一些生成排列与组合的讨论

（大量参考 Richard A. Brualdi 的 *Introductory Combinatorics* 课本）

为了行文方便以中英夹杂的方式写作，不适之处请谅解…

一. 基于递归的生成全排列

Johnson-Trotter Algorithm：

认识到，集合{1, 2, 3,…, n}的排列中，删去 n，我们将得到{1, 2, 3,…, n-1}的全排列（当然包括那些重复的排列）。也即，通过在前 n-1 个数的全排列中有顺序地插入 n，可以得到 n 的全排列。下面给出这种算法的描述：

n = 2:

$$\begin{array}{ccc} & 1 & \mathbf{2} \\ \mathbf{2} & 1 & \end{array}$$

n = 3:

$$\begin{array}{ccccc} & 1 & & 2 & \mathbf{3} \\ & 1 & \mathbf{3} & 2 & \\ \mathbf{3} & 1 & & 2 & \\ \mathbf{3} & 2 & & 1 & \\ & 2 & \mathbf{3} & 1 & \\ & 2 & & 1 & \mathbf{3} \end{array}$$

通过简单的归纳可以得到，每一次生成的排列都将从 1234…n 到 2134…n，由于交换 21 可以得到第一个排列，这个过程实际上是循环的。

从图论的角度来说，即，由{1, 2, 3, …, n}的全部排列作为顶点，存在边当且仅当两个排列可以仅交换两个相邻的数字得到的图，存在 Hamilton cycle。

更一般地，如果不使用 recursion，即，仅使用当前的一个排列来生成这个循环，我们定义一个整数序列，其中每一个整数上方用→和←来标示方向。一个整数为*可移动的*当且仅当它上方的箭头指向一个相邻的且比它小的整数。对于最大的整数 n，它一直可移动除非，它位于序列的两端且箭头没有指向任何数字。

算法：

　　从全部为左箭头的 1234…n 开始

　　当存在一个可移动整数时：

　　　　求出最大的可移动整数 m,

　　　　交换 m 和它箭头所指的相邻的整数

　　　　交换所有满足 p>m 的整数 p 上箭头的方向

　　例如，n = 4 时：

$$\begin{array}{cccc@{\qquad}cccc}
\overleftarrow{1} & \overleftarrow{2} & \overleftarrow{3} & \overleftarrow{4} & \overrightarrow{4} & \overrightarrow{3} & \overleftarrow{2} & \overleftarrow{1} \\
\overleftarrow{1} & \overleftarrow{2} & \overleftarrow{4} & \overleftarrow{3} & \overrightarrow{3} & \overrightarrow{4} & \overleftarrow{2} & \overleftarrow{1} \\
\overleftarrow{1} & \overleftarrow{4} & \overleftarrow{2} & \overleftarrow{3} & \overrightarrow{3} & \overleftarrow{2} & \overrightarrow{4} & \overleftarrow{1} \\
\overleftarrow{4} & \overleftarrow{1} & \overleftarrow{2} & \overleftarrow{3} & \overrightarrow{3} & \overleftarrow{2} & \overleftarrow{1} & \overrightarrow{4} \\
\overrightarrow{4} & \overleftarrow{1} & \overleftarrow{3} & \overleftarrow{2} & \overleftarrow{2} & \overrightarrow{3} & \overleftarrow{1} & \overleftarrow{4} \\
\overleftarrow{1} & \overrightarrow{4} & \overleftarrow{3} & \overleftarrow{2} & \overleftarrow{2} & \overrightarrow{3} & \overleftarrow{4} & \overleftarrow{1} \\
\overleftarrow{1} & \overleftarrow{3} & \overrightarrow{4} & \overleftarrow{2} & \overleftarrow{2} & \overleftarrow{4} & \overrightarrow{3} & \overleftarrow{1} \\
\overleftarrow{1} & \overleftarrow{3} & \overleftarrow{2} & \overrightarrow{4} & \overleftarrow{4} & \overleftarrow{2} & \overrightarrow{3} & \overleftarrow{1} \\
\overleftarrow{3} & \overleftarrow{1} & \overleftarrow{2} & \overleftarrow{4} & \overrightarrow{4} & \overleftarrow{2} & \overleftarrow{1} & \overrightarrow{3} \\
\overleftarrow{3} & \overleftarrow{1} & \overleftarrow{4} & \overleftarrow{2} & \overleftarrow{2} & \overrightarrow{4} & \overleftarrow{1} & \overrightarrow{3} \\
\overleftarrow{3} & \overleftarrow{4} & \overleftarrow{1} & \overleftarrow{2} & \overleftarrow{2} & \overleftarrow{1} & \overrightarrow{4} & \overrightarrow{3} \\
\overleftarrow{4} & \overleftarrow{3} & \overleftarrow{1} & \overleftarrow{2} & \overleftarrow{2} & \overleftarrow{1} & \overrightarrow{3} & \overrightarrow{4}
\end{array}$$

一个额外的例子

排列中的逆序对应着不以自然数顺序出现的一对数, 例如, 排列 31524 有 4 个逆序, (3, 1), (3, 2), (5, 2), (5, 4)。在排列中, 一个数的逆序数量为出现在它前面又比它大的数, 例如, 排列 31524 的逆序数列为, 1, 2, 0, 1, 0.

对于每一个自然数 k, 可能的逆序为 0, 1, …, n-k, 因此, {1, 2, 3, …, n}可能的逆序数列有 n!个, 这与它的排列数相同, 暗示着, 对于任何一个排列, 它的逆序数列是一一对应的。通过算法来证明这个猜想:

　　将 n 个空位置从左向右标为 1, 2, …, n

　　从 1 开始插入空位置, 若 1 的逆序为b_1, 则把 1 放在$b_1 + 1$的位置上。

　　(一般地) 若 k 的逆序为b_k, 则把 k 放在$b_k + 1$的位置上。

　　把 n 放在最后一个空位置上。

由此可以由逆序列唯一地确定一个排列。

另外, 逆序可以用来给出一个矩阵行列式的公式:

$$\det(\mathrm{A}) = \sum \varepsilon(i_1\, i_2 \dots i_n) a_{1i_1} a_{ni_2} \dots a_{ni_n}$$

(据说内地一般使用这个来定义行列式, 反正我已经看不懂了【笑着哭】)

逆序列的和表示了这个排列的无序程度。若和为 k, 意味着该排列可以由 123…n 经过 k 次连续交换相邻两个数得到。实际上, 这就是 bubble sort。

二. 生成子集

二进制数与子集的对应关系非常明显 (回忆对组合数求和的经典证明), 通过生成 0 到 2^n 的全部二进制数, 可以方便地生成全部子集。

更一般地, 当我们需要生成的是两个或两个以上的子集时, 可以用相应进制的数来代表。(例如, 考虑当有多于一个背包时 Knapsack Problem 的暴力解法)

三. 生成 Gray Code

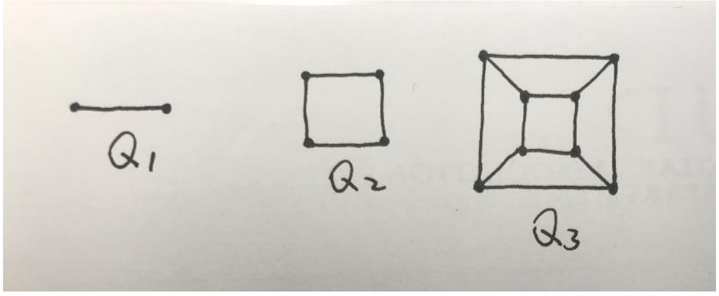
在说明这个算法之前, 先来考虑一个十分有趣的问题:

假设现在有一把十个数位的二进制锁 (或者说, 这把锁上是 10 个按钮, 只有当正确组合的按钮同时按下时, 锁才会打开)。

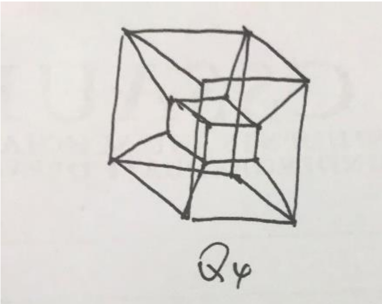
在不知道正确组合的情况下, 枚举是对于非洲人唯一可行的方法, 当然, 即使是枚举也需要技巧, 例如, 如果按照字典序来枚举, 0000000001 和 0000000010 需要改变两次按钮, 更不用说当需要从 0111111111 变成 1000000000 时了 (这样的情况将会出现很多次)。所以, 我们希望每次尝试只改变一个按钮, 并且遍历所有可能的结果。

这种顺序是可能的吗? 用图论来解释:

定义Q_n是 n 维空间里的立方体:



那么，Q_4是什么呢？可以看出，从Q_1到Q_2，我们复制了两次Q_1，并将对应的顶点连接。从Q_2到Q_3也是如此。归纳得到（是的，一个只有两步且没有证明的归纳），从Q_3到Q_4，也只需复制Q_3，并将对应的顶点连接。

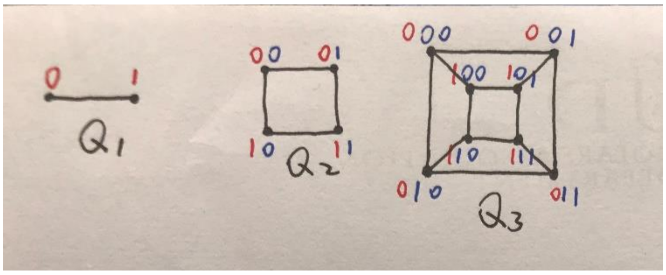


（这大概也是为什么许多解释四维空间的图都会这么画）

现在，将顶点进行编码，编码规则如下：

　　Q_1的两个顶点分别为 0 和 1。

　　其后的每一个图，都继承它复制而来的那个图的编码，并在两个复制部分的前面加上编号 0 和 1：



观察到，每两个顶点相邻，当且仅当它们的编码相差一个数字。

所以，问题变成了，Q_{10}有 Hamilton path 吗？

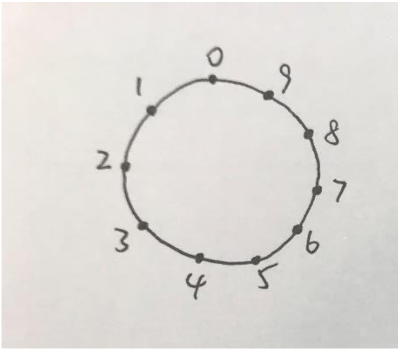
数学归纳法告诉我们，有。事实上，更严格地，当 n > 2 时，Q_n is Hamiltonian.

一个额外的例子

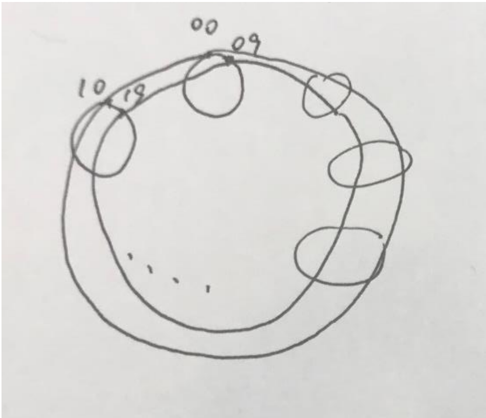
如果现在，我们的锁不再是二进制的，而是最常用的四位密码锁，还可以每次只改变一个数位（注意到 0 与 9 相邻，如，3610，3609，3601 都可以由 3600 只改变一次得到）来遍历全部可能吗？

下面给出我自己的一个解释：

考虑一个数位时，代表的图是一个C_{10}



考虑两个数位时，将C_{10}复制 10 次，对应的数字之间形成了 10 个C_{10}（确实是一个十分有趣的结构）



考虑 n 个数位时，将代表 n-1 的结构复制 10 次，形成更大的循环。
这个大的结构是 Hamiltonian 的吗？由数学归纳法可以得到，是的。

回到正题，下面描述一种无需利用递归生成 Gray Code 的算法：
令 n 元组$a_{n-1}a_{n-2}\dots a_0$=00…0。
当$a_{n-1}a_{n-2}\dots a_0 \neq$10…0 时：
　　计算$\sum a_k$
　　若和是偶数，改变a_0（从 1 变到 0 或从 0 变到 1）
　　否则，确定从右边开始的第一个为 1 的数a_j，然后，改变a_{j+1}.
对这个算法正确性的证明并不是本文的重点，故略过（可以对 n 实施归纳法，分别讨论开头为 0 和开头为 1 两种情况来证明）。

四. 生成 r 子集
前面讨论了如何生成一个集合的全部子集，如果我们希望生成有确定个数元素的子集，即，希望生成C_n^r组合，当然可以从所有集合中取出长度为 r 的那些，然而这并不现实。
为了讨论方便，以下所有集合以递增的顺序写出。
现在，考虑以字典序（也就是大家所熟知的 string 大小比较）生成组合。为了确定这样的顺序，我们需要知道在字典序中，一个组合的直接后继是什么。
例，12389 是以 123 开头的最后一个组合，所以它的直接后继是 12456。
推广这个例子，可以确定任何一个组合（除了最后一个）的直接后继：
设$a_1a_2\dots a_r$是{1，2，…，n}的 r 子集。在字典序中，第一个 r 子集是 12…r，最后一个 r 子集是(n-r+1)(n-r+2)…n。假设$a_1a_2\dots a_r \neq$(n-r+1)(n-r+2)…n。寻找位置 k，使 k 是满足

a_k<n 且使得$a_k + 1$不等于$a_{k+1}, \ldots, a_r$中任何一个数的最大整数。那么，$a_1 a_2 \ldots a_r$的直接后继是：

$$a_1 \ldots a_{k-1}(a_k + 1)(a_k + 2) \ldots (a_k + r - k + 1)$$

根据这个结论，我们从$a_1 a_2 \ldots a_r = 12 \ldots r$开始，每次用它的直接后继替换这个组合，即可生成全部的 r 子集。

将生成 r 子集的方法与生成集合排列的方法组合起来，我们就可以得到生成含 n 个元素的集合的 r 排列的方法。

为什么要写这个东西

今年的 MATH3600 由于时间关系跳过了 Chapter 4: Generating Permutations and Combinations，以及 Chapter 10: Combinatorial Design. 这两章确实属于主干知识的分支，但正如 LKH 所说，如果觉得 Combinatorics 有趣的话，这两章的内容也会很有趣。这一个星期压力太大，看课本时摸鱼打了鸡血，也就产生了这篇文档。如有疏漏，不吝赐教。两个读 DA 的编程菜鸡（不，PC 老师不是菜鸡，主要是我）可能后续会给出代码实现（不存在的），诚挚欢迎 CS 大佬们 show the code【给 dalao 递茶】
总而言之，谢谢阅读！

Neithen 2017/11/18