字数，建立位置和数字间的对应关系，进位不频繁。邻位对换法的中介数是递减进位制数，用$(b_2,b_3,,b_n)\downarrow$表示，对应的每一位的进制为(2,3,⋯,n),记录背向i的方向所有比数字i小的数字数，建立位置和数字间的对应关系，通过保存数字的方向性来快速得到下一个排列。而针对存在重复元素的全排列数，我们也思考能否找到一种新的的中介数解释方法来唯一映射对应的排列，因此提出我们的算法。

4 算法思想

4.1 整体思想

首先考察不存在重复元素的全排列数对应的递增进位制的中介数的每一位的进制值，则从最高位到最低位的进位制值依次为n,n-1,...2，该列值的另外一种表达形式为C(n,1),C(n-1,1),C(n-2,1),C(n-3,1),...,C(2,1).

假设中介数的最高位为第0位，中介数的最低位为第n-2位，则对于中介数第i位的进制值$(0\leq i\leq n-2)$可解释为，在剩余n-i个空位中选取一个空位填充第i+1大的数字的总的可能行，该位的元素的值共有0到n-i-1种可能性，对应了在剩余n-i个空位中选取一个位置的n-i种可能，则中介数每一位选取数字的所有可能性遍历了排列数的所有可能。

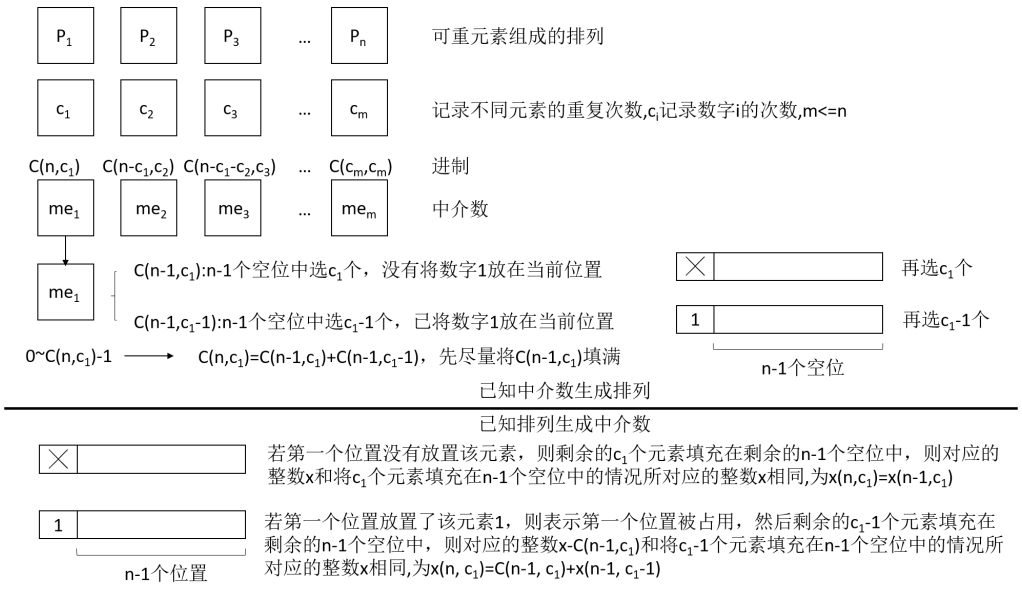

Figure 1: 有重复元素的全排列算法

将该解释推广到可重复元素全排列数中去，形式如下：

假设存在m种不同的元素，每种元素用e(i)来表示$(0 \leq i \leq m-1)$，对于每个i，元素e(i)的元素数量为c(i)。

现在构造一个m-1位的中介数，设定中介数的最高位为第0位，最低位为第m-2位，中介数从最高位到最低位的进位制依次为

$$C(\sum_{i=0}^{m-1} c(i), c(0)), C(\sum_{i=1}^{m-1} c(i), c(1)), ..., C(\sum_{i=m-2}^{m-2} c(i), c(m-2))$$

即中介数的第k位的进位制为$C(\sum_{i=k}^{m-1} c(i), c(k))$。

则对于第k位的进位制的解释即为，从剩余的$\sum_{i=k}^{m-1} c(i)$个空位中，选取c(k)个位置填充元素e(k)的所有可能性，该中位数第k位元素的大小在0到$C(\sum_{i=k}^{m-1} c(i), c(k))$-1之间，每种值都代表了一种填充的可能性，则中介数的每一位选取数字的所有可能性遍历了可重复元素排列数的所有可能。

现只需要找到一种映射方案，使得对于给定中介数第k位元素的值，都只有一种填充的可能性与之对应，对于每一种填充的可能性也仅存在唯一的元素值与之对应，即可实现可重复元素全排列数的中介数法，现在考虑将m个元素填入到n个空位中(m<n)，则所有的方案总数为C(n,m)，可将该组合数进行分解为：C(n,m)=C(n-1,m)+C(n-1,m-1) ，其中左侧C(n-1,m)代表为第一个空位未被占用，然后将剩余的m个元素放入剩余的n-1个空位中的总的可能情况，右侧C(n-1,m-1)代表为第一个空位被占用，则将剩余的m-1个元素放入剩余的n-1个空位中。

4.2 十进制序号转为排列数

现考察将整数转换为对应的组合选择情况，考虑整数x，所对应的排列情况，其中0≤x<C(n,m) ，考虑将整数x分两种情况讨论:

(1)若x<C(n-1,m) 则表示第一种C(n-1,m)的情况，即第一个空位未被占用，然后将剩余的m个元素放入剩余的n-1个空位中的放置方法，则将第一个空位空开，该问题被归约为对于指定整数x，将m个小球放入n-1个空位中的子问题。

(2)若x≥C(n-1,m) 则表示第二种C(n-1,m-1)的情况，即第一个空位被占用，然后将剩余的m-1个元素放入剩余的n-1个空位中的放置方法，则将第一个空位填入元素，该问题被规约为对于指定整数x-C(n-1,m)，将m-1个小球放入n-1个空位中的子问题。

考察该问题的结束情况，当没有元素需要被放置时，即当存在规约子问题P(n,m)其中m为0时该问题结束，则将整数x转换为对应的将m个元素放置入n个空位中的对应选择情况的方法执行结束。

4.3 排列数转为十进制序号

现考察将组合的选择情况转换为对应的整数x，考察将m个元素放入n个空位中的一种填充情况。

(1)若第一个位置为空，则表示第一个位置没有被占用，然后剩余的m个元素填充在剩余的n-1个空位中，则对应的整数x和将m个元素填充在n-1个空位中的情况所对应的整数x相同，该问题被规约为m个元素填充在剩余n-1个空位中对应的整数x的子问题，即x(n,m)=x(n-1,m)。

(2)若第一个位置放置了元素，则表示第一个位置被占用，然后剩余的m-1个元素填充在剩余的n-1个空位中，则对应的整数x-C(n-1,m)和将m-1个元素填充在n-1个空位中的情况所对应的整数x相同，该问题被规约为m个元素填充在剩余n-1个空位中对应的整数x的子问题，即x(n,m)=C(n-1,m)+x(n-1,m-1)。

考察该问题的结束情况，当元素全部被放置时，则该问题结束，即当存在规约子问题P(n,m)，其中m=0时，该问题结束。

5 算法分析

5.1 伪代码描述

程序实现时数组下标从0开始，因此c_i表示数字i+1的重复次数。

Algorithm 1 IndexToPerm:根据十进制序号生成排列数

Input: 十进制的序号index,给定的要排序的元素的数量n，存储不重复元素的重复次数的数组c，如c[1]表示2的重复次数。

Output: 用vector存储的排列数

1: 将十进制index转为我们规定的组合数进位制数mc_number，并获得vector存储的每一位的值mc_vector；

2: 定义一个大小为N的vector< int > p_vector来存储新生成的排列数，ava_flag来存储当前可以分配的排列数的空闲位置，先将p_vector初始化为0，并将每位的下标存入ava_flag中;定义cur_n来表示当前还可以分配位置的个数，初始值为N;

3: 遍历c中元素，因为c[i]存储的是i+1这个元素的重复次数，因此记录该元素为num=i+1。根据我们提出的PerNumberToPerm算法对num的c[i]重元素递归分配排列数中的位置，更新p_vector。之后清空并重新初始化ava_flag当前可以分配的排列数的位置；

4: **return** 由p_vector生成的新的排列数。

Algorithm 2 PerNumberToPerm:将每个元素分配到排列数中

Input: 输入：p是当前已经生成的排列，ava_flag存储p中可选的下标,n是当前可选的下标个数，m是当前元素要选的位置个数，put_number是当前要放置的元素,mc 是当前位的中介数;

Output: 给put_number分配m个位置后的排列p;

1: **if** $m == 0$ **then**

2: **return** p;

3: **end if**

4: 计算tag=C(n-1,m),即对当前位中介数进行拆分;

5: **if** $mc < tag$ **then**

6: 更新ava_flag，清空该位的下标;

7: 递归调用PerNumberToPerm (p,ava_flag,n-1,m,put_number,mc)并赋给p;

8: **else**

9: mc=mc-tag;

10: 将排列p中该位的元素更新为put_number;

11: 更新ava_flag，清空该位的下标;

12: 递归调用PerNumberToPerm (p,ava_flag,n-1,m-1,put_number,mc) 并赋给p;

13: **end if**

14: 遍历c中元素，因为c[i]存储的是i+1这个元素的重复次数，因此记录该元素为num=i+1。

15: 根据我们提出的PerNumberToPerm算法对num的c[i]重元素递归分配排列数中的位置，更新p_vector。

16: 之后清空并重新初始化ava_flag当前可以分配的排列数的位置；

17: **return** 给put_number分配m个位置后的排列p。

Algorithm 3 PermToIndex:根据排列数生成十进制序号

Input: 给定的排列数p,用vector存储每一位的值。
Output: long类型的十进制序号index
1: 获得排列数$vector < int > p_vector$和包含重复元素的排列长度N；
2: 定义一个大小为N的$vector < int > ava_flag$来存储当前可以分配的排列数的空闲位置，遍历p_vector初始化ava_flag并获得输入排列数中的最大值m，先将p_vector初始化为0，并将每位的下标存入ava_flag中；
3: 定义一个大小为m的$vector < int > c_vector$来存储重复元素的个数，$c\_vector[p\_vector[i]-1] = c\_vector[p\_vector[i]-1]+1, i=0,1,...,N$，通过遍历$p_vector$初始化。
4: 定义cur_n来表示当前还可以分配位置的个数，初始值为N;
5: 定义一个大小为m的$vector < int > mc_vector$来存储我们定义的组合数进位制中介数，初始值全为0;
6: 遍历c_vector中元素，因为$c_vector[i]$存储的是i+1这个元素的重复次数，因此记录该元素为num=i+1。根据我们提出的PermPerNumToIndex算法对该元素求该位上的中介数$mc_vector[i]$，并更新$cur_n = cur_n - c_vector[i]$。之后清空并重新初始化$ava_flag$当前可以分配的排列数的位置；
7: **return** 将我们定义的组合数进位制中介数mc_vector转化的为十进制序号index。

Algorithm 4 PermPerNumToIndex:求当前数字位在中介数中的值

Input: p是生成的排列，ava_flag是p中可选的下标,n是当前可选的下标个数，m是当前元素要选的位置个数，put_number是当前元素;
Output: 排列的中介数中put_number所在位置对应的值idx;
1: **if** $m == 0$ **then**
2: **return** 0;
3: **end if**
4: idx初始化为0;
5: tag=p[$ava\_flag[ava\_flag.size()-1]$]; 当前空位的值；
6: **if** $tag == put\_number$ **then**
7: 更新ava_flag，清空该位的下标;
8: idx=$get\_cni(n-1,m) + PermPerNumToIndex(p, ava\_flag, n-1, m-1, put\_number)$，$get_cni(n-1,m)$是获取排列数C(n-1,m);
9: **else**
10: 更新ava_flag，清空该位的下标;
11: idx=$PermPerNumToIndex(p, ava\_flag, n-1, m, put\_number)$;
12: **end if**
13: **return** idx

5.2 时间复杂度分析

对于基于中介数的n个不重复元素的排列生成算法，如之前提到的字典序法、递增进位制数法、递减进位制数法和邻位对换法的时间复杂度都是线性的。包括十进制序号跟递增或递减进制数的中介数之间的转换O(n)和根据中介数放置n个元素的时间O(n)，因此总的时间复杂度是O(n)。

对于我们提出的基于中介数的包含重复元素的全排列生成算法，假设有n个包含重复数字的数，一共有m个不重复元素(n≥m)，使用我们提出的基于中介数的方法求解全排列时，算法包括两个步骤：一是对于十进制序号和每一位的进制为C(n,c_1),C(n-c_1,c_2),C(n-c_1-c_2,c_3),⋯,C(c_m,c_m)的组合数进制数的中介数之间的进制转换，二是通过中介数根据算法求得排列。根据序号转化为中介数的部分的时间复杂度为O(m)，递归调用根据序号生成排列的算法PerNumberToPerm 和根据排列生成序号的PermPerNumToIndex 算法的时间复杂度都为O(n)，因此我们提出的有重复元素的全排列生成算法的时间复杂度为O(m+n)=O(2n)=O(n)

6 实验

在实验中，我们使用CodeBlocks c++11中vector实现了目前已有四种无元素重复的全排列数的中介数方法，分别实现了其中的字典序法，递增进位制数法，递减进位制数法以及邻位对换法，每种方法都分别实现了将给定10进制序号转换为与之对应的全排列数，已经将全排列数重新转换为10进制序号，分别对3 个全排列数，4个全排列数，6个全排列数，9 个全排列数进行了输出、验证以及运行时间统计，验证了算法的正确性并考察了四种算法的时间效率，同时也跟我们提出的基于中介数的包含重复元素的排列方法生成不重复全

排列相比较，五种算法所消耗的时间如下图表所示。

Table 1: 非重复元素算法的运行时间比较

	n			
	3	4	6	9
字典序法	0ms	31ms	859ms	240062ms
递增进位制数法	0ms	16ms	844ms	232264ms
递减进位制数法	0ms	15ms	812ms	276886ms
邻位对换法	15ms	16ms	953ms	256661ms
我们的方法	0ms	31ms	687ms	271128ms

可以看出在排列数为3时，几种方法的运行时间相差不大；n=4时，字典序法的速度开始明显慢于其他几种方法；n=6时，递减进位制法最快，邻位对换法的速度最慢，我们的方法这时明显达到了最快；n=9时递增进位制法的速度却明显更快，字典序次之，然后才是邻位对换法，我们的方法稍快于递减进位制法。可以看出在排列数较少时邻位对换法的速度可能没有字典序法等方法快，而在排列数较大时递增进位制和字典序法的速度的优势有所显现。可以看出我们的算法在无重复元素的全排列上依然有效，并且速度与其他全排列生成算法相差不大，因此我们的算法可以用在无重复元素的全排列生成上。

Figure 2: 测试四种方法和我们的方法(最后一张)生成1,2,3,4数字组成的所有全排列

另外我们发现从生成的排列数规律上，邻位对换法具有明显的优势，相邻两个排列数仅有相邻的一对数字

发生了对换，相对于其他方法生成的排列数具有明显的规律性。

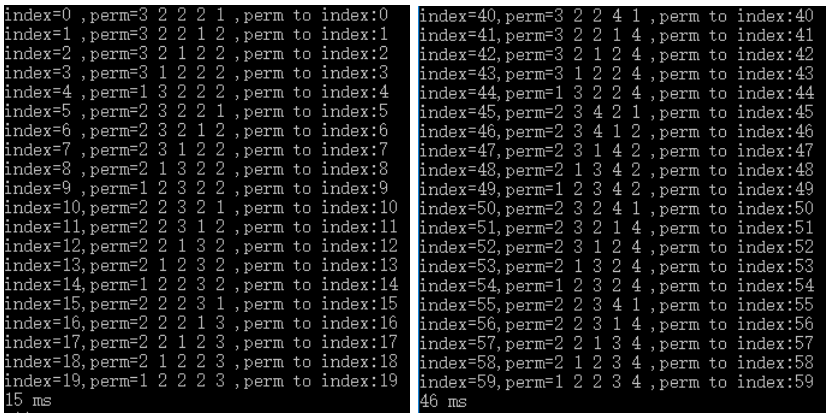

Figure 3: 我们的算法生成不带重复元素的排列和带重复元素的排列的例子

在已有全排列方法的基础上，我们还提出新的元素可重复的全排列数的中介数方法，并进行了代码实现，我们可以看到该方法可以针对元素可重复的全排列数实现中介数和排列数之间的转换，我们可以看到，当设定每种元素的数量为1时，该方法即为一个全排列方法，即为不重复元素的全排列方法的泛化方法。并且由上图可以看出当重复元素多时，算法的时间效率要比重复元素少时的高。因此我们的算法在生成可重复元素的全排列时会比生成不可重复元素的全排列有时更好的效果。

7 结论

我们提出了一种元素可重复的全排列数的中介数法。该方法建立了包含重复元素的排列数和特定进制的中介数之间的一一映射关系，同时使得任意的带有重复元素排列数都存在对应的中介数表示，从而实现了通过任意数值索引在固定时间内获得该索引所对应的带有重复元素的排列数的方法。我们还对该方法进行了代码实现，验证了该方法的可行性。