组合数学 Project1 Paper

Members: 李其乐(2015210918)

黎健成(2015210936)

谭川奇(2015310609)

我们小组成员共实现和测试了几种全排列生成算法，包括：深度搜索法、中介数法、多线程版本的中介数法、字典序法、交换法、递增进位制数法、递减进位数法，还考察了 C++ 标准库中的<algorithm>中的全排列实现 next_permutation()，考察研究了它的实现机制。排列组合问题都是枚举类型的问题，这类问题的关键是采取怎样的枚机策略保证：完整性、准确性、效率。下面分别对我们考察的几个算法作一个简单的讨论。

深度搜索法：

也可以视为回溯法，回溯法的是来解决多维向量的枚举问题的常见方法。具体步骤为：

- 递归的枚举每个维度的值
- 既然是递归，那就用终止条件。一般的终止条件是找到了满足条件的解，或者已经超出了解的范围，无需再递归下去。
- 找到某一维的候选值。解排列组合的重中之重。常用的技巧是使用标志位来记录某一维是否可用。
- f[i]=true 和 f[i]=false]，保证了回溯法的正确！

中介数法（以及多线程版本）：

将每个组合与一个数对应起来，中介数和排列是一一对应的，不存在一个中介数对应多个排列的情况。所以，可以通过枚举这些中介数，在枚举这些中介数的同时，枚举每个排列。

这种方式表面上看起来比较简单，但实际实现起来是比较复杂的（相对于后面的递增递减进制位法）。

多线程版本对其实现采用了多线程优化来加快速度。

字典序法：

对给定的字符集中的字符规定一个先后关系，在此基础上按照顺序依次产生每个排列。例如字符集{1,2,3}, 按字典序生成的全排列是:123,132,213,231,312,321 例如：839647521 的下一个排列

Step1 :从最右开始, 找到第一个比右边小的数字 4（因为 4<7，而 7>5>2>1），再从最右开始，找到 4 右边比 4 大的数字 5（因为 4>2>1 而 4<5）。4、5 也即从右往左找到的第一个对严格的升序对。

Step2：交换 4、5。

Step3：倒置 5 右边的 7421 为 1247.即得到下一个排列 839651247.

交换（Swap）法：

核心思想是交换，具体来说，对于一个长度为 n 的串，要得到其所有排列，可以这样做：

- 把当前位上的元素依次与其后的所有元素进行交换

- 对下一位做相同处理，直到当前位是最后一位为止，输出序列

递增进位制数法：

为了改变由中介数求原排列的复杂性，修改定义中介数的方法，定义一种新的中介数，称为递增进位制数，它的下标对应了数字大小。它是从中介数演变过来而已。实际上，经过这样的一个演变，每一位上的数都不会超过位的大小。

相比于原来的中介数要简便得多，不需要进行额外的比较，而且得到中介数的方法也基本类似，虽然它的本质是一个递增进制数可能有些困扰，如果不去管它的本质，则是以下简单的事情：

- 用原排列求新的中介序。
- 用中介序方便得求得原排列。
- 用中介序求得新的序号。

要注意的是，此时的序号并不是按照字典序排序的，而是一个新的排序方式。但是范围是一样的，而且始末点是一样的。

递减进位制数法：

主要思想和步骤同递增进位制数法一样，仅仅区别为一个是递增进位制，一个是递减进位制。

C++ <algorithm> 中的 next_permutation：

经过考察和研究，得知 C++ algorithm 中的全排列实现原理为：在当前序列中，从尾端往前寻找两个相邻元素，前一个记为*i，后一个记为*ii，并且满足*i <

*ii。然后再从尾端寻找另一个元素*j，如果满足*i < *j，即将第 i 个元素与第 j 个元素对调，并将第 ii 个元素之后（包括 ii）的所有元素颠倒排序，即求出下一个序列了。

甚至 Leetcode 上也有很多对全排列算法的讨论，参见以下几个问题：

- Permutations
- Permutations II
- Next Permutation
- Permutation Sequence