

经典计算机科学著作最新版

计算机程序设计艺术

第4卷 第2册 （双语版）

生成所有元组和排列

The Art of Computer Programming, Volume 4 Generating All Tuples and Permutations

Fascicle 2

苏运霖 译

（美）Donald E. Knuth 著

机械工业出版社
China Machine Press

关于算法分析的这多卷论著已经长期被公认为经典计算机科学的定义性描述。作为关于组合查找的冗长一章的一部分，这个分册开始关于如何生成所有可能性的讨论。具体地说，它讨论所有n元组的生成，然后把这些思想扩充到所有排列上。这样一些算法提供了一个自然的导引，借助于此，关于组合数学的许多关键思想都可加以介绍和剖析。

Simplified Chinese edition copyright © 2006 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *The Art of Computer Programming, Volume 4, Fascicle 2, Generating All Tuples and Permutations* (0-201-85393-0), by Donald E.Knuth , Copyright © 2005.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as ADDISON WESLEY.

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2005-3614

图书在版编目（CIP）数据

计算机程序设计艺术 第4卷 第2册　生成所有元组和排列（双语版）/（美）克努特（Knuth, D. E.）著. 苏运霖译. －北京：机械工业出版社，2006.7

书名原文：The Art of Computer Programming, Volume 4, Fascicle 2, Generating All Tuples and Permutations

ISBN 7-111-17773-8

Ⅰ.计…　Ⅱ.①克…　②苏…　Ⅲ.程序设计　Ⅳ.TP311.1

中国版本图书馆CIP数据核字（2005）第125978号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：朱起飞

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2006年7月第1版第1次印刷

170mm × 242mm · 17.25印张

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

PREFACE

I am grateful to all my friends,
and record here and now my most especial appreciation
to those friends who, after a decent interval,
stopped asking me, "How's the book coming?"

— PETER J. GOMES, *The Good Book* (1996)

THIS BOOKLET is Fascicle 2 of *The Art of Computer Programming*, Volume 4: *Combinatorial Algorithms*. As explained in the preface to Fascicle 1 of Volume 1, I'm circulating the material in this preliminary form because I know that the task of completing Volume 4 will take many years; I can't wait for people to begin reading what I've written so far and to provide valuable feedback.

There will also be a Fascicle 1 for Volume 4. But I've written Fascicle 2 first. Experienced programmers will understand that the initialization of a program usually can't be written properly until after the main body has been fleshed out.

To put the material in context, this fascicle contains Sections 7.2.1.1 and 7.2.1.2 of a long, long chapter on combinatorial searching. Chapter 7 will eventually fill three volumes (namely Volumes 4A, 4B, and 4C), assuming that I'm able to remain healthy. It will begin with a short review of graph theory, with emphasis on some highlights of significant graphs in The Stanford GraphBase, from which I will be drawing many examples. Then comes Section 7.1, which deals with bitwise manipulation and with algorithms relating to Boolean functions. Section 7.2 is about generating all possibilities, and it begins with Section 7.2.1: Generating Basic Combinatorial Patterns. That sets the stage for the main contents of the present booklet, namely Section 7.2.1.1 (where I get the ball rolling by dealing with the generation of n-tuples) and Section 7.2.1.2 (which extends the ideas to permutations). Then will come Section 7.2.1.3 (about combinations), Section 7.2.1.4 (about integer partitions), and Section 7.2.1.5 (about set partitions), all in Fascicle 3. Section 7.2.1.6 (about trees) and Section 7.2.1.7 (about the history of combinatorial generation) will comprise Fascicle 4. Section 7.2.2 will deal with backtracking in general. And so it will go on, if all goes well; an outline of the entire Chapter 7 as currently envisaged appears on the `taocp` webpage that is cited on page ii.

I had great pleasure writing this material, akin to the thrill of excitement that I felt when writing Volume 2 many years ago. As in Volume 2, where I found to my delight that the basic principles of elementary probability theory and number theory arose naturally in the study of algorithms for random number generation and arithmetic, I learned while preparing Section 7.2.1 that the basic

principles of elementary combinatorics arise naturally and in a highly motivated way when we study algorithms for combinatorial generation. Thus, I found once again that a beautiful story was "out there" waiting to be told.

My original intention was to devote far less space to this topic. But when I saw how fundamental the ideas were for combinatorial studies in general, I knew that I could never be happy unless I covered the basics quite thoroughly. Therefore I've done my best to build a solid foundation of theoretical and practical ideas that will support many kinds of reliable superstructures.

The topic of combinatorial generation has not only given me a chance to discuss important mathematical theories, it also has led to other kinds of fun, because of its many connections to amusing games and puzzles. Good puzzles are great aids to education, and I intend to continue focusing on recreational topics throughout Volume 4.

> *The average boy who abhors square root or algebra finds delight in working puzzles which involve similar principles, and may be led into a course of study which would develop the mathematical and inventive bumps in a way to astonish the family phrenologist.*
> — SAM LOYD, *The World of Puzzledom* (1896)

I shall happily pay a finder's fee of \$2.56 for each error in this fascicle when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually reward you with immortal glory instead of mere money, by publishing your name in the eventual book:–)

I wish to thank Yoichi Hariguchi for helping me to build and rebuild the computer on which this booklet was written. And I also want to thank Frank Ruskey for bravely foisting an early draft of this material on college students and for telling me about his classroom experiences.

Notations that are used here and not otherwise explained can be found in the Index to Notations at the end of Volumes 1, 2, or 3. Those indexes point to the places where further information is available. Of course Volume 4 will some day contain its own Index to Notations.

Machine-language examples in all future editions of *The Art of Computer Programming* will be based on the `MMIX` computer, which is defined in Section 1.3.1′ of Volume 1, Fascicle 1. Cross-references to Sections 1.3.1′, 1.3.2′, 1.4.1′, 1.4.2′, and 1.4.3′ in the present booklet refer to that fascicle.

Cross references to yet-unwritten material sometimes appear as '00' in the following pages; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

Stanford, California
December 2004

D. E. K.

CONTENTS

Preface ····· *III*

Chapter 7 Combinatorial Searching

7.2 **Generating All Possibilities** ····· *2*

7.2.1 Generating Basic Combinatorial Patterns ····· *2*

7.2.1.1 Generating all *n*-tuples ····· *28*

7.2.1.2 Generating all permutations ····· *51*

Answers to Exercises ····· *76*

目　录

译者序 ····· *125*

前言 ····· *129*

第7章 组合查找

7.2 **生成所有可能性** ····· *132*

7.2.1 生成基本的组合模式 ····· *132*

7.2.1.1 生成所有*n*元组 ····· *132*

7.2.1.2 生成所有排列 ····· *170*

习题答案 ····· *205*

索引和词汇表 ····· *256*

我认为，写一些关于艺术或科学的规则及其他可能不会湮没和不为人所知的东西，值得花费数日劳动。

——理查德·达克沃思（Richard Duckworth），*Tintinnalogia*（钟声叮当）（1668）

7.2. GENERATING ALL POSSIBILITIES

All present or accounted for, sir.
— Traditional American military saying

All present and correct, sir.
— Traditional British military saying

7.2.1. Generating Basic Combinatorial Patterns

OUR GOAL in this section is to study methods for running through all of the possibilities in some combinatorial universe, because we often face problems in which an exhaustive examination of all cases is necessary or desirable. For example, we might want to look at all permutations of a given set.

Some authors call this the task of *enumerating* all of the possibilities; but that's not quite the right word, because "enumeration" most often means that we merely want to *count* the total number of cases, not that we actually want to look at them all. If somebody asks you to enumerate the permutations of $\{1,2,3\}$, you are quite justified in replying that the answer is $3! = 6$; you needn't give the more complete answer $\{123, 132, 213, 231, 312, 321\}$.

Other authors speak of *listing* all the possibilities; but that's not such a great word either. No sensible person would want to make a list of the $10! = 3{,}628{,}800$ permutations of $\{0,1,2,3,4,5,6,7,8,9\}$ by printing them out on thousands of sheets of paper, nor even by writing them all in a computer file. All we really want is to have them present momentarily in some data structure, so that a program can examine each permutation one at a time.

So we will speak of *generating* all of the combinatorial objects that we need, and *visiting* each object in turn. Just as we studied algorithms for tree traversal in Section 2.3.1, where the goal was to visit every node of a tree, we turn now to algorithms that systematically traverse a combinatorial space of possibilities.

He's got 'em on the list—
he's got 'em on the list;
And they'll none of 'em be missed—
they'll none of 'em be missed.
— WILLIAM S. GILBERT, *The Mikado* (1885)

7.2.1.1. Generating all n-tuples. Let's start small, by considering how to run through all 2^n strings that consist of n binary digits. Equivalently, we want to visit all n-tuples $(a_1, \ldots, a_n)$ where each a_j is either 0 or 1. This task is also, in essence, equivalent to examining all subsets of a given set $\{x_1, \ldots, x_n\}$, because we can say that x_j is in the subset if and only if $a_j = 1$.

Of course such a problem has an absurdly simple solution. All we need to do is start with the binary number $(0\ldots00)_2 = 0$ and repeatedly add 1 until we reach $(1\ldots11)_2 = 2^n - 1$. We will see, however, that even this utterly trivial problem has astonishing points of interest when we look into it more deeply. And our study of n-tuples will pay off later when we turn to the generation of more difficult kinds of patterns.

In the first place, we can see that the binary-notation trick extends to other kinds of n-tuples. If we want, for example, to generate all $(a_1, \ldots, a_n)$ in which each a_j is one of the decimal digits $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, we can simply count from $(0 \ldots 00)_{10} = 0$ to $(9 \ldots 99)_{10} = 10^n - 1$ in the decimal number system. And if we want more generally to run through all cases in which

$$0 \le a_j < m_j \qquad \text{for } 1 \le j \le n, \tag{1}$$

where the upper limits m_j might be different in different components of the vector $(a_1, \ldots, a_n)$, the task is essentially the same as repeatedly adding unity to the number

$$\begin{bmatrix} a_1, & a_2, & \ldots, & a_n \\ m_1, & m_2, & \ldots, & m_n \end{bmatrix} \tag{2}$$

in a mixed-radix number system; see Eq. 4.1–(9) and exercise 4.3.1–9.

We might as well pause to describe the process more formally:

Algorithm M (*Mixed-radix generation*). This algorithm visits all n-tuples that satisfy (1), by repeatedly adding 1 to the mixed-radix number in (2) until overflow occurs. Auxiliary variables a_0 and m_0 are introduced for convenience.

M1. [Initialize.] Set $a_j \leftarrow 0$ for $0 \le j \le n$, and set $m_0 \leftarrow 2$.

M2. [Visit.] Visit the n-tuple $(a_1, \ldots, a_n)$. (The program that wants to examine all n-tuples now does its thing.)

M3. [Prepare to add one.] Set $j \leftarrow n$.

M4. [Carry if necessary.] If $a_j = m_j - 1$, set $a_j \leftarrow 0$, $j \leftarrow j - 1$, and repeat this step.

M5. [Increase, unless done.] If $j = 0$, terminate the algorithm. Otherwise set $a_j \leftarrow a_j + 1$ and go back to step M2. ∎

Algorithm M is simple and straightforward, but we shouldn't forget that nested loops are even simpler, when n is a fairly small constant. When $n = 4$, we could for example write out the following instructions:

```
For a1 = 0, 1, ..., m1 − 1 (in this order) do the following:
 For a2 = 0, 1, ..., m2 − 1 (in this order) do the following:
  For a3 = 0, 1, ..., m3 − 1 (in this order) do the following:
   For a4 = 0, 1, ..., m4 − 1 (in this order) do the following:
    Visit (a1, a2, a3, a4).
```
(3)

These instructions are equivalent to Algorithm M, and they are easily expressed in any programming language.

Gray binary code. Algorithm M runs through all $(a_1, \ldots, a_n)$ in lexicographic order, as in a dictionary. But there are many situations in which we prefer to visit those n-tuples in some other order. The most famous alternative arrangement is the so-called Gray binary code, which lists all 2^n strings of n bits in such a way

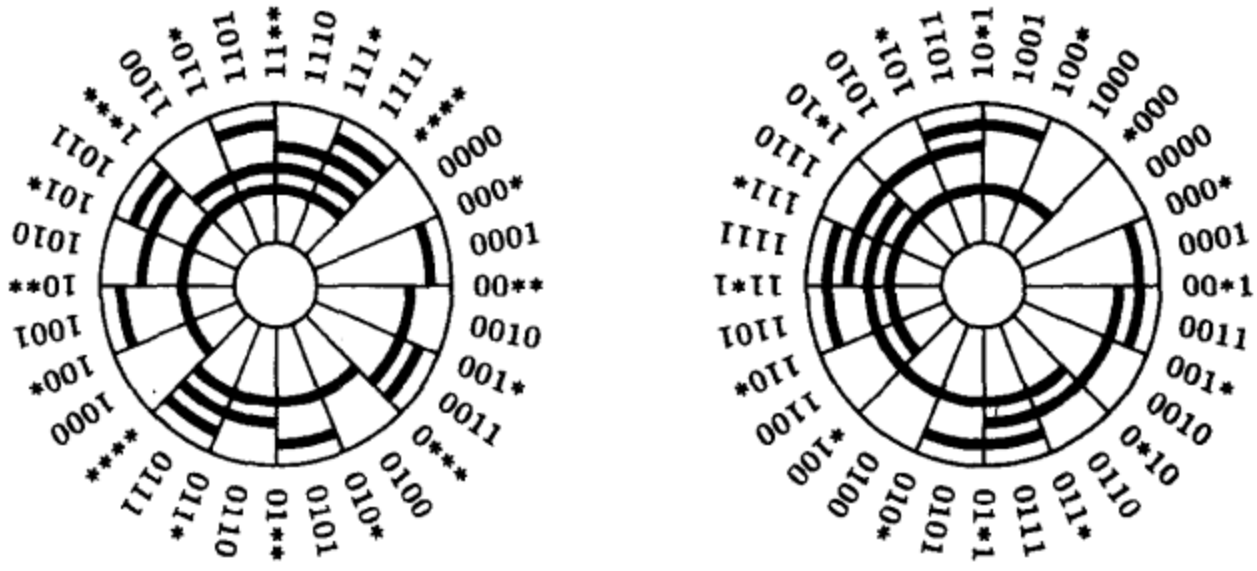


Fig. 10. (a) Lexicographic binary code. (b) Gray binary code.

that only one bit changes each time, in a simple and regular way. For example, the Gray binary code for $n = 4$ is

$$0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,$$
$$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000. \tag{4}$$

Such codes are especially important in applications where analog information is being converted to digital or vice versa. For example, suppose we want to identify our current position on a rotating disk that has been divided into 16 sectors, using four sensors that each distinguish black from white. If we use lexicographic order to mark the tracks from 0000 to 1111, as in Fig. 10(a), wildly inaccurate measurements can occur at the boundaries between sectors; but the code in Fig. 10(b) never gives a bad reading.

Gray binary code can be defined in many equivalent ways. For example, if Γ_n stands for the Gray binary sequence of n-bit strings, we can define Γ_n recursively by the two rules

$$\Gamma_0 = \epsilon;$$
$$\Gamma_{n+1} = 0\Gamma_n,\ 1\Gamma_n^R. \tag{5}$$

Here ϵ denotes the empty string, $0\Gamma_n$ denotes the sequence Γ_n with 0 prefixed to each string, and $1\Gamma_n^R$ denotes the sequence Γ_n in *reverse order* with 1 prefixed to each string. Since the last string of Γ_n equals the first string of Γ_n^R, it is clear from (5) that exactly one bit changes in every step of Γ_{n+1} if Γ_n enjoys the same property.

Another way to define the sequence $\Gamma_n = g(0), g(1), \ldots, g(2^n-1)$ is to give an explicit formula for its individual elements $g(k)$. Indeed, since Γ_{n+1} begins with $0\Gamma_n$, the infinite sequence

$$\begin{aligned}\Gamma_\infty &= g(0), g(1), g(2), g(3), g(4), \ldots \\ &= (0)_2, (1)_2, (11)_2, (10)_2, (110)_2, \ldots\end{aligned} \tag{6}$$

is a permutation of all the nonnegative integers, if we regard each string of 0s and 1s as a binary integer with optional leading 0s. Then Γ_n consists of the first 2^n elements of (6), converted to n-bit strings by inserting 0s at the left if needed.

When $k = 2^n + r$, where $0 \le r < 2^n$, relation (5) tells us that $g(k)$ is equal to $2^n + g(2^n - 1 - r)$. Therefore we can prove by induction on n that the integer k whose binary representation is $(\ldots b_2 b_1 b_0)_2$ has a Gray binary equivalent $g(k)$ with the representation $(\ldots a_2 a_1 a_0)_2$, where

$$a_j = b_j \oplus b_{j+1}, \qquad \text{for } j \ge 0. \tag{7}$$

(See exercise 6.) For example, $g\big((111001000011)_2\big) = (100101100010)_2$. Conversely, if $g(k) = (\ldots a_2 a_1 a_0)_2$ is given, we can find $k = (\ldots b_2 b_1 b_0)_2$ by inverting the system of equations (7), obtaining

$$b_j = a_j \oplus a_{j+1} \oplus a_{j+2} \oplus \cdots, \qquad \text{for } j \ge 0; \tag{8}$$

this infinite sum is really finite because $a_{j+t} = 0$ for all large t.

One of the many pleasant consequences of Eq. (7) is that $g(k)$ can be computed very easily with bitwise arithmetic:

$$g(k) = k \oplus \lfloor k/2 \rfloor. \tag{9}$$

Similarly, the inverse function in (8) satisfies

$$g^{[-1]}(l) = l \oplus \lfloor l/2 \rfloor \oplus \lfloor l/4 \rfloor \oplus \cdots; \tag{10}$$

this function, however, requires more computation (see exercise 7.1–00). We can also deduce from (7) that, if k and k' are any nonnegative integers,

$$g(k \oplus k') = g(k) \oplus g(k'). \tag{11}$$

Yet another consequence is that the $(n+1)$-bit Gray binary code can be written

$$\Gamma_{n+1} = 0\Gamma_n,\ (0\Gamma_n) \oplus 110\ldots0;$$

this pattern is evident, for example, in (4). Comparing with (5), we see that reversing the order of Gray binary code is equivalent to complementing the first bit:

$$\Gamma_n^R = \Gamma_n \oplus 1\overbrace{0\ldots0}^{n-1}. \tag{12}$$

The exercises below show that the function $g(k)$ defined in (7), and its inverse $g^{[-1]}$ defined in (8), have many further properties and applications of interest. Sometimes we think of these as functions taking binary strings to binary strings; at other times we regard them as functions from integers to integers, via binary notation, with leading zeros irrelevant.

Gray binary code is named after Frank Gray, a physicist who became famous for helping to devise the method long used for compatible color television broadcasting [*Bell System Tech. J.* **13** (1934), 464–515]. He invented Γ_n for applications to pulse code modulation, a method for analog transmission of digital signals [see *Bell System Tech. J.* **30** (1951), 38–40; *U.S. Patent 2632058* (17 March 1953); W. R. Bennett, *Introduction to Signal Transmission* (1971), 238–240]. But the idea of "Gray binary code" was known long before he worked on it; for example, it appeared in *U.S. Patent 2307868* by George Stibitz (12 January 1943). More significantly, Γ_5 was used in a telegraph machine demonstrated in 1878 by Émile Baudot, after whom the term "baud" was later named. At

about the same time, a similar but less systematic code for telegraphy was independently devised by Otto Schäffler [see *Journal Télégraphique* **4** (1878), 252–253; *Annales Télégraphiques* **6** (1879), 361, 382–383].*

In fact, Gray binary code is implicitly present in a classic toy that has fascinated people for centuries, now generally known as the "Chinese ring puzzle" in English, although Englishmen used to call it the "tiring irons." Figure 11 shows a seven-ring example. The challenge is to remove the rings from the bar, and the rings are interlocked in such a way that only two basic types of move are possible (although this may not be immediately apparent from the illustration):

a) The rightmost ring can be removed or replaced at any time;
b) Any other ring can be removed or replaced if and only if the ring to its right is on the bar and all rings to the right of that one are off.

We can represent the current state of the puzzle in binary notation, writing 1 if a ring is on the bar and 0 if it is off; thus Fig. 11 shows the rings in state 1011000. (The second ring from the left is encoded as 0, because it lies entirely above the bar.)

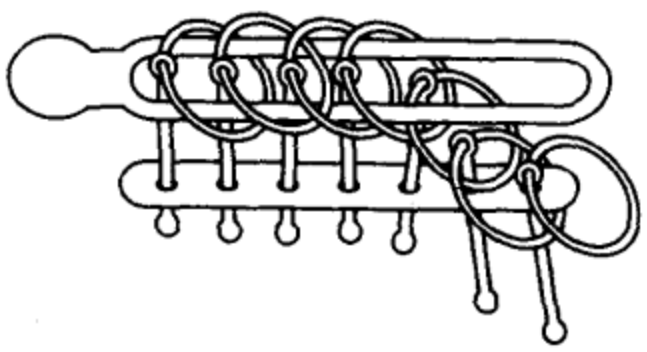


Fig. 11.
The Chinese ring puzzle.

A French magistrate named Louis Gros demonstrated an explicit connection between Chinese rings and binary numbers, in a booklet called *Théorie du Baguenodier* [sic] (Lyon: Aimé Vingtrinier, 1872) that was published anonymously. If the rings are in state $a_{n-1}\ldots a_0$, and if we define the binary number $k = (b_{n-1}\ldots b_0)_2$ by Eq. (8), he showed that exactly k more steps are necessary and sufficient to solve the puzzle. Thus Gros is the true inventor of Gray binary code.

Certainly no home should be without
this fascinating, historic, and instructive puzzle.
— HENRY E. DUDENEY (1901)

When the rings are in any state other than $00\ldots0$ or $10\ldots0$, exactly two moves are possible, one of type (a) and one of type (b). Only one of these moves advances toward the desired goal; the other is a step backward that will need to be undone. A type (a) move changes k to $k \oplus 1$; thus we want to do it when k is odd, since this will decrease k. A type (b) move from a position that ends in $(10^{j-1})_2$ for $1 \le j < n$ changes k to $k \oplus (1^{j+1})_2 = k \oplus (2^{j+1} - 1)$. When k

* Some authors have asserted that Gray code was invented by Elisha Gray, who developed a printing telegraph machine at the same time as Baudot and Schäffler. Such claims are untrue, although Elisha did get a raw deal with respect to priority for inventing the telephone [see L. W. Taylor, *Amer. Physics Teacher* **5** (1937), 243–251].

is even, we want $k \oplus (2^{j+1} - 1)$ to equal $k - 1$, which means that k must be a multiple of 2^j but not a multiple of 2^{j+1}; in other words,

$$j = \rho(k), \tag{13}$$

where ρ is the "ruler function" of Eq. 7.1–(oo). Therefore the rings follow a nice pattern when the puzzle is solved properly: If we number them $0, 1, \ldots, n-1$ (starting at the free end), the sequence of ring moves on or off the bar is the sequence of numbers that ends with $\ldots, \rho(4), \rho(3), \rho(2), \rho(1)$.

Going backwards, successively putting rings on or off until we reach the ultimate state $10\ldots0$ (which, as John Wallis observed in 1693, is more difficult to reach than the supposedly harder state $11\ldots1$), yields an algorithm for counting in Gray binary code:

Algorithm G (*Gray binary generation*). This algorithm visits all binary n-tuples $(a_{n-1}, \ldots, a_1, a_0)$ by starting with $(0, \ldots, 0, 0)$ and changing only one bit at a time, also maintaining a parity bit a_∞ such that

$$a_\infty = a_{n-1} \oplus \cdots \oplus a_1 \oplus a_0. \tag{14}$$

It successively complements bits $\rho(1), \rho(2), \rho(3), \ldots, \rho(2^n - 1)$ and then stops.

G1. [Initialize.] Set $a_j \leftarrow 0$ for $0 \le j < n$; also set $a_\infty \leftarrow 0$.

G2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

G3. [Change parity.] Set $a_\infty \leftarrow 1 - a_\infty$.

G4. [Choose j.] If $a_\infty = 1$, set $j \leftarrow 0$. Otherwise let $j \ge 1$ be minimum such that $a_{j-1} = 1$. (After the kth time we have performed this step, $j = \rho(k)$.)

G5. [Complement coordinate j.] Terminate if $j = n$; otherwise set $a_j \leftarrow 1 - a_j$ and return to G2. ∎

The parity bit a_∞ comes in handy if we are computing a sum like

$$X_{000} - X_{001} - X_{010} + X_{011} - X_{100} + X_{101} + X_{110} - X_{111}$$

or

$$X_\emptyset - X_a - X_b + X_{ab} - X_c + X_{ac} + X_{bc} - X_{abc},$$

where the sign depends on the parity of a binary string or the number of elements in a subset. Such sums arise frequently in "inclusion-exclusion" formulas such as Eq. 1.3.3–(29). The parity bit is also necessary, for efficiency: Without it we could not easily choose between the two ways of determining j, which correspond to performing a type (a) or type (b) move in the Chinese ring puzzle. But the most important feature of Algorithm G is that step G5 makes only a single coordinate change. Therefore only a simple change is usually needed to the terms X that we are summing, or to whatever other structures we are concerned with as we visit each n-tuple.

> *It is impossible, of course, to remove all ambiguity in the lowest-order digit except by a scheme like one the Irish railways are said to have used of removing the last car of every train because it is too susceptible to collision damage.*
>
> — G. R. STIBITZ and J. A. LARRIVEE, *Mathematics and Computers* (1957)

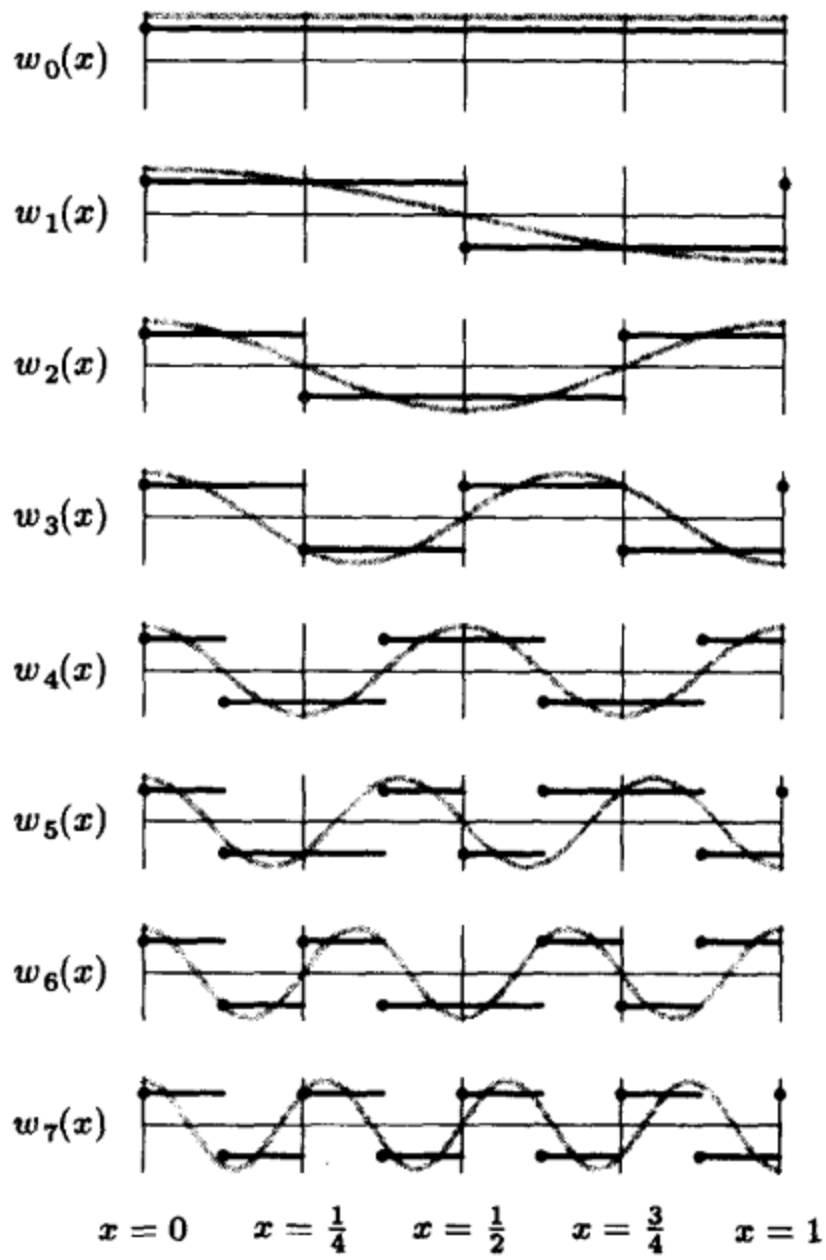


Fig. 12. Walsh functions $w_k(x)$ for $0 \le k < 8$, with the analogous trigonometric functions $\sqrt{2}\cos k\pi x$ shown in gray for comparison.

Another key property of Gray binary code was discovered by J. L. Walsh in connection with an important sequence of functions now known as *Walsh functions* [see *Amer. J. Math.* **45** (1923), 5–24]. Let $w_0(x) = 1$ for all real numbers x, and

$$w_k(x) = (-1)^{\lfloor 2x \rfloor \lceil k/2 \rceil} w_{\lfloor k/2 \rfloor}(2x), \qquad \text{for } k > 0. \tag{15}$$

For example, $w_1(x) = (-1)^{\lfloor 2x \rfloor}$ changes sign whenever x is an integer or an integer plus $\frac{1}{2}$. It follows that $w_k(x) = w_k(x+1)$ for all k, and that $w_k(x) = \pm 1$ for all x. More significantly, $w_k(0) = 1$ and $w_k(x)$ *has exactly* k *sign changes in the interval* $(0\,..\,1)$, so that it approaches $(-1)^k$ as x approaches 1 from the left. Therefore $w_k(x)$ behaves rather like a trigonometric function $\cos k\pi x$ or $\sin k\pi x$, and we can represent other functions as a linear combination of Walsh functions in much the same way as they are traditionally represented as Fourier series. This fact, together with the simple discrete nature of $w_k(x)$, makes Walsh functions extremely useful in computer calculations related to information transmission, image processing, and many other applications.

Figure 12 shows the first eight Walsh functions together with their trigonometric cousins. Engineers commonly call $w_k(x)$ the Walsh function of *sequency* k, by analogy with the fact that $\cos k\pi x$ and $\sin k\pi x$ have *frequency* $k/2$. [See, for example, the book *Sequency Theory: Foundations and Applications* (New York: Academic Press, 1977), by H. F. Harmuth.]

Although Eq. (15) may look formidable at first glance, it actually provides an easy way to see by induction why $w_k(x)$ has exactly k sign changes as claimed. If k is even, say $k = 2l$, we have $w_{2l}(x) = w_l(2x)$ for $0 \le x < \frac{1}{2}$; the effect is simply to compress the function $w_l(x)$ into half the space, so $w_{2l}(x)$ has accumulated l sign changes so far. Then $w_{2l}(x) = (-1)^l w_l(2x) = (-1)^l w_l(2x-1)$ in the range $\frac{1}{2} \le x < 1$; this concatenates another copy of $w_l(2x)$, flipping the sign if necessary to avoid a sign change at $x = \frac{1}{2}$. The function $w_{2l+1}(x)$ is similar, but it *forces* a sign change when $x = \frac{1}{2}$.

What does this have to do with Gray binary code? Walsh discovered that his functions could all be expressed neatly in terms of simpler functions called *Rademacher functions* [Hans Rademacher, *Math. Annalen* **87** (1922), 112–138],

$$r_k(x) \;=\; (-1)^{\lfloor 2^k x \rfloor}, \tag{16}$$

which take the value $(-1)^{c-k}$ when $(\ldots c_2 c_1 c_0 . c_{-1} c_{-2} \ldots)_2$ is the binary representation of x. Indeed, we have $w_1(x) = r_1(x)$, $w_2(x) = r_1(x) r_2(x)$, $w_3(x) = r_2(x)$, and in general

$$w_k(x) = \prod_{j \ge 0} r_{j+1}(x)^{b_j \oplus b_{j+1}} \qquad \text{when } k = (b_{n-1} \ldots b_1 b_0)_2. \tag{17}$$

(See exercise 33.) Thus the exponent of $r_{j+1}(x)$ in $w_k(x)$ is the jth bit of the Gray binary number $g(k)$, according to (7), and we have

$$w_k(x) \;=\; r_{\rho(k)+1}(x)\, w_{k-1}(x), \qquad \text{for } k > 0. \tag{18}$$

Equation (17) implies the handy formula

$$w_k(x)\, w_{k'}(x) \;=\; w_{k \oplus k'}(x), \tag{19}$$

which is much simpler than the corresponding product formulas for sines and cosines. This identity follows easily because $r_j(x)^2 = 1$ for all j and x, hence $r_j(x)^{a \oplus b} = r_j(x)^{a+b}$. It implies in particular that $w_k(x)$ is *orthogonal* to $w_{k'}(x)$ when $k \ne k'$, in the sense that the average value of $w_k(x) w_{k'}(x)$ is zero. We also can use (17) to define $w_k(x)$ for fractional values of k like $1/2$ or $13/8$.

The *Walsh transform* of 2^n numbers $(X_0, \ldots, X_{2^n-1})$ is the vector defined by the equation $(x_0, \ldots, x_{2^n-1})^T = W_n (X_0, \ldots, X_{2^n-1})^T$, where W_n is the $2^n \times 2^n$ matrix having $w_j(k/2^n)$ in row j and column k, for $0 \le j, k < 2^n$. For example, Fig. 12 tells us that the Walsh transform when $n = 3$ is

$$\begin{pmatrix} x_{000} \\ x_{001} \\ x_{010} \\ x_{011} \\ x_{100} \\ x_{101} \\ x_{110} \\ x_{111} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \bar{1} & \bar{1} & \bar{1} & \bar{1} \\ 1 & 1 & \bar{1} & \bar{1} & \bar{1} & \bar{1} & 1 & 1 \\ 1 & 1 & \bar{1} & \bar{1} & 1 & 1 & \bar{1} & \bar{1} \\ 1 & \bar{1} & \bar{1} & 1 & 1 & \bar{1} & \bar{1} & 1 \\ 1 & \bar{1} & \bar{1} & 1 & \bar{1} & 1 & 1 & \bar{1} \\ 1 & \bar{1} & 1 & \bar{1} & \bar{1} & 1 & \bar{1} & 1 \\ 1 & \bar{1} & 1 & \bar{1} & 1 & \bar{1} & 1 & \bar{1} \end{pmatrix} \begin{pmatrix} X_{000} \\ X_{001} \\ X_{010} \\ X_{011} \\ X_{100} \\ X_{101} \\ X_{110} \\ X_{111} \end{pmatrix}. \tag{20}$$

(Here $\bar{1}$ stands for -1, and the subscripts are conveniently regarded as binary strings 000–111 instead of as the integers 0–7.) The *Hadamard transform* is defined similarly, but with the matrix H_n in place of W_n, where H_n has $(-1)^{j\cdot k}$ in row j and column k; here '$j\cdot k$' denotes the dot product $a_{n-1}b_{n-1}+\cdots+a_0b_0$ of the binary representations $j=(a_{n-1}\ldots a_0)_2$ and $k=(b_{n-1}\ldots b_0)_2$. For example, the Hadamard transform for $n=3$ is

$$\begin{pmatrix} x'_{000}\\ x'_{001}\\ x'_{010}\\ x'_{011}\\ x'_{100}\\ x'_{101}\\ x'_{110}\\ x'_{111}\end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 1&\bar{1}&1&\bar{1}&1&\bar{1}&1&\bar{1}\\ 1&1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}\\ 1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}&1\\ 1&1&1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}\\ 1&\bar{1}&1&\bar{1}&\bar{1}&1&\bar{1}&1\\ 1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}&1&1\\ 1&\bar{1}&\bar{1}&1&\bar{1}&1&1&\bar{1}\end{pmatrix} \begin{pmatrix} X_{000}\\ X_{001}\\ X_{010}\\ X_{011}\\ X_{100}\\ X_{101}\\ X_{110}\\ X_{111}\end{pmatrix}. \tag{21}$$

This is the same as the discrete Fourier transform on an n-dimensional cube, Eq. 4.6.4–(38), and we can evaluate it quickly "in place" by adapting the method of Yates discussed in Section 4.6.4:

Given	First step	Second step	Third step
X_{000}	$X_{000}+X_{001}$	$X_{000}+X_{001}+X_{010}+X_{011}$	$X_{000}+X_{001}+X_{010}+X_{011}+X_{100}+X_{101}+X_{110}+X_{111}$
X_{001}	$X_{000}-X_{001}$	$X_{000}-X_{001}+X_{010}-X_{011}$	$X_{000}-X_{001}+X_{010}-X_{011}+X_{100}-X_{101}+X_{110}-X_{111}$
X_{010}	$X_{010}+X_{011}$	$X_{000}+X_{001}-X_{010}-X_{011}$	$X_{000}+X_{001}-X_{010}-X_{011}+X_{100}+X_{101}-X_{110}-X_{111}$
X_{011}	$X_{010}-X_{011}$	$X_{000}-X_{001}-X_{010}+X_{011}$	$X_{000}-X_{001}-X_{010}+X_{011}+X_{100}-X_{101}-X_{110}+X_{111}$
X_{100}	$X_{100}+X_{101}$	$X_{100}+X_{101}+X_{110}+X_{111}$	$X_{000}+X_{001}+X_{010}+X_{011}-X_{100}-X_{101}-X_{110}-X_{111}$
X_{101}	$X_{100}-X_{101}$	$X_{100}-X_{101}+X_{110}-X_{111}$	$X_{000}-X_{001}+X_{010}-X_{011}-X_{100}+X_{101}-X_{110}+X_{111}$
X_{110}	$X_{110}+X_{111}$	$X_{100}+X_{101}-X_{110}-X_{111}$	$X_{000}+X_{001}-X_{010}-X_{011}-X_{100}-X_{101}+X_{110}+X_{111}$
X_{111}	$X_{110}-X_{111}$	$X_{100}-X_{101}-X_{110}+X_{111}$	$X_{000}-X_{001}-X_{010}+X_{011}-X_{100}+X_{101}+X_{110}-X_{111}$

Notice that the rows of H_3 are a permutation of the rows of W_3. This is true in general, so we can obtain the Walsh transform by permuting the elements of the Hadamard transform. Exercise 36 discusses the details.

Going faster. When we're running through 2^n possibilities, we usually want to reduce the computation time as much as possible. Algorithm G needs to complement only one bit a_j per visit to $(a_{n-1},\ldots,a_0)$, but it loops in step G4 while choosing an appropriate value of j. Another approach has been suggested by Gideon Ehrlich [*JACM* **20** (1973), 500–513], who introduced the notion of *loopless* combinatorial generation: With a loopless algorithm, the number of operations performed between successive visits is required to be bounded in advance, so there never is a long wait before a new pattern has been generated.

We learned some tricks in Section 7.1 about quick ways to determine the number of leading or trailing 0s in a binary number. Those methods could be used in step G4 to make Algorithm G loopless, assuming that n isn't unreasonably large. But Ehrlich's method is quite different, and much more versatile, so it provides us with a new weapon in our arsenal of techniques for efficient computation. Here is how his approach can be used to generate binary n-tuples [see Bitner, Ehrlich, and Reingold, *CACM* **19** (1976), 517–521]:

Algorithm L (*Loopless Gray binary generation*). This algorithm, like Algorithm G, visits all binary n-tuples $(a_{n-1}, \ldots, a_0)$ in the order of the Gray binary code. But instead of maintaining a parity bit, it uses an array of "focus pointers" $(f_n, \ldots, f_0)$, whose significance is discussed below.

L1. [Initialize.] Set $a_j \leftarrow 0$ and $f_j \leftarrow j$ for $0 \le j < n$; also set $f_n \leftarrow n$. (A loopless algorithm is allowed to have loops in its initialization step, as long as the initial setup is reasonably efficient; after all, every program needs to be loaded and launched.)

L2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

L3. [Choose j.] Set $j \leftarrow f_0$, $f_0 \leftarrow 0$. (If this is the kth time we are performing the present step, j is now equal to $\rho(k)$.) Terminate if $j = n$; otherwise set $f_j \leftarrow f_{j+1}$ and $f_{j+1} \leftarrow j + 1$.

L4. [Complement coordinate j.] Set $a_j \leftarrow 1 - a_j$ and return to L2. ∎

For example, the computation proceeds as follows when $n = 4$. Elements a_j have been underlined in this table if the corresponding bit b_j is 1 in the binary string $b_3b_2b_1b_0$ such that $a_3a_2a_1a_0 = g(b_3b_2b_1b_0)$:

a_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
a_2	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
a_1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
a_0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
f_3	3	3	3	3	3	3	3	3	4	4	4	4	3	3	3	3
f_2	2	2	2	2	3	3	2	2	2	2	2	2	4	4	2	2
f_1	1	1	2	1	1	1	3	1	1	1	2	1	1	1	4	1
f_0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

Although the binary number $k = (b_{n-1} \ldots b_0)_2$ never appears explicitly in Algorithm L, the focus pointers f_j represent it implicitly in a clever way, so that we can repeatedly form $g(k) = (a_{n-1} \ldots a_0)_2$ by complementing bit $a_{\rho(k)}$ as we should. Let's say that a_j is *passive* when it is underlined, *active* otherwise. Then the focus pointers satisfy the following invariant relations:

1) If a_j is passive and a_{j-1} is active, then f_j is the smallest index $j' > j$ such that $a_{j'}$ is active. (Bits a_n and a_{-1} are considered to be active for purposes of this rule, although they aren't really present in the algorithm.)
2) Otherwise $f_j = j$.

Thus, the rightmost element a_j of a block of passive elements $a_{i-1} \ldots a_{j+1}a_j$, with decreasing subscripts, has a focus f_j that points to the element a_i just to the left of that block. All other elements a_j have f_j pointing to themselves.

In these terms, the first two operations '$j \leftarrow f_0$, $f_0 \leftarrow 0$' in step L3 are equivalent to saying, "Set j to the index of the rightmost active element, and activate all elements to the right of a_j." Notice that if $f_0 = 0$, the operation $f_0 \leftarrow 0$ is redundant; but it doesn't do any harm. The other two operations of L3, '$f_j \leftarrow f_{j+1}$, $f_{j+1} \leftarrow j + 1$', are equivalent to saying, "Make a_j passive," because we know that a_j and a_{j-1} are both active at this point in the computation.

(Again the operation $f_{j+1} \leftarrow j+1$ might be harmlessly redundant.) The net effect of activation and passivation is therefore equivalent to counting in binary notation, as in Algorithm M, with 1-bits passive and 0-bits active.

Algorithm L is almost blindingly fast, because it does only five assignment operations and one test for termination between each visit to a generated n-tuple. But we can do even better. In order to see how, let's consider an application to recreational linguistics: Rudolph Castown, in *Word Ways* **1** (1968), 165–169, noted that all 16 of the ways to intermix the letters of **sins** with the corresponding letters of **fate** produce words that are found in a sufficiently large dictionary of English: **sine**, **sits**, **site**, etc.; and all but three of those words (namely **fane**, **fite**, and **sats**) are sufficiently common as to be unquestionably part of standard English. Therefore it is natural to ask the analogous question for five-letter words: What two strings of five letters will produce the maximum number of words in the Stanford GraphBase, when letters in corresponding positions are swapped in all 32 possible ways?

To answer this question, we need not examine all $\binom{26}{2}^5 = 3{,}625{,}908{,}203{,}125$ essentially different pairs of strings; it suffices to look at all $\binom{5757}{2} = 16{,}568{,}646$ pairs of words in the GraphBase, provided that at least one of those pairs produces at least 17 words, because every set of 17 or more five-letter words obtainable from two five-letter strings must contain two that are "antipodal" (with no corresponding letters in common). For every antipodal pair, we want to determine as rapidly as possible whether the 32 possible subset-swaps produce a significant number of English words.

Every 5-letter word can be represented as a 25-bit number using 5 bits per letter, from "**a**" = 00000 to "**z**" = 11001. A table of 2^{25} bits or bytes will then determine quickly whether a given five-letter string is a word. So the problem is reduced to generating the bit patterns of the 32 potential words obtainable by mixing the letters of two given words, and looking those patterns up in the table. We can proceed as follows, for each pair of 25-bit words w and w':

W1. [Check the difference.] Set $z \leftarrow w \oplus w'$. Reject the word pair (w, w') if $\big((z-m)\oplus z\oplus m\big)\&m' \neq 0$, where $m = 2^{20}+2^{15}+2^{10}+2^5+1$ and $m' = 2^5 m$; this test eliminates cases where w and w' have a common letter in some position. (See 7.1–(oo); it turns out that 10,614,085 of the 16,568,646 word pairs have no such common letters.)

W2. [Form individual masks.] Set $m_0 \leftarrow z\,\&\,(2^5-1)$, $m_1 \leftarrow z\,\&\,(2^{10}-2^5)$, $m_2 \leftarrow z\,\&\,(2^{15}-2^{10})$, $m_3 \leftarrow z\,\&\,(2^{20}-2^{15})$, and $m_4 \leftarrow z\,\&\,(2^{25}-2^{20})$, in preparation for the next step.

W3. [Count words.] Set $l \leftarrow 1$ and $A_0 \leftarrow w$; the variable l will count how many words starting with w we have found so far. Then perform the operations *swap*(4) defined below.

W4. [Print a record-setting solution.] If l exceeds or equals the current maximum, print A_j for $0 \le j < l$. ∎

The heart of this high-speed method is the sequence of operations ***swap***(4), which should be expanded inline (for example with a macro-processor) to eliminate all

unnecessary overhead. It is defined in terms of the basic operation

$sw(j)$: Set $w \leftarrow w \oplus m_j$. Then if w is a word, set $A_l \leftarrow w$ and $l \leftarrow l+1$.

Given $sw(j)$, which flips the letters in position j, we define

$$\begin{aligned} &swap(0) = sw(0);\\ &swap(1) = swap(0), sw(1), swap(0);\\ &swap(2) = swap(1), sw(2), swap(1);\\ &swap(3) = swap(2), sw(3), swap(2);\\ &swap(4) = swap(3), sw(4), swap(3). \end{aligned} \tag{22}$$

Thus $swap(4)$ expands into a sequence of 31 steps $sw(0)$, $sw(1)$, $sw(0)$, $sw(2)$, $\ldots$, $sw(0) = sw(\rho(1))$, $sw(\rho(2))$, $\ldots$, $sw(\rho(31))$; these steps will be used 10 million times. We clearly gain speed by embedding the ruler function values $\rho(k)$ directly into our program, instead of recomputing them repeatedly for each word pair via Algorithm M, G, or L.

The winning pair of words generates a set of 21, namely

```
ducks  ducky  duces  dunes  dunks  dinks  dinky
dines  dices  dicey  dicky  dicks  picks  picky
pines  piney  pinky  pinks  punks  punky  pucks
```
(23)

If, for example, $w =$ **ducks** and $w' =$ **piney**, then $m_0 =$ **s** $\oplus$ **y**, so the first operation $sw(0)$ changes **ducks** to **ducky**, which is seen to be a word. The next operation $sw(1)$ applies m_1, which is **k** $\oplus$ **e** in the next-to-last letter position, so it produces the nonword **ducey**. Another application of $sw(0)$ changes **ducey** to **duces** (a legal term generally followed by the word **tecum**). And so on. All word pairs can be processed by this method in at most a few seconds.

Further streamlining is also possible. For example, once we have found a pair that yields k words, we can reject later pairs as soon as they generate $33-k$ nonwords. But the method we've discussed is already quite fast, and it demonstrates the fact that even the loopless Algorithm L can be beaten.

Fans of Algorithm L may, of course, complain that we have speeded up the process only in the small special case $n = 5$, while Algorithm L solves the generation problem for n in general. A similar idea does, however, work also for general values of $n > 5$: We can expand out a program so that it rapidly generates all 32 settings of the rightmost bits $a_4a_3a_2a_1a_0$, as above; then we can apply Algorithm L after every 32 steps, using it to generate successive changes to the other bits $a_{n-1}\ldots a_5$. This approach reduces the amount of unnecessary work done by Algorithm L by nearly a factor of 32.

Other binary Gray codes. The Gray binary code $g(0), g(1), \ldots, g(2^n-1)$ is only one of many ways to traverse all possible n-bit strings while changing only a single bit at each step. Let us say that, in general, a "Gray cycle" on binary n-tuples is *any* sequence $(v_0, v_1, \ldots, v_{2^n-1})$ that includes every n-tuple and has the property that v_k differs from $v_{(k+1) \bmod 2^n}$ in just one bit position. Thus, in the terminology of graph theory, a Gray cycle is an oriented Hamiltonian

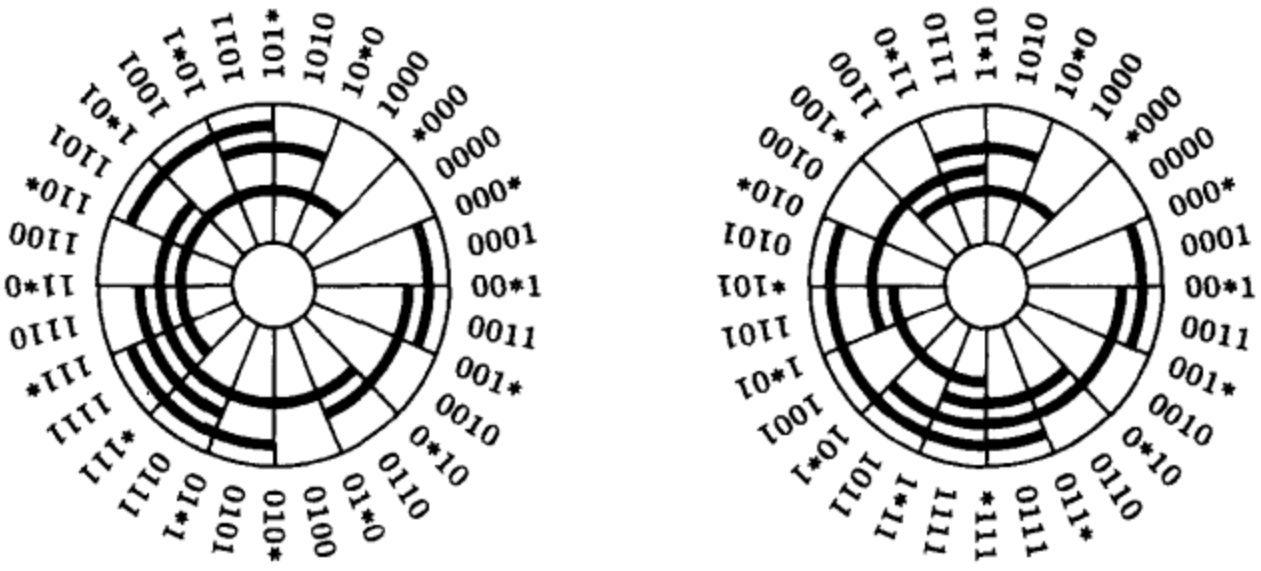


Fig. 13. (a) Complementary Gray code. (b) Balanced Gray code.

cycle on the n-cube. We can assume that subscripts have been chosen so that $v_0 = 0 \ldots 0$.

If we think of the v's as binary numbers, there are integers $\delta_0 \ldots \delta_{2^n-1}$ such that

$$v_{(k+1) \bmod 2^n} = v_k \oplus 2^{\delta_k}, \qquad \text{for } 0 \le k < 2^n; \tag{24}$$

this so-called "delta sequence" is another way to describe a Gray cycle. For example, the delta sequence for standard Gray binary when $n = 3$ is 01020102; it is essentially the ruler function $\delta_k = \rho(k+1)$ of (13), but the final value δ_{2^n-1} is $n-1$ instead of n, so that the cycle closes. The individual elements δ_k always lie in the range $0 \le \delta_k < n$, and they are called "coordinates."

Let $d(n)$ be the number of different delta sequences that define an n-bit Gray cycle, and let $c(n)$ be the number of "canonical" delta sequences in which each coordinate k appears before the first appearance of $k+1$. Then $d(n) = n!\,c(n)$, because every permutation of the coordinate numbers in a delta sequence obviously produces another delta sequence. The only possible canonical delta sequences for $n \le 3$ are easily seen to be

$$00; \qquad 0101; \qquad 01020102 \quad \text{and} \quad 01210121. \tag{25}$$

Therefore $c(1) = c(2) = 1$, $c(3) = 2$; $d(1) = 1$, $d(2) = 2$, and $d(3) = 12$. A straightforward computer calculation, using techniques for the enumeration of Hamiltonian cycles that we will study later, establishes the next values,

$$\begin{aligned} c(4) &= 112; & d(4) &= 2688; \\ c(5) &= 15{,}109{,}096; & d(5) &= 1{,}813{,}091{,}520. \end{aligned} \tag{26}$$

No simple pattern is evident, and the numbers grow quite rapidly (see exercise 45); therefore it's a fairly safe bet that nobody will ever know the exact values of $c(8)$ and $d(8)$.

Since the number of possibilities is so huge, people have been encouraged to look for Gray cycles that have additional useful properties. For example, Fig. 13(a) shows a 4-bit Gray cycle in which every string $a_3a_2a_1a_0$ is diametrically opposite to its complement $\bar{a}_3\bar{a}_2\bar{a}_1\bar{a}_0$. Such coding schemes are possible whenever the number of bits is even (see exercise 49).

An even more interesting Gray cycle, found by G. C. Tootill [*Proc. IEE* **103**, Part B Supplement (1956), 435], is shown in Fig. 13(b). This one has the same number of changes in each of the four coordinate tracks, hence all coordinates share equally in the activities. Gray cycles that are balanced in a similar way can in fact be constructed for all larger values of n, by using the following versatile method to extend a cycle from n bits to $n+2$ bits:

Theorem D. *Let $\alpha_1 j_1 \alpha_2 j_2 \ldots \alpha_l j_l$ be a delta sequence for an n-bit Gray cycle, where each j_k is a single coordinate, each α_k is a possibly empty sequence of coordinates, and l is odd. Then*

$$\begin{aligned}&\alpha_1(n{+}1)\alpha_1^R n\alpha_1\\ &\quad j_1\alpha_2 n\alpha_2^R(n{+}1)\alpha_2\ j_2\alpha_3(n{+}1)\alpha_3^R n\alpha_3\ \ldots\ j_{l-1}\alpha_l(n{+}1)\alpha_l^R n\alpha_l\\ &\qquad\qquad (n{+}1)\alpha_l^R j_{l-1}\alpha_{l-1}^R\ldots\alpha_2^R j_1\alpha_1^R n\end{aligned}\tag{27}$$

is the delta sequence of an $(n+2)$-bit Gray cycle.

For example, if we start with the sequence $01\underline{0}201\underline{0}\underline{2}$ for $n=3$ and let the three underlined elements be j_1, j_2, j_3, the new sequence (27) for a 5-bit cycle is

$$0141030102013102420104340102010 3.\tag{28}$$

Proof. Let α_k have length m_k and let v_{kt} be the vertex reached if we start at $0\ldots0$ and apply the coordinate changes $\alpha_1 j_1 \ldots \alpha_{k-1} j_{k-1}$ and the first t of α_k. We need to prove that all vertices $00v_{kt}$, $01v_{kt}$, $10v_{kt}$, and $11v_{kt}$ occur when (27) is used, for $1 \le k \le l$ and $0 \le t \le m_k$. (The leftmost coordinate is $n{+}1$.)

Starting with $000\ldots0 = 00v_{10}$, we proceed to obtain the vertices

$$00v_{11},\ \ldots,\ 00v_{1m_1},\ 10v_{1m_1},\ \ldots,\ 10v_{10}, 11v_{10},\ \ldots,\ 11v_{1m_1};$$

then j_1 yields $11v_{20}$, which is followed by

$$11v_{21},\ \ldots,\ 11v_{2m_2},\ 10v_{2m_2},\ \ldots,\ 10v_{20}, 00v_{20},\ \ldots,\ 00v_{2m_2};$$

then comes $00v_{30}$, etc., and we eventually reach $11v_{lm_l}$. The glorious finale then uses the third line of (27) to generate all the missing vertices $01v_{lm_l}$, $\ldots$, $01v_{10}$ and take us back to $000\ldots0$. ∎

The *transition counts* $(c_0, \ldots, c_{n-1})$ of a delta sequence are defined by letting c_j be the number of times $\delta_k = j$. For example, (28) has transition counts $(12, 8, 4, 4, 4)$, and it arose from a sequence with transition counts $(4, 2, 2)$. If we choose the original delta sequence carefully and underline appropriate elements j_k, we can obtain transition counts that are as equal as possible:

Corollary B. *For all $n \ge 1$, there is an n-bit Gray cycle with transition counts $(c_0, c_1, \ldots, c_{n-1})$ that satisfy the condition*

$$|c_j - c_k| \le 2 \qquad \text{for } 0 \le j < k < n.\tag{29}$$

(This is the best possible balance condition, because each c_j must be an even number, and we must have $c_0 + c_1 + \cdots + c_{n-1} = 2^n$. Indeed, condition (29)

holds if and only if $n-r$ of the counts are equal to $2q$ and r are equal to $2q+2$, where $q = \lfloor 2^{n-1}/n \rfloor$ and $r = 2^{n-1} \bmod n$.)

Proof. Given a delta sequence for an n-bit Gray cycle with transition counts $(c_0, \ldots, c_{n-1})$, the counts for cycle (27) are obtained by starting with the values $(c'_0, \ldots, c'_{n-1}, c'_n, c'_{n+1}) = (4c_0, \ldots, 4c_{n-1}, l+1, l+1)$, then subtracting 2 from c'_{j_k} for $1 \le k < l$ and subtracting 4 from c'_{j_l}. For example, when $n = 3$ we can obtain a balanced 5-bit Gray cycle having transition counts $(8-2, 16-10, 8, 6, 6) = (6, 6, 8, 6, 6)$ if we apply Theorem D to the delta sequence $\underline{0}\underline{1}2\underline{1}0\underline{1}2\underline{1}$. Exercise 51 works out the details for other values of n. ∎

Another important class of n-bit Gray cycles in which each of the coordinate tracks has equal responsibility arises when we consider *run lengths*, namely the distances between consecutive appearances of the same δ value. Standard Gray binary code has run length 2 in the least significant position, and this can lead to a loss of accuracy when precise measurements need to be made [see, for example, the discussion by G. M. Lawrence and W. E. McClintock, *Proc. SPIE* **2831** (1996), 104–111]. But all runs have length 4 or more in the remarkable 5-bit Gray cycle whose delta sequence is

$$(0123042103210423)^2. \tag{30}$$

Let $r(n)$ be the maximum value r such that an n-bit Gray cycle can be found in which all runs have length $\ge r$. Clearly $r(1) = 1$, and $r(2) = r(3) = r(4) = 2$; and it is easy to see that $r(n)$ must be less than n when $n > 2$, hence (30) proves that $r(5) = 4$. Exhaustive computer searches establish the values $r(6) = 4$ and $r(7) = 5$. Indeed, a fairly straightforward backtrack calculation for the case $n = 7$ needs a tree of only about 60 million nodes to determine that $r(7) < 6$, and exercise 61(a) constructs a 7-bit cycle with no run shorter than 5. The exact values of $r(n)$ are unknown for $n \ge 8$; but $r(10)$ is almost certainly 8, and interesting constructions are known by which we can prove that $r(n) = n - O(\log n)$ as $n \to \infty$. (See exercises 60–64.)

***Binary Gray paths.** We have defined an n-bit Gray cycle as a way to arrange all binary n-tuples into a sequence $(v_0, v_1, \ldots, v_{2^n-1})$ with the property that v_k is adjacent to v_{k+1} in the n-cube for $0 \le k < 2^n$, and such that v_{2^n-1} is also adjacent to v_0. The cyclic property is nice, but not always essential; and sometimes we can do better without it. Therefore we say that an n-bit *Gray path*, also commonly called a *Gray code*, is any sequence that satisfies the conditions of a Gray cycle except that the last element need not be adjacent to the first. In other words, a Gray cycle is a Hamiltonian *cycle* on the vertices of the n-cube, but a Gray code is simply a Hamiltonian *path* on that graph.

The most important binary Gray paths that are not also Gray cycles are n-bit sequences $(v_0, v_1, \ldots, v_{2^n-1})$ that are *monotonic*, in the sense that

$$\nu(v_k) \le \nu(v_{k+2}) \qquad \text{for } 0 \le k < 2^n - 2. \tag{31}$$

(Here, as elsewhere, we use ν to denote the "weight" or the "sideways sum" of a binary string, namely the number of 1s that it has.) Trial and error shows that

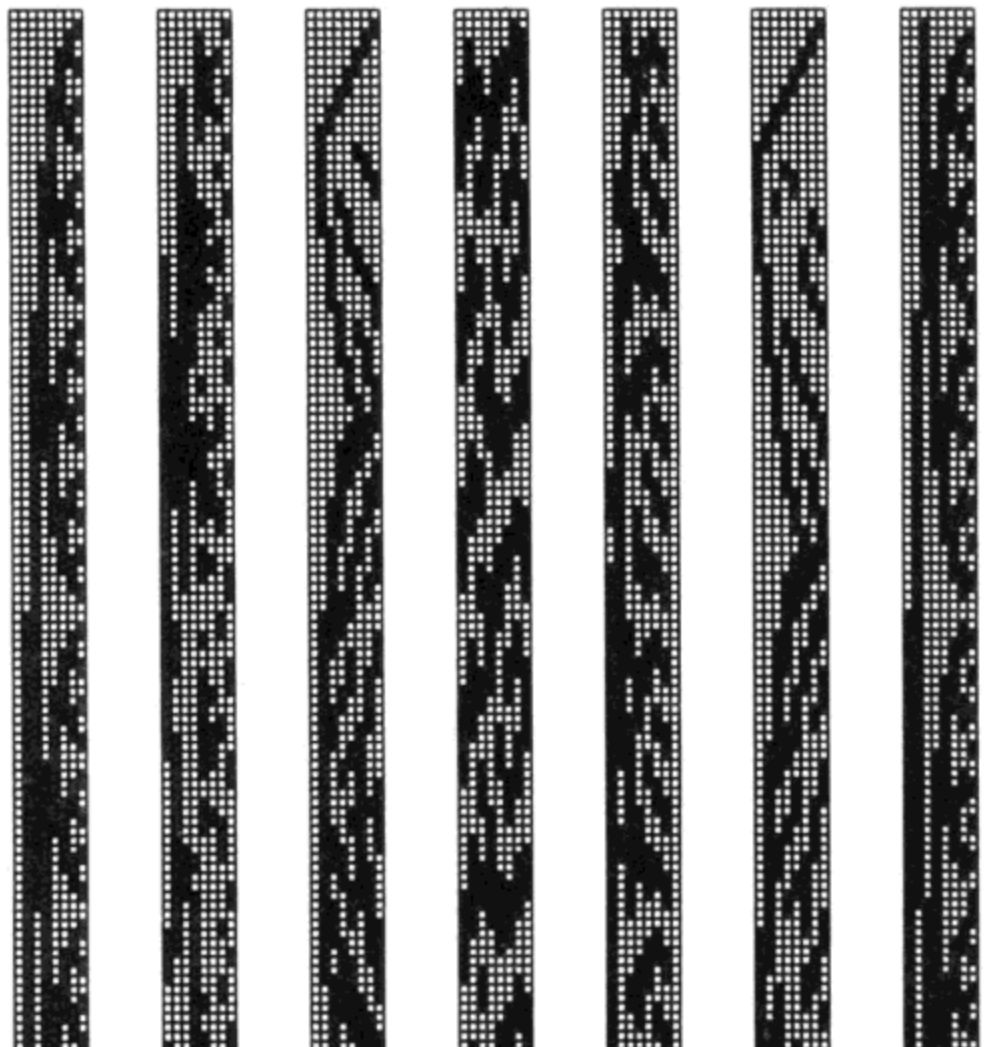


Fig. 14. Examples of 8-bit Gray codes:

a) standard;
b) balanced;
c) complementary;
d) long-run;
e) nonlocal;
f) monotonic;
g) trend-free.

there are essentially only two monotonic n-bit Gray codes for each $n \le 4$, one starting with 0^n and the other starting with $0^{n-1}1$. The two for $n = 3$ are

$$000, 001, 011, 010, 110, 100, 101, 111; \tag{32}$$

$$001, 000, 010, 110, 100, 101, 111, 011. \tag{33}$$

The two for $n = 4$ are slightly less obvious, but not really difficult to discover.

Since $\nu(v_{k+1}) = \nu(v_k) \pm 1$ whenever v_k is adjacent to v_{k+1}, we obviously can't strengthen (31) to the requirement that all n-tuples be strictly sorted by weight. But relation (31) is strong enough to determine the weight of each v_k, given k and the weight of v_0, because we know that exactly $\binom{n}{j}$ of the n-tuples have weight j.

Figure 14 summarizes our discussions so far, by illustrating seven of the zillions of Gray codes that make a grand tour through all 256 of the possible 8-bit bytes. Black squares represent ones and white squares represent zeros. Figure 14(a) is the standard Gray binary code, while Fig. 14(b) is balanced with exactly $256/8 = 32$ transitions in each coordinate position. Fig. 14(c) is a Gray code analogous to Fig. 13(a), in which the bottom 128 codes are complements of the top 128. In Fig. 14(d), the transitions in each coordinate position never occur closer than five steps apart; in other words, all run lengths are at least 5. The cycle in Fig. 14(e) is *nonlocal* in the sense of exercise 59. Fig. 14(f) shows a monotonic path for $n = 8$; notice how black it gets near the bottom. Finally, Fig. 14(g) illustrates a Gray code that is totally nonmonotonic, in the sense that the center of gravity of the black squares lies exactly at the halfway point in each column. Standard Gray binary code has this property in seven of the coordinate positions, but Fig. 14(g) achieves perfect black-white weight balance in all eight. Such codes are called *trend-free*; they are important in the design of agricultural and other experiments (see exercises 75 and 76).

Carla Savage and Peter Winkler [*J. Combinatorial Theory* **A70** (1995), 230–248] found an elegant way to construct monotonic binary Gray codes for all $n > 0$. Such paths are necessarily built from subpaths P_{nj} in which all transitions are between n-tuples of weights j and $j + 1$. Savage and Winkler defined suitable subpaths recursively by letting $P_{10} = 0, 1$ and, for all $n > 0$,

$$P_{(n+1)j} = 1P_{n(j-1)}^{\pi_n},\ 0P_{nj}; \tag{34}$$

$$P_{nj} = \emptyset \quad \text{if } j < 0 \text{ or } j \ge n. \tag{35}$$

Here π_n is a permutation of the coordinates that we will specify later, and the notation P^π means that every element $a_{n-1} \ldots a_1 a_0$ of the sequence P is replaced by $b_{n-1} \ldots b_1 b_0$, where $b_{j\pi} = a_j$. (We don't define P^π by letting $b_j = a_{j\pi}$, because we want $(2^j)^\pi$ to be $2^{j\pi}$.) It follows, for example, that

$$P_{20} = 0P_{10} = 00,\ 01 \tag{36}$$

because $P_{1(-1)}$ is vacuous; also

$$P_{21} = 1P_{10}^{\pi_1} = 10,\ 11 \tag{37}$$

because P_{11} is vacuous and π_1 must be the identity permutation. In general, P_{nj} is a sequence of n-bit strings containing exactly $\binom{n-1}{j}$ strings of weight j interleaved with $\binom{n-1}{j}$ strings of weight $j+1$.

Let α_{nj} and ω_{nj} be the first and last elements of P_{nj}. Then we easily find

$$\omega_{nj} = 0^{n-j-1}1^{j+1}, \qquad \text{for } 0 \le j < n; \tag{38}$$

$$\alpha_{n0} = 0^n, \qquad \text{for } n > 0; \tag{39}$$

$$\alpha_{nj} = 1\alpha_{(n-1)(j-1)}^{\pi_{n-1}}, \qquad \text{for } 1 \le j < n. \tag{40}$$

In particular, α_{nj} always has weight j, and ω_{nj} always has weight $j+1$. We will define permutations π_n of $\{0, 1, \ldots, n-1\}$ so that both of the sequences

$$P_{n0},\ P_{n1}^R,\ P_{n2},\ P_{n3}^R,\ \ldots \tag{41}$$

$$\text{and } P_{n0}^R,\ P_{n1},\ P_{n2}^R,\ P_{n3},\ \ldots \tag{42}$$

are monotonic binary Gray paths for $n = 1, 2, 3, \ldots$. In fact, the monotonicity is clear, so only the Grayness is in doubt; and the sequences (41), (42) link up nicely because the adjacencies

$$\alpha_{n0} \mathrel{—} \alpha_{n1} \mathrel{—} \cdots \mathrel{—} \alpha_{n(n-1)}, \qquad \omega_{n0} \mathrel{—} \omega_{n1} \mathrel{—} \cdots \mathrel{—} \omega_{n(n-1)} \tag{43}$$

follow immediately from (34), regardless of the permutations π_n. Thus the crucial point is the transition at the comma in formula (34), which makes $P_{(n+1)j}$ a Gray subpath if and only if

$$\omega_{n(j-1)}^{\pi_n} = \alpha_{nj} \qquad \text{for } 0 < j < n. \tag{44}$$

For example, when $n = 2$ and $j = 1$ we need $(01)^{\pi_2} = \alpha_{21} = 10$, by (38)–(40); hence π_2 must transpose coordinates 0 and 1. The general formula (see exercise 71) turns out to be

$$\pi_n = \sigma_n \pi_{n-1}^2, \tag{45}$$

where σ_n is the n-cycle $(n{-}1\ \ldots\ 1\,0)$. The first few cases are therefore

$$\pi_1 = (0), \qquad \pi_4 = (0\,3),$$
$$\pi_2 = (0\,1), \qquad \pi_5 = (0\,4\,3\,2\,1),$$
$$\pi_3 = (0\,2\,1), \qquad \pi_6 = (0\,5\,2\,4\,1\,3);$$

no simple "closed form" for the magic permutations π_n is apparent. Exercise 73 shows that the Savage–Winkler codes can be generated efficiently.

Nonbinary Gray codes. We have studied the case of binary n-tuples in great detail, because it is the simplest, most classical, most applicable, and most thoroughly explored part of the subject. But of course there are numerous applications in which we want to generate $(a_1, \ldots, a_n)$ with coordinates in the more general ranges $0 \le a_j < m_j$, as in Algorithm M. Gray codes apply nicely to this case as well.

Consider, for example, decimal digits, where we want $0 \le a_j < 10$ for each j. Is there a decimal way to count that is analogous to the Gray binary code, changing only one digit at a time? Yes; in fact, *two* natural schemes are

available. In the first, called *reflected Gray decimal*, the sequence for counting up to a thousand with 3-digit strings has the form

$$000, 001, \ldots, 009, 019, 018, \ldots, 011, 010, 020, 021, \ldots, 091, 090, 190, 191, \ldots, 900,$$

with each coordinate moving alternately from 0 up to 9 and then back down from 9 to 0. In the second, called *modular Gray decimal*, the digits always increase by 1 mod 10, therefore they "wrap around" from 9 to 0:

$$000, 001, \ldots, 009, 019, 010, \ldots, 017, 018, 028, 029, \ldots, 099, 090, 190, 191, \ldots, 900.$$

In both cases the digit that changes on step k is determined by the radix-ten ruler function $\rho_{10}(k)$, the largest power of 10 that divides k. Therefore each n-tuple of digits occurs exactly once: We generate 10^j different settings of the rightmost j digits before changing any of the others, for $1 \le j \le n$.

In general, the reflected Gray code in any mixed-radix system can be regarded as a permutation of the nonnegative integers, a function that maps an ordinary mixed-radix number

$$k = \left[\begin{matrix} b_{n-1}, & \ldots, & b_1, & b_0 \\ m_{n-1}, & \ldots, & m_1, & m_0 \end{matrix}\right] = b_{n-1} m_{n-2} \ldots m_1 m_0 + \cdots + b_1 m_0 + b_0 \qquad (46)$$

into its reflected-Gray equivalent

$$\hat{g}(k) = \left[\begin{matrix} a_{n-1}, & \ldots, & a_1, & a_0 \\ m_{n-1}, & \ldots, & m_1, & m_0 \end{matrix}\right] = a_{n-1} m_{n-2} \ldots m_1 m_0 + \cdots + a_1 m_0 + a_0, \qquad (47)$$

just as (7) does this in the special case of binary numbers. Let

$$A_j = \left[\begin{matrix} a_{n-1}, & \ldots, & a_j \\ m_{n-1}, & \ldots, & m_j \end{matrix}\right], \qquad B_j = \left[\begin{matrix} b_{n-1}, & \ldots, & b_j \\ m_{n-1}, & \ldots, & m_j \end{matrix}\right], \qquad (48)$$

with $A_n = B_n = 0$, so that when $0 \le j < n$ we have

$$A_j = m_j A_{j+1} + a_j \qquad \text{and} \qquad B_j = m_j B_{j+1} + b_j. \qquad (49)$$

The rule connecting the a's and b's is not difficult to derive by induction on $n - j$:

$$a_j = \begin{cases} b_j, & \text{if } B_{j+1} \text{ is even;} \\ m_j - 1 - b_j, & \text{if } B_{j+1} \text{ is odd.} \end{cases} \qquad (50)$$

(Here we are numbering the coordinates of the n-tuples $(a_{n-1}, \ldots, a_1, a_0)$ and $(b_{n-1}, \ldots, b_1, b_0)$ from right to left, for consistency with (7) and the conventions of mixed-radix notation in Eq. 4.1–(9). Readers who prefer notations like $(a_1, \ldots, a_n)$ can change j to $n - j$ in all the formulas if they wish.) Going the other way, we have

$$b_j = \begin{cases} a_j, & \text{if } a_{j+1} + a_{j+2} + \cdots \text{ is even;} \\ m_j - 1 - a_j, & \text{if } a_{j+1} + a_{j+2} + \cdots \text{ is odd.} \end{cases} \qquad (51)$$

Curiously, rule (50) and its inverse in (51) are exactly the same when all of the radices m_j are odd. In Gray ternary code, for example, when $m_0 = m_1 = \cdots = 3$, we have $\hat{g}\big((10010211012)_3\big) = (12210211010)_3$ and also $\hat{g}\big((12210211010)_3\big) =$

$(10010211012)_3$. Exercise 78 proves (50) and (51), and discusses similar formulas that hold in the modular case.

We can in fact generate such Gray sequences looplessly, generalizing Algorithms M and L:

Algorithm H (*Loopless reflected mixed-radix Gray generation*). This algorithm visits all n-tuples $(a_{n-1}, \ldots, a_0)$ such that $0 \le a_j < m_j$ for $0 \le j < n$, changing only one coordinate by ± 1 at each step. It maintains an array of focus pointers $(f_n, \ldots, f_0)$ to control the actions as in Algorithm L, together with an array of directions $(o_{n-1}, \ldots, o_0)$. We assume that each radix m_j is ≥ 2.

H1. [Initialize.] Set $a_j \leftarrow 0$, $f_j \leftarrow j$, and $o_j \leftarrow 1$, for $0 \le j < n$; also set $f_n \leftarrow n$.

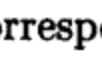
H2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

H3. [Choose j.] Set $j \leftarrow f_0$ and $f_0 \leftarrow 0$. (As in Algorithm L, j was the rightmost active coordinate; all elements to its right have now been reactivated.)

H4. [Change coordinate j.] Terminate if $j = n$; otherwise set $a_j \leftarrow a_j + o_j$.

H5. [Reflect?] If $a_j = 0$ or $a_j = m_j - 1$, set $o_j \leftarrow -o_j$, $f_j \leftarrow f_{j+1}$, and $f_{j+1} \leftarrow j + 1$. (Coordinate j has thus become passive.) Return to H2. ∎

A similar algorithm generates the modular variation (see exercise 77).

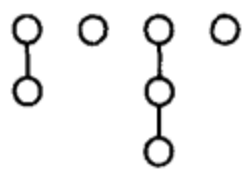
***Subforests.** An interesting and instructive generalization of Algorithm H, discovered by Y. Koda and F. Ruskey [*J. Algorithms* **15** (1993), 324–340], sheds further light on the subject of Gray codes and loopless generation. Suppose we have a forest of n nodes, and we want to visit all of its "principal subforests," namely all subsets of nodes S such that if x is in S and x is not a root, the parent of x is also in S. For example, the 7-node forest  has 33 such subsets, corresponding to the black nodes in the following 33 diagrams:



(52)

Notice that if we read the top row from left to right, the middle row from right to left, and the bottom row from left to right, the status of exactly one node changes at each step.

If the given forest consists of degenerate nonbranching trees, the principal subforests are equivalent to mixed-radix numbers. For example, a forest like

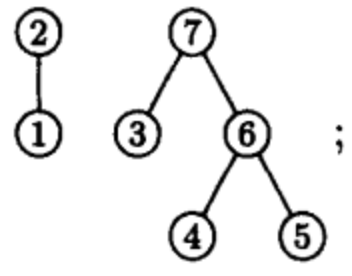


has $3 \times 2 \times 4 \times 2$ principal subforests, corresponding to 4-tuples (x_1, x_2, x_3, x_4) such that $0 \le x_1 < 3$, $0 \le x_2 < 2$, $0 \le x_3 < 4$, and $0 \le x_4 < 2$; the value of x_j is the number of nodes selected in the jth tree. When the algorithm of Koda

and Ruskey is applied to such a forest, it will visit the subforests in the same order as the reflected Gray code on radices $(3, 2, 4, 2)$.

Algorithm K (*Loopless reflected subforest generation*). Given a forest whose nodes are $(1, \ldots, n)$ when arranged in postorder, this algorithm visits all binary n-tuples $(a_1, \ldots, a_n)$ such that $a_p \geq a_q$ whenever p is a parent of q. (Thus, $a_p = 1$ means that p is a node in the current subforest.) Exactly one bit a_j changes between one visit and the next. Focus pointers $(f_0, f_1, \ldots, f_n)$ analogous to those of Algorithm L are used together with additional arrays of pointers $(l_0, l_1, \ldots, l_n)$ and $(r_0, r_1, \ldots, r_n)$, which represent a doubly linked list called the "current fringe." The current fringe contains all nodes of the current subforest and their children; r_0 points to its leftmost node and l_0 to its rightmost.

An auxiliary array $(c_0, c_1, \ldots, c_n)$ defines the forest as follows: If p has no children, $c_p = 0$; otherwise c_p is the leftmost (smallest) child of p. Also c_0 is the leftmost root of the forest itself. When the algorithm begins, we assume that $r_p = q$ and $l_q = p$ whenever p and q are consecutive children of the same family. Thus, for example, the forest in (52) has the postorder numbering



;

therefore we should have $(c_0, \ldots, c_7) = (2, 0, 1, 0, 0, 0, 4, 3)$ and $r_2 = 7$, $l_7 = 2$, $r_3 = 6$, $l_6 = 3$, $r_4 = 5$, and $l_5 = 4$ at the beginning of step K1 in this case.

K1. [Initialize.] Set $a_j \leftarrow 0$ and $f_j \leftarrow j$ for $1 \leq j \leq n$, thereby making the initial subforest empty and all nodes active. Set $f_0 \leftarrow 0$, $l_0 \leftarrow n$, $r_n \leftarrow 0$, $r_0 \leftarrow c_0$, and $l_{c_0} \leftarrow 0$, thereby putting all roots into the current fringe.

K2. [Visit.] Visit the subforest defined by $(a_1, \ldots, a_n)$.

K3. [Choose p.] Set $q \leftarrow l_0$, $p \leftarrow f_q$. (Now p is the rightmost active node of the fringe.) Also set $f_q \leftarrow q$ (thereby activating all nodes to p's right).

K4. [Check a_p.] Terminate the algorithm if $p = 0$. Otherwise go to K6 if $a_p = 1$.

K5. [Insert p's children.] Set $a_p \leftarrow 1$. Then, if $c_p \neq 0$, set $q \leftarrow r_p$, $l_q \leftarrow p - 1$, $r_{p-1} \leftarrow q$, $r_p \leftarrow c_p$, $l_{c_p} \leftarrow p$ (thereby putting p's children to the right of p in the fringe). Go to K7.

K6. [Delete p's children.] Set $a_p \leftarrow 0$. Then, if $c_p \neq 0$, set $q \leftarrow r_{p-1}$, $r_p \leftarrow q$, $l_q \leftarrow p$ (thereby removing p's children from the fringe).

K7. [Make p passive.] (At this point we know that p is active.) Set $f_p \leftarrow f_{l_p}$ and $f_{l_p} \leftarrow l_p$. Return to K2. ∎

The reader is encouraged to play through this algorithm on examples like (52), in order to understand the beautiful mechanism by which the fringe grows and shrinks at just the right times.

***Shift register sequences.** A completely different way to generate all n-tuples of m-ary digits is also possible: We can generate one digit at a time, and repeatedly work with the n most recently generated digits, thus passing from one n-tuple $(x_0, x_1, \ldots, x_{n-1})$ to another one $(x_1, \ldots, x_{n-1}, x_n)$ by shifting an appropriate new digit in at the right. For example, Fig. 15 shows how all 5-bit numbers can be obtained as blocks of 5 consecutive bits in a certain cyclic pattern of length 32. This general idea has already been discussed in some of the exercises of Sections 2.3.4.2 and 3.2.2, and we now are ready to explore it further.

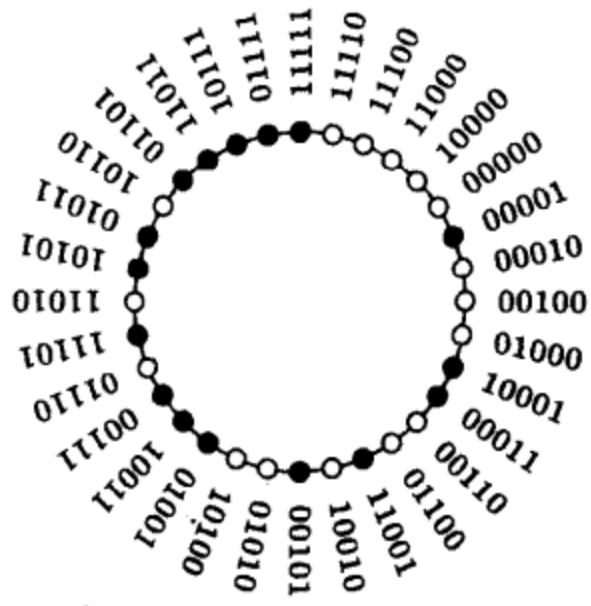


Fig. 15.
A de Bruijn cycle for 5-bit numbers.

Algorithm S (*Generic shift register generation*). This algorithm visits all n-tuples $(a_1, \ldots, a_n)$ such that $0 \le a_j < m$ for $1 \le j \le n$, provided that a suitable function f is used in step S3.

S1. [Initialize.] Set $a_j \leftarrow 0$ for $-n < j \le 0$ and $k \leftarrow 1$.

S2. [Visit.] Visit the n-tuple $(a_{k-n}, \ldots, a_{k-1})$. Terminate if $k = m^n$.

S3. [Advance.] Set $a_k \leftarrow f(a_{k-n}, \ldots, a_{k-1})$, $k \leftarrow k+1$, and return to S2. ∎

Every function f that makes Algorithm S valid corresponds to a cycle of m^n radix-m digits such that every combination of n digits occurs consecutively in the cycle. For example, the case $m = 2$ and $n = 5$ illustrated in Fig. 15 corresponds to the binary cycle

$$00000100011001010011101011011111; \tag{53}$$

and the first m^2 digits of the infinite sequence

$$00110212203132330414243 44\ldots \tag{54}$$

yield an appropriate cycle for $n = 2$ and arbitrary m. Such cycles are commonly called m-ary *de Bruijn cycles*, because N. G. de Bruijn treated the binary case for arbitrary n in *Indagationes Mathematicæ* **8** (1946), 461–467.

Exercise 2.3.4.2–23 proves that exactly $m!^{m^{n-1}}/m^n$ functions f have the required properties. That's a huge number, but only a few of those functions are known to be efficiently computable. We will discuss three kinds of f that appear to be the most useful.

Table 1

PARAMETERS FOR ALGORITHM A

3 : 1	8 : 1, 5	13 : 1, 3	18 : 7	23 : 5	28 : 3
4 : 1	9 : 4	14 : 1, 11	19 : 1, 5	24 : 1, 3	29 : 2
5 : 2	10 : 3	15 : 1	20 : 3	25 : 3	30 : 1, 15
6 : 1	11 : 2	16 : 2, 3	21 : 2	26 : 1, 7	31 : 3
7 : 1	12 : 3, 4	17 : 3	22 : 1, 7	27 : 1, 7	32 : 1, 27

The entries '$n : s$' or '$n : s, t$' mean that the polynomials $x^n + x^s + 1$ or $x^n + (x^s + 1)(x^t + 1)$ are primitive modulo 2. Additional values up to $n = 168$ have been tabulated by W. Stahnke, *Math. Comp.* **27** (1973), 977–980.

The first important case occurs when m is a prime number, and f is the almost-linear recurrence

$$f(x_1, \ldots, x_n) = \begin{cases} c_1, & \text{if } (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0); \\ 0, & \text{if } (x_1, x_2, \ldots, x_n) = (1, 0, \ldots, 0); \\ (c_1 x_1 + c_2 x_2 + \cdots + c_n x_n) \bmod m, & \text{otherwise.} \end{cases} \tag{55}$$

Here the coefficients $(c_1, \ldots, c_n)$ must be such that

$$x^n - c_n x^{n-1} - \cdots - c_2 x - c_1 \tag{56}$$

is a primitive polynomial modulo m, in the sense discussed following Eq. 3.2.2–(9). The number of such polynomials is $\varphi(m^n - 1)/n$, large enough to allow us to find one in which only a few of the c's are nonzero. [This construction goes back to a pioneering paper of Willem Mantel, *Nieuw Archief voor Wiskunde* (2) **1** (1897), 172–184.]

For example, suppose $m = 2$. We can generate binary n-tuples with a very simple loopless procedure:

Algorithm A (*Almost-linear bit-shift generation*). This algorithm visits all n-bit vectors, by using either a special offset s [Case 1] or two special offsets s and t [Case 2], as found in Table 1.

A1. [Initialize.] Set $(x_0, x_1, \ldots, x_{n-1}) \leftarrow (1, 0, \ldots, 0)$ and $k \leftarrow 0$, $j \leftarrow s$. In Case 2, also set $i \leftarrow t$ and $h \leftarrow s + t$.

A2. [Visit.] Visit the n-tuple $(x_{k-1}, \ldots, x_0, x_{n-1}, \ldots, x_{k+1}, x_k)$.

A3. [Test for end.] If $x_k \neq 0$, set $r \leftarrow 0$; otherwise set $r \leftarrow r + 1$, and go to A6 if $r = n - 1$. (We have just seen r consecutive zeros.)

A4. [Shift.] Set $k \leftarrow (k - 1) \bmod n$ and $j \leftarrow (j - 1) \bmod n$. In Case 2 also set $i \leftarrow (i - 1) \bmod n$ and $h \leftarrow (h - 1) \bmod n$.

A5. [Compute a new bit.] Set $x_k \leftarrow x_k \oplus x_j$ [Case 1] or $x_k \leftarrow x_k \oplus x_j \oplus x_i \oplus x_h$ [Case 2]. Return to A2.

A6. [Finish.] Visit $(0, \ldots, 0)$ and terminate. ∎

Appropriate offset parameters s and possibly t almost certainly exist for all n, because primitive polynomials are so abundant; for example, eight different choices of (s, t) would work when $n = 32$, and Table 1 merely lists the smallest.

However, a rigorous proof of existence in all cases lies well beyond the present state of mathematical knowledge.

Our first construction of de Bruijn cycles, in (55), was algebraic, relying for its validity on the theory of finite fields. A similar method that works when m is not a prime number appears in exercise 3.2.2–21. Our next construction, by contrast, will be purely combinatorial. In fact, it is strongly related to the idea of modular Gray m-ary codes.

Algorithm R (*Recursive de Bruijn cycle generation*). Suppose $f()$ is a coroutine that will output the successive digits of an m-ary de Bruijn cycle of length m^n, beginning with n zeros, when it is invoked repeatedly. This algorithm is a similar coroutine that outputs a cycle of length m^{n+1}, provided that $n \geq 2$. It maintains three private variables x, y, and t; variable x should initially be zero.

R1. [Output.] Output x. Go to R3 if $x \neq 0$ and $t \geq n$.

R2. [Invoke f.] Set $y \leftarrow f()$.

R3. [Count ones.] If $y = 1$, set $t \leftarrow t + 1$; otherwise set $t \leftarrow 0$.

R4. [Skip one?] If $t = n$ and $x \neq 0$, go back to R2.

R5. [Adjust x.] Set $x \leftarrow (x + y) \bmod m$ and return to R1. ∎

For example, let $m = 3$ and $n = 2$. If $f()$ produces the infinite 9-cycle

$$001102122\ 001102122\ 0\ldots, \tag{57}$$

then Algorithm R will produce the following infinite 27-cycle at step R1:

$$\begin{aligned} y &= 001021220011110212200102122\ 001\ldots \\ t &= 001001000012340010000100100\ 001\ldots \\ x &= 000110102220120020211122121\ 0001\ldots \end{aligned}$$

The proof that Algorithm R works correctly is interesting and instructive (see exercise 93). And the proof of the next algorithm, which *doubles* the window size n, is even more so (see exercise 95).

Algorithm D (*Doubly recursive de Bruijn cycle generation*). Suppose $f()$ and $f'()$ are coroutines that each will output the successive digits of an m-ary de Bruijn cycle of length m^n when invoked repeatedly, beginning with n zeros. (The two cycles are identical, but they must be generated by independent coroutines, because we will consume their values at different rates.) This algorithm is a similar coroutine that outputs a cycle of length m^{2n}. It maintains six private variables x, y, t, x', y', and t'; variables x and x' should initially be m.

The special parameter r must be set to a constant value such that

$$0 \leq r \leq m \qquad \text{and} \qquad \gcd(m^n - r,\ m^n + r) = 2. \tag{58}$$

The best choice is usually $r = 1$ when m is odd and $r = 2$ when m is even.

D1. [Possibly invoke f.] If $t \neq n$ or $x \geq r$, set $y \leftarrow f()$.

D2. [Count repeats.] If $x \neq y$, set $x \leftarrow y$ and $t \leftarrow 1$. Otherwise set $t \leftarrow t + 1$.

D3. [Output from f.] Output the current value of x.

D4. [Invoke f'.] Set $y' \leftarrow f'()$.

D5. [Count repeats.] If $x' \neq y'$, set $x' \leftarrow y'$ and $t' \leftarrow 1$. Otherwise set $t' \leftarrow t'+1$.

D6. [Possibly reject f'.] If $t' = n$ and $x' < r$ and either $t < n$ or $x' < x$, go to D4. If $t' = n$ and $x' < r$ and $x' = x$, go to D3.

D7. [Output from f'.] Output the current value of x'. Return to D3 if $t' = n$ and $x' < r$; otherwise return to D1. ∎

The basic idea of Algorithm D is to output from $f()$ and $f'()$ alternately, making special adjustments when either sequence generates n consecutive x's for $x < r$. For example, when $f()$ and $f'()$ produce the 9-cycle (57), we take $r = 1$ and get

t in step D2: 12 31211112 12312111 12123121 11121231 21111212 ...
x in step D3: 00001102122 00011021 22000110 21220001 102122000 ...
t' in step D6: 12121111212121111212121111212121111212121111212121 ...
x' in step D7: 0 11021220 11021220 11021220 11021220 11021220 1 ...;

so the 81-cycle produced in steps D3 and **D7** is **000010110122222 00001**....

The case $m = 2$ of Algorithm R was discovered by Abraham Lempel [*IEEE Trans.* **C-19** (1970), 1204–1209]; Algorithm D was not discovered until more than 25 years later [C. J. Mitchell, T. Etzion, and K. G. Paterson, *IEEE Trans.* **IT-42** (1996), 1472–1478]. By using them together, starting with simple coroutines for $n = 2$ based on (54), we can build up an interesting family of cooperating coroutines that will generate a de Bruijn cycle of length m^n for any desired $m \geq 2$ and $n \geq 2$, using only $O(\log n)$ simple computations for each digit of output. (See exercise 96.) Furthermore, in the simplest case $m = 2$, this combination "R&D method" has the property that its kth output can be computed directly, as a function of k, by doing $O(n \log n)$ simple operations on n-bit numbers. Conversely, given any n-bit pattern β, the position of β in the cycle can also be computed in $O(n \log n)$ steps. (See exercises 97–99.) No other family of binary de Bruijn cycles is presently known to have the latter property.

Our third construction of de Bruijn cycles is based on the theory of prime strings, which will be of great importance to us when we study pattern matching in Chapter 9. Suppose $\gamma = \alpha\beta$ is the concatenation of two strings; we say that α is a *prefix* of γ and β is a *suffix*. A prefix or suffix of γ is called *proper* if its length is positive but less than the length of γ. Thus β is a proper suffix of $\alpha\beta$ if and only if $\alpha \neq \epsilon$ and $\beta \neq \epsilon$.

Definition P. *A string is prime if it is nonempty and (lexicographically) less than all of its proper suffixes.*

For example, 01101 is not prime, because it is greater than 01; but 01102 is prime, because it is less than 1102, 102, 02, and 2. (We assume that strings are composed of letters, digits, or other symbols from a linearly ordered alphabet. Lexicographic or dictionary order is the normal way to compare strings, so we write $\alpha < \beta$ and say that α is less than β when α is lexicographically less than β. In particular, we always have $\alpha \leq \alpha\beta$, and $\alpha < \alpha\beta$ if and only if $\beta \neq \epsilon$.)

Prime strings have often been called *Lyndon words*, because they were introduced by R. C. Lyndon [*Trans. Amer. Math. Soc.* **77** (1954), 202–215]; Lyndon called them "standard sequences." The simpler term "prime" is justified because of the fundamental factorization theorem in exercise 101. We will, however, continue to pay respect to Lyndon implicitly by often using the letter λ to denote strings that are prime.

Several of the most important properties of prime strings were derived by Chen, Fox, and Lyndon in an important paper on group theory [*Annals of Math.* **68** (1958), 81–95], including the following easy but basic result:

Theorem P. *A nonempty string that is less than all its cyclic shifts is prime.*

(The cyclic shifts of $a_1 \dots a_n$ are $a_2 \dots a_n a_1$, $a_3 \dots a_n a_1 a_2$, $\dots$, $a_n a_1 \dots a_{n-1}$.)

Proof. Suppose $\gamma = \alpha\beta$ is not prime, because $\alpha \neq \epsilon$ and $\gamma \geq \beta \neq \epsilon$; but suppose γ is also less than its cyclic shift $\beta\alpha$. Then the conditions $\beta \leq \gamma < \beta\alpha$ imply that $\gamma = \beta\theta$ for some string $\theta < \alpha$. Therefore, if γ is also less than its cyclic shift $\theta\beta$, we have $\theta < \alpha < \alpha\beta < \theta\beta$. But that is impossible, because α and θ have the same length. ∎

Let $L_m(n)$ be the number of m-ary primes of length n. Every string $a_1 \dots a_n$, together with its cyclic shifts, yields d distinct strings for some divisor d of n, corresponding to exactly one prime of length d. For example, from 010010 we get also 100100 and 001001 by cyclic shifting, and the smallest of the periodic parts $\{010, 100, 001\}$ is the prime 001. Therefore we must have

$$\sum_{d \backslash n} d L_m(d) = m^n, \qquad \text{for all } m, n \geq 1. \tag{59}$$

This family of equations can be solved for $L_m(n)$ using exercise 4.5.3–28(a), and we obtain

$$L_m(n) = \frac{1}{n} \sum_{d \backslash n} \mu(d) m^{n/d}. \tag{60}$$

During the 1970s, Harold Fredricksen and James Maiorana discovered a beautifully simple way to generate all of the m-ary primes of length n or less, in increasing order [*Discrete Math.* **23** (1978), 207–210]. Before we are ready to understand their algorithm, we need to consider the *n-extension* of a nonempty string λ, namely the first n characters of the infinite string $\lambda\lambda\lambda\dots$. For example, the 10-extension of 123 is 1231231231. In general if $|\lambda| = k$, its n-extension is $\lambda^{\lfloor n/k \rfloor}\lambda'$, where λ' is the prefix of λ whose length is $n \bmod k$.

Definition Q. *A string is preprime if it is a nonempty prefix of a prime, on some alphabet.*

Theorem Q. *A string of length $n > 0$ is preprime if and only if it is the n-extension of a prime string λ of length $k \leq n$. This prime string is uniquely determined.*

Proof. See exercise 105. ∎

Theorem Q states, in essence, that there is a one-to-one correspondence between primes of length $\le n$ and preprimes of length n. The following algorithm generates all of the m-ary instances, in increasing order.

Algorithm F (*Prime and preprime string generation*). This algorithm visits all m-ary n-tuples $(a_1, \ldots, a_n)$ such that the string $a_1 \ldots a_n$ is preprime. It also identifies the index j such that $a_1 \ldots a_n$ is the n-extension of the prime $a_1 \ldots a_j$.

F1. [Initialize.] Set $a_1 \leftarrow \cdots \leftarrow a_n \leftarrow 0$ and $j \leftarrow 1$; also set $a_0 \leftarrow -1$.

F2. [Visit.] Visit $(a_1, \ldots, a_n)$ with index j.

F3. [Prepare to increase.] Set $j \leftarrow n$. Then if $a_j = m-1$, decrease j until finding $a_j < m-1$.

F4. [Add one.] Terminate if $j = 0$. Otherwise set $a_j \leftarrow a_j + 1$. (Now $a_1 \ldots a_j$ is prime, by exercise 105(a).)

F5. [Make n-extension.] For $k \leftarrow j+1, \ldots, n$ (in this order) set $a_k \leftarrow a_{k-j}$. Return to F2. ∎

For example, Algorithm F visits 32 ternary preprimes when $m = 3$ and $n = 4$:

$$\begin{array}{cccccccc} 0\hat{0}00 & 00\hat{1}1 & 00\hat{2}2 & 01\hat{1}1 & 01\hat{2}2 & 02\hat{1}2 & 11\hat{1}1 & 12\hat{1}2 \\ 000\hat{1} & 001\hat{2} & 010\hat{1} & 011\hat{2} & 020\hat{2} & 022\hat{0} & 111\hat{2} & 122\hat{1} \\ 000\hat{2} & 002\hat{0} & 010\hat{2} & 012\hat{0} & 021\hat{0} & 022\hat{1} & 112\hat{1} & 122\hat{2} \\ 001\hat{0} & 002\hat{1} & 011\hat{0} & 012\hat{1} & 021\hat{1} & 022\hat{2} & 112\hat{2} & 222\hat{2} \end{array} \tag{61}$$

(The digits preceding '$\hat{}$' are the prime strings 0, 0001, 0002, 001, 0011, ..., 2.)

Theorem Q explains why this algorithm is correct, because steps F3 and F4 obviously find the smallest m-ary prime of length $\le n$ that exceeds the previous preprime $a_1 \ldots a_n$. Notice that after a_1 increases from 0 to 1, the algorithm proceeds to visit all the $(m-1)$-ary primes and preprimes, increased by $1 \ldots 1$.

Algorithm F is quite beautiful, but what does it have to do with de Bruijn cycles? Here now comes the punch line: If we output the digits $a_1, \ldots, a_j$ in step F2 whenever j is a divisor of n, the sequence of all such digits forms a de Bruijn cycle! For example, in the case $m = 3$ and $n = 4$, the following 81 digits are output:

$$\begin{aligned} &0\,0001\,0002\,0011\,0012\,0021\,0022\,01\,0102\,0111\,0112 \\ &\qquad 0121\,0122\,02\,0211\,0212\,0221\,0222\,1\,1112\,1122\,12\,1222\,2. \end{aligned} \tag{62}$$

(We omit the primes 001, 002, 011, ..., 122 of (61) because their length does not divide 4.) The reasons underlying this almost magical property are explored in exercise 108. Notice that the cycle has the correct length, by (59).

There is a sense in which the outputs of this procedure are actually equivalent to the "granddaddy" of all de Bruijn cycle constructions that work for all m and n, namely the construction first published by M. H. Martin in *Bull. Amer. Math. Soc.* **40** (1934), 859–864: Martin's original cycle for $m = 3$ and $n = 4$ was $2222122202211\ldots10000$, the twos' complement of (62). In fact, Fredricksen and Maiorana discovered Algorithm F almost by accident while looking for a

simple way to generate Martin's sequence. The explicit connection between their algorithm and preprime strings was not noticed until many years later, when Ruskey, Savage, and Wang carried out a careful analysis of the running time [*J. Algorithms* **13** (1992), 414–430]. The principal results of that analysis appear in exercise 107, namely

i) The average value of $n - j$ in steps F3 and F5 is approximately $1/(m-1)$.
ii) The total running time to produce a de Bruijn cycle like (62) is $O(m^n)$.

EXERCISES

1. [*10*] Explain how to generate all n-tuples $(a_1, \ldots, a_n)$ in which $l_j \le a_j \le u_j$, given lower bounds l_j and upper bounds u_j for each coordinate. (Assume that $l_j \le u_j$.)

2. [*15*] What is the 1000000th n-tuple visited by Algorithm M if $n = 10$ and $m_j = j$ for $1 \le j \le n$? *Hint:* $\left[\begin{smallmatrix}0,&0,&1,&2,&3,&0,&2,&7,&1,&0\\1,&2,&3,&4,&5,&6,&7,&8,&9,&10\end{smallmatrix}\right] = 1000000$.

▶ **3.** [*M20*] How many times does Algorithm M perform step M4?

▶ **4.** [*18*] On most computers it is faster to count down to 0 rather than up to m. Revise Algorithm M so that it visits all n-tuples in the opposite order, starting with $(m_1 - 1, \ldots, m_n - 1)$ and finishing with $(0, \ldots, 0)$.

▶ **5.** [*20*] Algorithms such as the "fast Fourier transform" (exercise 4.6.4–14) often end with an array of answers in bit-reflected order, having $A[(b_0 \ldots b_{n-1})_2]$ in the place where $A[(b_{n-1} \ldots b_0)_2]$ is desired. What is a good way to rearrange the answers into proper order? [*Hint:* Reflect Algorithm M.]

6. [*M17*] Prove (7), the basic formula for Gray binary code.

7. [*20*] Figure 10(b) shows the Gray binary code for a disk that is divided into 16 sectors. What would be a good Gray-like code to use if the number of sectors were 12 or 60 (for hours or minutes on a clock), or 360 (for degrees in a circle)?

8. [*15*] What's an easy way to run through all n-bit strings of even parity, changing only two bits at each step?

9. [*16*] What move should follow Fig. 11, when solving the Chinese ring puzzle?

▶ **10.** [*M21*] Find a simple formula for the total number of steps A_n or B_n in which a ring is (a) removed or (b) replaced, in the shortest procedure for removing n Chinese rings. For example, $A_3 = 4$ and $B_3 = 1$.

11. [*M22*] (H. J. Purkiss, 1865.) The two smallest rings of the Chinese ring puzzle can actually be taken on or off the bar simultaneously. How many steps does the puzzle require when such accelerated moves are permitted?

▶ **12.** [*25*] The *compositions* of n are the sequences of positive integers that sum to n. For example, the compositions of 4 are 1111, 112, 121, 13, 211, 22, 31, and 4. An integer n has exactly 2^{n-1} compositions, corresponding to all subsets of the points $\{1, \ldots, n-1\}$ that might be used to break the interval $(0\,..\,n)$ into integer-sized subintervals.

a) Design a loopless algorithm to generate all compositions of n, representing each composition as a sequential array of integers $s_1 s_2 \ldots s_j$.
b) Similarly, design a loopless algorithm that represents the compositions implicitly in an array of pointers $q_0 q_1 \ldots q_t$, where the elements of the composition are $(q_0 - q_1)(q_1 - q_2) \ldots (q_{t-1} - q_t)$ and we have $q_0 = n$, $q_t = 0$. For example, the composition 211 would be represented under this scheme by the pointers $q_0 = 4$, $q_1 = 2$, $q_2 = 1$, $q_3 = 0$, and with $t = 3$.

13. [*21*] Continuing the previous exercise, compute also the multinomial coefficient $C = \binom{n}{s_1,\ldots,s_j}$ for use as the composition $s_1 \ldots s_j$ is being visited.

14. [*20*] Design an algorithm to generate all strings $a_1 \ldots a_j$ such that $0 \le j \le n$ and $0 \le a_i < m_i$ for $1 \le i \le j$, in lexicographic order. For example, if $m_1 = m_2 = n = 2$, your algorithm should successively visit ϵ, 0, 00, 01, 1, 10, 11.

▶ **15.** [*25*] Design a *loopless* algorithm to generate the strings of the previous exercise. All strings of the same length should be visited in lexicographic order as before, but strings of different lengths can be intermixed in any convenient way. For example, 0, 00, 01, ϵ, 10, 11, 1 is an acceptable order when $m_1 = m_2 = n = 2$.

16. [*23*] A loopless algorithm obviously cannot generate all binary vectors $(a_1, \ldots, a_n)$ in lexicographic order, because the number of coordinates a_j that need to change between successive visits is not bounded. Show, however, that loopless lexicographic generation does become possible if a *linked* representation is used instead of a sequential one: Suppose there are $2n + 1$ nodes $\{0, 1, \ldots, 2n\}$, each containing a LINK field. The binary n-tuple $(a_1, \ldots, a_n)$ is represented by letting

$$\begin{aligned}
&\texttt{LINK}(0) = 1 + na_1;\\
&\texttt{LINK}(j - 1 + na_{j-1}) = j + na_j, \qquad \text{for } 1 < j \le n;\\
&\texttt{LINK}(n + na_n) = 0;
\end{aligned}$$

the other n LINK fields can have any convenient values.

17. [*20*] A well-known construction called the *Karnaugh map* [M. Karnaugh, *Amer. Inst. Elect. Eng. Trans.* **72**, part I (1953), 593–599] uses Gray binary code in two dimensions to display all 4-bit numbers in a 4×4 torus:

0000	0001	0011	0010
0100	0101	0111	0110
1100	1101	1111	1110
1000	1001	1011	1010

(The entries of a torus "wrap around" at the left and right and also at the top and bottom—just as if they were tiles, replicated infinitely often in a plane.) Show that, similarly, all 6-bit numbers can be arranged in an 8×8 torus so that only one coordinate changes when we move north, south, east, or west from any point.

▶ **18.** [*20*] The *Lee weight* of a vector $u = (u_1, \ldots, u_n)$, where each component satisfies $0 \le u_j < m_j$, is defined to be

$$\nu_L(u) = \sum_{j=1}^{n} \min(u_j, m_j - u_j);$$

and the *Lee distance* between two such vectors u and v is

$$d_L(u, v) = \nu_L(u - v), \qquad \text{where } u - v = ((u_1 - v_1) \bmod m_1, \ldots, (u_n - v_n) \bmod m_n).$$

(This is the minimum number of steps needed to change u to v if we adjust some component u_j by ± 1 (modulo m_j) in each step.)

A quaternary vector has $m_j = 4$ for $1 \le j \le n$, and a binary vector has all $m_j = 2$. Find a simple one-to-one correspondence between quaternary vectors $u = (u_1, \ldots, u_n)$ and binary vectors $u' = (u'_1, \ldots, u'_{2n})$, with the property that $\nu_L(u) = \nu(u')$ and $d_L(u, v) = \nu(u' \oplus v')$.

19. [*21*] (*The octacode.*) Let $g(x) = x^3 + 2x^2 + x - 1$.

a) Use one of the algorithms in this section to evaluate $\sum z_{u_0} z_{u_1} z_{u_2} z_{u_3} z_{u_4} z_{u_5} z_{u_6} z_{u_\infty}$, summed over all 256 polynomials

$$(v_0 + v_1 x + v_2 x^2 + v_3 x^3) g(x) \bmod 4 = u_0 + u_1 x + u_2 x^2 + u_3 x^3 + u_4 x^4 + u_5 x^5 + u_6 x^6$$

for $0 \le v_0, v_1, v_2, v_3 < 4$, where u_∞ is chosen so that $0 \le u_\infty < 4$ and $(u_0 + u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_\infty) \bmod 4 = 0$.

b) Construct a set of 256 16-bit numbers that differ from each other in at least six different bit positions. (Such a set, first discovered by Nordstrom and Robinson [*Information and Control* **11** (1967), 613–616], is essentially unique.)

20. [*M36*] The 16-bit codewords in the previous exercise can be used to transmit 8 bits of information, allowing transmission errors to be corrected if any one or two bits are corrupted; furthermore, mistakes will be detected (but not necessarily correctable) if any three bits are received incorrectly. Devise an algorithm that either finds the nearest codeword to a given 16-bit number u' or determines that at least three bits of u' are erroneous. How does your algorithm decode the number $(1100100100001111)_2$? [*Hint:* Use the facts that $x^7 \equiv 1$ (modulo $g(x)$ and 4), and that every quaternary polynomial of degree < 3 is congruent to $x^j + 2x^k$ (modulo $g(x)$ and 4) for some $j, k \in \{0, 1, 2, 3, 4, 5, 6, \infty\}$, where $x^\infty = 0$.]

21. [*M30*] A t-subcube of an n-cube can be represented by a string like $**10**0*$, containing t asterisks and $n - t$ specified bits. If all 2^n binary n-tuples are written in lexicographic order, the elements belonging to such a subcube appear in $2^{t'}$ clusters of consecutive entries, where t' is the number of asterisks that lie to the left of the rightmost specified bit. (In the example given, $n = 8$, $t = 5$, and $t' = 4$.) But if the n-tuples are written in Gray binary order, the number of clusters might be reduced. For example, the $(n-1)$-subcubes $*\ldots*0$ and $*\ldots*1$ occur in only $2^{n-2}+1$ and 2^{n-2} clusters, respectively, when Gray binary order is used, not in 2^{n-1} of them.

a) Explain how to compute $C(\alpha)$, the number of Gray binary clusters of the subcube defined by a given string α of asterisks, 0s, and 1s. What is $C(**10**0*)$?

b) Prove that $C(\alpha)$ always lies between $2^{t'-1}$ and $2^{t'}$, inclusive.

c) What is the average value of $C(\alpha)$, over all $2^{n-t}\binom{n}{t}$ possible t-subcubes?

▸ **22.** [*22*] A "right subcube" is a subcube such as $0110**$ in which all the asterisks appear after all the specified digits. Any binary trie (Section 6.3) can be regarded as a way to partition a cube into disjoint right subcubes, as in Fig. 16(a). If we interchange the left and right subtries of every right subtrie, proceeding downward from the root, we obtain a *Gray binary trie*, as in Fig. 16(b).

Prove that if the "lieves" of a Gray binary trie are traversed in order, from left to right, consecutive lieves correspond to adjacent subcubes. (Subcubes are adjacent if they contain adjacent vertices. For example, $00**$ is adjacent to $011*$ because the first contains 0010 and the second contains 0110; but $011*$ is not adjacent to $10**$.)

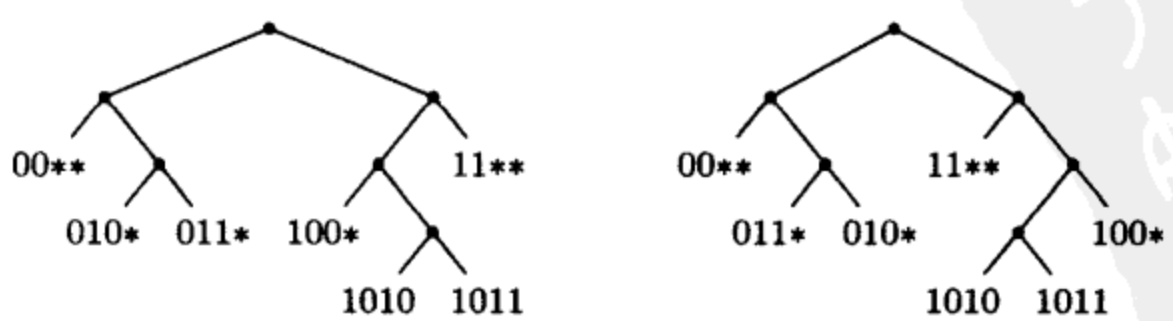


Fig. 16. (a) Normal binary trie. (b) Gray binary trie.

23. [*20*] Suppose $g(k) \oplus 2^j = g(l)$. What is a simple way to find l, given j and k?

24. [*M21*] Consider extending the Gray binary function g to all 2-adic integers (see exercise 4.1–31). What is the corresponding inverse function $g^{[-1]}$?

▸ **25.** [*M25*] Prove that if $g(k)$ and $g(l)$ differ in $t > 0$ bits, and if $0 \le k, l < 2^n$, then $\lceil 2^t/3 \rceil \le |k - l| \le 2^n - \lceil 2^t/3 \rceil$.

26. [*25*] (Frank Ruskey.) For which integers N is it possible to generate all of the nonnegative integers less than N in such a way that only one bit of the binary representation changes at each step?

▸ **27.** [*20*] Let $S_0 = \{1\}$ and $S_{n+1} = 1/(2 + S_n) \cup 1/(2 - S_n)$; thus, for example,

$$S_2 = \left\{ \frac{1}{2+\frac{1}{2+1}}, \frac{1}{2+\frac{1}{2-1}}, \frac{1}{2-\frac{1}{2+1}}, \frac{1}{2-\frac{1}{2-1}} \right\} = \left\{ \frac{3}{7}, \frac{1}{3}, \frac{3}{5}, 1 \right\},$$

and S_n has 2^n elements that lie between $\frac{1}{3}$ and 1. Compute the 10^{10}th smallest element of S_{100}.

28. [*M27*] A *median* of n-bit strings $\{\alpha_1, \ldots, \alpha_t\}$, where α_k has the binary representation $\alpha_k = a_{k(n-1)} \ldots a_{k0}$, is a string $\hat{\alpha} = a_{n-1} \ldots a_0$ whose bits a_j for $0 \le j < n$ agree with the majority of the bits a_{kj} for $1 \le k \le t$. (If t is even and the bits α_{kj} are half 0 and half 1, the median bit a_j can be either 0 or 1.) For example, the strings $\{0010, 0100, 0101, 1110\}$ have two medians, 0100 and 0110, which we can denote by 01∗0.

a) Find a simple way to describe the medians of $G_t = \{g(0), \ldots, g(t-1)\}$, the first t Gray binary strings, when $0 < t \le 2^n$.

b) Prove that if $\alpha = a_{n-1} \ldots a_0$ is such a median, and if $2^{n-1} < t < 2^n$, then the string β obtained from α by complementing any bit a_j is also an element of G_t.

29. [*M24*] If integer values k are transmitted as n-bit Gray binary codes $g(k)$ and received with errors described by a bit pattern $p = (p_{n-1} \ldots p_0)_2$, the average numerical error is

$$\frac{1}{2^n} \sum_{k=0}^{2^n-1} \Big| (g^{[-1]}(k) \oplus p) - k \Big|,$$

assuming that all values of k are equally likely. Show that this sum is equal to $\sum_{k=0}^{2^n-1} |(k \oplus p) - k|/2^n$, just as if Gray binary code were not used, and evaluate it explicitly.

▸ **30.** [*M27*] (*Gray permutation.*) Design a one-pass algorithm to replace the array elements $(X_0, X_1, X_2, \ldots, X_{2^n-1})$ by $(X_{g(0)}, X_{g(1)}, X_{g(2)}, \ldots, X_{g(2^n-1)})$, using only a constant amount of auxiliary storage. *Hint:* Considering the function $g(n)$ as a permutation of all nonnegative integers, show that the set

$$L = \{0, 1, (10)_2, (100)_2, (100*)_2, (100*0)_2, (100*0*)_2, \ldots\}$$

is the set of *cycle leaders* (the smallest elements of the cycles).

31. [*HM35*] (*Gray fields.*) Let $f_n(x) = g(r_n(x))$ denote the operation of reflecting the bits of an n-bit binary string as in exercise 5 and then converting to Gray binary code. For example, the operation $f_3(x)$ takes $(001)_2 \mapsto (110)_2 \mapsto (010)_2 \mapsto (011)_2 \mapsto (101)_2 \mapsto (111)_2 \mapsto (100)_2 \mapsto (001)_2$, hence all of the nonzero possibilities appear in

a single cycle. Therefore we can use f_3 to define a field of 8 elements, with $\oplus$ as the addition operator and with multiplication defined by the rule

$$f_3^{[j]}(1) \times f_3^{[k]}(1) = f_3^{[j+k]}(1) = f_3^{[j]}(f_3^{[k]}(1)).$$

The functions f_2, f_5, and f_6 have the same nice property. But f_4 does not, because $f_4((1011)_2) = (1011)_2$.

Find all $n \le 100$ for which f_n defines a field of 2^n elements.

32. [*M20*] True or false: Walsh functions satisfy $w_k(-x) = (-1)^k w_k(x)$.

▶ **33.** [*M20*] Prove the Rademacher-to-Walsh law (17).

34. [*M21*] The *Paley functions* $p_k(x)$ are defined by

$$p_0(x) = 1 \qquad \text{and} \qquad p_k(x) = (-1)^{\lfloor 2x \rfloor k} p_{\lfloor k/2 \rfloor}(2x).$$

Show that $p_k(x)$ has a simple expression in terms of Rademacher functions, analogous to (17), and relate Paley functions to Walsh functions.

35. [*HM23*] The $2^n \times 2^n$ Paley matrix P_n is obtained from Paley functions just as the Walsh matrix W_n is obtained from Walsh functions. (See (20).) Find interesting relations between P_n, W_n, and the Hadamard matrix H_n. Prove that all three matrices are symmetric.

36. [*21*] Spell out the details of an efficient algorithm to compute the Walsh transform $(x_0, \ldots, x_{2^n-1})$ of a given vector $(X_0, \ldots, X_{2^n-1})$.

37. [*HM23*] Let z_{kl} be the location of the lth sign change in $w_k(x)$, for $1 \le l \le k$ and $0 < z_{kl} < 1$. Prove that $|z_{kl} - l/(k+1)| = O((\log k)/k)$.

▶ **38.** [*M25*] Devise a ternary generalization of Walsh functions.

▶ **39.** [*HM30*] (J. J. Sylvester.) The rows of $\left(\begin{smallmatrix} a & b \\ b & -a \end{smallmatrix}\right)$ are orthogonal to each other and have the same magnitude; therefore the matrix identity

$$\begin{aligned}(A\ B)\begin{pmatrix} a^2+b^2 & 0 \\ 0 & a^2+b^2 \end{pmatrix}\begin{pmatrix} A \\ B \end{pmatrix} &= (A\ B)\begin{pmatrix} a & b \\ b & -a \end{pmatrix}\begin{pmatrix} a & b \\ b & -a \end{pmatrix}\begin{pmatrix} A \\ B \end{pmatrix} \\ &= (Aa+Bb\ \ Ab-Ba)\begin{pmatrix} aA+bB \\ bA-aB \end{pmatrix}\end{aligned}$$

implies the sum-of-two-squares identity $(a^2+b^2)(A^2+B^2) = (aA+bB)^2 + (bA-aB)^2$. Similarly, the matrix

$$\begin{pmatrix} a & b & c & d \\ b & -a & d & -c \\ d & c & -b & -a \\ c & -d & -a & b \end{pmatrix}$$

leads to the sum-of-four-squares identity

$$\begin{aligned}(a^2+b^2+c^2+d^2)(A^2+B^2+C^2+D^2) = {} & (aA+bB+cC+dD)^2 + (bA-aB+dC-cD)^2 \\ & + (dA+cB-bC-aD)^2 + (cA-dB-aC+bD)^2.\end{aligned}$$

a) Attach the signs of the matrix H_3 in (21) to the symbols $\{a, b, c, d, e, f, g, h\}$, obtaining a matrix with orthogonal rows and a sum-of-eight-squares identity.

b) Generalize to H_4 and higher-order matrices.

▶ **40.** [*21*] Would the text's five-letter word computation scheme produce correct answers also if the masks in step W2 were computed as $m_j = x \,\&\, (2^{5j} - 1)$ for $0 \le j < 5$?

41. [25] If we restrict the five-letter word problem to the most common 3000 words — thereby eliminating **ducky**, **duces**, **dunks**, **dinks**, **dinky**, **dices**, **dicey**, **dicky**, **dicks**, **picky**, **pinky**, **punky**, and **pucks** from (23) — how many valid words can still be generated from a single pair?

42. [35] (M. L. Fredman.) Algorithm L uses $\Theta(n \log n)$ bits of auxiliary memory for focus pointers as it decides what Gray binary bit a_j should be complemented next. On each step L3 it examines $\Theta(\log n)$ of the auxiliary bits, and it occasionally changes $\Omega(\log n)$ of them.

Show that, from a theoretical standpoint, we can do better: The n-bit Gray binary code can be generated by changing at most 2 auxiliary bits between visits. (We still allow ourselves to examine $O(\log n)$ of the auxiliary bits on each step, so that we know which of them should be changed.)

43. [47] Determine $d(6)$, the number of 6-bit Gray cycles.

44. [M37] Show that arbitrary delta sequences for Gray cycles on $n-1$ or $n-2$ bits can be used to construct a large number of delta sequences for n-bit Gray cycles with the property that exactly (a) one or (b) two of the coordinate names occur only twice.

45. [M25] Prove that the sequence $d(n)$ has doubly exponential growth: There is a constant $A > 1$ such that $d(n) = \Omega(A^{2^n})$.

46. [HM48] Determine the asymptotic behavior of $d(n)^{1/2^n}$ as $n \to \infty$.

47. [M46] (Silverman, Vickers, and Sampson.) Let $S_k = \{g(0), \ldots, g(k-1)\}$ be the first k elements of the standard Gray binary code, and let $H(k, v)$ be the number of Hamiltonian paths in S_k that begin with 0 and end with v. Prove or disprove: $H(k, v) \le H(k, g(k-1))$ for all $v \in S_k$ that are adjacent to $g(k)$.

48. [36] Prove that $d(n) \le 4(n/2)^{2^n}$ if the conjecture in the previous exercise is true. [*Hint:* Let $d(n, k)$ be the number of n-bit Gray cycles that begin with $g(0) \ldots g(k-1)$; the conjecture implies that $d(n) \le c_{n1} \ldots c_{n(k-1)} d(n, k)$, where c_{nk} is the number of vertices adjacent to $g(k-1)$ in the n-cube but not in S_k.]

49. [20] Prove that for all $n \ge 1$ there is a $2n$-bit Gray cycle in which $v_{k+2^{2n-1}}$ is the complement of v_k, for all $k \ge 0$.

▶ **50.** [21] Find a construction like that of Theorem D but with l even.

51. [M24] Complete the proof of Corollary B to Theorem D.

52. [M20] Prove that if the transition counts of an n-bit Gray cycle satisfy $c_0 \le c_1 \le \cdots \le c_{n-1}$, we must have $c_0 + \cdots + c_{j-1} \ge 2^j$, with equality when $j = n$.

53. [M46] If the numbers $(c_0, \ldots, c_{n-1})$ are even and satisfy the condition of the previous exercise, is there always an n-bit Gray cycle with these transition counts?

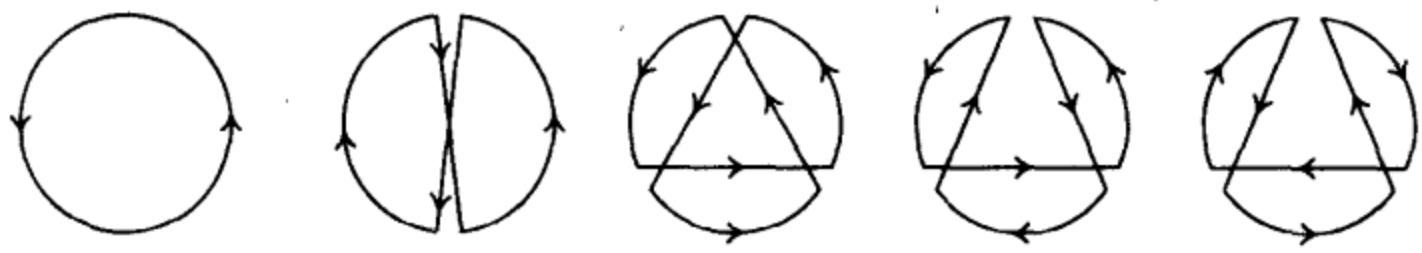
54. [M20] (H. S. Shapiro, 1953.) Show that if a sequence of integers $(a_1, \ldots, a_{2^n})$ contains only n distinct values, then there is a subsequence whose product $a_{k+1} a_{k+2} \ldots a_l$ is a perfect square, for some $0 \le k < l \le 2^n$. However, this conclusion might not be true if we disallow the case $l = 2^n$.

55. [47] (F. Ruskey and C. Savage, 1993.) If $(v_0, \ldots, v_{2^n-1})$ is an n-bit Gray cycle, the pairs $\{\{v_{2k}, v_{2k+1}\} \mid 0 \le k < 2^{n-1}\}$ form a perfect matching between the vertices of even and odd parity in the n-cube. Conversely, does every such perfect matching arise as "half" of some n-bit Gray cycle?

56. [M30] (E. N. Gilbert, 1958.) Say that two Gray cycles are equivalent if their delta sequences can be made equal by permuting the coordinate names, or by reversing the

cycle and/or starting the cycle at a different place. Show that the 2688 different 4-bit Gray cycles fall into just 9 equivalence classes.

57. [*32*] Consider a graph whose vertices are the 2688 possible 4-bit Gray cycles, where two such cycles are adjacent if they are related by one of the following simple transformations:



Before | After Type 1 | After Type 2 | After Type 3 | After Type 4

(Type 1 changes arise when the cycle can be broken into two parts and reassembled with one part reversed. Types 2, 3, and 4 arise when the cycle can be broken into three parts and reassembled after reversing 0, 1, or 2 of the parts. The parts need not have equal size. Such transformations of Hamiltonian cycles are often possible.)

Write a program to discover which 4-bit Gray cycles are transformable into each other, by finding the connected components of the graph; restrict consideration to only one of the four types at a time.

▸ **58.** [*21*] Let α be the delta sequence of an n-bit Gray cycle, and obtain β from α by changing q occurrences of 0 to n, where q is odd. Prove that $\beta\beta$ is the delta sequence of an $(n+1)$-bit Gray cycle.

59. [*22*] The 5-bit Gray cycle of (30) is *nonlocal* in the sense that no 2^t consecutive elements belong to a single t-subcube, for $1 < t < n$. Prove that nonlocal n-bit Gray cycles exist for all $n \geq 5$. [*Hint:* See the previous exercise.]

60. [*20*] Show that the run-length-bound function satisfies $r(n+1) \geq r(n)$.

61. [*M30*] Show that $r(m+n) \geq r(m) + r(n) - 1$ if (a) $m = 2$ and $2 < r(n) < 8$; or (b) $m \leq n$ and $r(n) \leq 2^{m-3}$.

62. [*46*] Does $r(8) = 6$?

63. [*30*] (Luis Goddyn.) Prove that $r(10) \geq 8$.

▸ **64.** [*HM35*] (L. Goddyn and P. Gvozdjak.) An n-bit *Gray stream* is a sequence of permutations $(\sigma_0, \sigma_1, \ldots, \sigma_{l-1})$ where each σ_k is a permutation of the vertices of the n-cube, taking every vertex to one of its neighbors.

a) Suppose $(u_0, \ldots, u_{2^m-1})$ is an m-bit Gray cycle and $(\sigma_0, \sigma_1, \ldots, \sigma_{2^m-1})$ is an n-bit Gray stream. Let $v_0 = 0 \ldots 0$ and $v_{k+1} = v_k \sigma_k$, where $\sigma_k = \sigma_{k \bmod 2^m}$ if $k \geq 2^m$. Under what conditions is the sequence

$$W = (u_0v_0,\ u_0v_1,\ u_1v_1,\ u_1v_2,\ \ldots,\ u_{2^{m+n-1}-1}v_{2^{m+n-1}-1},\ u_{2^{m+n-1}-1}v_{2^{m+n-1}})$$

an $(m+n)$-bit Gray cycle?

b) Show that if m is sufficiently large, there is an n-bit Gray stream satisfying the conditions of (a) for which all run lengths of the sequence $(v_0, v_1, \ldots)$ are $\geq n-2$.

c) Apply these results to prove that $r(n) \geq n - O(\log n)$.

65. [*30*] (Brett Stevens.) In Samuel Beckett's play *Quad*, the stage begins and ends empty; n actors enter and exit one at a time, running through all 2^n possible subsets, and the actor who leaves is always the one whose previous entrance was earliest. When $n = 4$, as in the actual play, some subsets are necessarily repeated. Show, however, that there is a perfect pattern with exactly 2^n entrances and exits when $n = 5$.

66. [*40*] Is there a perfect Beckett–Gray pattern for 8 actors?

67. [*20*] Sometimes it is desirable to run through all n-bit binary strings by changing as *many* bits as possible from one step to the next, for example when testing a physical circuit for reliable behavior in worst-case conditions. Explain how to traverse all binary n-tuples in such a way that each step changes n or $n-1$ bits, alternately.

68. [*21*] Rufus Q. Perverse decided to construct an *anti-Gray* ternary code, in which each n-trit number differs from its neighbors in *every* digit position. Is such a code possible for all n?

▶ **69.** [*M25*] Modify the definition of Gray binary code (7) by letting

$$h(k) = (\ldots(b_6 \oplus b_5)(b_5 \oplus b_4)(b_4 \oplus b_3 \oplus b_2 \oplus b_0)(b_3 \oplus b_0)(b_2 \oplus b_1 \oplus b_0)b_1)_2,$$

when $k = (\ldots b_5 b_4 b_3 b_2 b_1 b_0)_2$.

a) Show that the sequence $h(0), h(1), \ldots, h(2^n-1)$ runs through all n-bit numbers in such a way that exactly 3 bits change each time, when $n > 3$.

b) Generalize this rule to obtain sequences in which exactly t bits change at each step, when t is odd and $n > t$.

70. [*21*] How many monotonic n-bit Gray codes exist for $n = 5$ and $n = 6$?

71. [*M22*] Derive (45), the recurrence that defines the Savage–Winkler permutations.

72. [*20*] What is the Savage–Winkler code from 00000 to 11111?

▶ **73.** [*32*] Design an efficient algorithm to construct the delta sequence of an n-bit monotonic Gray code.

74. [*M25*] (Savage and Winkler.) How far apart can adjacent vertices of the n-cube be, in a monotonic Gray code?

75. [*32*] Find all 5-bit Gray paths $v_0, \ldots, v_{31}$ that are *trend-free*, in the sense that $\sum_{k=0}^{31} k(-1)^{v_{kj}} = 0$ in each coordinate position j.

76. [*M25*] Prove that trend-free n-bit Gray codes exist for all $n \geq 5$.

77. [*21*] Modify Algorithm H in order to visit mixed-radix n-tuples in *modular* Gray order.

78. [*M26*] Prove the conversion formulas (50) and (51) for reflected mixed-radix Gray codes, and derive analogous formulas for the modular case.

▶ **79.** [*M22*] When is the last n-tuple of the (a) reflected (b) modular mixed-radix Gray code adjacent to the first?

80. [*M20*] Explain how to run through all divisors of a number, given its prime factorization $p_1^{e_1} \ldots p_t^{e_t}$, repeatedly multiplying or dividing by a single prime at each step.

81. [*M21*] Let (a_0, b_0), (a_1, b_1), $\ldots$, (a_{m^2-1}, b_{m^2-1}) be the 2-digit m-ary modular Gray code. Show that, if $m > 2$, every edge (x, y) — $(x, (y+1) \bmod m)$ and (x, y) — $((x+1) \bmod m, y)$ occurs in one of the two cycles

$$(a_0, b_0) — (a_1, b_1) — \cdots — (a_{m^2-1}, b_{m^2-1}) — (a_0, b_0),$$
$$(b_0, a_0) — (b_1, a_1) — \cdots — (b_{m^2-1}, a_{m^2-1}) — (b_0, a_0).$$

▶ **82.** [*M25*] (G. Ringel, 1956.) Use the previous exercise to deduce that there exist four 8-bit Gray cycles that, together, cover all edges of the 8-cube.

83. [*41*] Can four *balanced* 8-bit Gray cycles cover all edges of the 8-cube?

▸ **84.** [*25*] (Howard L. Dyckman.) Figure 17 shows a fascinating puzzle called Loony Loop or the Gordian Knot, in which the object is to remove a flexible cord from the rigid loops that surround it. Show that the solution to this puzzle is inherently related to the reflected Gray ternary code.

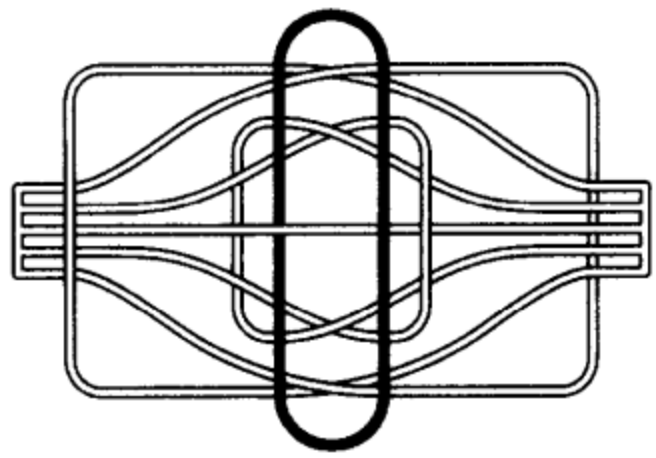


Fig. 17. The Loony Loop puzzle.

▸ **85.** [*M25*] (Dana Richards.) If $\Gamma = (\alpha_0, \ldots, \alpha_{t-1})$ is any sequence of t strings and $\Gamma' = (\alpha'_0, \ldots, \alpha'_{t'-1})$ is any sequence of t' strings, the *boustrophedon product* $\Gamma \boxtimes \Gamma'$ is the sequence of tt' strings that begins

$$(\alpha_0\alpha'_0, \ldots, \alpha_0\alpha'_{t'-1}, \alpha_1\alpha'_{t'-1}, \ldots, \alpha_1\alpha'_0, \alpha_2\alpha'_0, \ldots, \alpha_2\alpha'_{t'-1}, \alpha_3\alpha'_{t'-1}, \ldots)$$

and ends with $\alpha_{t-1}\alpha'_0$ if t is even, $\alpha_{t-1}\alpha'_{t'-1}$ if t is odd. For example, the basic definition of Gray binary code in (5) can be expressed in this notation as $\Gamma_n = (0,1) \boxtimes \Gamma_{n-1}$ when $n > 0$. Prove that the operation $\boxtimes$ is associative, hence $\Gamma_{m+n} = \Gamma_m \boxtimes \Gamma_n$.

▸ **86.** [*26*] Define an infinite Gray code that runs through all possible nonnegative integer n-tuples $(a_1, \ldots, a_n)$ in such a way that $\max(a_1, \ldots, a_n) \le \max(a'_1, \ldots, a'_n)$ when $(a_1, \ldots, a_n)$ is followed by $(a'_1, \ldots, a'_n)$.

87. [*27*] Continuing the previous exercise, define an infinite Gray code that runs through *all* integer n-tuples $(a_1, \ldots, a_n)$, in such a way that $\max(|a_1|, \ldots, |a_n|) \le \max(|a'_1|, \ldots, |a'_n|)$ when $(a_1, \ldots, a_n)$ is followed by $(a'_1, \ldots, a'_n)$.

▸ **88.** [*25*] After Algorithm K has terminated in step K4, what would happen if we immediately restarted it in step K2?

▸ **89.** [*25*] (*Gray code for Morse code.*) The Morse code words of length n (exercise 4.5.3–32) are strings of dots and dashes, where n is the number of dots plus twice the number of dashes.

a) Show that it is possible to generate all Morse code words of length n by successively changing a dash to two dots or vice versa. For example, the path for $n = 3$ must be $\bullet-$, $\bullet\bullet\bullet$, $-\bullet$ or its reverse.

b) What string follows $\bullet-\bullet\bullet-\bullet-\bullet$ in your sequence for $n = 15$?

90. [*26*] For what values of n can the Morse code words be arranged in a *cycle*, under the ground rules of exercise 89? [*Hint:* The number of code words is F_{n+1}.]

▸ **91.** [*34*] Design a loopless algorithm to visit all binary n-tuples $(a_1, \ldots, a_n)$ such that $a_1 \le a_2 \ge a_3 \le a_4 \ge \cdots$. [The number of such n-tuples is F_{n+2}.]

92. [*M30*] Is there an infinite sequence Φ_n whose first m^n elements form an m-ary de Bruijn cycle, for all m? [The case $n = 2$ is solved in (54).]

▸ **93.** [*M28*] Prove that Algorithm R outputs a de Bruijn cycle as advertised.

94. [*22*] What is the output of Algorithm D when $m = 5$, $n = 1$, and $r = 3$, if the coroutines $f()$ and $f'()$ generate the trivial cycles 01234 01234 01...?

▸ **95.** [*M23*] Suppose an infinite sequence $a_0a_1a_2\ldots$ of period p is interleaved with an infinite sequence $b_0b_1b_2\ldots$ of period q to form the infinite cyclic sequence

$$c_0c_1c_2c_3c_4c_5\ldots = a_0b_0a_1b_1a_2b_2\ldots.$$

a) Under what circumstances does $c_0c_1c_2\ldots$ have period pq? (The "period" of a sequence $a_0a_1a_2\ldots$, for the purposes of this exercise, is the smallest integer $p>0$ such that $a_k = a_{k+p}$ for all $k \ge 0$.)

b) Which $2n$-tuples would occur as consecutive outputs of Algorithm D if step D6 were changed to say simply "If $t' = n$ and $x' < r$, go to D4"?

c) Prove that Algorithm D outputs a de Bruijn cycle as advertised.

▸ **96.** [*M28*] Suppose a family of coroutines has been set up to generate a de Bruijn cycle of length m^n using Algorithms R and D, based recursively on simple coroutines like Algorithm S for the base case $n = 2$.

a) How many coroutines (R_n, D_n, S_n) of each type will there be?

b) What is the maximum number of coroutine activations needed to get one top-level digit of output?

97. [*M29*] The purpose of this exercise is to analyze the de Bruijn cycles constructed by Algorithms R and D in the important special case $m = 2$. Let $f_n(k)$ be the $(k+1)$st bit of the 2^n-cycle, so that $f_n(k) = 0$ for $0 \le k < n$. Also let j_n be the index such that $0 \le j_n < 2^n$ and $f_n(k) = 1$ for $j_n \le k < j_n + n$.

a) Write out the cycles $(f_n(0)\ldots f_n(2^n-1))$ for $n = 2$, 3, 4, and 5.

b) Prove that, for all even values of n, there is a number $\delta_n = \pm1$ such that we have

$$f_{n+1}(k) \equiv \begin{cases} \Sigma f_n(k), & \text{if } 0 < k \le j_n \text{ or } 2^n + j_n < k \le 2^{n+1}, \\ 1 + \Sigma f_n(k+\delta_n), & \text{if } j_n < k \le 2^n + j_n, \end{cases}$$

where the congruence is modulo 2. (In this formula Σf stands for the summation function $\Sigma f(k) = \sum_{j=0}^{k-1} f(j)$.) Hence $j_{n+1} = 2^n - \delta_n$ when n is even.

c) Let $(c_n(0)c_n(1)\ldots c_n(2^{2n}-5))$ be the cycle produced when the simplified version of Algorithm D in exercise 95(b) is applied to $f_n()$. Where do the $(2n-1)$-tuples 1^{2n-1} and $(01)^{n-1}0$ occur in this cycle?

d) Use the results of (c) to express $f_{2n}(k)$ in terms of $f_n()$.

e) Find a (somewhat) simple formula for j_n as a function of n.

98. [*M34*] Continuing the previous exercise, design an efficient algorithm to compute $f_n(k)$, given $n \ge 2$ and $k \ge 0$.

▸ **99.** [*M23*] Exploit the technology of the previous exercises to design an efficient algorithm that locates any given n-bit string in the cycle $(f_n(0)f_n(1)\ldots f_n(2^n-1))$.

100. [*40*] Do the de Bruijn cycles of exercise 97 provide a useful source of pseudo-random bits when n is large?

▸ **101.** [*M30*] (*Unique factorization of strings into nonincreasing primes.*)

a) Prove that if λ and λ' are prime, then $\lambda\lambda'$ is prime if $\lambda < \lambda'$.

b) Consequently every string α can be written in the form

$$\alpha = \lambda_1\lambda_2\ldots\lambda_t, \qquad \lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_t, \qquad \text{where each } \lambda_j \text{ is prime.}$$

c) In fact, only one such factorization is possible. *Hint:* Show that λ_t must be the lexicographically smallest nonempty suffix of α.

d) True or false: λ_1 is the longest prime prefix of α.

e) What are the prime factors of 31415926535897932384626433832795028841971?

102. [*HM28*] Deduce the number of m-ary primes of length n from the unique factorization theorem in the previous exercise.

103. [*M20*] Use Eq. (59) to prove Fermat's theorem that $m^p \equiv m$ (modulo p).

104. [*17*] According to formula (60), about $1/n$ of all n-letter words are prime. How many of the 5757 five-letter GraphBase words are prime? Which of them is the smallest nonprime? The largest prime?

105. [*M31*] Let α be a preprime string of length n on an infinite alphabet.

a) Show that if the final letter of α is increased, the resulting string is prime.

b) If α has been factored as in exercise 101, show that it is the n-extension of λ_1.

c) Furthermore α cannot be the n-extension of two different primes.

▸ **106.** [*M30*] By reverse-engineering Algorithm F, design an algorithm that visits all m-ary primes and preprimes in *decreasing* order.

107. [*HM30*] Analyze the running time of Algorithm F.

108. [*M35*] Let $\lambda_1 < \cdots < \lambda_t$ be the m-ary prime strings whose lengths divide n, and let $a_1 \ldots a_n$ be any m-ary string. The object of this exercise is to prove that $a_1 \ldots a_n$ appears in $\lambda_1 \ldots \lambda_t \lambda_1 \lambda_2$; hence $\lambda_1 \ldots \lambda_t$ is a de Bruijn cycle (since it has length m^n). For convenience we may assume that $m = 10$ and that strings correspond to decimal numbers; the same arguments will apply for arbitrary $m \geq 2$.

a) Show that if $a_1 \ldots a_n = \alpha\beta$ is distinct from all its cyclic shifts, and if $\beta\alpha = \lambda_k$ is prime, then $\alpha\beta$ is a substring of $\lambda_k \lambda_{k+1}$, unless $\alpha = 9^j$ for some $j \geq 1$.

b) Where does $\alpha\beta$ appear in $\lambda_1 \ldots \lambda_t$ if $\beta\alpha$ is prime and α consists of all 9s? *Hint:* Show that if $a_{n+1-l} \ldots a_n = 9^l$ in step F2 for some $l > 0$, and if j is not a divisor of n, the previous step F2 had $a_{n-l} \ldots a_n = 9^{l+1}$.

c) Now consider n-tuples of the form $(\alpha\beta)^d$, where $d > 1$ is a divisor of n and $\beta\alpha = \lambda_k$ is prime.

d) Where do 899135, 997879, 913131, 090909, 909090, and 911911 occur when $n=6$?

e) Is $\lambda_1 \ldots \lambda_t$ the lexicographically least m-ary de Bruijn cycle of length m^n?

109. [*M22*] An m-ary de Bruijn torus of size $m^2 \times m^2$ for 2×2 windows is a matrix of m-ary digits d_{ij} such that each of the m^4 submatrices

$$\begin{pmatrix} d_{ij} & d_{i(j+1)} \\ d_{(i+1)j} & d_{(i+1)(j+1)} \end{pmatrix}, \qquad 0 \leq i, j < m^2$$

is different, where subscripts wrap around modulo m^2. Thus every possible m-ary 2×2 submatrix occurs exactly once; Ian Stewart [*Game, Set, and Math* (Oxford: Blackwell, 1989), Chapter 4] has therefore called it an m-ary *ourotorus*. For example,

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

is a binary ourotorus; indeed, it is essentially the only such matrix when $m = 2$, except for shifting and/or transposition.

Consider the infinite matrix D whose entry in row $i = (\ldots a_2 a_1 a_0)_2$ and column $j = (\ldots b_2 b_1 b_0)_2$ is $d_{ij} = (\ldots c_2 c_1 c_0)_2$, where

$$c_0 = (a_0 \oplus b_0)(a_1 \oplus b_1) \oplus b_1;$$
$$c_k = (a_{2k} a_0 \oplus b_{2k}) b_0 \oplus (a_{2k+1} a_0 \oplus b_{2k+1})(b_0 \oplus 1), \quad \text{for } k > 0.$$

Show that the upper left $2^{2n} \times 2^{2n}$ submatrix of D is a 2^n-ary ourotorus for all $n \geq 0$.

110. [*M25*] Continuing the previous exercise, construct m-ary ourotoruses for all m.

111. [*20*] We can obtain the number 100 in twelve ways by inserting + and − signs into the sequence 123456789; for example, $100 = 1 + 23 - 4 + 5 + 6 + 78 - 9 = 123 - 45 - 67 + 89 = -1 + 2 - 3 + 4 + 5 + 6 + 78 + 9$.

a) What is the smallest positive integer that cannot be represented in such a way?

b) Consider also inserting signs into the 10-digit sequence 9876543210.

▸ **112.** [*25*] Continuing the previous exercise, how far can we go by inserting signs into 12345678987654321? For example, $100 = -1234 - 5 - 6 + 7898 - 7 - 6543 - 2 - 1$.

Tin tan din dan bim bam bom bo—
tan tin din dan bam bim bo bom—
tin tan dan din bim bam bom bo—
tan tin dan din bam bim bo bom—
tan dan tin bam din bo bim bom—
.... Tin tan din dan bim bam bom bo.

— DOROTHY L. SAYERS, *The Nine Tailors* (1934)

A permutation on the ten decimal digits is simply a 10 digit decimal number in which all digits are distinct. Hence all we need to do is to produce all 10 digit numbers and select only those whose digits are distinct. Isn't it wonderful how high speed computing saves us from the drudgery of thinking! We simply program $k + 1 \to k$ and examine the digits of k for undesirable equalities. This gives us the permutations in dictionary order too! On second sober thought ... we do need to think of something else.

— D. H. LEHMER (1957)

7.2.1.2. Generating all permutations. After n-tuples, the next most important item on nearly everybody's wish list for combinatorial generation is the task of visiting all *permutations* of some given set or multiset. Many different ways have been devised to solve this problem. In fact, almost as many different algorithms have been published for unsorting as for sorting! We will study the most important permutation generators in this section, beginning with a classical method that is both simple and flexible:

Algorithm L (*Lexicographic permutation generation*). Given a sequence of n elements $a_1 a_2 \dots a_n$, initially sorted so that

$$a_1 \le a_2 \le \cdots \le a_n, \tag{1}$$

this algorithm generates all permutations of $\{a_1, a_2, \dots, a_n\}$, visiting them in lexicographic order. (For example, the permutations of $\{1, 2, 2, 3\}$ are

1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221,

ordered lexicographically.) An auxiliary element a_0 is assumed to be present for convenience; a_0 must be strictly less than the largest element a_n.

L1. [Visit.] Visit the permutation $a_1 a_2 \dots a_n$.

L2. [Find j.] Set $j \leftarrow n-1$. If $a_j \geq a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate the algorithm if $j = 0$. (At this point j is the smallest subscript such that we have already visited all permutations beginning with $a_1 \ldots a_j$. Therefore the lexicographically next permutation will increase the value of a_j.)

L3. [Increase a_j.] Set $l \leftarrow n$. If $a_j \geq a_l$, decrease l by 1 repeatedly until $a_j < a_l$. Then interchange $a_j \leftrightarrow a_l$. (Since $a_{j+1} \geq \cdots \geq a_n$, element a_l is the smallest element greater than a_j that can legitimately follow $a_1 \ldots a_{j-1}$ in a permutation. Before the interchange we had $a_{j+1} \geq \cdots \geq a_{l-1} \geq a_l > a_j \geq a_{l+1} \geq \cdots \geq a_n$; after the interchange, we have $a_{j+1} \geq \cdots \geq a_{l-1} \geq a_j > a_l \geq a_{l+1} \geq \cdots \geq a_n$.)

L4. [Reverse $a_{j+1} \ldots a_n$.] Set $k \leftarrow j+1$ and $l \leftarrow n$. Then, if $k < l$, interchange $a_k \leftrightarrow a_l$, set $k \leftarrow k+1$, $l \leftarrow l-1$, and repeat until $k \geq l$. Return to L1. ∎

This algorithm goes back to Nārāyaṇa Paṇḍita in 14th-century India (see Section 7.2.1.7); it also appeared in C. F. Hindenburg's preface to *Specimen Analyticum de Lineis Curvis Secundi Ordinis* by C. F. Rüdiger (Leipzig: 1784), xlvi–xlvii, and it has been frequently rediscovered ever since. The parenthetical remarks in steps L2 and L3 explain why it works.

In general, the lexicographic successor of any combinatorial pattern $a_1 \ldots a_n$ is obtainable by a three-step procedure:

1) Find the largest j such that a_j can be increased.
2) Increase a_j by the smallest feasible amount.
3) Find the lexicographically least way to extend the new $a_1 \ldots a_j$ to a complete pattern.

Algorithm L follows this general procedure in the case of permutation generation, just as Algorithm 7.2.1.1M followed it in the case of n-tuple generation; we will see numerous further instances later, as we consider other kinds of combinatorial patterns. Notice that we have $a_{j+1} \geq \cdots \geq a_n$ at the beginning of step L4. Therefore the first permutation beginning with the current prefix $a_1 \ldots a_j$ is $a_1 \ldots a_j a_n \ldots a_{j+1}$, and step L4 produces it by doing $\lfloor (n-j)/2 \rfloor$ interchanges.

In practice, step L2 finds $j = n-1$ half of the time when the elements are distinct, because exactly $n!/2$ of the $n!$ permutations have $a_{n-1} < a_n$. Therefore Algorithm L can be speeded up by recognizing this special case, without making it significantly more complicated. (See exercise 1.) Similarly, the probability that $j \leq n-t$ is only $1/t!$ when the a's are distinct; hence the loops in steps L2–L4 usually go very fast. Exercise 6 analyzes the running time in general, showing that Algorithm L is reasonably efficient even when equal elements are present, unless some values appear much more often than others do in the multiset $\{a_1, a_2, \ldots, a_n\}$.

Adjacent interchanges. We saw in Section 7.2.1.1 that Gray codes are advantageous for generating n-tuples, and similar considerations apply when we want to generate permutations. The simplest possible change to a permutation is to interchange adjacent elements, and we know from Chapter 5 that any

permutation can be sorted into order if we make a suitable sequence of such interchanges. (For example, Algorithm 5.2.2B works in this way.) Hence we can go backward and obtain any desired permutation, by starting with all elements in order and then exchanging appropriate pairs of adjacent elements.

A natural question now arises: Is it possible to run through *all* permutations of a given multiset in such a way that only two adjacent elements change places at every step? If so, the overall program that is examining all permutations will often be simpler and faster, because it will only need to calculate the effect of an exchange instead of to reprocess an entirely new array $a_1 \ldots a_n$ each time.

Alas, when the multiset has repeated elements, we can't always find such a Gray-like sequence. For example, the six permutations of $\{1,1,2,2\}$ are connected to each other in the following way by adjacent interchanges:

$$1122 \text{ — } 1212 \begin{array}{l} \diagup \\ \diagdown \end{array} \begin{array}{c} 2112 \\ \\ 1221 \end{array} \begin{array}{l} \diagdown \\ \diagup \end{array} 2121 \text{ — } 2211; \tag{2}$$

this graph has no Hamiltonian path.

But most applications deal with permutations of *distinct* elements, and for this case there is good news: A simple algorithm makes it possible to generate all $n!$ permutations by making just $n! - 1$ adjacent interchanges. Furthermore, another such interchange returns to the starting point, so we have a Hamiltonian cycle analogous to Gray binary code.

The idea is to take such a sequence for $\{1,\ldots,n-1\}$ and to insert the number n into each permutation in all ways. For example, if $n = 4$ the sequence $(123, 132, 312, 321, 231, 213)$ leads to the columns of the array

$$\begin{array}{cccccc} 1234 & 1324 & 3124 & 3214 & 2314 & 2134 \\ 1243 & 1342 & 3142 & 3241 & 2341 & 2143 \\ 1423 & 1432 & 3412 & 3421 & 2431 & 2413 \\ 4123 & 4132 & 4312 & 4321 & 4231 & 4213 \end{array} \tag{3}$$

when 4 is inserted in all four possible positions. Now we obtain the desired sequence by reading downwards in the first column, upwards in the second, downwards in the third, ..., upwards in the last: $(1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324, 3124, 3142, \ldots, 2143, 2134)$.

In Section 5.1.1 we studied the inversions of a permutation, namely the pairs of elements (not necessarily adjacent) that are out of order. Every interchange of adjacent elements changes the total number of inversions by ± 1. In fact, when we consider the so-called inversion table $c_1 \ldots c_n$ of exercise 5.1.1–7, where c_j is the number of elements lying to the right of j that are less than j, we find that the permutations in (3) have the following inversion tables:

$$\begin{array}{cccccc} 0000 & 0010 & 0020 & 0120 & 0110 & 0100 \\ 0001 & 0011 & 0021 & 0121 & 0111 & 0101 \\ 0002 & 0012 & 0022 & 0122 & 0112 & 0102 \\ 0003 & 0013 & 0023 & 0123 & 0113 & 0103 \end{array} \tag{4}$$

And if we read these columns alternately down and up as before, we obtain precisely the reflected Gray code for mixed radices $(1,2,3,4)$, as in Eqs. (46)–(51)

of Section 7.2.1.1. The same property holds for all n, as noticed by E. W. Dijkstra [*Acta Informatica* **6** (1976), 357–359], and it leads us to the following formulation:

Algorithm P (*Plain changes*). Given a sequence $a_1 a_2 \ldots a_n$ of n distinct elements, this algorithm generates all of their permutations by repeatedly interchanging adjacent pairs. It uses an auxiliary array $c_1 c_2 \ldots c_n$, which represents inversions as described above, running through all sequences of integers such that

$$0 \le c_j < j \qquad \text{for } 1 \le j \le n. \tag{5}$$

Another array $o_1 o_2 \ldots o_n$ governs the directions by which the entries c_j change.

P1. [Initialize.] Set $c_j \leftarrow 0$ and $o_j \leftarrow 1$ for $1 \le j \le n$.

P2. [Visit.] Visit the permutation $a_1 a_2 \ldots a_n$.

P3. [Prepare for change.] Set $j \leftarrow n$ and $s \leftarrow 0$. (The following steps determine the coordinate j for which c_j is about to change, preserving (5); variable s is the number of indices $k > j$ such that $c_k = k - 1$.)

P4. [Ready to change?] Set $q \leftarrow c_j + o_j$. If $q < 0$, go to P7; if $q = j$, go to P6.

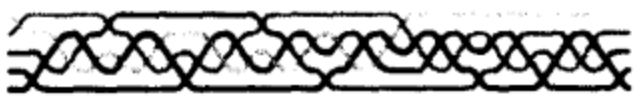
P5. [Change.] Interchange $a_{j-c_j+s} \leftrightarrow a_{j-q+s}$. Then set $c_j \leftarrow q$ and return to P2.

P6. [Increase s.] Terminate if $j = 1$; otherwise set $s \leftarrow s + 1$.

P7. [Switch direction.] Set $o_j \leftarrow -o_j$, $j \leftarrow j - 1$, and go back to P4. ∎

This procedure, which clearly works for all $n \ge 1$, originated in 17th-century England, when bell ringers began the delightful custom of ringing a set of bells in all possible permutations. They called Algorithm P the method of *plain changes*. Figure 18(a) illustrates the "Cambridge Forty-Eight," an irregular and ad hoc sequence of 48 permutations on 5 bells that had been used in the early 1600s, before the plain-change principle revealed how to achieve all $5! = 120$ possibilities. The venerable history of Algorithm P has been traced to a manuscript by Peter Mundy now in the Bodleian Library, written about 1653 and transcribed by Ernest Morris in *The History and Art of Change Ringing* (1931), 29–30. Shortly afterwards, a famous book called *Tintinnalogia*, published anonymously in 1668 but now known to have been written by Richard Duckworth and Fabian Stedman, devoted its first 60 pages to a detailed description of plain changes, working up from $n = 3$ to the case of arbitrarily large n.

> Cambridge Forty-eight, *for many years,*
> *was the greatest* Peal *that was* Rang *or invented; but now,*
> *neither* Forty-eight, *nor a* Hundred, *nor* Seven-hundred and twenty,
> *nor any Number can confine us; for we can* Ring Changes, Ad infinitum.
> ... *On four Bells, there are* Twenty four several Changes,
> *in* Ringing *of which, there is one Bell called the* Hunt,
> *and the other three are* Extream *Bells;*
> *the* Hunt *moves, and* hunts *up and down continually ...;*
> *two of the* Extream *Bells makes a* Change
> *every time the* Hunt *comes before or behind them.*
>
> — DUCKWORTH and STEDMAN, *Tintinnalogia* (1668)



(incomplete)

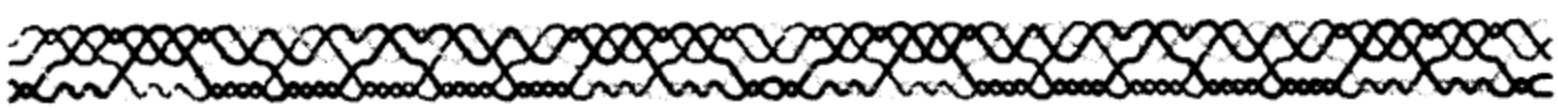
(a) The Cambridge Forty-Eight.



(b) Plain Changes.



(c) Grandsire Doubles.



(d) Stedman Doubles.

Fig. 18. Four patterns that were used in 17th-century England to ring permutations of five different church-bells. Pattern (b) corresponds to Algorithm P.

British bellringing enthusiasts soon went on to develop more complicated schemes in which two or more pairs of bells change places simultaneously. For example, they devised the pattern in Fig. 18(c) known as Grandsire Doubles, "the best and most ingenious Peal that ever was composed, to be rang on five bells" [*Tintinnalogia*, page 95]. Such fancier methods are more interesting than Algorithm P from a musical standpoint, but they are less useful in computer applications, so we shall not dwell on them here. Interested readers can learn more by reading W. G. Wilson's book, *Change Ringing* (1965); see also A. T. White, *AMM* **103** (1996), 771–778.

H. F. Trotter published the first computer implementation of plain changes in *CACM* **5** (1962), 434–435. The algorithm is quite efficient, especially when it is streamlined as in exercise 16, because $n-1$ out of every n permutations are generated without using steps P6 and P7. By contrast, Algorithm L enjoys its best case only about half of the time.

The fact that Algorithm P does exactly one interchange per visit means that the permutations it generates are alternately even and odd (see exercise 5.1.1–13). Therefore we can generate all the even permutations by simply bypassing the odd ones. In fact, the c and o tables make it easy to keep track of the current total number of inversions, $c_1 + \cdots + c_n$, as we go.

Many programs need to generate the same permutations repeatedly, and in such cases we needn't run through the steps of Algorithm P each time. We can simply prepare a list of suitable transitions, using the following method:

Algorithm T (*Plain change transitions*). This algorithm computes a table $t[1]$, $t[2]$, ..., $t[n!-1]$ such that the actions of Algorithm P are equivalent to the successive interchanges $a_{t[k]} \leftrightarrow a_{t[k]+1}$ for $1 \le k < n!$. We assume that $n \ge 2$.

T1. [Initialize.] Set $N \leftarrow n!$, $d \leftarrow N/2$, $t[d] \leftarrow 1$, and $m \leftarrow 2$.

T2. [Loop on m.] Terminate if $m = n$. Otherwise set $m \leftarrow m+1$, $d \leftarrow d/m$, and $k \leftarrow 0$. (We maintain the condition $d = n!/m!$.)

T3. [Hunt down.] Set $k \leftarrow k+d$ and $j \leftarrow m-1$. Then while $j > 0$, set $t[k] \leftarrow j$, $j \leftarrow j-1$, and $k \leftarrow k+d$, until $j = 0$.

T4. [Offset.] Set $t[k] \leftarrow t[k]+1$ and $k \leftarrow k+d$.

T5. [Hunt up.] While $j < m-1$, set $j \leftarrow j+1$, $t[k] \leftarrow j$, and $k \leftarrow k+d$. Return to T3 if $k < N$, otherwise return to T2. ∎

For example, if $n = 4$ we get the table $(t[1], t[2], \ldots, t[23]) = (3,2,1,3,1,2,3,1,3,2,1,3,1,2,3,1,3,2,1,3,1,2,3)$.

Alphametics. Now let's consider a simple kind of puzzle in which permutations are useful: How can the pattern

$$\begin{array}{r} \texttt{SEND} \\ +\,\texttt{MORE} \\ \hline \texttt{MONEY} \end{array} \tag{6}$$

represent a correct sum, if every letter stands for a different decimal digit? [H. E. Dudeney, *Strand* **68** (1924), 97, 214.] Such puzzles are often called "alphametics," a word coined by J. A. H. Hunter [*Globe and Mail* (Toronto: 27 October 1955), 27]; another term, "cryptarithm," has also been suggested by S. Vatriquant [*Sphinx* **1** (May 1931), 50].

The classic alphametic (6) can easily be solved by hand (see exercise 21). But let's suppose we want to deal with a large set of complicated alphametics, some of which may be unsolvable while others may have dozens of solutions. Then we can save time by programming a computer to try out all permutations of digits that match a given pattern, seeing which permutations yield a correct sum. [A computer program for solving alphametics was published by John Beidler in *Creative Computing* **4**, 6 (November–December 1978), 110–113.]

We might as well raise our sights slightly and consider additive alphametics in general, dealing not only with simple sums like (6) but also with examples like

$$\texttt{VIOLIN} + \texttt{VIOLIN} + \texttt{VIOLA} \;=\; \texttt{TRIO} + \texttt{SONATA}.$$

Equivalently, we want to solve puzzles such as

$$2(\texttt{VIOLIN}) + \texttt{VIOLA} - \texttt{TRIO} - \texttt{SONATA} \;=\; 0, \tag{7}$$

where a sum of terms with integer coefficients is given and the goal is to obtain zero by substituting distinct decimal digits for the different letters. Each letter in such a problem has a "signature" obtained by substituting 1 for that letter and 0 for the others; for example, the signature for I in (7) is

$$2(010010) + 01000 - 0010 - 000000,$$

namely 21010. If we arbitrarily assign the codes $(1, 2, \ldots, 10)$ to the letters $(\texttt{V}, \texttt{I}, \texttt{O}, \texttt{L}, \texttt{N}, \texttt{A}, \texttt{T}, \texttt{R}, \texttt{S}, \texttt{X})$, the respective signatures corresponding to (7) are

$$\begin{aligned} &s_1 = 210000, \quad s_2 = 21010, \quad s_3 = -7901, \quad s_4 = 210, \quad s_5 = -998, \\ &s_6 = -100, \quad s_7 = -1010, \quad s_8 = -100, \quad s_9 = -100000, \quad s_{10} = 0. \end{aligned} \tag{8}$$

(An additional letter, X, has been added because we need ten of them.) The problem now is to find all permutations $a_1 \ldots a_{10}$ of $\{0, 1, \ldots, 9\}$ such that

$$a \cdot s = \sum_{j=1}^{10} a_j s_j = 0. \tag{9}$$

There also is a side condition, because the numbers in alphametics should not have zero as a leading digit. For example, the sums

$$\begin{array}{r} 7316 \\ +\,0823 \\ \hline 08139 \end{array} \quad \text{and} \quad \begin{array}{r} 5731 \\ +\,0647 \\ \hline 06378 \end{array} \quad \text{and} \quad \begin{array}{r} 6524 \\ +\,0735 \\ \hline 07259 \end{array} \quad \text{and} \quad \begin{array}{r} 2817 \\ +\,0368 \\ \hline 03185 \end{array}$$

and numerous others are *not* considered to be valid solutions of (6). In general there is a set F of first letters such that we must have

$$a_j \neq 0 \qquad \text{for all } j \in F; \tag{10}$$

the set F corresponding to (7) and (8) is $\{1, 7, 9\}$.

One way to tackle a family of additive alphametics is to start by using Algorithm T to prepare a table of $10! - 1$ transitions $t[k]$. Then, for each problem defined by a signature sequence $(s_1, \ldots, s_{10})$ and a first-letter set F, we can exhaustively look for solutions as follows:

A1. [Initialize.] Set $a_1 a_2 \ldots a_{10} \leftarrow 01 \ldots 9$, $v \leftarrow \sum_{j=1}^{10} (j-1)s_j$, $k \leftarrow 1$, and $\delta_j \leftarrow s_{j+1} - s_j$ for $1 \le j < 10$.

A2. [Test.] If $v = 0$ and if (10) holds, output the solution $a_1 \ldots a_{10}$.

A3. [Swap.] Stop if $k = 10!$. Otherwise set $j \leftarrow t[k]$, $v \leftarrow v - (a_{j+1} - a_j)\delta_j$, $a_{j+1} \leftrightarrow a_j$, $k \leftarrow k + 1$, and return to A2. ∎

Step A3 is justified by the fact that swapping a_j with a_{j+1} simply decreases $a \cdot s$ by $(a_{j+1} - a_j)(s_{j+1} - s_j)$. Even though 10! is 3,628,800, a fairly large number, the operations in step A3 are so simple that the whole job takes only a fraction of a second on a modern computer.

An alphametic is said to be *pure* if it has a unique solution. Unfortunately (7) is not pure; the permutations 1764802539 and 3546281970 both solve (9) and (10), hence we have both

$$176478 + 176478 + 17640 = 2576 + 368020$$

and

$$354652 + 354652 + 35468 = 1954 + 742818.$$

Furthermore $s_6 = s_8$ in (8), so we can obtain two more solutions by interchanging the digits assigned to A and R.

On the other hand (6) *is* pure, yet the method we have described will find two different permutations that solve it. The reason is that (6) involves only eight distinct letters, hence we will set it up for solution by using two dummy signatures $s_9 = s_{10} = 0$. In general, an alphametic with m distinct letters will have $10 - m$ dummy signatures $s_{m+1} = \cdots = s_{10} = 0$, and each of its solutions will be found $(10 - m)!$ times unless we insist that, say, $a_{m+1} < \cdots < a_{10}$.

A general framework. A great many algorithms have been proposed for generating permutations of distinct objects, and the best way to understand them is to apply the multiplicative properties of permutations that we studied in Section 1.3.3. For this purpose we will change our notation slightly, by using 0-origin indexing and writing $a_0 a_1 \dots a_{n-1}$ for permutations of $\{0, 1, \dots, n-1\}$ instead of writing $a_1 a_2 \dots a_n$ for permutations of $\{1, 2, \dots, n\}$. More importantly, we will consider schemes for generating permutations in which most of the action takes place at the *left*, so that all permutations of $\{0, 1, \dots, k-1\}$ will be generated during the first $k!$ steps, for $1 \le k \le n$. For example, one such scheme for $n = 4$ is

$$\begin{aligned} &0123, 1023, 0213, 2013, 1203, 2103, 0132, 1032, 0312, 3012, 1302, 3102, \\ &0231, 2031, 0321, 3021, 2301, 3201, 1230, 2130, 1320, 3120, 2310, 3210; \end{aligned} \tag{11}$$

this is called "reverse colex order," because if we reflect the strings from right to left we get 3210, 3201, 3120, ..., 0123, the reverse of lexicographic order. Another way to think of (11) is to view the entries as $(n-a_n)\dots(n-a_2)(n-a_1)$, where $a_1 a_2 \dots a_n$ runs lexicographically through the permutations of $\{1, 2, \dots, n\}$.

Let's recall from Section 1.3.3 that a permutation like $\alpha = 250143$ can be written either in the two-line form

$$\alpha = \begin{pmatrix} 012345 \\ 250143 \end{pmatrix}$$

or in the more compact cycle form

$$\alpha = (0\ 2)(1\ 5\ 3),$$

with the meaning that α takes $0 \mapsto 2$, $1 \mapsto 5$, $2 \mapsto 0$, $3 \mapsto 1$, $4 \mapsto 4$, and $5 \mapsto 3$; a 1-cycle like '(4)' need not be indicated. Since 4 is a fixed point of this permutation we say that "α fixes 4." We also write $0\alpha = 2$, $1\alpha = 5$, and so on, saying that $j\alpha$ is "the image of j under α." Multiplication of permutations, like α times β where $\beta = 543210$, is readily carried out either in the two-line form

$$\alpha\beta = \begin{pmatrix} 012345 \\ 250143 \end{pmatrix}\begin{pmatrix} 012345 \\ 543210 \end{pmatrix} = \begin{pmatrix} 012345 \\ 250143 \end{pmatrix}\begin{pmatrix} 250143 \\ 305412 \end{pmatrix} = \begin{pmatrix} 012345 \\ 305412 \end{pmatrix}$$

or in the cycle form

$$\alpha\beta = (0\ 2)(1\ 5\ 3)\ \cdot\ (0\ 5)(1\ 4)(2\ 3)\ =\ (0\ 3\ 4\ 1)(2\ 5).$$

Notice that the image of 1 under $\alpha\beta$ is $1(\alpha\beta) = (1\alpha)\beta = 5\beta = 0$, etc. *Warning:* About half of all books that deal with permutations multiply them the other way (from right to left), imagining that $\alpha\beta$ means that β should be applied before α. The reason is that traditional functional notation, in which one writes $\alpha(1) = 5$, makes it natural to think that $\alpha\beta(1)$ should mean $\alpha(\beta(1)) = \alpha(4) = 4$. However, the present book subscribes to the other philosophy, and we shall always multiply permutations from left to right.

The order of multiplication needs to be understood carefully when permutations are represented by arrays of numbers. For example, if we "apply" the reflection $\beta = 543210$ to the permutation $\alpha = 250143$, the result 341052 is not $\alpha\beta$

but $\beta\alpha$. In general, the operation of replacing a permutation $\alpha = a_0 a_1 \dots a_{n-1}$ by some rearrangement $a_{0\beta} a_{1\beta} \dots a_{(n-1)\beta}$ takes $k \mapsto a_{k\beta} = k\beta\alpha$. Permuting the *positions* by β corresponds to *premultiplication* by β, changing α to $\beta\alpha$; permuting the *values* by β corresponds to *postmultiplication* by β, changing α to $\alpha\beta$. Thus, for example, a permutation generator that interchanges $a_1 \leftrightarrow a_2$ is premultiplying the current permutation by (1 2), postmultiplying it by $(a_1\ a_2)$.

Following a proposal made by Évariste Galois in 1830, a nonempty set G of permutations is said to form a *group* if it is closed under multiplication, that is, if the product $\alpha\beta$ is in G whenever α and β are elements of G [see *Écrits et Mémoires Mathématiques d'Évariste Galois* (Paris: 1962), 47]. Consider, for example, the 4-cube represented as a 4×4 torus

$$\begin{array}{|cccc|}\hline 0 & 1 & 3 & 2 \\ 4 & 5 & 7 & 6 \\ c & d & f & e \\ 8 & 9 & b & a \\ \hline\end{array} \tag{12}$$

as in exercise 7.2.1.1–17, and let G be the set of all permutations of the vertices $\{0, 1, \dots, \mathtt{f}\}$ that preserve adjacency: A permutation α is in G if and only if u — v implies $u\alpha$ — $v\alpha$ in the 4-cube. (Here we are using hexadecimal digits $(0, 1, \dots, \mathtt{f})$ to stand for the integers $(0, 1, \dots, 15)$. The labels in (12) are chosen so that u — v if and only if u and v differ in only one bit position.) This set G is obviously a group, and its elements are called the symmetries or "automorphisms" of the 4-cube.

Groups of permutations G are conveniently represented inside a computer by means of a *Sims table*, introduced by Charles C. Sims [*Computational Methods in Abstract Algebra* (Oxford: Pergamon, 1970), 169–183], which is a family of subsets $S_1, S_2, \dots$ of G having the following property: S_k contains exactly one permutation σ_{kj} that takes $k \mapsto j$ and fixes the values of all elements greater than k, whenever G contains such a permutation. We let σ_{kk} be the identity permutation, which is always present in G; but when $0 \le j < k$, any suitable permutation can be selected to play the role of σ_{kj}. The main advantage of a Sims table is that it provides a convenient representation of the entire group:

Lemma S. *Let $S_1, S_2, \dots, S_{n-1}$ be a Sims table for a group G of permutations on $\{0, 1, \dots, n-1\}$. Then every element α of G has a unique representation*

$$\alpha = \sigma_1 \sigma_2 \dots \sigma_{n-1}, \qquad \text{where } \sigma_k \in S_k \text{ for } 1 \le k < n. \tag{13}$$

Proof. If α has such a representation and if σ_{n-1} is the permutation $\sigma_{(n-1)j} \in S_{n-1}$, then α takes $n-1 \mapsto j$, because all elements of $S_1 \cup \dots \cup S_{n-2}$ fix the value of $n-1$. Conversely, if α takes $n-1 \mapsto j$ we have $\alpha = \alpha' \sigma_{(n-1)j}$, where

$$\alpha' = \alpha \sigma^{-}_{(n-1)j}$$

is a permutation of G that fixes $n-1$. (As in Section 1.3.3, σ^- denotes the inverse of σ.) The set G' of all such permutations is a group, and $S_1, \dots, S_{n-2}$ is a Sims table for G'; therefore the result follows by induction on n. ∎

For example, a bit of calculation shows that one possible Sims table for the automorphism group of the 4-cube is

$$\begin{aligned}
S_{\mathtt{f}} &= \{(),\ (\mathtt{01})(\mathtt{23})(\mathtt{45})(\mathtt{67})(\mathtt{89})(\mathtt{ab})(\mathtt{cd})(\mathtt{ef}),\ \ldots,\\
&\qquad\qquad (\mathtt{0f})(\mathtt{1e})(\mathtt{2d})(\mathtt{3c})(\mathtt{4b})(\mathtt{5a})(\mathtt{69})(\mathtt{78})\};\\
S_{\mathtt{e}} &= \{(),\ (\mathtt{12})(\mathtt{56})(\mathtt{9a})(\mathtt{de}),\ (\mathtt{14})(\mathtt{36})(\mathtt{9c})(\mathtt{be}),\ (\mathtt{18})(\mathtt{3a})(\mathtt{5c})(\mathtt{7e})\};\\
S_{\mathtt{d}} &= \{(),\ (\mathtt{24})(\mathtt{35})(\mathtt{ac})(\mathtt{bd}),\ (\mathtt{28})(\mathtt{39})(\mathtt{6c})(\mathtt{7d})\};\\
S_{\mathtt{c}} &= \{()\};\\
S_{\mathtt{b}} &= \{(),\ (\mathtt{48})(\mathtt{59})(\mathtt{6a})(\mathtt{7b})\};\\
S_{\mathtt{a}} &= S_{\mathtt{9}} = \cdots = S_{\mathtt{1}} = \{()\};
\end{aligned} \tag{14}$$

here $S_{\mathtt{f}}$ contains 16 permutations $\sigma_{\mathtt{f}j}$ for $0 \le j \le 15$, which respectively take $i \mapsto i \oplus (15 - j)$ for $0 \le i \le 15$. The set $S_{\mathtt{e}}$ contains only four permutations, because an automorphism that fixes $\mathtt{f}$ must take $\mathtt{e}$ into a neighbor of $\mathtt{f}$; thus the image of $\mathtt{e}$ must be either $\mathtt{e}$ or $\mathtt{d}$ or $\mathtt{b}$ or 7. The set $S_{\mathtt{c}}$ contains only the identity permutation, because an automorphism that fixes $\mathtt{f}$, $\mathtt{e}$, and $\mathtt{d}$ must also fix $\mathtt{c}$. Most groups have $S_k = \{()\}$ for all small values of k, as in this example; hence a Sims table usually needs to contain only a fairly small number of permutations although the group itself might be quite large.

The Sims representation (13) makes it easy to test if a given permutation α lies in G: First we determine $\sigma_{n-1} = \sigma_{(n-1)j}$, where α takes $n-1 \mapsto j$, and we let $\alpha' = \alpha\sigma_{n-1}^{-}$; then we determine $\sigma_{n-2} = \sigma_{(n-2)j'}$, where α' takes $n-2 \mapsto j'$, and we let $\alpha'' = \alpha'\sigma_{n-2}^{-}$; and so on. If at any stage the required σ_{kj} does not exist in S_k, the original permutation α does not belong to G. In the case of (14), this process must reduce α to the identity after finding $\sigma_{\mathtt{f}}$, $\sigma_{\mathtt{e}}$, $\sigma_{\mathtt{d}}$, $\sigma_{\mathtt{c}}$, and $\sigma_{\mathtt{b}}$.

For example, let α be the permutation $(\mathtt{14})(\mathtt{28})(\mathtt{3c})(\mathtt{69})(\mathtt{7d})(\mathtt{be})$, which corresponds to transposing (12) about its main diagonal $\{\mathtt{0}, \mathtt{5}, \mathtt{f}, \mathtt{a}\}$. Since α fixes $\mathtt{f}$, $\sigma_{\mathtt{f}}$ will be the identity permutation $()$, and $\alpha' = \alpha$. Then $\sigma_{\mathtt{e}}$ is the member of $S_{\mathtt{e}}$ that takes $\mathtt{e} \mapsto \mathtt{b}$, namely $(\mathtt{14})(\mathtt{36})(\mathtt{9c})(\mathtt{be})$, and we find $\alpha'' = (\mathtt{28})(\mathtt{39})(\mathtt{6c})(\mathtt{7d})$. This permutation belongs to $S_{\mathtt{d}}$, so α is indeed an automorphism of the 4-cube.

Conversely, (13) also makes it easy to generate all elements of the corresponding group. We simply run through all permutations of the form

$$\sigma(1, c_1)\sigma(2, c_2)\ldots\sigma(n-1, c_{n-1}),$$

where $\sigma(k, c_k)$ is the $(c_k + 1)$st element of S_k for $0 \le c_k < s_k = |S_k|$ and $1 \le k < n$, using any algorithm of Section 7.2.1.1 that runs through all $(n-1)$-tuples $(c_1, \ldots, c_{n-1})$ for the respective radices $(s_1, \ldots, s_{n-1})$.

Using the general framework. Our chief concern is the group of *all* permutations on $\{0, 1, \ldots, n-1\}$, and in this case every set S_k of a Sims table will contain $k+1$ elements $\{\sigma(k,0), \sigma(k,1), \ldots, \sigma(k,k)\}$, where $\sigma(k,0)$ is the identity and the others take k to the values $\{0, \ldots, k-1\}$ in some order. (The permutation $\sigma(k,j)$ need not be the same as σ_{kj}, and it usually is different.) Every such Sims table leads to a permutation generator, according to the following outline:

Algorithm G (*General permutation generator*). Given a Sims table $(S_1, S_2, \ldots, S_{n-1})$ where each S_k has $k+1$ elements $\sigma(k,j)$ as just described, this algorithm generates all permutations $a_0 a_1 \ldots a_{n-1}$ of $\{0, 1, \ldots, n-1\}$, using an auxiliary control table $c_n \ldots c_2 c_1$.

G1. [Initialize.] Set $a_j \leftarrow j$ and $c_{j+1} \leftarrow 0$ for $0 \le j < n$.

G2. [Visit.] (At this point the mixed-radix number $\left[\begin{smallmatrix} c_{n-1}, & \ldots, & c_2, & c_1 \\ n, & \ldots, & 3, & 2 \end{smallmatrix}\right]$ is the number of permutations visited so far.) Visit the permutation $a_0 a_1 \ldots a_{n-1}$.

G3. [Add 1 to $c_n \ldots c_2 c_1$.] Set $k \leftarrow 1$. If $c_k = k$, set $c_k \leftarrow 0$, $k \leftarrow k+1$, and repeat until $c_k < k$. Terminate the algorithm if $k = n$; otherwise set $c_k \leftarrow c_k + 1$.

G4. [Permute.] Apply the permutation $\tau(k, c_k)\omega(k-1)^-$ to $a_0 a_1 \ldots a_{n-1}$, as explained below, and return to G2. ∎

Applying a permutation π to $a_0 a_1 \ldots a_{n-1}$ means replacing a_j by $a_{j\pi}$ for $0 \le j < n$; this corresponds to premultiplication by π as explained earlier. Let us define

$$\tau(k,j) = \sigma(k,j)\sigma(k,j-1)^- \qquad \text{for } 1 \le j \le k; \tag{15}$$

$$\omega(k) = \sigma(1,1)\ldots\sigma(k,k). \tag{16}$$

Then steps G3 and G4 maintain the property that

$$a_0 a_1 \ldots a_{n-1} \text{ is the permutation } \sigma(1,c_1)\sigma(2,c_2)\ldots\sigma(n-1,c_{n-1}), \tag{17}$$

and Lemma S proves that every permutation is visited exactly once.

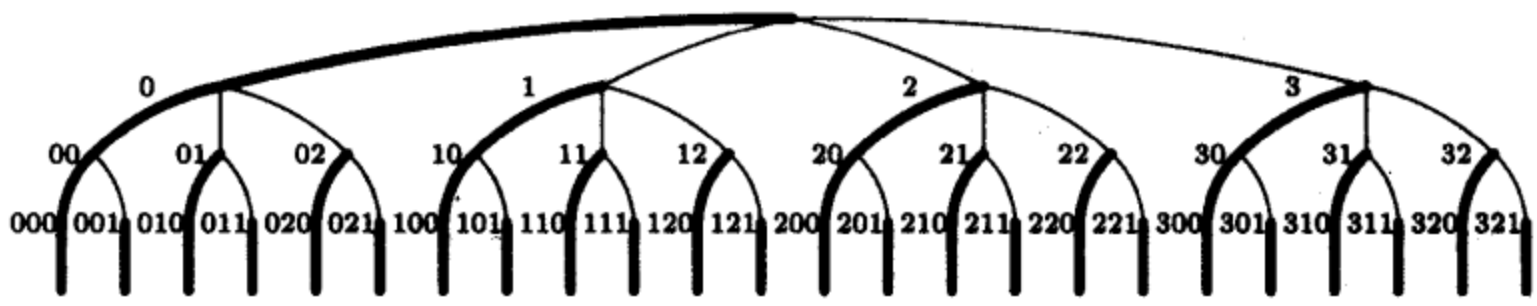


Fig. 19. Algorithm G implicitly traverses this tree when $n = 4$.

The tree in Fig. 19 illustrates Algorithm G in the case $n = 4$. According to (17), every permutation $a_0 a_1 a_2 a_3$ of $\{0,1,2,3\}$ corresponds to a three-digit control string $c_3 c_2 c_1$, with $0 \le c_3 \le 3$, $0 \le c_2 \le 2$, and $0 \le c_1 \le 1$. Some nodes of the tree are labeled by a single digit c_3; these correspond to the permutations $\sigma(3, c_3)$ of the Sims table being used. Other nodes, labeled with two digits $c_3 c_2$, correspond to the permutations $\sigma(2, c_2)\sigma(3, c_3)$. A heavy line connects node c_3 to node $c_3 0$ and node $c_3 c_2$ to node $c_3 c_2 0$, because $\sigma(2,0)$ and $\sigma(1,0)$ are the identity permutation and these nodes are essentially equivalent. Adding 1 to the mixed-radix number $c_3 c_2 c_1$ in step G3 corresponds to moving from one node of Fig. 19 to its successor in preorder, and the transformation in step G4 changes the permutations accordingly. For example, when $c_3 c_2 c_1$ changes from 121 to 200, step G4 premultiplies the current permutation by

$$\tau(3,2)\omega(2)^- = \tau(3,2)\sigma(2,2)^-\sigma(1,1)^-;$$

premultiplying by $\sigma(1,1)^-$ takes us from node 121 to node 12, premultiplying by $\sigma(2,2)^-$ takes us from node 12 to node 1, and premultiplying by $\tau(3,2) = \sigma(3,2)\sigma(3,1)^-$ takes us from node 1 to node $2 \equiv 200$, which is the preorder successor of node 121. Stating this another way, premultiplication by $\tau(3,2)\omega(2)^-$ is exactly what is needed to change $\sigma(1,1)\sigma(2,2)\sigma(3,1)$ to $\sigma(1,0)\sigma(2,0)\sigma(3,2)$, preserving (17).

Algorithm G defines a huge number of permutation generators (see exercise 37), so it is no wonder that many of its special cases have appeared in the literature. Of course some of its variants are much more efficient than others, and we want to find examples where the operations are particularly well suited to the computer we are using.

We can, for instance, obtain permutations in reverse colex order as a special case of Algorithm G (see (11)), by letting $\sigma(k,j)$ be the $(j+1)$-cycle

$$\sigma(k,j) \;=\; (k{-}j \;\; k{-}j{+}1 \;\; \ldots \;\; k). \tag{18}$$

The reason is that $\sigma(k,j)$ should be the permutation that corresponds to $c_n \ldots c_1$ in reverse colex order when $c_k = j$ and $c_i = 0$ for $i \neq k$, and this permutation $a_0 a_1 \ldots a_{n-1}$ is $01\ldots(k{-}j{-}1)(k{-}j{+}1)\ldots(k)(k{-}j)(k{+}1)\ldots(n{-}1)$. For example, when $n = 8$ and $c_n \ldots c_1 = 00030000$ the corresponding reverse colex permutation is 01345267, which is $(2\,3\,4\,5)$ in cycle form. When $\sigma(k,j)$ is given by (18), Eqs. (15) and (16) lead to the formulas

$$\tau(k,j) = (k{-}j \;\; k); \tag{19}$$

$$\omega(k) = (0\,1)(0\,1\,2)\ldots(0\,1\ldots k) = (0\,k)(1\,k{-}1)(2\,k{-}2)\ldots = \phi(k); \tag{20}$$

here $\phi(k)$ is the "$(k{+}1)$-flip" that changes $a_0 \ldots a_k$ to $a_k \ldots a_0$. In this case $\omega(k)$ turns out to be the same as $\omega(k)^-$, because $\phi(k)^2 = ()$.

Equations (19) and (20) are implicitly present behind the scenes in Algorithm L and in its reverse colex equivalent (exercise 2), where step L3 essentially applies a transposition and step L4 does a flip. Step G4 actually does the flip first; but the identity

$$(k{-}j\;k)\phi(k-1) \;=\; \phi(k-1)(j{-}1\;k) \tag{21}$$

shows that a flip followed by a transposition is the same as a (different) transposition followed by the flip.

In fact, equation (21) is a special case of the important identity

$$\pi^-(j_1\;j_2\;\ldots\;j_t)\pi \;=\; (j_1\pi\;j_2\pi\;\ldots\;j_t\pi), \tag{22}$$

which is valid for *any* permutation π and any t-cycle $(j_1\;j_2\;\ldots\;j_t)$. On the left of (22) we have, for example, $j_1\pi \mapsto j_1 \mapsto j_2 \mapsto j_2\pi$, in agreement with the cycle on the right. Therefore if α and π are any permutations whatsoever, the permutation $\pi^-\alpha\pi$ (called the *conjugate* of α by π) has exactly the same cycle structure as α; we simply replace each element j in each cycle by $j\pi$.

Another significant special case of Algorithm G was introduced by R. J. Ord-Smith [*CACM* **10** (1967), 452; **12** (1969), 638; see also *Comp. J.* **14** (1971),

136–139], whose algorithm is obtained by setting

$$\sigma(k,j) = (k\ \ldots\ 1\ 0)^j. \tag{23}$$

Now it is clear from (15) that

$$\tau(k,j) = (k\ \ldots\ 1\ 0); \tag{24}$$

and once again we have

$$\omega(k) = (0\ k)(1\ k{-}1)(2\ k{-}2)\ldots = \phi(k), \tag{25}$$

because $\sigma(k,k) = (0\ 1\ \ldots\ k)$ is the same as before. The nice thing about this method is that the permutation needed in step G4, namely $\tau(k,c_k)\omega(k-1)^-$, does not depend on c_k:

$$\tau(k,j)\omega(k-1)^- = (k\ \ldots\ 1\ 0)\phi(k-1)^- = \phi(k). \tag{26}$$

Thus, Ord-Smith's algorithm is the special case of Algorithm G in which step G4 simply interchanges $a_0 \leftrightarrow a_k$, $a_1 \leftrightarrow a_{k-1}$, ...; this operation is usually quick, because k is small, and it saves some of the work of Algorithm L. (See exercise 38 and the reference to G. S. Klügel in Section 7.2.1.7.)

We can do even better by rigging things so that step G4 needs to do only a single transposition each time, somewhat as in Algorithm P but not necessarily on adjacent elements. Many such schemes are possible. The best is probably to let

$$\tau(k,j)\omega(k-1)^- = \begin{cases} (k\ 0), & \text{if } k \text{ is even,} \\ (k\ j{-}1), & \text{if } k \text{ is odd,} \end{cases} \tag{27}$$

as suggested by B. R. Heap [*Comp. J.* **6** (1963), 293–294]. Notice that Heap's method always transposes $a_k \leftrightarrow a_0$ except when $k = 3, 5, \ldots$; and the value of k, in 5 of every 6 steps, is either 1 or 2. Exercise 40 proves that Heap's method does indeed generate all permutations.

Bypassing unwanted blocks. One noteworthy advantage of Algorithm G is that it runs through all permutations of $a_0 \ldots a_{k-1}$ before touching a_k; then it performs another $k!$ cycles before changing a_k again, and so on. Therefore if at any time we reach a setting of the final elements $a_k \ldots a_{n-1}$ that is unimportant to the problem we're working on, we can skip quickly over all permutations that end with the undesirable suffix. More precisely, we could replace step G2 by the following substeps:

G2.0. [Acceptable?] If $a_k \ldots a_{n-1}$ is not an acceptable suffix, go to G2.1. Otherwise set $k \leftarrow k-1$. Then if $k > 0$, repeat this step; if $k = 0$, proceed to step G2.2.

G2.1. [Skip this suffix.] If $c_k = k$, apply $\sigma(k,k)^-$ to $a_0 \ldots a_{n-1}$, set $c_k \leftarrow 0$, $k \leftarrow k+1$, and repeat until $c_k < k$. Terminate if $k = n$; otherwise set $c_k \leftarrow c_k + 1$, apply $\tau(k,c_k)$ to $a_0 \ldots a_{n-1}$, and return to G2.0.

G2.2. [Visit.] Visit the permutation $a_0 \ldots a_{n-1}$. ∎

Step G1 should also set $k \leftarrow n-1$. Notice that the new steps are careful to preserve condition (17). The algorithm has become more complicated, because

we need to know the permutations $\tau(k,j)$ and $\sigma(k,k)$ in addition to the permutations $\tau(k,j)\omega(k-1)^-$ that appear in G4. But the additional complications are often worth the effort, because the resulting program might run significantly faster.

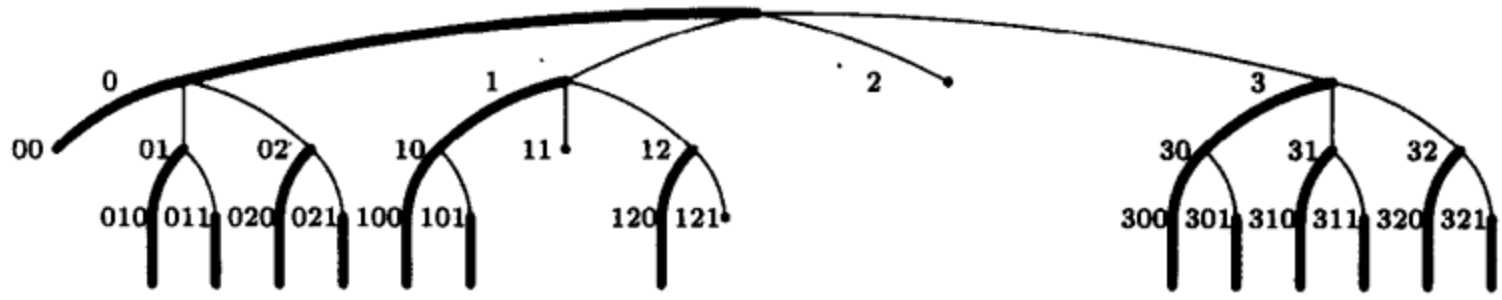


Fig. 20. Unwanted branches can be pruned from the tree of Fig. 19, if Algorithm G is suitably extended.

For example, Fig. 20 shows what happens to the tree of Fig. 19 when the suffixes of $a_0a_1a_2a_3$ that correspond to nodes 00, 11, 121, and 2 are not acceptable. (Each suffix $a_k \ldots a_{n-1}$ of the permutation $a_0 \ldots a_{n-1}$ corresponds to a *prefix* $c_n \ldots c_k$ of the control string $c_n \ldots c_1$, because the permutations $\sigma(1,c_1)\ldots\sigma(k-1,c_{k-1})$ do not affect $a_k \ldots a_{n-1}$.) Step G2.1 premultiplies by $\tau(k,j)$ to move from node $c_{n-1}\ldots c_{k+1}j$ to its right sibling $c_{n-1}\ldots c_{k+1}(j+1)$, and it premultiplies by $\sigma(k,k)^-$ to move up from node $c_{n-1}\ldots c_{k+1}k$ to its parent $c_{n-1}\ldots c_{k+1}$. Thus, to get from the rejected prefix 121 to its preorder successor, the algorithm premultiplies by $\sigma(1,1)^-$, $\sigma(2,2)^-$, and $\tau(3,2)$, thereby moving from node 121 to 12 to 1 to 2. (This is a somewhat exceptional case, because a prefix with $k=1$ is rejected only if we don't want to visit the unique permutation $a_0a_1\ldots a_{n-1}$ that has suffix $a_1\ldots a_{n-1}$.) After node 2 is rejected, $\tau(3,3)$ takes us to node 3, etc.

Notice, incidentally, that bypassing a suffix $a_k \ldots a_{n-1}$ in this extension of Algorithm G is essentially the same as bypassing a prefix $a_1 \ldots a_j$ in our original notation, if we go back to the idea of generating permutations $a_1 \ldots a_n$ of $\{1,\ldots,n\}$ and doing most of the work at the right-hand end. Our original notation corresponds to choosing a_1 first, then a_2, ..., then a_n; the notation in Algorithm G essentially chooses a_{n-1} first, then a_{n-2}, ..., then a_0. Algorithm G's conventions may seem backward, but they make the formulas for Sims table manipulation a lot simpler. A good programmer soon learns to switch without difficulty from one viewpoint to another.

We can apply these ideas to alphametics, because it is clear for example that most choices of the values for the letters `D`, `E`, and `Y` will make it impossible for `SEND` plus `MORE` to equal `MONEY`: We need to have $(\texttt{D}+\texttt{E}-\texttt{Y}) \bmod 10 = 0$ in that problem. Therefore many permutations can be eliminated from consideration.

In general, if r_k is the maximum power of 10 that divides the signature value s_k, we can sort the letters and assign codes $\{0,1,\ldots,9\}$ so that $r_0 \ge r_1 \ge \cdots \ge r_9$. For example, to solve the trio sonata problem (7), we could use $(0,1,\ldots,9)$ respectively for $(\texttt{X},\texttt{S},\texttt{V},\texttt{A},\texttt{R},\texttt{I},\texttt{L},\texttt{T},\texttt{O},\texttt{N})$, obtaining the signatures

$$s_0 = 0, \quad s_1 = -100000, \quad s_2 = 210000, \quad s_3 = -100, \quad s_4 = -100,$$
$$s_5 = 21010, \quad s_6 = 210, \quad s_7 = -1010, \quad s_8 = -7901, \quad s_9 = -998;$$

hence $(r_0, \ldots, r_9) = (\infty, 5, 4, 2, 2, 1, 1, 1, 0, 0)$. Now if we get to step G2.0 for a value of k with $r_{k-1} \neq r_k$, we can say that the suffix $a_k \ldots a_9$ is unacceptable unless $a_k s_k + \cdots + a_9 s_9$ is a multiple of $10^{r_k - 1}$. Also, (10) tells us that $a_k \ldots a_9$ is unacceptable if $a_k = 0$ and $k \in F$; the first-letter set F is now $\{1, 2, 7\}$.

Our previous approach to alphametics with steps A1–A3 above used brute force to run through 10! possibilities. It operated rather fast under the circumstances, since the adjacent-transposition method allowed it to get by with only 6 memory references per permutation; but still, 10! is 3,628,800, so the entire process cost almost 22 megamems, regardless of the alphametic being solved. By contrast, the extended Algorithm G with Heap's method and the cutoffs just described will find all four solutions to (7) with fewer than 128 *kilo*mems! Thus the suffix-skipping technique runs more than 170 times faster than the previous method, which simply blasted away blindly.

Most of the 128 kilomems in the new approach are spent applying $\tau(k, c_k)$ in step G2.1. The other memory references come primarily from applications of $\sigma(k, k)^-$ in that step, but τ is needed 7812 times while σ^- is needed only 2162 times. The reason is easy to understand from Fig. 20, because the "shortcut move" $\tau(k, c_k)\omega(k-1)^-$ in step G4 hardly ever applies; in this case it is used only four times, once for each solution. Thus, preorder traversal of the tree is accomplished almost entirely by τ steps that move to the right and σ^- steps that move upward. The τ steps dominate in a problem like this, where very few complete permutations are actually visited, because each step $\sigma(k, k)^-$ is preceded by k steps $\tau(k, 1)$, $\tau(k, 2)$, $\ldots$, $\tau(k, k)$.

This analysis reveals that Heap's method — which goes to great lengths to optimize the permutations $\tau(k, j)\omega(k-1)^-$ so that each transition in step G4 is a simple transposition — is *not* especially good for the extended Algorithm G unless comparatively few suffixes are rejected in step G2.0. The simpler reverse colex order, for which $\tau(k, j)$ itself is always a simple transposition, is now much more attractive (see (19)). Indeed, Algorithm G with reverse colex order solves the alphametic (7) with only 97 kilomems.

Similar results occur with respect to other alphametic problems. For example, if we apply the extended Algorithm G to the alphametics in exercise 24, parts (a) through (h), the computations involve respectively

$$\begin{aligned} &(551,\ 110,\ 14,\ 8,\ 350,\ 84,\ 153,\ 1598) \text{ kilomems with Heap's method;} \\ &(429,\ \ 84,\ 10,\ 5,\ 256,\ 63,\ 117,\ 1189) \text{ kilomems with reverse colex.} \end{aligned} \tag{28}$$

The speedup factor for reverse colex in these examples, compared to brute force with Algorithm T, ranges from 18 in case (h) to 4200 in case (d), and it is about 80 on the average; Heap's method gives an average speedup of about 60.

We know from Algorithm L, however, that lexicographic order is easily handled *without* the complication of the control table $c_n \ldots c_1$ used by Algorithm G. And a closer look at Algorithm L shows that we can improve its behavior when permutations are frequently being skipped, by using a linked list instead of a sequential array. The improved algorithm is well-suited to a wide variety of algorithms that wish to generate restricted classes of permutations:

Algorithm X (*Lexicographic permutations with restricted prefixes*). This algorithm generates all permutations $a_1 a_2 \dots a_n$ of $\{1, 2, \dots, n\}$ that pass a given sequence of tests

$$t_1(a_1), \quad t_2(a_1, a_2), \quad \dots, \quad t_n(a_1, a_2, \dots, a_n),$$

visiting them in lexicographic order. It uses an auxiliary table of links l_0, l_1, $\dots$, l_n to maintain a cyclic list of unused elements, so that if the currently available elements are

$$\{1, \dots, n\} \setminus \{a_1, \dots, a_k\} = \{b_1, \dots, b_{n-k}\}, \qquad \text{where } b_1 < \cdots < b_{n-k}, \tag{29}$$

then we have

$$l_0 = b_1, \quad l_{b_j} = b_{j+1} \quad \text{for } 1 \le j < n-k, \quad \text{and} \quad l_{b_{n-k}} = 0. \tag{30}$$

It also uses an auxiliary table $u_1 \dots u_n$ to undo operations that have been performed on the l array.

X1. [Initialize.] Set $l_k \leftarrow k+1$ for $0 \le k < n$, and $l_n \leftarrow 0$. Then set $k \leftarrow 1$.

X2. [Enter level k.] Set $p \leftarrow 0$, $q \leftarrow l_0$.

X3. [Test $a_1 \dots a_k$.] Set $a_k \leftarrow q$. If $t_k(a_1, \dots, a_k)$ is false, go to X5. Otherwise, if $k = n$, visit $a_1 \dots a_n$ and go to X6.

X4. [Increase k.] Set $u_k \leftarrow p$, $l_p \leftarrow l_q$, $k \leftarrow k+1$, and return to X2.

X5. [Increase a_k.] Set $p \leftarrow q$, $q \leftarrow l_p$. If $q \neq 0$ return to X3.

X6. [Decrease k.] Set $k \leftarrow k-1$, and terminate if $k = 0$. Otherwise set $p \leftarrow u_k$, $q \leftarrow a_k$, $l_p \leftarrow q$, and go to X5. ∎

The basic idea of this elegant algorithm is due to M. C. Er [*Comp. J.* **30** (1987), 282]. We can apply it to alphametics by changing notation slightly, obtaining permutations $a_0 \dots a_9$ of $\{0, \dots, 9\}$ and letting l_{10} play the former role of l_0. The resulting algorithm needs only 49 kilomems to solve the trio-sonata problem (7), and it solves the alphametics of exercise 24(a)–(h) in

$$(248,\ 38,\ 4,\ 3,\ 122,\ 30,\ 55,\ 553) \text{ kilomems}, \tag{31}$$

respectively. Thus it runs about 165 times faster than the brute-force approach.

Another way to apply Algorithm X to alphametics is often faster yet (see exercise 49).

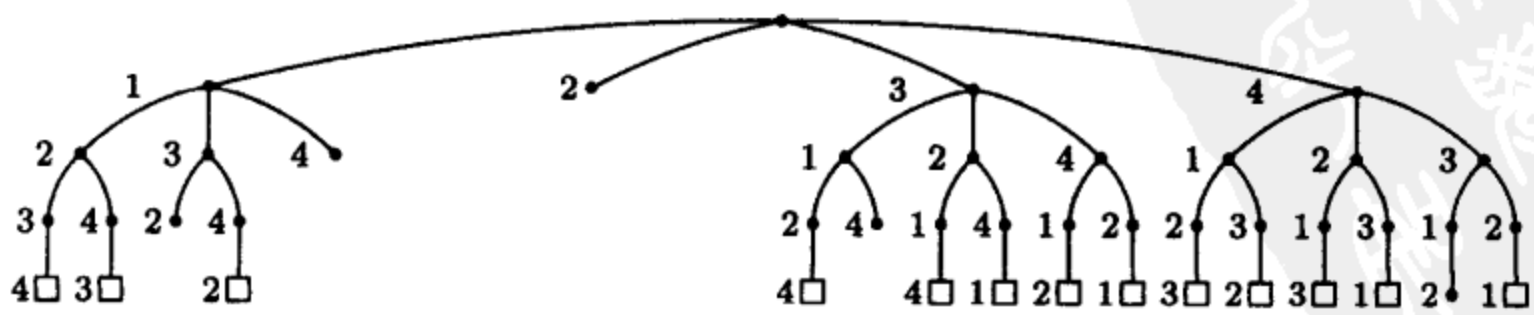


Fig. 21. The tree implicitly traversed by Algorithm X when $n = 4$, if all permutations are visited except those beginning with 132, 14, 2, 314, or 4312.

***Dual methods.** If $S_1, \ldots, S_{n-1}$ is a Sims table for a permutation group G, we learned in Lemma S that every element of G can be expressed uniquely as a product $\sigma_1 \ldots \sigma_{n-1}$, where $\sigma_k \in S_k$; see (13). Exercise 50 shows that every element α can also be expressed uniquely in the dual form

$$\alpha = \sigma_{n-1}^- \ldots \sigma_2^- \sigma_1^-, \qquad \text{where } \sigma_k \in S_k \text{ for } 1 \le k < n, \tag{32}$$

and this fact leads to another large family of permutation generators. In particular, when G is the group of all $n!$ permutations, every permutation can be written

$$\sigma(n-1, c_{n-1})^- \ldots \sigma(2, c_2)^- \sigma(1, c_1)^-, \tag{33}$$

where $0 \le c_k \le k$ for $1 \le k < n$ and the permutations $\sigma(k, j)$ are the same as in Algorithm G. Now, however, we want to vary c_{n-1} most rapidly and c_1 least rapidly, so we arrive at an algorithm of a different kind:

Algorithm H (*Dual permutation generator*). Given a Sims table as in Algorithm G, this algorithm generates all permutations $a_0 \ldots a_{n-1}$ of $\{0, \ldots, n-1\}$, using an auxiliary table $c_0 \ldots c_{n-1}$.

H1. [Initialize.] Set $a_j \leftarrow j$ and $c_j \leftarrow 0$ for $0 \le j < n$.

H2. [Visit.] (At this point the mixed-radix number $\left[\begin{smallmatrix} c_1, & c_2, & \ldots, & c_{n-1} \\ 2, & 3, & \ldots, & n \end{smallmatrix}\right]$ is the number of permutations visited so far.) Visit the permutation $a_0 a_1 \ldots a_{n-1}$.

H3. [Add 1 to $c_0 c_1 \ldots c_{n-1}$.] Set $k \leftarrow n-1$. If $c_k = k$, set $c_k \leftarrow 0$, $k \leftarrow k-1$, and repeat until $k = 0$ or $c_k < k$. Terminate the algorithm if $k = 0$; otherwise set $c_k \leftarrow c_k + 1$.

H4. [Permute.] Apply the permutation $\tau(k, c_k)\omega(k+1)^-$ to $a_0 a_1 \ldots a_{n-1}$, as explained below, and return to H2. ∎

Although this algorithm looks almost identical to Algorithm G, the permutations τ and ω that it needs in step H4 are quite different from those needed in step G4. The new rules, which replace (15) and (16), are

$$\tau(k, j) = \sigma(k, j)^- \sigma(k, j-1), \qquad \text{for } 1 \le j \le k, \tag{34}$$

$$\omega(k) = \sigma(n-1, n-1)^- \sigma(n-2, n-2)^- \ldots \sigma(k, k)^-. \tag{35}$$

The number of possibilities is just as vast as it was for Algorithm G, so we will confine our attention to a few cases that have special merit. One natural case to try is, of course, the Sims table that makes Algorithm G produce reverse colex order, namely

$$\sigma(k, j) \;=\; (k{-}j \;\; k{-}j{+}1 \;\; \ldots \;\; k) \tag{36}$$

as in (18). The resulting permutation generator turns out to be very nearly the same as the method of plain changes; so we can say that Algorithms L and P are essentially dual to each other. (See exercise 52.)

Another natural idea is to construct a Sims table for which step H4 always makes a single transposition of two elements, by analogy with the construction of (27) that achieves maximum efficiency in step G4. But such a mission now turns out to be impossible: We cannot achieve it even when $n = 4$. For if

we start with the identity permutation $a_0a_1a_2a_3 = 0123$, the transitions that take us from control table $c_0c_1c_2c_3 = 0000$ to 0001 to 0002 to 0003 must move the 3; so, if they are transpositions, they must be $(3\,a)$, $(a\,b)$, and $(b\,c)$ for some permutation abc of $\{0,1,2\}$. The permutation corresponding to $c_0c_1c_2c_3 = 0003$ is now $\sigma(3,3)^- = (b\,c)(a\,b)(3\,a) = (3\,a\,b\,c)$; and the next permutation, which corresponds to $c_0c_1c_2c_3 = 0010$, will be $\sigma(2,1)^-$, which must fix the element 3. The only suitable transposition is $(3\,c)$, hence $\sigma(2,1)^-$ must be $(3\,c)(3\,a\,b\,c) = (a\,b\,c)$. Similarly we find that $\sigma(2,2)^-$ must be $(a\,c\,b)$, and the permutation corresponding to $c_0c_1c_2c_3 = 0023$ will be $(3\,a\,b\,c)(a\,c\,b) = (3\,c)$. Step H4 is now supposed to convert this to the permutation $\sigma(1,1)^-$, which corresponds to the control table 0100 that follows 0023. But the only transposition that will convert $(3\,c)$ into a permutation that fixes 2 and 3 is $(3\,c)$; and the resulting permutation also fixes 1, so it cannot be $\sigma(1,1)^-$.

The proof in the preceding paragraph shows that we cannot use Algorithm H to generate all permutations with the minimum number of transpositions. But it also suggests a simple generation scheme that comes very close to the minimum, and the resulting algorithm is quite attractive because it needs to do extra work only once per $n(n-1)$ steps. (See exercise 53.)

Finally, let's consider the dual of Ord-Smith's method, when

$$\sigma(k,j) = (k\ \ldots\ 1\ 0)^j \tag{37}$$

as in (23). Once again the value of $\tau(k,j)$ is independent of j,

$$\tau(k,j) = (0\ 1\ \ldots\ k), \tag{38}$$

and this fact is particularly advantageous in Algorithm H because it allows us to dispense with the control table $c_0c_1\ldots c_{n-1}$. The reason is that $c_{n-1} = 0$ in step H3 if and only if $a_{n-1} = n-1$, because of (32); and indeed, when $c_j = 0$ for $k < j < n$ in step H3 we have $c_k = 0$ if and only if $a_k = k$. Therefore we can reformulate this variant of Algorithm H as follows.

Algorithm C (*Permutation generation by cyclic shifts*). This algorithm visits all permutations $a_1\ldots a_n$ of the distinct elements $\{x_1,\ldots,x_n\}$.

C1. [Initialize.] Set $a_j \leftarrow x_j$ for $1 \le j \le n$.

C2. [Visit.] Visit the permutation $a_1\ldots a_n$, and set $k \leftarrow n$.

C3. [Shift.] Replace $a_1a_2\ldots a_k$ by the cyclic shift $a_2\ldots a_ka_1$, and return to C2 if $a_k \neq x_k$.

C4. [Decrease k.] Set $k \leftarrow k-1$, and go back to C3 if $k > 1$. ∎

For example, the successive permutations of $\{1,2,3,4\}$ generated when $n = 4$ are

1234, 2341, 3412, 4123, (1234),
2314, 3142, 1423, 4231, (2314),
3124, 1243, 2431, 4312, (3124), (1234),
2134, 1342, 3421, 4213, (2134),
1324, 3241, 2413, 4132, (1324),
3214, 2143, 1432, 4321, (3214), (2134), (1234),

with unvisited intermediate permutations shown in parentheses. This algorithm may well be the simplest permutation generator of all, in terms of minimum program length. It is due to G. G. Langdon, Jr. [*CACM* **10** (1967), 298–299; **11** (1968), 392]; similar methods had been published previously by C. Tompkins [*Proc. Symp. Applied Math.* **6** (1956), 202–205] and, more explicitly, by R. Seitz [*Unternehmensforschung* **6** (1962), 2–15]. The procedure is particularly well suited to applications in which cyclic shifting is efficient, for example when successive permutations are being kept in a machine register instead of in an array.

The main disadvantage of dual methods is that they usually do not adapt well to situations where large blocks of permutations need to be skipped, because the set of all permutations with a given value of the first control entries $c_0 c_1 \ldots c_{k-1}$ is usually not of importance. The special case (36) is, however, sometimes an exception, because the $n!/k!$ permutations with $c_0 c_1 \ldots c_{k-1} = 00\ldots0$ in that case are precisely those $a_0 a_1 \ldots a_{n-1}$ in which 0 precedes 1, 1 precedes 2, ..., and $k-2$ precedes $k-1$.

***Ehrlich's swap method.** Gideon Ehrlich has discovered a completely different approach to permutation generation, based on yet another way to use a control table $c_1 \ldots c_{n-1}$. His method obtains each permutation from its predecessor by interchanging the leftmost element with another:

Algorithm E (*Ehrlich swaps*). This algorithm generates all permutations of the distinct elements $a_0 \ldots a_{n-1}$ by using auxiliary tables $b_0 \ldots b_{n-1}$ and $c_1 \ldots c_n$.

E1. [Initialize.] Set $b_j \leftarrow j$ and $c_{j+1} \leftarrow 0$ for $0 \le j < n$.

E2. [Visit.] Visit the permutation $a_0 \ldots a_{n-1}$.

E3. [Find k.] Set $k \leftarrow 1$. Then if $c_k = k$, set $c_k \leftarrow 0$, $k \leftarrow k+1$, and repeat until $c_k < k$. Terminate if $k = n$, otherwise set $c_k \leftarrow c_k + 1$.

E4. [Swap.] Interchange $a_0 \leftrightarrow a_{b_k}$.

E5. [Flip.] Set $j \leftarrow 1$, $k \leftarrow k-1$. If $j < k$, interchange $b_j \leftrightarrow b_k$, set $j \leftarrow j+1$, $k \leftarrow k-1$, and repeat until $j \ge k$. Return to E2. ∎

Notice that steps E2 and E3 are identical to steps G2 and G3 of Algorithm G. The most amazing thing about this algorithm, which Ehrlich communicated to Martin Gardner in 1987, is that it works; exercise 55 contains a proof. A similar method, which simplifies the operations of step E5, can be validated in the same way (see exercise 56). The average number of interchanges performed in step E5 is less than 0.18 (see exercise 57).

As it stands, Algorithm E isn't faster than other methods we have seen. But it has the nice property that it changes each permutation in a minimal way, using only $n-1$ different kinds of transpositions. Whereas Algorithm P used adjacent interchanges, $a_{t-1} \leftrightarrow a_t$, Algorithm E uses first-element swaps, $a_0 \leftrightarrow a_t$, also called *star transpositions*, for some well-chosen sequence of indices $t[1]$, $t[2]$, ..., $t[n!-1]$. And if we are generating permutations repeatedly for the same fairly small value of n, we can precompute this sequence, as we did in Algorithm T

for the index sequence of Algorithm P. Notice that star transpositions have an advantage over adjacent interchanges, because we always know the value of a_0 from the previous swap; we need not read it from memory.

Let E_n be the sequence of $n!-1$ indices t such that Algorithm E swaps a_0 with a_t in step E4. Since E_{n+1} begins with E_n, we can regard E_n as the first $n!-1$ elements of an infinite sequence

$$E_\infty = 12121321212312121321212431321313 12 \ldots . \tag{39}$$

For example, if $n = 4$ and $a_0a_1a_2a_3 = 1234$, the permutations visited by Algorithm E are

$$\begin{aligned} &1234,\ 2134,\ 3124,\ 1324,\ 2314,\ 3214,\\ &4213,\ 1243,\ 2143,\ 4123,\ 1423,\ 2413,\\ &3412,\ 4312,\ 1342,\ 3142,\ 4132,\ 1432,\\ &2431,\ 3421,\ 4321,\ 2341,\ 3241,\ 4231. \end{aligned} \tag{40}$$

***Using fewer generators.** After seeing Algorithms P and E, we might naturally ask whether all permutations can be obtained by using just *two* basic operations, instead of $n-1$. For example, Nijenhuis and Wilf [*Combinatorial Algorithms* (1975), Exercise 6] noticed that all permutations can be generated for $n = 4$ if we replace $a_1a_2a_3\ldots a_n$ at each step by either $a_2a_3\ldots a_na_1$ or $a_2a_1a_3\ldots a_n$, and they wondered whether such a method exists for all n.

In general, if G is any group of permutations and if $\alpha_1, \ldots, \alpha_k$ are elements of G, the *Cayley graph* for G with generators $(\alpha_1, \ldots, \alpha_k)$ is the directed graph whose vertices are the permutations π of G and whose arcs go from π to $\alpha_1\pi, \ldots, \alpha_k\pi$. [Arthur Cayley, *American J. Math.* **1** (1878), 174–176.] The question of Nijenhuis and Wilf is equivalent to asking whether the Cayley graph for all permutations of $\{1, 2, \ldots, n\}$, with generators σ and τ where σ is the cyclic permutation $(1\ 2\ \ldots\ n)$ and τ is the transposition $(1\ 2)$, has a Hamiltonian path.

A basic theorem due to R. A. Rankin [*Proc. Cambridge Philos. Soc.* **44** (1948), 17–25] allows us to conclude in many cases that Cayley graphs with two generators do not have a Hamiltonian cycle:

Theorem R. *Let G be a group consisting of g permutations. If the Cayley graph for G with generators (α, β) has a Hamiltonian cycle, and if the permutations $(\alpha, \beta, \alpha\beta^-)$ are respectively of order (a, b, c), then either c is even or g/a and g/b are odd.*

(The *order* of a permutation α is the least positive integer a such that α^a is the identity.)

Proof. See exercise 73. ∎

In particular, when $\alpha = \sigma$ and $\beta = \tau$ as above, we have $g = n!$, $a = n$, $b = 2$, and $c = n-1$, because $\sigma\tau^- = (2\ \ldots\ n)$. Therefore we conclude that no Hamiltonian cycle is possible when $n \geq 4$ is even. However, a Hamiltonian *path* is easy to

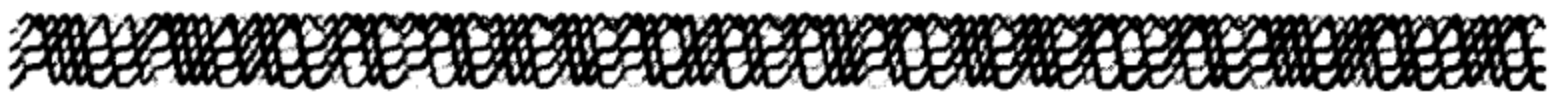
construct when $n = 4$, because we can join up the 12-cycles

$$\begin{aligned}&1234 \to 2341 \to 3412 \to 4312 \to 3124 \to 1243 \to 2431\\&\qquad\to 4231 \to 2314 \to 3142 \to 1423 \to 4123 \to 1234,\\&2134 \to 1342 \to 3421 \to 4321 \to 3214 \to 2143 \to 1432\\&\qquad\to 4132 \to 1324 \to 3241 \to 2413 \to 4213 \to 2134,\end{aligned} \tag{41}$$

by starting at 2341 and jumping from 1234 to 2134, ending at 4213.

Ruskey, Jiang, and Weston [*Discrete Applied Math.* **57** (1995), 75–83] undertook an exhaustive search in the σ–τ graph for $n = 5$ and discovered that it has five essentially distinct Hamiltonian cycles, one of which (the "most beautiful") is illustrated in Fig. 22(a). They also found a Hamiltonian path for $n = 6$; this was a difficult feat, because it is the outcome of a 720-stage binary decision tree. Unfortunately the solution they discovered has no apparent logical structure. A somewhat less complex path is described in exercise 70, but even that path cannot be called simple. Therefore a σ–τ approach will probably not be of practical interest for larger values of n unless a new construction is discovered. R. C. Compton and S. G. Williamson [*Linear and Multilinear Algebra* **35** (1993), 237–293] have proved that Hamiltonian cycles exist for all n if the three generators σ, σ^-, and τ are allowed instead of just σ and τ; their cycles have the interesting property that every nth transformation is τ, and the intervening $n-1$ transformations are either all σ or all σ^-. But their method is too complicated to explain in a short space.

Exercise 69 describes a general permutation algorithm that is reasonably simple and needs only three generators, each of order 2. Figure 22(b) illustrates the case $n = 5$ of this method, which was motivated by examples of bell-ringing.



(a) Using only transitions (1 2 3 4 5) and (1 2).



(b) Using only transitions (1 2)(3 4), (2 3)(4 5), and (3 4).

Fig. 22. Hamiltonian cycles for 5! permutations.

Faster, faster. What is the fastest way to generate permutations? This question has often been raised in computer publications, because people who examine $n!$ possibilities want to keep the running time as small as possible. But the answers have generally been contradictory, because there are many different ways to formulate the question. Let's try to understand the related issues by studying how permutations might be generated most rapidly on the MMIX computer.

Suppose first that our goal is to produce permutations in an array of n consecutive memory words (octabytes). The fastest way to do this, of all those we've seen in this section, is to streamline Heap's method (27), as suggested by R. Sedgewick [*Computing Surveys* **9** (1977), 157–160].

The key idea is to optimize the code for the most common cases of steps G2 and G3, namely the cases in which all activity occurs at the beginning of the array. If registers u, v, and w contain the contents of the first three words, and if the next six permutations to be generated involve permuting those words in all six possible ways, we can clearly do the job as follows:

```
PUSHJ 0,Visit
STO v,A0;  STO u,A1;  PUSHJ 0,Visit
STO w,A0;  STO v,A2;  PUSHJ 0,Visit
STO u,A0;  STO w,A1;  PUSHJ 0,Visit
STO v,A0;  STO u,A2;  PUSHJ 0,Visit
STO w,A0;  STO v,A1;  PUSHJ 0,Visit
```
(42)

(Here A0 is the address of octabyte a_0, etc.) A complete permutation program, which takes care of getting the right things into u, v, and w, appears in exercise 77, but the other instructions are less important because they need to be performed only $\frac{1}{6}$ of the time. The total cost per permutation, not counting the $4v$ needed for PUSHJ and POP on each call to Visit, comes to approximately $2.77\mu + 5.69v$ with this approach. If we use four registers u, v, w, x, and if we expand (42) to 24 calls on Visit, the running time per permutation drops to about $2.19\mu + 3.07v$. And with r registers and $r!$ Visits, exercise 78 shows that the cost is $(2 + O(1/r!))(\mu + v)$, which is very nearly the cost of two STO instructions.

The latter is, of course, the minimum possible time for any method that generates all permutations in a sequential array. ... Or is it? We have assumed that the visiting routine wants to see permutations in consecutive locations, but perhaps that routine is able to read the permutations from different starting points. Then we can arrange to keep a_{n-1} fixed and to keep two copies of the other elements in its vicinity:

$$a_0 a_1 \ldots a_{n-2} a_{n-1} a_0 a_1 \ldots a_{n-2}. \tag{43}$$

If we now let $a_0 a_1 \ldots a_{n-2}$ run through $(n-1)!$ permutations, always changing both copies simultaneously by doing two STO commands instead of one, we can let every call to Visit look at the n permutations

$$a_0 a_1 \ldots a_{n-1}, \quad a_1 \ldots a_{n-1} a_0, \quad \ldots, \quad a_{n-1} a_0 \ldots a_{n-2}, \tag{44}$$

which all appear consecutively. The cost per permutation is now reduced to the cost of three simple instructions like ADD, CMP, PBNZ, plus $O(1/n)$. [See Varol and Rotem, *Comp. J.* **24** (1981), 173–176.]

Furthermore, we might not want to waste time storing permutations into memory at all. Suppose, for example, that our goal is to generate all permutations of $\{0, 1, \ldots, n-1\}$. The value of n will probably be at most 16, because $16! = 20{,}922{,}789{,}888{,}000$ and $17! = 355{,}687{,}428{,}096{,}000$. Therefore an entire permutation will fit in the 16 nybbles of an octabyte, and we can keep it in a single register. This will be advantageous only if the visiting routine doesn't need to unpack the individual nybbles; but let's suppose that it doesn't. How fast can we generate permutations in the nybbles of a 64-bit register?

One idea, suggested by a technique due to A. J. Goldstein [*U. S. Patent 3383661* (14 May 1968)], is to precompute the table $(t[1],\ldots,t[5039])$ of plain-change transitions for seven elements, using Algorithm T. These numbers $t[k]$ lie between 1 and 6, so we can pack 20 of them into a 64-bit word. It is convenient to put the number $\sum_{k=1}^{20} 2^{3k-1}t[20j+k]$ into word j of an auxiliary table, for $0 \le j < 252$, with $t[5040] = 1$; for example, the table begins with the codeword

00|001|010|011|100|101|110|100|110|101|100|011|010|001|110|001|010|011|100|101|110|00.

The following program reads such codes efficiently:

```
Perm  ⟨Set register a to the first permutation⟩
0H    LDA  p,T       p ← address of first codeword.
      JMP  3F
1H    ⟨Visit the permutation in register a⟩
      ⟨Swap the nybbles of a that lie t bits from the right⟩
      SRU  c,c,3     c ← c ≫ 3.
2H    AND  t,c,#1c   t ← c & (11100)₂.                    (45)
      PBNZ t,1B      Branch if t ≠ 0.
      ADD  p,p,8
3H    LDO  c,p,0     c ← next codeword.
      PBNZ c,2B      (The final codeword is followed by 0.)
      ⟨If not done, advance the leading n − 7 nybbles and return to 0B⟩
```

Exercise 79 shows how to ⟨Swap the nybbles ...⟩ with seven instructions, using bit manipulation operations that are found on most computers. Therefore the cost per permutation is just a bit more than $10v$. (The instructions that fetch new codewords cost only $(\mu + 5v)/20$; and the instructions that advance the leading $n-7$ nybbles are even more negligible since their cost is divided by 5040.) Notice that there is now no need for PUSHJ and POP as there was with (42); we ignored those instructions before, but they did cost $4v$.

We can, however, do even better by adapting Langdon's cyclic-shift method, Algorithm C. Suppose we start with the lexicographically largest permutation and operate as follows:

```
      GREG @
0H    OCTA #fedcba9876543210&(1<<(4*N)-1)
Perm  LDOU a,0B          Set a ← #...3210.
      JMP  2F
1H    SRU  a,a,4*(16-N)  a ← ⌊a/16^(16−n)⌋.
      OR   a,a,t         a ← a | t.                       (46)
2H    ⟨Visit the permutation in register a⟩
      SRU  t,a,4*(N-1)   t ← ⌊a/16^(n−1)⌋.
      SLU  a,a,4*(17-N)  a ← 16^(17−n) a mod 16^16.
      PBNZ t,1B          To 1B if t ≠ 0.
      ⟨Continue with Langdon's method⟩
```

The running time per permutation is now only $5v + O(1/n)$, again without the need for PUSHJ and POP. See exercise 81 for an interesting way to extend (46) to a complete program, obtaining a remarkably short and fast routine.

Fast permutation generators are amusing, but in practice we can usually save more time by streamlining the visiting routine than by speeding up the generator.

Topological sorting. Instead of working with all $n!$ permutations of $\{1,\ldots,n\}$, we often want to look only at permutations that obey certain restrictions. For example, we might be interested only in permutations for which 1 precedes 3, 2 precedes 3, and 2 precedes 4; there are five such permutations of $\{1,2,3,4\}$, namely

$$1234,\ 1243,\ 2134,\ 2143,\ 2413. \tag{47}$$

The problem of *topological sorting*, which we studied in Section 2.2.3 as a first example of nontrivial data structures, is the general problem of finding a permutation that satisfies m such conditions $x_1 \prec y_1, \ldots, x_m \prec y_m$, where $x \prec y$ means that x should precede y in the permutation. This problem arises frequently in practice, so it has several different names; for example, it is often called the *linear embedding* problem, because we want to arrange objects in a line while preserving certain order relationships. It is also the problem of extending a partial ordering to a total ordering (see exercise 2.2.3–14).

Our goal in Section 2.2.3 was to find a *single* permutation that satisfies all the relations. But now we want rather to find *all* such permutations, all topological sorts. Indeed, we will assume in the present section that the elements x and y on which the relations are defined are integers between 1 and n, and that we have $x < y$ whenever $x \prec y$. Consequently the permutation $12\ldots n$ will always be topologically correct. (If this simplifying assumption is not met, we can preprocess the data by using Algorithm 2.2.3T to rename the objects appropriately.)

Many important classes of permutations are special cases of this topological ordering problem. For example, the permutations of $\{1,\ldots,8\}$ such that

$$1 \prec 2, \quad 2 \prec 3, \quad 3 \prec 4, \quad 6 \prec 7, \quad 7 \prec 8$$

are equivalent to permutations of the multiset $\{1,1,1,1,2,3,3,3\}$, because we can map $\{1,2,3,4\} \mapsto 1$, $5 \mapsto 2$, and $\{6,7,8\} \mapsto 3$. We know how to generate permutations of a multiset using Algorithm L, but now we will learn another way.

Notice that x precedes y in a permutation $a_1\ldots a_n$ if and only if $a'_x < a'_y$ in the inverse permutation $a'_1\ldots a'_n$. Therefore the algorithm we are about to study will also find all permutations $a'_1\ldots a'_n$ such that $a'_j < a'_k$ whenever $j \prec k$. For example, we learned in Section 5.1.4 that a Young tableau is an arrangement of $\{1,\ldots,n\}$ in rows and columns so that each row is increasing from left to right and each column is increasing from top to bottom. The problem of generating all 3×3 Young tableaux is therefore equivalent to generating all $a'_1\ldots a'_9$ such that

$$\begin{aligned} &a'_1 < a'_2 < a'_3, \quad a'_4 < a'_5 < a'_6, \quad a'_7 < a'_8 < a'_9, \\ &a'_1 < a'_4 < a'_7, \quad a'_2 < a'_5 < a'_8, \quad a'_3 < a'_6 < a'_9, \end{aligned} \tag{48}$$

and this is a special kind of topological sorting.

We might also want to find all *matchings* of $2n$ elements, namely all ways to partition $\{1,\ldots,2n\}$ into n pairs. There are $(2n-1)(2n-3)\ldots(1) = (2n)!/(2^n n!)$ ways to do this, and they correspond to permutations that satisfy

$$a'_1 < a'_2, \quad a'_3 < a'_4, \quad \ldots, \quad a'_{2n-1} < a'_{2n}, \qquad a'_1 < a'_3 < \cdots < a'_{2n-1}. \tag{49}$$

An elegant algorithm for exhaustive topological sorting was discovered by Y. L. Varol and D. Rotem [*Comp. J.* **24** (1981), 83–84], who realized that a method analogous to plain changes (Algorithm P) can be used. Suppose we have found a way to arrange $\{1,\ldots,n-1\}$ topologically, so that $a_1\ldots a_{n-1}$ satisfies all the conditions that do not involve n. Then we can easily write down all the allowable ways to insert the final element n without changing the relative order of $a_1\ldots a_{n-1}$: We simply start with $a_1\ldots a_{n-1}n$, then shift n left one step at a time, until it cannot move further. Applying this idea recursively yields the following straightforward procedure.

Algorithm V (*All topological sorts*). Given a relation $\prec$ on $\{1,\ldots,n\}$ with the property that $x \prec y$ implies $x < y$, this algorithm generates all permutations $a_1\ldots a_n$ and their inverses $a'_1\ldots a'_n$ with the property that $a'_j < a'_k$ whenever $j \prec k$. We assume for convenience that $a_0 = 0$ and that $0 \prec k$ for $1 \le k \le n$.

V1. [Initialize.] Set $a_j \leftarrow j$ and $a'_j \leftarrow j$ for $0 \le j \le n$.

V2. [Visit.] Visit the permutation $a_1\ldots a_n$ and its inverse $a'_1\ldots a'_n$. Then set $k \leftarrow n$.

V3. [Can k move left?] Set $j \leftarrow a'_k$ and $l \leftarrow a_{j-1}$. If $l \prec k$, go to V5.

V4. [Yes, move it.] Set $a_{j-1} \leftarrow k$, $a_j \leftarrow l$, $a'_k \leftarrow j-1$, and $a'_l \leftarrow j$. Go to V2.

V5. [No, put k back.] While $j < k$, set $l \leftarrow a_{j+1}$, $a_j \leftarrow l$, $a'_l \leftarrow j$, and $j \leftarrow j+1$. Then set $a_k \leftarrow a'_k \leftarrow k$. Decrease k by 1 and return to V3 if $k > 0$. ∎

For example, Theorem 5.1.4H tells us that there are exactly 42 Young tableaux of size 3×3. If we apply Algorithm V to the relations (48) and write the inverse permutation in array form

$$\begin{array}{|ccc|}\hline a'_1 & a'_2 & a'_3 \\ a'_4 & a'_5 & a'_6 \\ a'_7 & a'_8 & a'_9 \\ \hline\end{array}, \tag{50}$$

we get the following 42 results:

123 456 789	123 457 689	123 458 679	123 467 589	123 468 579	124 356 789	124 357 689	124 358 679	124 367 589	124 368 579	125 367 489	125 368 479	125 346 789	125 347 689
125 348 679	126 347 589	126 348 579	127 348 569	126 357 489	126 358 479	127 358 469	134 256 789	134 257 689	134 258 679	134 267 589	134 268 579	135 267 489	135 268 479
145 267 389	145 268 379	135 246 789	135 247 689	135 248 679	136 247 589	136 248 579	137 248 569	136 257 489	136 258 479	137 258 469	146 257 389	146 258 379	147 258 369

Let t_r be the number of topological sorts for which the final $n-r$ elements are in their initial position $a_j = j$ for $r < j \le n$. Equivalently, t_r is the number of topological sorts $a_1 \dots a_r$ of $\{1, \dots, r\}$, when we ignore the relations involving elements greater than r. Then the recursive mechanism underlying Algorithm V shows that step V2 is performed N times and step V3 is performed M times, where

$$M = t_n + \cdots + t_1 \qquad \text{and} \qquad N = t_n. \tag{51}$$

Also, step V4 and the loop operations of V5 are performed $N-1$ times; the rest of step V5 is done $M - N + 1$ times. Therefore the total running time of the algorithm is a linear combination of M, N, and n.

If the element labels are chosen poorly, M might be much larger than N. For example, if the constraints input to Algorithm V are

$$2 \prec 3, \quad 3 \prec 4, \quad \dots, \quad n-1 \prec n, \tag{52}$$

then $t_j = j$ for $1 \le j \le n$ and we have $M = \frac{1}{2}(n^2 + n)$, $N = n$. But those constraints are also equivalent to

$$1 \prec 2, \quad 2 \prec 3, \quad \dots, \quad n-2 \prec n-1, \tag{53}$$

under renaming of the elements; then M is reduced to $2n - 1 = 2N - 1$.

Exercise 89 shows that a simple preprocessing step will find element labels so that a slight modification of Algorithm V is able to generate all topological sorts in $O(N+n)$ steps. Thus topological sorting can always be done efficiently.

Think twice before you permute. We have seen several attractive algorithms for permutation generation in this section, but many algorithms are known by which permutations that are optimum for particular purposes can be found *without* running through all possibilities. For example, Theorem 6.1S showed that we can find the best way to arrange records on a sequential storage simply by sorting them with respect to a certain cost criterion, and this process takes only $O(n \log n)$ steps. In Section 7.5.2 we will study the *assignment problem*, which asks how to permute the columns of a square matrix so that the sum of the diagonal elements is maximized. That problem can be solved in at most $O(n^3)$ operations, so it would be foolish to use a method of order $n!$ unless n is extremely small. Even in cases like the traveling salesrep problem, when no efficient algorithm is known, we can usually find a much better approach than to examine every possible solution. Permutation generation is best used when there is good reason to look at each permutation individually.

EXERCISES

▶ **1.** [*20*] Explain how to make Algorithm L run faster, by streamlining its operations when the value of j is near n.

2. [*20*] Rewrite Algorithm L so that it produces all permutations of $a_1 \dots a_n$ in reverse colex order. (In other words, the values of the reflections $a_n \dots a_1$ should be lexicographically decreasing, as in (11). This form of the algorithm is often simpler and faster than the original, because fewer calculations depend on the value of n.)

▸ **3.** [*M21*] The *rank* of a combinatorial arrangement X with respect to a generation algorithm is the number of other arrangements that the algorithm visits prior to X. Explain how to compute the rank of a given permutation $a_1 \dots a_n$ with respect to Algorithm L, if $\{a_1, \dots, a_n\} = \{1, \dots, n\}$. What is the rank of 314592687?

4. [*M23*] Generalizing exercise 3, explain how to compute the rank of $a_1 \dots a_n$ with respect to Algorithm L when $\{a_1, \dots, a_n\}$ is the multiset $\{n_1 \cdot x_1, \dots, n_t \cdot x_t\}$; here $n_1 + \cdots + n_t = n$ and $x_1 < \cdots < x_t$. (The total number of permutations is, of course, the multinomial coefficient

$$\binom{n}{n_1, \dots, n_t} = \frac{n!}{n_1! \dots n_t!};$$

see Eq. 5.1.2–(3).) What is the rank of 314159265?

5. [*HM25*] Compute the mean and variance of the number of comparisons made by Algorithm L in (a) step L2, (b) step L3, when the elements $\{a_1, \dots, a_n\}$ are distinct.

6. [*HM34*] Derive generating functions for the mean number of comparisons made by Algorithm L in (a) step L2, (b) step L3, when $\{a_1, \dots, a_n\}$ is a general multiset as in exercise 4. Also give the results in closed form when $\{a_1, \dots, a_n\}$ is the binary multiset $\{s \cdot 0, (n-s) \cdot 1\}$.

7. [*HM35*] What is the limit as $t \to \infty$ of the average number of comparisons made per permutation in step L2 when Algorithm L is being applied to the multiset (a) $\{2 \cdot 1, 2 \cdot 2, \dots, 2 \cdot t\}$? (b) $\{1 \cdot 1, 2 \cdot 2, \dots, t \cdot t\}$? (c) $\{2 \cdot 1, 4 \cdot 2, \dots, 2^t \cdot t\}$?

▸ **8.** [*21*] The *variations* of a multiset are the permutations of all its submultisets. For example, the variations of $\{1, 2, 2, 3\}$ are

ϵ, 1, 12, 122, 1223, 123, 1232, 13, 132, 1322,
2, 21, 212, 2123, 213, 2132, 22, 221, 2213, 223, 2231, 23, 231, 2312, 232, 2321,
3, 31, 312, 3122, 32, 321, 3212, 322, 3221.

Show that simple changes to Algorithm L will generate all variations of a given multiset $\{a_1, a_2, \dots, a_n\}$.

9. [*22*] Continuing the previous exercise, design an algorithm to generate all r-variations of a given multiset $\{a_1, a_2, \dots, a_n\}$, also called its r-permutations, namely all permutations of its r-element submultisets. (For example, the solution to an alphametic with r distinct letters is an r-variation of $\{0, 1, \dots, 9\}$.)

10. [*20*] What are the values of $a_1 a_2 \dots a_n$, $c_1 c_2 \dots c_n$, and $o_1 o_2 \dots o_n$ at the end of Algorithm P, if $a_1 a_2 \dots a_n = 12 \dots n$ at the beginning?

11. [*M22*] How many times is each step of Algorithm P performed? (Assume that $n \geq 2$.)

▸ **12.** [*M23*] What is the 1000000th permutation visited by (a) Algorithm L, (b) Algorithm P, (c) Algorithm C, if $\{a_1, \dots, a_n\} = \{0, \dots, 9\}$? *Hint:* In mixed-radix notation we have $1000000 = \left[\begin{smallmatrix} 2, & 6, & 6, & 2, & 5, & 1, & 2, & 2, & 0, & 0 \\ 10, & 9, & 8, & 7, & 6, & 5, & 4, & 3, & 2, & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0, & 0, & 1, & 2, & 3, & 0, & 2, & 7, & 1, & 0 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10 \end{smallmatrix}\right]$.

13. [*M21*] (Martin Gardner, 1974.) True or false: If $a_1 a_2 \dots a_n$ is initially $12 \dots n$, Algorithm P begins by visiting all $n!/2$ permutations in which 1 precedes 2; then the next permutation is $n \dots 21$.

14. [*M22*] True or false: If $a_1 a_2 \dots a_n$ is initially $x_1 x_2 \dots x_n$ in Algorithm P, we always have $a_{j - c_j + s} = x_j$ at the beginning of step P5.

15. [*M23*] (Selmer Johnson, 1963.) Show that the offset variable s never exceeds 2 in Algorithm P.

16. [*21*] Explain how to make Algorithm P run faster, by streamlining its operations when the value of j is near n. (This problem is analogous to exercise 1.)

▶ **17.** [*20*] Extend Algorithm P so that the *inverse permutation* $a'_1 \ldots a'_n$ is available for processing when $a_1 \ldots a_n$ is visited in step P2. (The inverse satisfies $a'_k = j$ if and only if $a_j = k$.)

18. [*21*] (*Rosary permutations.*) Devise an efficient way to generate $(n-1)!/2$ permutations that represent all possible undirected cycles on the vertices $\{1, \ldots, n\}$; that is, no cyclic shift of $a_1 \ldots a_n$ or $a_n \ldots a_1$ will be generated if $a_1 \ldots a_n$ is generated. The permutations (1234, 1324, 3124) could, for example, be used when $n = 4$.

19. [*25*] Construct an algorithm that generates all permutations of n distinct elements *looplessly* in the spirit of Algorithm 7.2.1.1L.

▶ **20.** [*20*] The n-cube has $2^n n!$ symmetries, one for each way to permute and/or complement the coordinates. Such a symmetry is conveniently represented as a *signed permutation*, namely a permutation with optional signs attached to the elements. For example, $2\bar{3}1$ is a signed permutation that transforms the vertices of the 3-cube by changing $x_1x_2x_3$ to $x_2\bar{x}_3x_1$, so that $000 \mapsto 001$, $001 \mapsto 011$, ..., $111 \mapsto 110$. Design a simple algorithm that generates all signed permutations of $\{1, 2, \ldots, n\}$, where each step either interchanges two adjacent elements or negates the first element.

21. [*M21*] (E. P. McCravy, 1971.) How many solutions does the alphametic (6) have in radix b?

22. [*M15*] True or false: If an alphametic has a solution in radix b, it has a solution in radix $b + 1$.

23. [*M20*] True or false: A pure alphametic cannot have two identical signatures $s_j = s_k \neq 0$ when $j \neq k$.

24. [*25*] Solve the following alphametics by hand or by computer:

a) SEND + A + TAD + MORE = MONEY.

b) ZEROES + ONES = BINARY. (Peter MacDonald, 1977)

c) DCLIX + DLXVI = MCCXXV. (Willy Enggren, 1972)

d) COUPLE + COUPLE = QUARTET. (Michael R. W. Buckley, 1977)

e) FISH + N + CHIPS = SUPPER. (Bob Vinnicombe, 1978)

f) SATURN + URANUS + NEPTUNE + PLUTO = PLANETS. (Willy Enggren, 1968)

g) EARTH + AIR + FIRE + WATER = NATURE. (Herman Nijon, 1977)

h) AN + ACCELERATING + INFERENTIAL + ENGINEERING + TALE + ELITE + GRANT + FEE + ET + CETERA = ARTIFICIAL + INTELLIGENCE.

i) HARDY + NESTS = NASTY + HERDS.

▶ **25.** [*M21*] Devise a fast way to compute $\min(a \cdot s)$ and $\max(a \cdot s)$ over all valid permutations $a_1 \ldots a_{10}$ of $\{0, \ldots, 9\}$, given the signature vector $s = (s_1, \ldots, s_{10})$ and the first-letter set F of an alphametic problem. (Such a procedure makes it possible to rule out many cases quickly when a large family of alphametics is being considered, as in several of the exercises that follow, because a solution can exist only when $\min(a \cdot s) \leq 0 \leq \max(a \cdot s)$.)

26. [*25*] What is the unique alphametic solution to

$$\texttt{NIIHAU} \pm \texttt{KAUAI} \pm \texttt{OAHU} \pm \texttt{MOLOKAI} \pm \texttt{LANAI} \pm \texttt{MAUI} \pm \texttt{HAWAII} = 0?$$

27. [*30*] Construct pure additive alphametics in which all words have five letters.

28. [*M25*] A *partition* of the integer n is an expression of the form $n = n_1 + \cdots + n_t$ with $n_1 \geq \cdots \geq n_t > 0$. Such a partition is called *doubly true* if $\alpha(n) = \alpha(n_1) + \cdots + \alpha(n_t)$ is also a pure alphametic, where $\alpha(n)$ is the "name" of n in some language. Doubly true partitions were introduced by Alan Wayne in *AMM* **54** (1947), 38, 412–414, where he suggested solving TWENTY = SEVEN + SEVEN + SIX and a few others.

a) Find all partitions that are doubly true in English when $1 \leq n \leq 20$.

b) Wayne also gave the example EIGHTY = FIFTY + TWENTY + NINE + ONE. Find all doubly true partitions for $1 \leq n \leq 100$ in which the parts are *distinct*, using the names ONE, TWO, ..., NINETYNINE, ONEHUNDRED.

▶ **29.** [*M25*] Continuing the previous exercise, find all equations of the form $n_1 + \cdots + n_t = n'_1 + \cdots + n'_{t'}$ that are both mathematically and alphametically true in English, when $\{n_1, \ldots, n_t, n'_1, \ldots, n'_{t'}\}$ are distinct positive integers less than 20. For example,

$$\texttt{TWELVE} + \texttt{NINE} + \texttt{TWO} = \texttt{ELEVEN} + \texttt{SEVEN} + \texttt{FIVE};$$

the alphametics should all be pure.

30. [*25*] Solve these multiplicative alphametics by hand or by computer:

a) TWO × TWO = SQUARE. (H. E. Dudeney, 1929)

b) HIP × HIP = HURRAY. (Willy Enggren, 1970)

c) PI × R × R = AREA. (Brian Barwell, 1981)

d) NORTH/SOUTH = EAST/WEST. (Nob Yoshigahara, 1995)

e) NAUGHT × NAUGHT = ZERO × ZERO × ZERO. (Alan Wayne, 2003)

31. [*M22*] (Nob Yoshigahara.) What is the unique solution to A/BC+D/EF+G/HI = 1, when $\{\texttt{A}, \ldots, \texttt{I}\} = \{1, \ldots, 9\}$?

32. [*M25*] (H. E. Dudeney, 1901.) Find all ways to represent 100 by inserting a plus sign and a slash into a permutation of the digits $\{1, \ldots, 9\}$. For example, $100 = 91 + 5742/638$. The plus sign should precede the slash.

33. [*25*] Continuing the previous exercise, find all positive integers less than 150 that (a) cannot be represented in such a fashion; (b) have a unique representation.

34. [*M26*] Make the equation EVEN + ODD + PRIME = x doubly true when (a) x is a perfect 5th power; (b) x is a perfect 7th power.

▶ **35.** [*M20*] The automorphisms of a 4-cube have many different Sims tables, only one of which is shown in (14). How many different Sims tables are possible for that group, when the vertices are numbered as in (12)?

36. [*M23*] Find a Sims table for the group of all automorphisms of the 4×4 tic-tac-toe board

0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

,

namely the permutations that take lines into lines, where a "line" is a set of four elements that belong to a row, column, or diagonal.

▶ **37.** [*HM22*] How many Sims tables can be used with Algorithms G or H? Estimate the logarithm of this number as $n \to \infty$.

38. [*HM21*] Prove that the average number of transpositions per permutation when using Ord-Smith's algorithm (26) is approximately $\sinh 1 \approx 1.175$.

39. [*16*] Write down the 24 permutations generated for $n = 4$ by (a) Ord-Smith's method (26); (b) Heap's method (27).

40. [*M23*] Show that Heap's method (27) corresponds to a valid Sims table.

▶ **41.** [*M33*] Design an algorithm that generates all r-variations of $\{0, 1, \ldots, n-1\}$ by interchanging just two elements when going from one variation to the next. (See exercise 9.) *Hint:* Generalize Heap's method (27), obtaining the results in positions $a_{n-r} \ldots a_{n-1}$ of an array $a_0 \ldots a_{n-1}$. For example, one solution when $n = 5$ and $r = 2$ uses the final two elements of the respective permutations 01234, 31204, 30214, 30124, 40123, 20143, 24103, 24013, 34012, 14032, 13042, 13402, 23401, 03421, 02431, 02341, 12340, 42310, 41320, 41230.

42. [*M20*] Construct a Sims table for all permutations in which every $\sigma(k, j)$ and every $\tau(k, j)$ for $1 \le j \le k$ is a cycle of length ≤ 3.

43. [*M24*] Construct a Sims table for all permutations in which every $\sigma(k, k)$, $\omega(k)$, and $\tau(k, j)\omega(k-1)^-$ for $1 \le j \le k$ is a cycle of length ≤ 3.

44. [*20*] When blocks of unwanted permutations are being skipped by the extended Algorithm G, is the Sims table of Ord-Smith's method (23) superior to the Sims table of the reverse colex method (18)?

45. [*20*] (a) What are the indices $u_1 \ldots u_9$ when Algorithm X visits the permutation 314592687? (b) What permutation is visited when $u_1 \ldots u_9 = 314157700$?

46. [*20*] True or false: When Algorithm X visits $a_1 \ldots a_n$, we have $u_k > u_{k+1}$ if and only if $a_k > a_{k+1}$, for $1 \le k < n$.

▶ **47.** [*M21*] Express the number of times that each step of Algorithm X is performed in terms of the numbers $N_0, N_1, \ldots, N_n$, where N_k is the number of prefixes $a_1 \ldots a_k$ that satisfy $t_j(a_1, \ldots, a_j)$ for $1 \le j \le k$.

▶ **48.** [*M25*] Compare the running times of Algorithm X and Algorithm L, in the case when the tests $t_1(a_1)$, $t_2(a_1, a_2)$, $\ldots$, $t_n(a_1, a_2, \ldots, a_n)$ always are true.

▶ **49.** [*28*] The text's suggested method for solving additive alphametics with Algorithm X essentially chooses digits from right to left; in other words, it assigns tentative values to the least significant digits before considering digits that correspond to higher powers of 10.

Explore an alternative approach that chooses digits from left to right. For example, such a method will deduce immediately that `M` $= 1$ when `SEND` + `MORE` = `MONEY`. *Hint:* See exercise 25.

50. [*M15*] Explain why the dual formula (32) follows from (13).

51. [*M16*] True or false: If the sets $S_k = \{\sigma(k, 0), \ldots, \sigma(k, k)\}$ form a Sims table for the group of all permutations, so also do the sets $S_k^- = \{\sigma(k, 0)^-, \ldots, \sigma(k, k)^-\}$.

▶ **52.** [*M22*] What permutations $\tau(k, j)$ and $\omega(k)$ arise when Algorithm H is used with the Sims table (36)? Compare the resulting generator with Algorithm P.

▶ **53.** [*M26*] (F. M. Ives.) Construct a Sims table for which Algorithm H will generate all permutations by making only $n! + O((n-2)!)$ transpositions.

54. [*20*] Would Algorithm C work properly if step C3 did a right-cyclic shift, setting $a_1 \ldots a_{k-1} a_k \leftarrow a_k a_1 \ldots a_{k-1}$, instead of a left-cyclic shift?

55. [*M27*] Consider the *factorial ruler function*

$$\rho_!(m) = \max\{k \mid m \bmod k! = 0\}.$$

Let σ_k and τ_k be permutations of the nonnegative integers such that $\sigma_j\tau_k = \tau_k\sigma_j$ whenever $j \le k$. Let α_0 and β_0 be the identity permutation, and for $m > 0$ define

$$\alpha_m = \beta_{m-1}^{-}\tau_{\rho_1(m)}\beta_{m-1}\alpha_{m-1}, \qquad \beta_m = \sigma_{\rho_1(m)}\beta_{m-1}.$$

For example, if σ_k is the flip operation $(1\,k{-}1)(2\,k{-}2)\ldots = (0\,k)\phi(k)$ and if $\tau_k = (0\,k)$, and if Algorithm E is started with $a_j = j$ for $0 \le j < n$, then α_m and β_m are the contents of $a_0\ldots a_{n-1}$ and $b_0\ldots b_{n-1}$ after step E5 has been performed m times.

a) Prove that $\beta_{(n+1)!}\alpha_{(n+1)!} = \sigma_{n+1}\sigma_n^{-}\tau_{n+1}\tau_n^{-}(\beta_{n!}\alpha_{n!})^{n+1}$.

b) Use the result of (a) to establish the validity of Algorithm E.

56. [*M22*] Prove that Algorithm E remains valid if step E5 is replaced by

E5′. [Transpose pairs.] If $k > 2$, interchange $b_{j+1} \leftrightarrow b_j$ for $j = k-2, k-4, \ldots,$ (2 or 1). Return to E2. ∎

57. [*HM22*] What is the average number of interchanges made in step E5?

58. [*M21*] True or false: If Algorithm E begins with $a_0\ldots a_{n-1} = x_1\ldots x_n$ then the final permutation visited begins with $a_0 = x_n$.

59. [*M20*] Some authors define the arcs of a Cayley graph as running from π to $\pi\alpha_j$ instead of from π to $\alpha_j\pi$. Are the two definitions essentially different?

▸ **60.** [*21*] A *Gray cycle for permutations* is a cycle $(\pi_0, \pi_1, \ldots, \pi_{n!-1})$ that includes every permutation of $\{1, 2, \ldots, n\}$ and has the property that π_k differs from $\pi_{(k+1) \bmod n!}$ by an adjacent transposition. It can also be described as a Hamiltonian cycle on the Cayley graph for the group of all permutations on $\{1, 2, \ldots, n\}$, with the $n{-}1$ generators $((1\,2), (2\,3), \ldots, (n{-}1\;n))$. The *delta sequence* of such a Gray cycle is the sequence of integers $\delta_0\delta_1\ldots\delta_{n!-1}$ such that

$$\pi_{(k+1) \bmod n!} = (\delta_k\;\delta_k{+}1)\,\pi_k.$$

(See 7.2.1.1–(24), which describes the analogous situation for binary n-tuples.) For example, Fig. 23 illustrates the Gray cycle defined by plain changes when $n = 4$; its delta sequence is $(32131231)^3$.

a) Find all Gray cycles for permutations of $\{1, 2, 3, 4\}$.

b) Two Gray cycles are considered to be equivalent if their delta sequences can be obtained from each other by cyclic shifting $(\delta_k\ldots\delta_{n!-1}\delta_0\ldots\delta_{k-1})$ and/or reversal $(\delta_{n!-1}\ldots\delta_1\delta_0)$ and/or complementation $((n{-}\delta_0)(n{-}\delta_1)\ldots(n{-}\delta_{n!-1}))$. Which of the Gray cycles in (a) are equivalent?

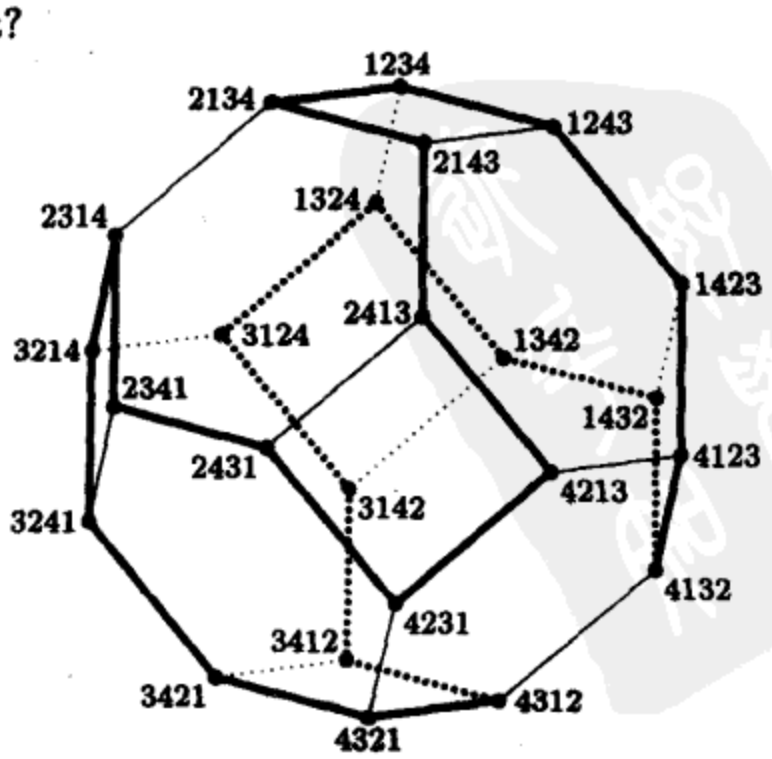


Fig. 23. Algorithm P traces out this Hamiltonian cycle on the truncated octahedron of Fig. 5–1.

61. [*21*] Continuing the previous exercise, a *Gray code for permutations* is like a Gray cycle except that the final permutation $\pi_{n!-1}$ is not required to be adjacent to the initial permutation π_0. Study the set of all Gray codes for $n = 4$ that start with 1234.

▸ **62.** [*M23*] What permutations can be reached as the final element of a Gray code that starts at $12\ldots n$?

63. [*M25*] Estimate the total number of Gray cycles for permutations of $\{1, 2, 3, 4, 5\}$.

64. [*23*] A "doubly Gray" code for permutations is a Gray cycle with the additional property that $\delta_{k+1} = \delta_k \pm 1$ for all k. Compton and Williamson have proved that such codes exist for all $n \geq 3$. How many doubly Gray codes exist for $n = 5$?

65. [*M25*] For which integers N is there a Gray path through the N lexicographically smallest permutations of $\{1, \ldots, n\}$? (Exercise 7.2.1.1–26 solves the analogous problem for binary n-tuples.)

66. [*22*] Ehrlich's swap method suggests another type of Gray cycle for permutations, in which the $n - 1$ generators are the star transpositions (1 2), (1 3), ..., (1 n). For example, Fig. 24 shows the relevant graph when $n = 4$. Analyze the Hamiltonian cycles of this graph.

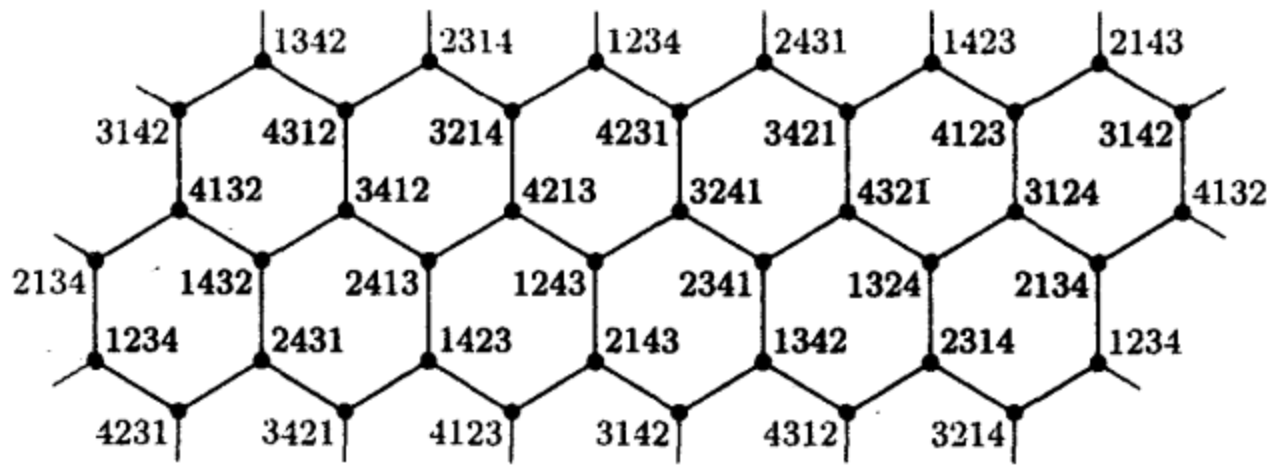


Fig. 24. The Cayley graph for permutations of $\{1, 2, 3, 4\}$, generated by the star transpositions (1 2), (1 3), and (1 4), drawn as a twisted torus.

67. [*26*] Continuing the previous exercise, find a first-element-swap Gray cycle for $n = 5$ in which each star transposition (1 j) occurs 30 times, for $2 \leq j \leq 5$.

68. [*M30*] (Kompel'makher and Liskovets, 1975.) Let G be the Cayley graph for all permutations of $\{1, \ldots, n\}$, with generators $(\alpha_1, \ldots, \alpha_k)$ where each α_j is a transposition $(u_j\ v_j)$; also let A be the graph with vertices $\{1, \ldots, n\}$ and edges u_j — v_j for $1 \leq j \leq k$. Prove that G has a Hamiltonian cycle if and only if A is connected. (Figure 23 is the special case when A is a path; Figure 24 is the special case when A is a "star.")

▸ **69.** [*28*] If $n \geq 4$, the following algorithm generates all permutations $A_1A_2A_3\ldots A_n$ of $\{1, 2, 3, \ldots, n\}$ using only three transformations,

$$\rho = (1\,2)(3\,4)(5\,6)\ldots, \qquad \sigma = (2\,3)(4\,5)(6\,7)\ldots, \qquad \tau = (3\,4)(5\,6)(7\,8)\ldots,$$

never applying ρ and τ next to each other. Explain why it works.

Z1. [Initialize.] Set $A_j \leftarrow j$ for $1 \leq j \leq n$. Also set $a_j \leftarrow 2j$ for $j \leq n/2$ and $a_{n-j} \leftarrow 2j + 1$ for $j < n/2$. Then invoke Algorithm P, but with parameter $n - 1$ instead of n. We will treat that algorithm as a coroutine, which should

return control to us whenever it "visits" $a_1 \dots a_{n-1}$ in step P2. We will also share its variables (except n).

Z2. [Set x and y.] Invoke Algorithm P again, obtaining a new permutation $a_1 \dots a_{n-1}$ and a new value of j. If $j = 2$, interchange $a_{1+s} \leftrightarrow a_{2+s}$ (thereby undoing the effect of step P5) and repeat this step; in such a case we are at the halfway point of Algorithm P. If $j = 1$ (so that Algorithm P has terminated), set $x \leftarrow y \leftarrow 0$ and go to Z3. Otherwise set

$$x \leftarrow a_{j-c_j+s+[o_j=-1]}, \qquad y \leftarrow a_{j-c_j+s-[o_j=+1]};$$

these are the two elements most recently interchanged in step P5.

Z3. [Visit.] Visit the permutation $A_1 \dots A_n$. Then go to Z5 if $A_1 = x$ and $A_2 = y$.

Z4. [Apply ρ, then σ.] Interchange $A_1 \leftrightarrow A_2$, $A_3 \leftrightarrow A_4$, $A_5 \leftrightarrow A_6$, ... Visit $A_1 \dots A_n$. Then interchange $A_2 \leftrightarrow A_3$, $A_4 \leftrightarrow A_5$, $A_6 \leftrightarrow A_7$, ... Terminate if $A_1 \dots A_n = 1 \dots n$, otherwise return to Z3.

Z5. [Apply τ, then σ.] Interchange $A_3 \leftrightarrow A_4$, $A_5 \leftrightarrow A_6$, $A_7 \leftrightarrow A_8$, ... Visit $A_1 \dots A_n$. Then interchange $A_2 \leftrightarrow A_3$, $A_4 \leftrightarrow A_5$, $A_6 \leftrightarrow A_7$, ..., and return to Z2. ∎

Hint: Show first that the algorithm works if modified so that $A_j \leftarrow n + 1 - j$ and $a_j \leftarrow j$ in step Z1, and if the "flip" permutations

$$\rho' = (1\ n)(2\ n{-}1)\dots, \qquad \sigma' = (2\ n)(3\ n{-}1)\dots, \qquad \tau' = (2\ n{-}1)(3\ n{-}2)\dots$$

are used instead of ρ, σ, τ in steps Z4 and Z5. In this modification, step Z3 should go to Z5 if $A_1 = x$ and $A_n = y$.

▸ **70.** [*M33*] The two 12-cycles (4_1) can be regarded as σ–τ cycles for the twelve permutations of $\{1, 1, 3, 4\}$:

$$1134 \to 1341 \to 3411 \to 4311 \to 3114 \to 1143 \to 1431 \\ \to 4131 \to 1314 \to 3141 \to 1413 \to 4113 \to 1134.$$

Replacing $\{1, 1\}$ by $\{1, 2\}$ yields disjoint cycles, and we obtained a Hamiltonian path by jumping from one to the other. Can a σ–τ path for all permutations of 6 elements be formed in a similar way, based on a 360-cycle for the permutations of $\{1, 1, 3, 4, 5, 6\}$?

71. [*48*] Does the Cayley graph with generators $\sigma = (1\,2 \dots n)$ and $\tau = (1\,2)$ have a Hamiltonian cycle whenever $n \geq 3$ is odd?

72. [*M21*] Given a Cayley graph with generators $(\alpha_1, \dots, \alpha_k)$, assume that each α_j takes $x \mapsto y$. (For example, both σ and τ in exercise 71 take $1 \mapsto 2$.) Prove that any Hamiltonian path starting at $12 \dots n$ in G must end at a permutation that takes $y \mapsto x$.

▸ **73.** [*M30*] Let α, β, and σ be permutations of a set X, where $X = A \cup B$. Assume that $x\sigma = x\alpha$ when $x \in A$ and $x\sigma = x\beta$ when $x \in B$, and that the order of $\alpha\beta^-$ is odd.

a) Prove that all three permutations α, β, σ have the same sign; that is, they are all even or all odd. *Hint:* A permutation has odd order if and only if its cycles all have odd length.

b) Derive Theorem R from part (a).

74. [*M30*] (R. A. Rankin.) Assuming that $\alpha\beta = \beta\alpha$ in Theorem R, prove that a Hamiltonian cycle exists if and only if there is a number k such that $0 \leq k \leq g/c$ and $t + k \perp c$, where $\beta^{g/c} = \gamma^t$, $\gamma = \alpha\beta^-$. *Hint:* Represent elements of the group in the form $\beta^j\gamma^k$.

75. [*M25*] The directed torus $O_m \times O_n$ has mn vertices (x, y) for $0 \le x < m$, $0 \le y < n$, and arcs $(x, y) \to (x, y)\alpha = ((x+1) \bmod m, y)$, $(x, y) \to (x, y)\beta = (x, (y+1) \bmod n)$. Prove that, if $m > 1$ and $n > 1$, the number of Hamiltonian cycles of this digraph is

$$\sum_{k=1}^{d-1} \binom{d}{k} [\gcd((d-k)m, kn) = d], \qquad d = \gcd(m, n).$$

76. [*M31*] The cells numbered 0, 1, ..., 63 in Fig. 25 illustrate a *northeasterly knight's tour* on an 8×8 torus: If k appears in cell (x_k, y_k), then $(x_{k+1}, y_{k+1}) \equiv (x_k + 2, y_k + 1)$ or $(x_k + 1, y_k + 2)$, modulo 8, and $(x_{64}, y_{64}) = (x_0, y_0)$. How many such tours are possible on an $m \times n$ torus, when $m, n \ge 3$?

29	24	19	14	49	44	39	34
58	53	48	43	38	9	4	63
23	18	13	8	3	62	33	28
52	47	42	37	32	27	22	57
17	12	7	2	61	56	51	46
6	41	36	31	26	21	16	11
35	30	1	60	55	50	45	40
0	59	54	25	20	15	10	5

Fig. 25. A northeasterly knight's tour.

▶ **77.** [*22*] Complete the MMIX program whose inner loop appears in (42), using Heap's method (27).

78. [*M23*] Analyze the running time of the program in exercise 77, generalizing it so that the inner loop does $r!$ visits (with $a_0 \dots a_{r-1}$ in global registers).

79. [*20*] What seven MMIX instructions will ⟨ Swap the nybbles ... ⟩ as (45) requires? For example, if register t contains the value 4 and register a contains the nybbles #12345678, register a should change to #12345687.

80. [*21*] Solve the previous exercise with only five MMIX instructions. *Hint:* Use MXOR.

▶ **81.** [*22*] Complete the MMIX program (46) by specifying how to ⟨ Continue with Langdon's method ⟩.

82. [*M21*] Analyze the running time of the program in exercise 81.

83. [*22*] Use the σ–τ path of exercise 70 to design an MMIX routine analogous to (42) that generates all permutations of #123456 in register a.

84. [*20*] Suggest a good way to generate all $n!$ permutations of $\{1, \dots, n\}$ on p processors that are running in parallel.

▶ **85.** [*25*] Assume that n is small enough that $n!$ fits in a computer word. What's a good way to convert a given permutation $\alpha = a_1 \dots a_n$ of $\{1, \dots, n\}$ into an integer $k = r(\alpha)$ in the range $0 \le k < n!$? Both functions $k = r(\alpha)$ and $\alpha = r^{[-1]}(k)$ should be computable in only $O(n)$ steps.

86. [*20*] A partial order relation is supposed to be transitive; that is, $x \prec y$ and $y \prec z$ should imply $x \prec z$. But Algorithm V does not require its input relation to satisfy this condition.

Show that if $x \prec y$ and $y \prec z$, Algorithm V will produce identical results whether or not $x \prec z$.

87. [*20*] (F. Ruskey.) Consider the inversion tables $c_1 \dots c_n$ of the permutations visited by Algorithm V. What noteworthy property do they have? (Compare with the inversion tables (4) in Algorithm P.)

88. [*21*] Show that Algorithm V can be used to generate all ways to partition the digits $\{0, 1, \ldots, 9\}$ into two 3-element sets and two 2-element sets.

▸ **89.** [*M30*] Consider the numbers $t_0, t_1, \ldots, t_n$ in (51). Clearly $t_0 = t_1 = 1$.

a) Say that index j is "trivial" if $t_j = t_{j-1}$. For example, 9 is trivial with respect to the Young tableau relations (48). Explain how to modify Algorithm V so that the variable k takes on only nontrivial values.

b) Analyze the running time of the modified algorithm. What formulas replace (51)?

c) Say that the interval $[j\,..\,k]$ is not a chain if we do not have $l \prec l+1$ for $j \le l < k$. Prove that in such a case $t_k \ge 2t_{j-1}$.

d) Every inverse topological sort $a'_1 \ldots a'_n$ defines a labeling that corresponds to relations $a'_{j_1} \prec a'_{k_1}, \ldots, a'_{j_m} \prec a'_{k_m}$, which are equivalent to the original relations $j_1 \prec k_1, \ldots, j_m \prec k_m$. Explain how to find a labeling such that $[j\,..\,k]$ is not a chain when j and k are consecutive nontrivial indices.

e) Prove that with such a labeling, $M < 4N$ in the formulas of part (b).

90. [*M21*] Algorithm V can be used to produce all permutations that are h-ordered for all h in a given set, namely all $a'_1 \ldots a'_n$ such that $a'_j < a'_{j+h}$ for $1 \le j \le n - h$ (see Section 5.2.1). Analyze the running time of Algorithm V when it generates all permutations that are both 2-ordered and 3-ordered.

91. [*HM21*] Analyze the running time of Algorithm V when it is used with the relations (49) to find matchings.

92. [*M18*] How many permutations is Algorithm V likely to visit, in a "random" case? Let P_n be the number of partial orderings on $\{1, \ldots, n\}$, namely the number of relations that are reflexive, antisymmetric, and transitive. Let Q_n be the number of such relations with the additional property that $j < k$ whenever $j \prec k$. Express the expected number of ways to sort n elements topologically, averaged over all partial orderings, in terms of P_n and Q_n.

93. [*35*] Prove that all topological sorts can be generated in such a way that only one or two adjacent transpositions are made at each step. (The example $1 \prec 2$, $3 \prec 4$ shows that a single transposition per step cannot always be achieved, even if we allow nonadjacent swaps, because only two of the six relevant permutations are odd.)

▸ **94.** [*25*] Show that in the case of matchings, using the relations in (49), all topological sorts can be generated with just one transposition per step.

95. [*21*] Discuss how to generate all *up-down permutations* of $\{1, \ldots, n\}$, namely those $a_1 \ldots a_n$ such that $a_1 < a_2 > a_3 < a_4 > \cdots$.

96. [*21*] Discuss how to generate all *cyclic permutations* of $\{1, \ldots, n\}$, namely those $a_1 \ldots a_n$ whose cycle representation consists of a single n-cycle.

97. [*21*] Discuss how to generate all *derangements* of $\{1, \ldots, n\}$, namely those $a_1 \ldots a_n$ such that $a_1 \ne 1$, $a_2 \ne 2$, $a_3 \ne 3$, ...

98. [*HM23*] Analyze the asymptotic running time of the method in the previous exercise.

99. [*M30*] Given $n \ge 3$, show that all derangements of $\{1, \ldots, n\}$ can be generated by making at most two transpositions between visits.

100. [*21*] Discuss how to generate all *indecomposable* permutations of $\{1, \ldots, n\}$, namely those $a_1 \ldots a_n$ such that $\{a_1, \ldots, a_j\} \ne \{1, \ldots, j\}$ for $1 \le j < n$.

101. [*21*] Discuss how to generate all *involutions* of $\{1, \ldots, n\}$, namely those permutations $a_1 \ldots a_n$ with $a_{a_1} \ldots a_{a_n} = 1 \ldots n$.

102. [*M30*] Show that all involutions of $\{1,\ldots,n\}$ can be generated by making at most two transpositions between visits.

103. [*M32*] Show that all even permutations of $\{1,\ldots,n\}$ can be generated by successive *rotations of three consecutive elements.*

▸ **104.** [*M22*] A permutation $a_1\ldots a_n$ of $\{1,\ldots,n\}$ is *well-balanced* if

$$\sum_{k=1}^{n} ka_k = \sum_{k=1}^{n}(n+1-k)a_k.$$

For example, 3142 is well-balanced when $n=4$.

a) Prove that no permutation is well-balanced when $n \bmod 4 = 2$.

b) Prove that if $a_1\ldots a_n$ is well-balanced, so are its reversal $a_n\ldots a_1$, its complement $(n+1-a_1)\ldots(n+1-a_n)$, and its inverse $a'_1\ldots a'_n$.

c) Determine the number of well-balanced permutations for small values of n.

▸ **105.** [*26*] A *weak order* is a relation $\preceq$ that is transitive ($x\preceq y$ and $y\preceq z$ implies $x\preceq z$) and complete ($x\preceq y$ or $y\preceq x$ always holds). We can write $x\equiv y$ if $x\preceq y$ and $y\preceq x$; $x\prec y$ if $x\preceq y$ and $y\not\preceq x$. There are thirteen weak orders on three elements $\{1,2,3\}$, namely

$$1\equiv 2\equiv 3,\quad 1\equiv 2\prec 3,\quad 1\prec 2\equiv 3,\quad 1\prec 2\prec 3,\quad 1\equiv 3\prec 2,\quad 1\prec 3\prec 2,$$
$$2\prec 1\equiv 3,\quad 2\prec 1\prec 3,\quad 2\equiv 3\prec 1,\quad 2\prec 3\prec 1,\quad 3\prec 1\equiv 2,\quad 3\prec 1\prec 2,\quad 3\prec 2\prec 1.$$

a) Explain how to generate all weak orders of $\{1,\ldots,n\}$ systematically, as sequences of digits separated by the symbols $\equiv$ or $\prec$.

b) A weak order can also be represented as a sequence $a_1\ldots a_n$ where $a_j=k$ if j is preceded by k $\prec$ signs. For example, the thirteen weak orders on $\{1,2,3\}$ are respectively 000, 001, 011, 012, 010, 021, 101, 102, 100, 201, 110, 120, 210 in this form. Find a simple way to generate all such sequences of length n.

106. [*M40*] Can exercise 105(b) be solved with a Gray-like code?

▸ **107.** [*30*] (John H. Conway, 1973.) To play the solitaire game of "topswops," start by shuffling a pack of n cards labeled $\{1,\ldots,n\}$ and place them face up in a pile. Then if the top card is $k>1$, deal out the top k cards and put them back on top of the pile, thereby changing the permutation from $a_1\ldots a_n$ to $a_k\ldots a_1 a_{k+1}\ldots a_n$. Continue until the top card is 1. For example, the 7-step sequence

$$31452 \to 41352 \to 53142 \to 24135 \to 42135 \to 31245 \to 21345 \to 12345$$

might occur when $n=5$. What is the longest sequence possible when $n=13$?

108. [*M27*] If the longest n-card game of topswops has length $f(n)$, prove that $f(n)\le F_{n+1}-1$.

109. [*M47*] Find good upper and lower bounds on the topswops function $f(n)$.

▸ **110.** [*25*] Find all permutations $a_0\ldots a_9$ of $\{0,\ldots,9\}$ such that

$$\{a_0,a_2,a_3,a_7\}=\{2,5,7,8\},$$
$$\{a_1,a_4,a_5\}=\{0,3,6\},$$
$$\{a_1,a_3,a_7,a_8\}=\{3,4,5,7\},$$
$$\{a_0,a_3,a_4\}=\{0,7,8\}.$$

Also suggest an algorithm for solving large problems of this type.

▸ **111.** [*M25*] Several permutation-oriented analogs of de Bruijn cycles have been proposed. The simplest and nicest of these is the notion of a *universal cycle of permutations*, introduced by B. W. Jackson in *Discrete Math.* **117** (1993), 141–150, namely a cycle of $n!$ digits such that each permutation of $\{1,\ldots,n\}$ occurs exactly once as a block of $n-1$ consecutive digits (with its redundant final element suppressed). For example, (121323) is a universal cycle of permutations for $n=3$, and it is essentially the only such cycle.

Find a universal cycle of permutations for $n=4$, and prove that such cycles exist for all $n \geq 2$.

▸ **112.** [*HM43*] Exactly how many universal cycles exist, for permutations of ≤ 9 objects?



ANSWERS TO EXERCISES

SECTION 7.2.1.1

1. Let $m_j = u_j - l_j + 1$, and visit $(a_1 + l_1, \ldots, a_n + l_n)$ instead of visiting $(a_1, \ldots, a_n)$ in Algorithm M. Or, change '$a_j \leftarrow 0$' to '$a_j \leftarrow l_j$' and '$a_j = m_j - 1$' to '$a_j = u_j$' in that algorithm, and set $l_0 \leftarrow 0$, $u_0 \leftarrow 1$ in step M1.

2. $(0, 0, 1, 2, 3, 0, 2, 7, 0, 9)$.

3. Step M4 is performed $m_1 m_2 \ldots m_k$ times when $j = k$; therefore the total is $\sum_{k=0}^{n} \prod_{j=1}^{k} m_j = m_1 \ldots m_n (1 + 1/m_n + 1/m_n m_{n-1} + \cdots + 1/m_n \ldots m_1)$. If all m_j are 2 or more, this is less than $2m_1 \ldots m_n$. [Thus, we should keep in mind that fancy Gray-code methods, which change only one digit per visit, actually reduce the total number of digit changes by at most a factor of 2.]

4. **N1.** [Initialize.] Set $a_j \leftarrow m_j - 1$ for $0 \le j \le n$, where $m_0 = 2$.

N2. [Visit.] Visit the n-tuple $(a_1, \ldots, a_n)$.

N3. [Prepare to subtract one.] Set $j \leftarrow n$.

N4. [Borrow if necessary.] If $a_j = 0$, set $a_j \leftarrow m_j - 1$, $j \leftarrow j - 1$, and repeat this step.

N5. [Decrease, unless done.] If $j = 0$, terminate the algorithm. Otherwise set $a_j \leftarrow a_j - 1$ and go back to step N2. ∎

5. Bit reflection is easy on a machine like MMIX, but on other computers we can proceed as follows:

R1. [Initialize.] Set $j \leftarrow k \leftarrow 0$.

R2. [Swap.] Interchange $A[j + 1] \leftrightarrow A[k + 2^{n-1}]$. Also, if $j < k$, interchange $A[j] \leftrightarrow A[k]$ and $A[j + 2^{n-1} + 1] \leftrightarrow A[k + 2^{n-1} + 1]$.

R3. [Advance k.] Set $k \leftarrow k + 2$, and terminate if $k \ge 2^{n-1}$.

R4. [Advance j.] Set $h \leftarrow 2^{n-2}$. If $j \ge h$, repeatedly set $j \leftarrow j - h$ and $h \leftarrow h/2$ until $j < h$. Then set $j \leftarrow j + h$. (Now $j = (b_0 \ldots b_{n-1})_2$ if $k = (b_{n-1} \ldots b_0)_2$.) Return to R2. ∎

6. If $g((0b_{n-1} \ldots b_1 b_0)_2) = (0(b_{n-1}) \ldots (b_2 \oplus b_1)(b_1 \oplus b_0))_2$ then $g((1b_{n-1} \ldots b_1 b_0)_2) = 2^n + g((0\bar{b}_{n-1} \ldots \bar{b}_1 \bar{b}_0)_2) = (1(\bar{b}_{n-1}) \ldots (\bar{b}_2 \oplus \bar{b}_1)(\bar{b}_1 \oplus \bar{b}_0))_2$, where $\bar{b} = b \oplus 1$.

7. To accommodate $2r$ sectors one can use $g(k)$ for $2^n - r \le k < 2^n + r$, where $n = \lceil \lg r \rceil$, because $g(2^n - r) \oplus g(2^n + r - 1) = 2^n$ by (5). [G. C. Tootill, *Proc. IEE* **103**, Part B Supplement (1956), 434.] See also exercise 26.

8. Use Algorithm G with $n \leftarrow n-1$ and include the parity bit a_∞ at the right. (This yields $g(0)$, $g(2)$, $g(4)$, ...)

9. Replace the rightmost ring, since $\nu(1011000)$ is odd.

10. $A_n + B_n = g^{[-1]}(2^n - 1) = \lfloor 2^{n+1}/3 \rfloor$ and $A_n = B_n + n$. Hence $A_n = \lfloor 2^n/3 + n/2 \rfloor$ and $B_n = \lfloor 2^n/3 - n/2 \rfloor$.

Historical notes: The early Japanese mathematician Yoriyuki Arima (1714–1783) treated this problem in his *Shūki Sanpō* (1769), Problem 44, observing that the n-ring puzzle reduces to an $(n-1)$-ring puzzle after a certain number of steps. Let $C_n = A_n - A_{n-1} = B_n - B_{n-1} + 1$ be the number of rings removed during this reduction. Arima noticed that $C_n = 2C_{n-1} - [n \text{ even}]$; thus he could compute $A_n = C_1 + C_2 + \cdots + C_n$ for $n = 9$ without actually knowing the formula $C_n = \lceil 2^{n-1}/3 \rceil$.

More than two centuries earlier, Cardano had already mentioned the "complicati annuli" in his *De Subtilitate Libri XXI* (Nuremberg: 1550), Book 15. He wrote that they are "useless yet admirably subtle," stating erroneously that 95 moves are needed to remove seven rings and 95 more to put them back. John Wallis devoted seven pages to this puzzle in the Latin edition of his *Algebra* **2** (Oxford: 1693), Chapter 111, presenting detailed but nonoptimum methods for the nine-ring case. He included the operation of sliding a ring through the bar as well as putting it on or off, and he hinted that shortcuts were available, but he did not attempt to find a shortest solution.

11. The solution to $S_n = S_{n-2} + 1 + S_{n-2} + S_{n-1}$ when $S_1 = S_2 = 1$ is $S_n = 2^{n-1} - [n \text{ even}]$. [*Math. Quest. Educational Times* **3** (1865), 66–67.]

12. (a) The theory of $n-1$ Chinese rings proves that Gray binary code yields the compositions in a convenient order (4, 31, 211, 22, 112, 1111, 121, 13):

A1. [Initialize.] Set $t \leftarrow 0$, $j \leftarrow 1$, $s_1 \leftarrow n$. (We assume that $n > 1$.)

A2. [Visit.] Visit $s_1 \ldots s_j$. Then set $t \leftarrow 1 - t$, and go to A4 if $t = 0$.

A3. [Odd step.] If $s_j > 1$, set $s_j \leftarrow s_j - 1$, $s_{j+1} \leftarrow 1$, $j \leftarrow j + 1$; otherwise set $j \leftarrow j - 1$ and $s_j \leftarrow s_j + 1$. Return to A2.

A4. [Even step.] If $s_{j-1} > 1$, set $s_{j-1} \leftarrow s_{j-1} - 1$, $s_{j+1} \leftarrow s_j$, $s_j \leftarrow 1$, $j \leftarrow j + 1$; otherwise set $j \leftarrow j - 1$, $s_j \leftarrow s_{j+1}$, $s_{j-1} \leftarrow s_{j-1} + 1$ (but terminate if $j - 1 = 0$). Return to A2. ▮

(b) Now $q_1, \ldots, q_{t-1}$ represent rings on the bar:

B1. [Initialize.] Set $t \leftarrow 1$, $q_0 \leftarrow n$. (We assume that $n > 1$.)

B2. [Visit.] Set $q_t \leftarrow 0$ and visit $(q_0 - q_1) \ldots (q_{t-1} - q_t)$. Go to B4 if t is even.

B3. [Odd step.] If $q_{t-1} = 1$, set $t \leftarrow t - 1$; otherwise set $q_t \leftarrow 1$ and $t \leftarrow t + 1$. Return to step B2.

B4. [Even step.] If $q_{t-2} = q_{t-1} + 1$, set $q_{t-2} \leftarrow q_{t-1}$ and $t \leftarrow t - 1$ (but terminate if $t = 2$); otherwise set $q_t \leftarrow q_{t-1}$, $q_{t-1} \leftarrow q_t + 1$, $t \leftarrow t + 1$. Return to B2. ▮

These algorithms [see J. Misra, *ACM Trans. Math. Software* **1** (1975), 285] are loopless even in their initialization steps.

13. In step A1, also set $C \leftarrow 1$. In step A3, set $C \leftarrow s_j C$ if $s_j > 1$, otherwise $C \leftarrow C/(s_{j-1}+1)$. In step A4, set $C \leftarrow s_{j-1} C$ if $s_{j-1} > 1$, otherwise $C \leftarrow C/(s_{j-2}+1)$.

Similar modifications apply to steps B1, B3, B4. Sufficient precision is needed to accommodate the value $C = n!$ for the composition $1 \ldots 1$; we are stretching the definition of looplessness by assuming that arithmetic operations take unit time.

14. **S1.** [Initialize.] Set $j \leftarrow 0$.

S2. [Visit.] Visit the string $a_1 \ldots a_j$.

S3. [Lengthen.] If $j < n$, set $j \leftarrow j+1$, $a_j \leftarrow 0$, and return to S2.

S4. [Increase.] If $a_j < m_j - 1$, set $a_j \leftarrow a_j + 1$ and return to S2.

S5. [Shorten.] Set $j \leftarrow j-1$, and return to S4 if $j > 0$. ∎

15. **T1.** [Initialize.] Set $j \leftarrow 0$.

T2. [Even visit.] If j is even, visit the string $a_1 \ldots a_j$.

T3. [Lengthen.] If $j < n$, set $j \leftarrow j+1$, $a_j \leftarrow 0$, and return to T2.

T4. [Odd visit.] If j is odd, visit the string $a_1 \ldots a_j$.

T5. [Increase.] If $a_j < m_j - 1$, set $a_j \leftarrow a_j + 1$ and return to T2.

T6. [Shorten.] Set $j \leftarrow j-1$, and return to T4 if $j > 0$. ∎

This algorithm is loopless, although it may appear at first glance to contain loops; at most four steps separate consecutive visits. The basic idea is related to exercise 2.3.1–5 and to "prepostorder" traversal (Algorithm 7.2.1.6Q).

16. Suppose $\texttt{LINK}(j-1) = j + nb_j$ for $1 \le j \le n$ and $\texttt{LINK}(j-1+n) = j + n(1-b_j)$ for $1 < j \le n$. These links represent $(a_1, \ldots, a_n)$ if and only if $g(b_1 \ldots b_n) = a_1 \ldots a_n$, so we can use a loopless Gray binary generator to achieve the desired result.

17. Put the concatenation of 3-bit codes $(g(j), g(k))$ in row j and column k, for $0 \le j, k < 8$. [It is not difficult to prove that this is essentially the *only* solution, except for permuting and/or complementing coordinates and/or rotating rows, because the coordinate that changes when moving north or south depends only on the row, and a similar statement applies to columns. Karnaugh's isomorphism between the 4-cube and the 4×4 torus can be traced back to *The Design of Switching Circuits* by W. Keister, A. E. Ritchie, and S. H. Washburn (1951), page 174. Incidentally, Keister went on to design an ingenious variant of Chinese rings called SpinOut, and a generalization called The Hexadecimal Puzzle, *U.S. Patents 3637215–3637216* (1972).]

18. Use 2-bit Gray code to represent the digits $u_j = (0, 1, 2, 3)$ respectively as the bit pairs $u'_{2j-1}u'_{2j} = (00, 01, 11, 10)$. [C. Y. Lee introduced his metric in *IEEE Trans.* **IT-4** (1958), 77–82. A similar $m/2$-bit encoding works for even values of m; for example, when $m = 8$ we can represent $(0, 1, 2, 3, 4, 5, 6, 7)$ by $(0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000)$. But such a scheme leaves out some of the binary patterns when $m > 4$.]

19. (a) A modular Gray quaternary algorithm needs slightly less computation than Algorithm M, but it doesn't matter because 256 is so small. The result is $z_0^8 + z_1^8 + z_2^8 + z_3^8 + 14(z_0^4 z_2^4 + z_1^4 z_3^4) + 56 z_0 z_1 z_2 z_3 (z_0^2 + z_2^2)(z_1^2 + z_3^2)$.

(b) Replacing (z_0, z_1, z_2, z_3) by $(1, z, z^2, z)$ gives $1 + 112z^6 + 30z^8 + 112z^{10} + z^{16}$; thus all of the nonzero Lee weights are ≥ 6. Now use the construction in the previous exercise to convert each $(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$ into a 16-bit number.

20. Recover the quaternary vector $(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$ from u', and use Algorithm 4.6.1D to find the remainder of $u_0 + u_1 x + \cdots + u_6 x^6$ divided by $g(x)$, mod 4; that algorithm can be used in spite of the fact that the coefficients do not belong to a field, because $g(x)$ is monic. Express the remainder as $x^j + 2x^k$ (modulo $g(x)$ and 4), and let $d = (k-j) \bmod 7$, $s = (u_0 + \cdots + u_6 + u_\infty) \bmod 4$.

Case 1, $s = 1$: If $k = \infty$, the error was x^j (in other words, the correct vector has $u_j \leftarrow (u_j - 1) \bmod 4$); otherwise there were three or more errors.

Case 2, $s = 3$: If $j = k$ the error was $-x^j$; otherwise ≥ 3 errors occurred.

Case 3, $s = 0$: If $j = k = \infty$, no errors were made; if $j = \infty$ and $k < \infty$, at least four errors were made. Otherwise the errors were $x^a - x^b$, where $a = (j + (\infty, 6, 5, 2, 3, 1, 4, 0)) \bmod 7$ according as $d = (0, 1, 2, 3, 4, 5, 6, \infty)$, and $b = (j+2d) \bmod 7$.

Case 4, $s = 2$: If $j = \infty$ the errors were $2x^k$. Otherwise the errors were

$$\begin{aligned} x^j + x^\infty&, \text{ if } k = \infty; \\ -x^j - x^\infty&, \text{ if } d = 0; \\ x^a + x^b&, \text{ if } d \in \{1, 2, 4\},\ a = (j - 3d) \bmod 7,\ b = (j - 2d) \bmod 7; \\ -x^a - x^b&, \text{ if } d \in \{3, 5, 6\},\ a = (j - 3d) \bmod 7,\ b = (j - d) \bmod 7. \end{aligned}$$

Given $u' = (1100100100001111)_2$, we have $u = (2, 0, 3, 1, 0, 0, 2, 2)$ and $2 + 3x^2 + x^3 + 2x^6 \equiv 1 + 3x + 3x^2 \equiv x^5 + 2x^6$; also $s = 2$. Thus the errors are $x^2 + x^3$, and the nearest errorfree codeword is $(2, 0, 2, 0, 0, 0, 2, 2)$. Algorithm 4.6.1D tells us that $2+2x^2+2x^6 \equiv (2+2x+2x^3)g(x)$ (modulo 4); so the eight information bits correspond to $(v_0, v_1, v_2, v_3) = (2, 2, 0, 2)$. [A more intelligent algorithm would also say, "Aha: The first 16 bits of π."]

For generalizations to other efficient coding schemes based on quaternary vectors, see the classic paper by Hammons, Kumar, Calderbank, Sloane, and Solé, *IEEE Trans.* **IT-40** (1994), 301–319.

21. (a) $C(\epsilon) = 1$, $C(0\alpha) = C(1\alpha) = C(\alpha)$, and $C(*\alpha) = 2C(\alpha) - [10\ldots0 \in \alpha]$. Iterating this recurrence gives $C(\alpha) = 2^t - 2^{t-1}e_t - 2^{t-2}e_{t-1} - \cdots - 2^0 e_1$, where $e_j = [10\ldots0 \in \alpha_j]$ and α_j is the suffix of α following the jth asterisk. In the example we have $\alpha_1 = *10{*}{*}0*$, $\alpha_2 = 10{*}{*}0*$, $\ldots$, $\alpha_5 = \epsilon$; thus $e_1 = 0$, $e_2 = 1$, $e_3 = 1$, $e_4 = 0$, and $e_5 = 1$ (by convention), hence $C(**10{*}{*}0*) = 2^5 - 2^4 - 2^2 - 2^1 = 10$.

(b) We may remove trailing asterisks so that $t = t'$. Then $e_t = 1$ implies $e_{t-1} = \cdots = e_1 = 0$. [The case $C(\alpha) = 2^{t'-1}$ occurs if and only if α ends in 10^j*^k.]

(c) To compute the sum of $C(\alpha)$ over all t-subcubes, note that $\binom{n}{t}$ clusters begin at the n-tuple $0\ldots0$, and $\binom{n-1}{t}$ begin at each succeeding n-tuple (namely one cluster for each t-subcube containing that n-tuple and specifying the bit that changed). Thus the average is $\left(\binom{n}{t} + (2^n - 1)\binom{n-1}{t}\right)/2^{n-t}\binom{n}{t} = 2^t(1 - t/n) + 2^{t-n}(t/n)$. [The formula in (c) holds for *any* n-bit Gray path, but (a) and (b) are specific to the reflected Gray binary code. These results are due to C. Faloutsos, *IEEE Trans.* **SE-14** (1988), 1381–1393.]

22. Let $\alpha*^j$ and $\beta*^k$ be consecutive lieves of a Gray binary trie, where α and β are binary strings and $j \leq k$. Then the last $k - j$ bits of α are a string α' such that α and $\beta\alpha'$ are consecutive elements of Gray binary code, hence adjacent. [Interesting applications of this property to cube-connected message-passing concurrent computers are discussed in *A VLSI Architecture for Concurrent Data Structures* by William J. Dally (Kluwer, 1987), Chapter 3.]

23. $2^j = g(k) \oplus g(l) = g(k \oplus l)$ implies that $l = k \oplus g^{[-1]}(2^j) = k \oplus (2^{j+1} - 1)$. In other words, if $k = (b_{n-1}\ldots b_0)_2$ we have $l = (b_{n-1}\ldots b_{j+1}\bar{b}_j\ldots\bar{b}_0)_2$.

24. Defining $g(k) = k \oplus \lfloor k/2 \rfloor$ as usual, we find $g(k) = g(-1 - k)$; hence there are *two* 2-adic integers k such that $g(k)$ has a given 2-adic value l. One of them is even, the other is odd. We can conveniently define $g^{[-1]}$ to be the solution that is even; then (8) is replaced by $b_j = a_{j-1} \oplus \cdots \oplus a_0$, for $j \geq 0$. For example, $g^{[-1]}(1) = -2$ by this definition; when l is a normal integer, the "sign" of $g^{[-1]}(l)$ is the parity of l.

25. Let $p = k \oplus l$; exercise 7.1–00 tells us that $2^{\lfloor \lg p \rfloor + 1} - p \le |k - l| \le p$. We have $\nu(g(p)) = \nu(g(k) \oplus g(l)) = t$ if and only if there are positive integers $j_1, \ldots, j_t$ such that $p = (1^{j_1}0^{j_2}1^{j_3}\ldots(0 \text{ or } 1)^{j_t})_2$. The largest possible $p < 2^n$ occurs when $j_1 = n + 1 - t$ and $j_2 = \cdots = j_t = 1$, yielding $p = 2^n - \lceil 2^t/3 \rceil$. The smallest possible $2^{\lfloor \lg p \rfloor + 1} - p = (1^{j_2}0^{j_3}\ldots(1 \text{ or } 0)^{j_t})_2 + 1$ occurs when $j_2 = \cdots = j_t = 1$, yielding $p = \lceil 2^t/3 \rceil$. [C. K. Yuen, *IEEE Trans.* **IT-20** (1974), 668; S. R. Cavior, *IEEE Trans.* **IT-21** (1975), 596.]

26. Let $N = 2^{n_t} + \cdots + 2^{n_1}$ where $n_t > \cdots > n_1 \ge 0$; also, let Γ_n be any Gray code for $\{0, 1, \ldots, 2^n - 1\}$ that begins at 0 and ends at 1, except that Γ_0 is simply 0. Use

$$\Gamma^R_{n_t},\ 2^{n_t}+\Gamma_{n_{t-1}},\ \ldots,\ 2^{n_t}+\cdots+2^{n_3}+\Gamma^R_{n_2},\ 2^{n_t}+\cdots+2^{n_2}+\Gamma_{n_1},\ \text{if } t \text{ is even;}$$

$$\Gamma_{n_t},\ 2^{n_t}+\Gamma^R_{n_{t-1}},\ \ldots,\ 2^{n_t}+\cdots+2^{n_3}+\Gamma^R_{n_2},\ 2^{n_t}+\cdots+2^{n_2}+\Gamma_{n_1},\ \text{if } t \text{ is odd.}$$

27. In general, if $k = (b_{n-1}\ldots b_0)_2$, the $(k+1)$st largest element of S_n is equal to

$$1/(2-(-1)^{a_{n-1}}/(2-\cdots/(2-(-1)^{a_1}/(2-(-1)^{a_0}))\ldots)),$$

corresponding to the sign pattern $g(k) = (a_{n-1}\ldots a_0)_2$. Thus we can compute any element of S_n in $O(n)$ steps, given its rank. Setting $k = 2^{100} - 10^{10}$ and $n = 100$ yields the answer 373065177/1113604409. [Whenever $f(x)$ is a positive and monotonic function, the 2^n elements $f(\pm f(\ldots \pm f(\pm x)\ldots))$ are ordered according to Gray binary code, as observed by H. E. Salzer, *CACM* **16** (1973), 180. In this particular case there is, however, another way to get the answer, because we also have $S_n = //2, \pm 2, \ldots, \pm 2, \pm 1//$ using the notation of Section 4.5.3; continued fractions in this form are ordered by complementing alternate bits of k.]

28. (a) As $t = 1, 2, \ldots$, bit a_j of median(G_t) runs through the periodic sequence

$$0, \ldots, 0, *, 1, \ldots, 1, *, 0, \ldots, 0, *, \ldots$$

with asterisks at every 2^{1+j}th step. Thus the strings that correspond to the binary representations of $\lfloor (t-1)/2 \rfloor$ and $\lfloor t/2 \rfloor$ are medians. And those strings are in fact "extreme" cases, in the sense that all medians agree with the common bits of $\lfloor (t-1)/2 \rfloor$ and $\lfloor t/2 \rfloor$, hence asterisks appear where they disagree. For example, when $t = 100 = (01100100)_2$ and $n = 8$, we have median$(G_{100}) = 001100{*}{*}$.

(b) Since $G_{2t} = 2G_t \cup (2G_t + 1)$, we may assume that $t = (a_{n-2}\ldots a_1 a_0 1)_2$ is odd. If α is $g(p)$ and β is $g(q)$ in Gray binary, we have $p = (p_{n-1}\ldots p_0)_2$ and $q = (p_{n-1}\ldots p_{j+1}\overline{p}_j\ldots\overline{p}_0)_2$; and $a_{n-1}a_{n-2} = 01 = p_{n-1}p_{n-2}$. We cannot have $p < t \le q$, because this would imply that $j = n - 1$ and $p_{n-3} = p_{n-4} = \cdots = p_0 = 1$. [See A. J. Bernstein, K. Steiglitz, and J. E. Hopcroft, *IEEE Trans.* **IT-12** (1966), 425–430.]

29. Assuming that $p \ne 0$, let $l = \lfloor \lg p \rfloor$ and $S_a = \{s \mid 2^l a \le s < 2^l(a+1)\}$ for $0 \le a < 2^{n-l}$. Then $(k \oplus p) - k$ has a constant sign for all $k \in S_a$, and

$$\sum_{k \in S_a} \Big|(k \oplus p) - k\Big| = 2^l |S_a| = 2^{2l}.$$

Also $g^{[-1]}(g(k) \oplus p) = k \oplus g^{[-1]}(p)$, and $\lfloor \lg g^{[-1]}(p) \rfloor = \lfloor \lg p \rfloor$. Therefore

$$\frac{1}{2^n}\sum_{k=0}^{2^n-1}\Big|g^{[-1]}(g(k)\oplus p) - k\Big| = \frac{1}{2^n}\sum_{a=0}^{2^{n-l}-1}\sum_{k\in S_a}\Big|(k \oplus g^{[-1]}(p)) - k\Big| = \frac{1}{2^n}\sum_{a=0}^{2^{n-l}-1} 2^{2l} = 2^l.$$

[See Morgan M. Buchner, Jr., *Bell System Tech. J.* **48** (1969), 3113–3130.]

30. The cycle containing $k > 1$ has length $2^{\lfloor \lg \lg k \rfloor + 1}$, because it is easy to show from Eq. (7) that if $k = (b_{n-1} \dots b_0)_2$ we have

$$g^{[2^l]}(k) = (c_{n-1} \dots c_0)_2, \qquad \text{where } c_j = b_j \oplus b_{j+l+1}.$$

To permute all elements k such that $\lfloor \lg k \rfloor = t$, there are two cases: If t is a power of 2, the cycle containing $2\lfloor k/2 \rfloor$ also contains $2\lfloor k/2 \rfloor + 1$, so we must double the cycle leaders for $t-1$. Otherwise the cycle containing $2\lfloor k/2 \rfloor$ is disjoint from the cycle containing $2\lfloor k/2 \rfloor + 1$, so $L_t = (2L_{t-1}) \cup (2L_{t-1}+1) = (L_{t-1}*)_2$. This argument, discovered by Jörg Arndt in 2001, establishes the hint and yields the following algorithm:

P1. [Initialize.] Set $t \leftarrow 1$, $m \leftarrow 0$. (We may assume that $n \geq 2$.)

P2. [Loop through leaders.] Set $r \leftarrow m$. Perform Algorithm Q with $k = 2^t + r$; then if $r > 0$, set $r \leftarrow (r-1) \& m$ and repeat until $r = 0$. [See exercise 7.1–00.]

P3. [Increase $\lg k$.] Set $t \leftarrow t+1$. Terminate if t is now equal to n; otherwise set $m \leftarrow 2m + [t \,\&\, (t-1) \neq 0]$ and return to P2. ▮

Q1. [Begin a cycle.] Set $s \leftarrow X_k$, $l \leftarrow k$, $j \leftarrow l \oplus \lfloor l/2 \rfloor$.

Q2. [Follow the cycle.] If $j \neq k$ set $X_l \leftarrow X_j$, $l \leftarrow j$, $j \leftarrow l \oplus \lfloor l/2 \rfloor$, and repeat until $j = k$. Then set $X_l \leftarrow s$. ▮

31. We get a field from f_n if and only if we get one from $f_n^{[2]}$, which takes $(a_{n-1} \dots a_0)_2$ to $((a_{n-1} \oplus a_{n-2})(a_{n-1} \oplus a_{n-3})(a_{n-2} \oplus a_{n-4}) \dots (a_2 \oplus a_0)(a_1))_2$. Let $c_n(x)$ be the characteristic polynomial of the matrix A defining this transformation, mod 2; then $c_1(x) = x+1$, $c_2(x) = x^2 + x + 1$, and $c_{j+1}(x) = xc_j(x) + c_{j-1}(x)$. Since $c_n(A)$ is the zero matrix, by the Cayley–Hamilton theorem, a field is obtained if and only if $c_n(x)$ is a primitive polynomial, and this condition can be tested as in Section 3.2.2. The first such values of n are 1, 2, 3, 5, 6, 9, 11, 14, 23, 26, 29, 30, 33, 35, 39, 41, 51, 53, 65, 69, 74, 81, 83, 86, 89, 90, 95.

[Running the recurrence backwards shows that $c_{-j-1}(x) = c_j(x)$, hence $c_j(x)$ divides $c_{(2j+1)k+j}(x)$; for example, $c_{3k+1}(x)$ is always a multiple of $x+1$. All numbers n of the form $2jk+j+k$ are therefore excluded when $j > 0$ and $k > 0$. The polynomials $c_{18}(x)$, $c_{50}(x)$, $c_{98}(x)$, and $c_{99}(x)$ are irreducible but not primitive.]

32. Mostly true, but false at the points where $w_k(x)$ changes sign. (Walsh originally suggested that $w_k(x)$ should be zero at such points; but the convention adopted here is better, because it makes simple formulas like (15)–(19) valid for all x.)

33. By induction on k, we have

$$w_k(x) = w_{\lfloor k/2 \rfloor}(2x) = r_1(2x)^{b_1+b_2} r_2(2x)^{b_2+b_3} \dots \doteq r_1(x)^{b_0+b_1} r_2(x)^{b_1+b_2} r_3(x)^{b_2+b_3} \dots$$

for $0 \leq x < \frac{1}{2}$, because $r_j(2x) = r_{j+1}(x)$ and $r_1(x) = 1$ in this range. And when $\frac{1}{2} \leq x < 1$,

$$\begin{aligned} w_k(x) = (-1)^{\lceil k/2 \rceil} w_{\lfloor k/2 \rfloor}(2x-1) &= r_1(x)^{b_0+b_1} r_1(2x-1)^{b_1+b_2} r_2(2x-1)^{b_2+b_3} \dots \\ &= r_1(x)^{b_0+b_1} r_2(x)^{b_1+b_2} r_3(x)^{b_2+b_3} \dots \end{aligned}$$

because $\lceil k/2 \rceil \equiv b_0 + b_1$ (modulo 2) and $r_j(2x-1) = r_{j+1}(x - \frac{1}{2}) = r_{j+1}(x)$ for $j \geq 1$.

34. $p_k(x) = \prod_{j \geq 0} r_{j+1}^{b_j}$; hence $w_k(x) = p_k(x) p_{\lfloor k/2 \rfloor}(x) = p_{g(k)}(x)$. [R. E. A. C. Paley, *Proc. London Math. Soc.* (2) **34** (1932), 241–279.]

35. If $j = (a_{n-1} \dots a_0)_2$ and $k = (b_{n-1} \dots b_0)_2$, the element in row j and column k is $(-1)^{f(j,k)}$, where $f(j,k)$ is the sum of all $a_r b_s$ such that: $r = s$ (Hadamard); $r+s = n-1$ (Paley); $r + s = n$ or $n-1$ (Walsh).

Let R_n, F_n, and G_n be permutation matrices for the permutations that take $j = (a_{n-1} \dots a_0)_2$ to $k = (a_0 \dots a_{n-1})_2$, $k = 2^n - 1 - j = (\bar{a}_{n-1} \dots \bar{a}_0)_2$, and $k = g^{[-1]}(j) = ((a_{n-1}) \dots (a_{n-1} \oplus \cdots \oplus a_0))_2$, respectively. Then, using the Kronecker product of matrices, we have the recursive formulas

$$R_{n+1} = \begin{pmatrix} R_n \otimes (1\ 0) \\ R_n \otimes (0\ 1) \end{pmatrix}, \qquad F_{n+1} = F_n \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad G_{n+1} = \begin{pmatrix} G_n & 0 \\ 0 & G_n F_n \end{pmatrix},$$

$$H_{n+1} = H_n \otimes \begin{pmatrix} 1 & 1 \\ 1 & \bar{1} \end{pmatrix}, \qquad P_{n+1} = \begin{pmatrix} P_n \otimes (1\ 1) \\ P_n \otimes (1\ \bar{1}) \end{pmatrix}, \qquad W_{n+1} = \begin{pmatrix} W_n \otimes (1\ 1) \\ F_n W_n \otimes (1\ \bar{1}) \end{pmatrix}.$$

Thus $W_n = G_n^T P_n = P_n G_n$; $H_n = P_n R_n = R_n P_n$; and $P_n = W_n G_n^T = G_n W_n = H_n R_n = R_n H_n$.

36. W1. [Hadamard transform.] For $k = 0, 1, \dots, n-1$, replace the pair (X_j, X_{j+2^k}) by $(X_j + X_{j+2^k}, X_j - X_{j+2^k})$ for all j with $\lfloor j/2^k \rfloor$ even, $0 \le j < 2^n$. (These operations effectively set $X^T \leftarrow H_n X^T$.)

W2. [Bit reversal.] Apply the algorithm of exercise 5 to the vector X. (These operations effectively set $X^T \leftarrow R_n X^T$, in the notation of exercise 35.)

W3. [Gray binary permutation.] Apply the algorithm of exercise 30 to the vector X. (These operations effectively set $X^T \leftarrow G_n^T X^T$.) ▮

If n has one of the special values in exercise 31, it may be faster to combine steps W2 and W3 into a single permutation step.

37. If $k = 2^{e_1} + \cdots + 2^{e_t}$ with $e_1 > \cdots > e_t \ge 0$, the sign changes occur at $S_{e_1} \cup \cdots \cup S_{e_t}$, where

$$S_0 = \left\{\frac{1}{2}\right\}, \quad S_1 = \left\{\frac{1}{4}, \frac{3}{4}\right\}, \quad \dots, \quad S_e = \left\{\frac{2j+1}{2^e} \,\middle|\, 0 \le j < 2^e\right\}.$$

Therefore the number of sign changes in $(0\,..\,x)$ is $\sum_{j=1}^{t} \lfloor 2^{e_j} x + \frac{1}{2} \rfloor$. Setting $x = l/(k+1)$ gives $l + O(t)$ changes; so the lth is at a distance of at most $O(\nu(k))/2^{\lfloor \lg k \rfloor}$ from $l/(k+1)$.

[This argument makes it plausible that infinitely many pairs (k,l) exist with $|z_{kl} - l/(k+1)| = \Omega((\log k)/k)$. But no explicit construction of such "bad" pairs is immediately apparent.]

38. Let $t_0(x) = 1$ and $t_k(x) = \omega^{\lfloor 3x \rfloor \lceil 2k/3 \rceil} t_{\lfloor k/3 \rfloor}(3x)$, where $\omega = e^{2\pi i/3}$. Then $t_k(x)$ winds around the origin $\frac{2}{3}k$ times as x increases from 0 to 1. If $s_k(x) = \omega^{\lfloor 3^k x \rfloor}$ is the ternary analog of the Rademacher function $r_k(x)$, we have $t_k(x) = \prod_{j \ge 0} s_{j+1}(x)^{b_j - b_{j+1}}$ when $k = (b_{n-1} \dots b_0)_3$, as in the modular ternary Gray code.

39. Let's call the symbols $\{x_0, x_1, \dots, x_7\}$ instead of $\{a, b, c, d, e, f, g, h\}$. We want to find a permutation p of $\{0, 1, \dots, 7\}$ such that the matrix with $(-1)^{j \cdot k} x_{p(j) \oplus k}$ in row j and column k has orthogonal rows; this condition is equivalent to requiring that

$$(j + j') \cdot (p(j) + p(j')) \equiv 1 \text{ (modulo 2)}, \qquad \text{for } 0 \le j < j' < 8.$$

One solution is $p(0) \dots p(7) = 01725634$, yielding the identity $(a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2)(A^2 + B^2 + C^2 + D^2 + E^2 + F^2 + G^2 + H^2) = \mathcal{A}^2 + \mathcal{B}^2 + \mathcal{C}^2 + \mathcal{D}^2 +$

$\mathcal{E}^2+\mathcal{F}^2+\mathcal{G}^2+\mathcal{H}^2$, where

$$\begin{pmatrix}\mathcal{A}\\ \mathcal{B}\\ \mathcal{C}\\ \mathcal{D}\\ \mathcal{E}\\ \mathcal{F}\\ \mathcal{G}\\ \mathcal{H}\end{pmatrix}=\begin{pmatrix}a&b&c&d&e&f&g&h\\ b&-a&d&-c&f&-e&h&-g\\ h&g&-f&-e&d&c&-b&-a\\ c&-d&-a&b&g&-h&-e&f\\ f&e&h&g&-b&-a&-d&-c\\ g&-h&e&-f&-c&d&-a&b\\ d&c&-b&-a&-h&-g&f&e\\ e&-f&-g&h&-a&b&c&-d\end{pmatrix}\begin{pmatrix}A\\ B\\ C\\ D\\ E\\ F\\ G\\ H\end{pmatrix}.$$

[This identity was discovered by C. F. Degen, *Mémoires de l'Acad. Sci. St. Petersbourg* (5) **8** (1818), 207–219. The related octonions are discussed in an interesting survey by J. C. Baez, *Bull. Amer. Math. Soc.* **39** (2002), 145–205.]

(b) There *is* no 16×16 solution. The closest one can come is

$$p(0)\ldots p(15) = 0\ 1\ 11\ 2\ 14\ 15\ 13\ 4\ 9\ 10\ 7\ 12\ 5\ 6\ 3\ 8,$$

which fails if and only if $j\oplus j'=5$. (See *Philos. Mag.* **34** (1867), 461–475. In §9, §10, §11, and §13 of this paper, Sylvester stated and proved the basic results about what has somehow come to be known as the Hadamard transform — although Hadamard himself gave credit to Sylvester [*Bull. des Sciences Mathématiques* (2) **17** (1893), 240–246]. Moreover, Sylvester introduced transforms of m^n elements in §14, using mth roots of unity.)

40. Yes; this change would in fact run through the swapped subsets in lexicographic binary order rather than in Gray binary order. (Any 5×5 matrix of 0s and 1s that is nonsingular mod 2 will generate all 32 possibilities when we run through all linear combinations of its rows.) The most important thing is the appearance of the ruler function, or some other Gray code delta sequence, not the fact that only one a_j changes per step, in cases like this where any number of the a_j can be changed simultaneously at the same cost.

41. At most 16; for example, **fired**, **fires**, **finds**, **fines**, **fined**, **fares**, **fared**, **wares**, **wards**, **wands**, **wanes**, **waned**, **wines**, **winds**, **wires**, **wired**. We also get 16 from **paced**/**links** and **paled**/**mints**; perhaps also from a word mixed with an antipodal nonword.

42. Suppose $n\le 2^{2^r}+r+1$, and let $s=2^r$. We use an auxiliary table of 2^{r+s} bits f_{jk} for $0\le j<2^s$ and $0\le k<s$, representing focus pointers as in Algorithm L, together with an auxiliary s-bit "register" $j=(j_{s-1}\ldots j_0)_2$ and an $(r+2)$-bit "program counter" $p=(p_{r+1}\ldots p_0)_2$. At each step we examine the program counter and possibly the j register and one of the f bits; then, based on the bits seen, we complement a bit of the Gray code, complement a bit of the program counter, and possibly change a j or f bit, thereby emulating step L3 with respect to the most significant $n-r-2$ bits.

For example, here is the construction when $r=1$:

$p_2p_1p_0$	Change	Set		$p_2p_1p_0$	Change	Set	
0 0 0	a_0,p_0	$j_0\leftarrow f_{00}$	$j\leftarrow f_0$	1 1 0	a_0,p_0	$f_{j0}\leftarrow f_{(j+1)0}$	$f_j\leftarrow f_{j+1}$
0 0 1	a_1,p_1	$j_1\leftarrow f_{01}$		1 1 1	a_1,p_1	$f_{j1}\leftarrow f_{(j+1)1}$	
0 1 1	a_0,p_0	$f_{00}\leftarrow 0$	$f_0\leftarrow 0$	1 0 1	a_0,p_0	$f_{(j+1)0}\leftarrow (j+1)_0$	$f_{j+1}\leftarrow j+1$
0 1 0	a_2,p_2	$f_{01}\leftarrow 0$		1 0 0	a_{j+3},p_2	$f_{(j+1)1}\leftarrow (j+1)_1$	

The process stops when it attempts to change bit a_n.

[In fact, we need change only *one* auxiliary bit per step if we allow ourselves to examine some Gray binary bits as well as the auxiliary bits, because $p_r \dots p_0 = a_r \dots a_0$, and we can set $f_0 \leftarrow 0$ in a more clever way when j doesn't have its final value $2^s - 1$. This construction, suggested by Fredman in 2001, improves on another that he had published in *SICOMP* **7** (1978), 134–146. With a more elaborate construction it is possible to reduce the number of auxiliary bits to $O(n)$.]

43. This number was estimated by Silverman, Vickers, and Sampson [*IEEE Trans.* **IT-29** (1983), 894–901] to be about 7×10^{22}. Exact calculation might be feasible because every 6-bit Gray cycle has only five or fewer segments that lie in a 5-cube corresponding to at least one of the six coordinates. (In unpublished work, Steve Winker had used a similar idea to evaluate $d(5)$ in less than 15 minutes on a "generic" computer in 1972.)

44. All $(n+1)$-bit delta sequences with just two occurrences of the coordinate j are produced by the following construction: Let $\delta_1 \dots \delta_{2^n-1}$ and $\varepsilon_1 \dots \varepsilon_{2^n-1}$ be n-bit delta sequences for Gray *paths*, with $2^{\delta_1} \oplus \cdots \oplus 2^{\delta_{2^n-1}} = 2^{\varepsilon_1} \oplus \cdots \oplus 2^{\varepsilon_{2^n-1}}$. Form the cycle

$$\delta_{k+1} \dots \delta_{2^n-1}\, n\, \varepsilon_1 \dots \varepsilon_{2^n-1}\, n\, \delta_1 \dots \delta_k$$

for some k with $0 \le k < 2^n$, then interchange $n \leftrightarrow j$.

All $(n+2)$-bit delta sequences with just two occurrences of coordinates h and j (with h before j) are, similarly, produced from four n-bit sequences $\delta_1 \dots \delta_{2^n-1}$, $\dots$, $\eta_1 \dots \eta_{2^n-1}$ where $2^{\delta_1} \oplus \cdots \oplus 2^{\eta_{2^n-1}} = 0$, by interchanging $n \leftrightarrow h$ and $n+1 \leftrightarrow j$ in

$$\delta_{k+1} \dots \delta_{2^n-1}\, n\, \varepsilon_1 \dots \varepsilon_{2^n-1}\, (n{+}1)\, \zeta_1 \dots \zeta_{2^n-1}\, n\, \eta_1 \dots \eta_{2^n-1}\, (n{+}1)\, \delta_1 \dots \delta_k.$$

Let $a(n)$ and $b(n)$ be the number of n-bit cycles defined in parts (a) and (b); then $(a(1), \dots, a(5)) = (1, 0, 0, 1920, 318996480)$ and $(b(1), \dots, b(5)) = (0, 2, 12, 384, 4200960)$. The constructions above prove that $a(n+1) + 2b(n+1) = 2^n(n+1)A(n)$ and $b(n+2) = 2^n(n+2)(n+1)B(n)$, if there are $A(n)$ and $B(n)$ ways to choose the respective sequences δ, ε, ζ, and η. If we restrict ourselves to cases where the Gray paths are extendible to Gray cycles, with $\delta_0 = \varepsilon_0 = \zeta_0 = \eta_0$, we get $a'(n+1)$ and $b'(n+2)$ sequences where $a'(n+1) + 2b'(n+1) = 2^n(n+1)d(n)^2/n$ and $b'(n+2) = 2^n(n+2)(n+1)d(n)^4/n^3$.

45. We have $d(n+1) \ge 2^n d(n)^2/n$, because $2^n d(n)^2/n$ is a lower bound on the number of $(n+1)$-bit delta sequences with exactly two appearances of 0. Hence $d(n+1)^{1/2^{n+1}} > d(n)^{1/2^n}$; and $d(n) \ge \frac{5}{32}\alpha^{2^n}$ for $n \ge 5$, where $\alpha = (\frac{32}{5}d(5))^{1/32} \approx 2.06$.

Indeed, we can establish even faster growth by using the previous exercise, because $d(n+1) \ge a'(n+1) + b'(n+1)$ and $b'(n+1) \le \frac{25}{64}(n+1)d(n)^2/n$ for $n \ge 5$. Hence $d(n+1) \ge (2^n - \frac{25}{64})(n+1)d(n)^2/n$ for $n \ge 5$, and iteration of this relation shows that

$$\lim_{n\to\infty} d(n)^{1/2^n} \ge d(5)^{1/32} \prod_{n=5}^{\infty} \left(2^n - \frac{25}{64}\right)^{1/2^{n+1}} \left(\frac{n+1}{n}\right)^{1/2^{n+1}} \approx 2.3606.$$

[See R. J. Douglas, *Disc. Math.* **17** (1977), 143–146; M. Mollard, *European J. Comb.* **9** (1988), 49–52.] The true value of this limit, however, is probably ∞.

46. Leo Moser (unpublished) has conjectured that it is $\sim n/e$. So far only an upper bound of about $n/\sqrt{2}$ has been established; see the references in the previous answer.

48. If $d(n,k,v)$ of the cycles begin with $g(0) \dots g(k-1)v$, the conjecture implies that $d(n,k,v) \le d(n,k,g(k))$, because the reverse of a Gray cycle is a Gray cycle. Thus the

hint follows from $d(n) = d(n,1)$ and

$$d(n,k) = \sum_v \{\, d(n,k,v) \mid v \mathrel{-\!\!-} g(k-1), v \notin S_k \,\} \le c_{nk} d(n,k,g(k)) = d(n,k+1).$$

Finally, $d(n,2^n) = 1$, hence $d(n) \le \prod_{k=1}^{2^n-1} c_{nk} = \prod_{k=1}^{n} k^{\binom{n}{k}} = n \prod_{k=1}^{n-1} (k(n-k))^{\binom{n}{k}/2} \le n \prod_{k=1}^{n-1} (n/2)^{\binom{n}{k}} = n(n/2)^{2^n-2}$. [*IEEE Trans.* **IT-29** (1983), 894–901.]

49. Take any Hamiltonian path P from $0\ldots0$ to $1\ldots1$ in the $(2n-1)$-cube, such as the Savage–Winkler code, and use $0P$, $1\overline{P}$. (All such cycles are obtained by this construction when $n = 1$ or $n = 2$, but many more possibilities exist when $n > 2$.)

50. $\alpha_1(n{+}1)\alpha_1^R n\alpha_1 j_1 \alpha_2 n\alpha_2^R(n{+}1)\alpha_2 \ldots j_{l-1}\alpha_l n\alpha_l^R(n{+}1)\alpha_l n\alpha_l^R j_{l-1}\ldots j_1\alpha_1^R n.$

51. We can assume that $n > 3$ and that we have an n-bit Gray cycle with transition counts $c_j = 2\lfloor(2^{n-1}+j)/n\rfloor$; we want to construct an $(n+2)$-bit cycle with transition counts $c'_j = 2\lfloor(2^{n+1}+j)/(n+2)\rfloor$. If $2^{n+1} \bmod (n+2) \ge 2$, we can use Theorem D with $l = 2\lfloor 2^{n+1}/(n+2)\rfloor + 1$, underlining b_j copies of j where $b_j = 4\lfloor(2^{n-1}+j)/n\rfloor - \lfloor(2^{n+1}+j)/(n+2)\rfloor - [j=0]$ and putting an underlined 0 last. This is always easy to do because $|b_j - 2^{n+2}/n(n+2)| < 5$. A similar construction works if $2^{n+1} \bmod (n+2) \le n$, with $l = 2\lfloor 2^{n+1}/(n+2)\rfloor - 1$ and $b_j = 4\lfloor(2^{n-1}+j)/n\rfloor - \lfloor(2^{n+1}+j+2)/(n+2)\rfloor - [j=0]$. In fact, $2^{n+1} \bmod (n+2)$ is always $\le n$ [see K. Kedlaya, *Electronic J. Combinatorics* **3** (1996), comment on #R25 (9 April 1997)]. The basic idea of this proof is due to J. P. Robinson and M. Cohn [*IEEE Trans.* **C-30** (1981), 17–23].

52. The number of different code patterns in the smallest j coordinate positions is at most $c_0 + \cdots + c_{j-1}$.

53. Notice that Theorem D produces only cycles with $c_j = c_{j+1}$ for some j, so it cannot produce the counts $(2,4,6,8,12)$. The extension in exercise 50 gives also $c_j = c_{j+1} - 2$, but it cannot produce $(6,10,14,18,22,26,32)$. The sets of numbers satisfying the conditions of exercise 52 are precisely those obtainable by starting with $\{2,2,4,\ldots,2^{n-1}\}$ and repeatedly replacing some pair $\{c_j, c_k\}$ for which $c_j < c_k$ by the pair $\{c_j+2, c_k-2\}$.

54. Suppose the values are $\{p_1,\ldots,p_n\}$, and let x_{jk} be the number of times p_j occurs in $(a_1,\ldots,a_k)$. We must have $(x_{1k},\ldots,x_{nk}) \equiv (x_{1l},\ldots,x_{nl})$ (modulo 2) for some $k < l$. But if the p's are prime numbers, varying as the delta sequence of an n-bit Gray cycle, the only solution is $k = 0$ and $l = 2^n$. [*AMM* **60** (1953), 418; **83** (1976), 54.]

56. [*Bell System Tech. J.* **37** (1958), 815–826.] The 112 canonical delta sequences yield

Class	Example	t	Class	Example	t	Class	Example	t
A	0102101302012023	2	*D*	0102013201020132	4	*G*	0102030201020302	8
B	0102303132101232	2	*E*	0102032021202302	4	*H*	0102101301021013	8
C	0102030130321013	2	*F*	0102013102010232	4	*I*	0102013121012132	1

Here B is the balanced code (Fig. 13(b)), G is standard Gray binary (Fig. 10(b)), and H is the complementary code (Fig. 13(a)). Class H is also equivalent to the modular $(4,4)$ Gray code under the correspondence of exercise 18. A class with t automorphisms corresponds to $32 \times 24/t$ of the 2688 different delta sequences $\delta_0\delta_1\ldots\delta_{15}$.

Similarly (see exercise 7.2.3–00), the 5-bit Gray cycles fall into 237,675 different equivalence classes.

57. With Type 1 only, 480 vertices are isolated, namely those of classes D, F, G in the previous answer. With Type 2 only, the graph has 384 components, 288 of which are

isolated vertices of classes F and G. There are 64 components of size 9, each containing 3 vertices from E and 6 from A; 16 components of size 30, each with 6 from H and 24 from C; and 16 components of size 84, each with 12 from D, 24 from B, 48 from I. With Type 3 (or Type 4) only, the entire graph is connected. [Similarly, all 91,392 of the 4-bit Gray *paths* are connected if path $\alpha\beta$ is considered adjacent to path $\alpha^R\beta$. Vickers and Silverman, *IEEE Trans.* **C-29** (1980), 329–331, have conjectured that Type 3 changes will suffice to connect the graph of n-bit Gray cycles for all $n \geq 3$.]

58. If some nonempty substring of $\beta\beta$ involves each coordinate an even number of times, that substring cannot have length $|\beta|$, so some cyclic shift of β has a prefix γ with the same evenness property. But then α doesn't define a Gray cycle, because we could change each n of γ back to 0.

59. If α is nonlocal in exercise 58, so is $\beta\beta$, provided that $q > 1$ and that 0 occurs more than $q+1$ times in α. Therefore, starting with the α of (30) but with 0 and 1 interchanged, we obtain nonlocal cycles for $n \geq 5$ in which coordinate 0 changes exactly 6 times. [Mark Ramras, *Discrete Math.* **85** (1990), 329–331.] On the other hand, a 4-bit Gray cycle cannot be nonlocal because it always has a run of length 2; if $\delta_k = \delta_{k+2}$, elements $\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}$ form a 2-subcube.

60. Use the construction of exercise 58 with $q = 1$.

61. The idea is to interleave an m-bit cycle $U = (u_0, u_1, u_2, \dots)$ with an n-bit cycle $V = (v_0, v_1, v_2, \dots)$, by forming concatenations

$$W = (u_{i_0}v_{j_0},\ u_{i_1}v_{j_1},\ u_{i_2}v_{j_2},\ \dots), \qquad i_k = \bar{a}_0 + \cdots + \bar{a}_{k-1}, \quad j_k = a_0 + \cdots + a_{k-1},$$

where $a_0a_1a_2\dots$ is a periodic string of control bits $\alpha\alpha\alpha\dots$; we advance to the next element of U when $a_k = 0$, otherwise to the next element of V.

If α is any string of length $2^m \leq 2^n$, containing s bits that are 0 and $t = 2^m - s$ bits that are 1, W will be an $(m+n)$-bit Gray cycle if s and t are odd. For we have $i_{k+l} \equiv i_k$ (modulo 2^m) and $j_{k+l} \equiv j_k$ (modulo 2^n) only if l is a multiple of 2^m, since $i_k + j_k = k$. Suppose $l = 2^m c$; then $j_{k+l} = j_k + tc$, so c is a multiple of 2^n.

(a) Let $\alpha = 0111$; then runs of length 8 occur in the left 2 bits and runs of length $\geq \lfloor \frac{4}{3} r(n) \rfloor$ occur in the right n bits.

(b) Let s be the largest odd number $\leq 2^m r(m)/(r(m)+r(n))$. Also let $t = 2^m - s$ and $a_k = \lfloor (k+1)t/2^m \rfloor - \lfloor kt/2^m \rfloor$, so that $i_k = \lceil ks/2^m \rceil$ and $j_k = \lfloor kt/2^m \rfloor$. If a run of length l occurs in the left m bits, we have $i_{k+l+1} \geq i_k + r(m) + 1$, hence $l + 1 > 2^m r(m)/s \geq r(m) + r(n)$. And if it occurs in the right n bits we have $j_{k+l+1} \geq j_k + r(n) + 1$, hence

$$l + 1 > 2^m r(n)/t > 2^m r(n)/(2^m r(n)/(r(m)+r(n)) + 2)$$

$$= r(m) + r(n) - \frac{2(r(m)+r(n))^2}{2^m r(n) + 2(r(m)+r(n))} > r(m) + r(n) - 1$$

because $r(m) \leq r(n)$.

The construction often works also in less restricted cases. See the paper that introduced the study of Gray-code runs: L. Goddyn, G. M. Lawrence, and E. Nemeth, *Utilitas Math.* **34** (1988), 179–192.

63. Set $a_k \leftarrow k \bmod 4$ for $0 \leq k < 2^{10}$, except that $a_k = 4$ when $k \bmod 16 = 15$ or $k \bmod 64 = 42$ or $k \bmod 256 = 133$. Also set $(j_0, j_1, j_2, j_3, j_4) \leftarrow (0, 2, 4, 6, 8)$. Then for $k = 0, 1, \dots, 1023$, set $\delta_k \leftarrow j_{a_k}$ and $j_{a_k} \leftarrow 1 + 4a_k - j_{a_k}$. (This construction generalizes the method of exercise 61.)

64. (a) Each element u_k appears together with $\{v_k, v_{k+2^m}, \ldots, v_{k+2^m(2^{n-1}-1)}\}$ and $\{v_{k+1}, v_{k+1+2^m}, \ldots, v_{k+1+2^m(2^{n-1}-1)}\}$. Thus the permutation $\sigma_0 \ldots \sigma_{2^m-1}$ must be a 2^{n-1}-cycle containing the n-bit vertices of even parity, times an arbitrary permutation of the other vertices. This condition is also sufficient.

(b) Let τ_j be the permutation that takes $v \mapsto v \oplus 2^j$, and let $\pi_j(u,w)$ be the permutation $(uw)\tau_j$. If $u \oplus w = 2^i + 2^j$ then $\pi_j(u,w)$ takes $u \mapsto u \oplus 2^i$ and $w \mapsto w \oplus 2^i$, while $v \mapsto v \oplus 2^j$ for all other vertices v, so it takes each vertex to a neighbor.

If S is any set $\subseteq \{0, \ldots, n-1\}$, let $\sigma(S)$ be the stream of all permutations τ_j for all $j \in \{0, \ldots, n-1\} \setminus S$, in increasing order of j, repeated twice; for example, if $n = 5$ we have $\sigma(\{1,2\}) = \tau_0\tau_3\tau_4\tau_0\tau_3\tau_4$. Then the Gray stream

$$\Sigma(i,j,u) = \sigma(\{i,j\})\pi_j(u, u\oplus 2^i \oplus 2^j)\sigma(\{i,j\})\tau_j\sigma(\{j\})$$

consists of $6n-8$ permutations whose product is the transposition $(u\ u\oplus 2^i\oplus 2^j)$. Moreover, when this stream is applied to any n-bit vertex v, its runs all have length $n-2$ or more.

We may assume that $n \geq 5$. Let $\delta_0 \ldots \delta_{2^n-1}$ be the delta sequence for an n-bit Gray cycle $(v_0, v_1, \ldots, v_{2^n-1})$ with all runs of length 3 or more. Then the product of all permutations in

$$\Sigma = \prod_{k=1}^{2^{n-1}-1} \big(\Sigma(\delta_{2k-1}, \delta_{2k}, v_{2k-1})\, \Sigma(\delta_{2k}, \delta_{2k+1}, v_{2k})\big)$$

is $(v_1\, v_3)(v_2\, v_4)\ldots(v_{2^n-3}\, v_{2^n-1})(v_{2^n-2}\, v_0) = (v_{2^n-1} \ldots v_1)(v_{2^n-2} \ldots v_0)$, so it satisfies the cycle condition of (a).

Moreover, all powers $(\sigma(\emptyset)\Sigma)^t$ produce runs of length $\geq n-2$ when applied to any vertex v. By repeating individual factors $\sigma(\{i,j\})$ or $\sigma(\{j\})$ in Σ as many times as we wish, we can adjust the length of $\sigma(\emptyset)\Sigma$, obtaining $2n + (2^{n-1}-1)(12n-16) + 2(n-2)a + 2(n-1)b$ for any integers $a, b \geq 0$; thus we can increase its length to exactly 2^m, provided that $2^m \geq 2n + (2^{n-1}-1)(12n-16) + 2(n^2-5n+6)$, by exercise 5.2.1–21.

(c) The bound $r(n) \geq n - 4\lg n + 8$ can be proved for $n \geq 5$ as follows. First we observe that it holds for $5 \leq n < 33$ by the methods of exercises 60–63. Then we observe that every integer $N \geq 33$ can be written as $N = m+n$ or $N = m+n+1$, for some $m \geq 20$, where

$$n = m - \lfloor 4\lg m \rfloor + 10.$$

If $m \geq 20$, 2^m is sufficiently large for the construction in part (b) to be valid; hence

$$\begin{aligned} r(N) \geq r(m+n) \geq 2\min(r(m), n-2) &\geq 2(m - \lfloor 4\lg m \rfloor + 8) \\ &= m+n+1 - \lfloor 4\lg(m+n) - 1 + \epsilon \rfloor + 8 \\ &\geq N - 4\lg N + 8 \end{aligned}$$

where $\epsilon = 4\lg(2m/(m+n)) < 1$. [*Electronic Journal of Combinatorics* **10** (2003), #R27, 1–10.] Recursive use of (b) gives, in fact, $r(1024) \geq 1000$.

65. A computer search reveals that eight essentially different patterns (and their reverses) are possible. One of them has the delta sequence 0102031420302404123 4214103234103, and it is close to two of the others.

66. (Solution by Mark Cooke.) One suitable delta sequence is 0123456070121324356576071021353462670153741236256701731426206570134214656057310246453757102043537614073630464273703564027132750541210275641502403654250136

025416156043125760325720431576243217604520417516354767035647570625437242132624161523417514367143164314. (Solutions for $n > 8$ are still unknown.)

67. Let $v_{2k+1} = \bar{v}_{2k}$ and $v_{2k} = 0u_k$, where $(u_0, u_1, \ldots, u_{2^n-1})$ is any $(n-1)$-bit Gray cycle. [See Robinson and Cohn, *IEEE Trans.* **C-30** (1981), 17–23.]

68. Yes. The simplest way is probably to take $(n-1)$-trit modular Gray ternary code and add $0\ldots0$, $1\ldots1$, $2\ldots2$ to each string (modulo 3). For example, when $n = 3$ the code is 000, 111, 222, 001, 112, 220, 002, 110, 221, 012, 120, 201, ..., 020, 101, 212.

69. (a) We need only verify the change in h when bits $b_{j-1}\ldots b_0$ are simultaneously complemented, for $j = 1, 2, \ldots$; and these changes are respectively $(1110)_2$, $(1101)_2$, $(0111)_2$, $(1011)_2$, $(10011)_2$, $(100011)_2$, ... To prove that every n-tuple occurs, note that $0 \le h(k) < 2^n$ when $0 \le k < 2^n$ and $n > 3$; also $h^{[-1]}((a_{n-1}\ldots a_0)_2) = (b_{n-1}\ldots b_0)_2$, where $b_0 = a_0 \oplus a_1 \oplus a_2 \oplus \cdots$, $b_1 = a_0$, $b_2 = a_2 \oplus a_3 \oplus a_4 \oplus \cdots$, $b_3 = a_0 \oplus a_1 \oplus a_3 \oplus \cdots$, and $b_j = a_j \oplus a_{j+1} \oplus \cdots$ for $j \ge 4$.

(b) Let $h(k) = (\ldots a_2 a_1 a_0)_2$ where $a_j = b_j \oplus b_{j+1} \oplus b_0[j \le t] \oplus b_{t-1}[t-1 \le j \le t]$.

70. As in (32) and (33), we can remove a factor of $n!$ by assuming that the strings of weight 1 occur in order. Then there are 14 solutions for $n = 5$ starting with 00000, and 21 starting with 00001. When $n = 6$ there are 46,935 of each type (related by reversal and complementation). When $n = 7$ the number is much, much larger, yet very small by comparison with the total number of 7-bit Gray codes.

71. Suppose that $\alpha_{n(j+1)}$ differs from α_{nj} in coordinate t_j, for $0 \le j < n-1$. Then $t_j = j\pi_n$, by (44) and (38). Now Eq. (34) tells us that $t_0 = n-1$; and if $0 < j < n-1$ we have $t_j = ((j-1)\pi_{n-1})\pi_{n-1}$ by (40). Thus $t_j = j\sigma_n\pi_{n-1}^2$ for $0 \le j < n-1$, and the value of $(n-1)\pi_n$ is whatever is left. (Notations for permutations are notoriously confusing, so it is always wise to check a few small cases carefully.)

72. The delta sequence is 0102132430201234012313041021323.

73. Let $Q_{nj} = P_{nj}^R$ and denote the sequences (41), (42) by S_n and T_n. Thus $S_n = P_{n0}Q_{n1}P_{n2}\ldots$ and $T_n = Q_{n0}P_{n1}Q_{n2}\ldots$, if we omit the commas; and we have

$$S_{n+1} = 0P_{n0}\ 0Q_{n1}\ 1Q_{n0}^{\pi}\ 1P_{n1}^{\pi}\ 0P_{n2}\ 0Q_{n3}\ 1Q_{n2}^{\pi}\ 1P_{n3}^{\pi}\ 0P_{n4}\ \ldots,$$
$$T_{n+1} = 0Q_{n0}\ 1P_{n0}^{\pi}\ 0P_{n1}\ 0Q_{n2}\ 1Q_{n1}^{\pi}\ 1P_{n2}^{\pi}\ 0P_{n3}\ 0Q_{n4}\ 1Q_{n3}^{\pi}\ \ldots,$$

where $\pi = \pi_n$, revealing a reasonably simple joint recursion between the delta sequences Δ_n and E_n of S_n and T_n. Namely, if we write

$$\Delta_n = \phi_1\, a_1\, \phi_2\, a_2 \ldots \phi_{n-1}\, a_{n-1}\, \phi_n, \qquad E_n = \psi_1\, b_1\, \psi_2\, b_2 \ldots \psi_{n-1}\, b_{n-1}\, \psi_n,$$

where each ϕ_j and ψ_j is a string of length $2\binom{n-1}{j-1} - 1$, the next sequences are

$$\Delta_{n+1} = \phi_1\ a_1\ \phi_2\ n\ \psi_1\pi\ b_1\pi\ \psi_2\pi\ n\ \phi_3\ a_3\ \phi_4\ n\ \psi_3\pi\ b_3\pi\ \psi_4\pi\ n\ \ldots$$
$$E_{n+1} = \psi_1\ n\ \phi_1\pi\ n\ \psi_2\ b_2\ \psi_3\ n\ \phi_2\pi\ a_2\pi\ \phi_3\pi\ n\ \psi_4\ b_4\ \psi_5\ n\ \phi_4\pi\ a_4\pi\ \phi_5\pi\ n\ \ldots$$

For example, we have $\Delta_3 = 0\underline{1}021\underline{0}1$ and $E_3 = 0\underline{2}120\underline{2}1$, if we underline the a's and b's to distinguish them from the ϕ's and ψ's; and

$$\Delta_4 = 0\ 1\ 0\ 2\ 1\ 3\ 0\pi\ 2\pi\ 1\pi\ 2\pi\ 0\pi\ 3\ 1\ 3\ 1\pi = 0\ \underline{1}\ 0\ 2\ 1\ 3\ 2\ \underline{1}\ 0\ 1\ 2\ 3\ 1\ \underline{3}\ 0,$$
$$E_4 = 0\ 3\ 0\pi\ 3\ 1\ 2\ 0\ 2\ 1\ 3\ 0\pi\ 2\pi\ 1\pi\ 0\pi\ 1\pi = 0\ \underline{3}\ 2\ 3\ 1\ 2\ 0\ \underline{2}\ 1\ 3\ 2\ 1\ 0\ \underline{2}\ 0;$$

here $a_3\phi_4$ and $b_3\psi_4$ are empty. Elements have been underlined for the next step.

Thus we can compute the delta sequences in memory as follows. Here $p[j] = j\pi_n$ for $1 \le j < n$; $s_k = \delta_k$, $t_k = \varepsilon_k$, and $u_k = [\delta_k$ and ε_k are underlined], for $0 \le k < 2^n - 1$.

R1. [Initialize.] Set $n \leftarrow 1$, $p[0] \leftarrow 0$, $s_0 \leftarrow t_0 \leftarrow u_0 \leftarrow 0$.

R2. [Advance n.] Perform Algorithm S below, which computes the arrays s', t', and u' for the next value of n; then set $n \leftarrow n+1$.

R3. [Ready?] If n is sufficiently large, the desired delta sequence Δ_n is in array s'; terminate. Otherwise keep going.

R4. [Compute π_n.] Set $p'[0] = n-1$, and $p'[j] = p[p[j-1]]$ for $1 \le j < n$.

R5. [Prepare to advance.] Set $p[j] \leftarrow p'[j]$ for $0 \le j < n$; set $s_k \leftarrow s'_k$, $t_k \leftarrow t'_k$, and $u_k \leftarrow u'_k$ for $0 \le k < 2^n - 1$. Return to R2. ∎

In the following steps, "Transmit stuff(l,j) while $u_j = 0$" is an abbreviation for "If $u_j = 0$, repeatedly stuff(l,j), $l \leftarrow l+1$, $j \leftarrow j+1$, until $u_j \neq 0$."

S1. [Prepare to compute Δ_{n+1}.] Set $j \leftarrow k \leftarrow l \leftarrow 0$ and $u_{2^n-1} \leftarrow -1$.

S2. [Advance j.] Transmit $s'_l \leftarrow s_j$ and $u'_l \leftarrow 0$ while $u_j = 0$. Then go to S5 if $u_j < 0$.

S3. [Advance j and k.] Set $s'_l \leftarrow s_j$, $u'_l \leftarrow 1$, $l \leftarrow l+1$, $j \leftarrow j+1$. Then transmit $s'_l \leftarrow s_j$ and $u'_l \leftarrow 0$ while $u_j = 0$. Then set $s'_l \leftarrow n$, $u'_l \leftarrow 0$, $l \leftarrow l+1$. Then transmit $s'_l \leftarrow p[t_k]$ and $u'_l \leftarrow 0$ while $u_k = 0$. Then set $s'_l \leftarrow p[t_k]$, $u'_l \leftarrow 1$, $l \leftarrow l+1$, $k \leftarrow k+1$. And once again transmit $s'_l \leftarrow p[t_k]$ and $u'_l \leftarrow 0$ while $u_k = 0$.

S4. [Done with Δ_{n+1}?] If $u_k < 0$, go to S6. Otherwise set $s'_l \leftarrow n$, $u'_l \leftarrow 0$, $l \leftarrow l+1$, $j \leftarrow j+1$, $k \leftarrow k+1$, and return to S2.

S5. [Finish Δ_{n+1}.] Set $s'_l \leftarrow n$, $u'_l \leftarrow 1$, $l \leftarrow l+1$. Then transmit $s'_l \leftarrow p[t[k]]$ and $u'_l \leftarrow 0$ while $u_k = 0$.

S6. [Prepare to compute E_{n+1}.] Set $j \leftarrow k \leftarrow l \leftarrow 0$. Transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then set $t'_l \leftarrow n$, $l \leftarrow l+1$.

S7. [Advance j.] Transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. Then terminate if $u_j < 0$; otherwise set $t'_l \leftarrow n$, $l \leftarrow l+1$, $j \leftarrow j+1$, $k \leftarrow k+1$.

S8. [Advance k.] Transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then go to S10 if $u_k < 0$.

S9. [Advance k and j.] Set $t'_l \leftarrow t_k$, $l \leftarrow l+1$, $k \leftarrow k+1$. Then transmit $t'_l \leftarrow t_k$ while $u_k = 0$. Then set $t'_l \leftarrow n$, $l \leftarrow l+1$. Then transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. Then set $t'_l \leftarrow p[s_j]$, $l \leftarrow l+1$, $j \leftarrow j+1$. Return to S7.

S10. [Finish E_{n+1}.] Set $t'_l \leftarrow n$, $l \leftarrow l+1$. Then transmit $t'_l \leftarrow p[s_j]$ while $u_j = 0$. ∎

To generate the monotonic Savage–Winkler code for fairly large n, one can first generate Δ_{10} and E_{10}, say, or even Δ_{20} and E_{20}. Using these tables, a suitable recursive procedure will then be able to reach higher values of n with very little computational overhead per step, on the average.

74. If the monotonic path is $v_0, \ldots, v_{2^n-1}$ and if v_k has weight j, we have

$$2\sum_{t>0}\binom{n}{j-2t} + ((j+\nu(v_0)) \bmod 2) \le k \le 2\sum_{t\ge 0}\binom{n}{j-2t} + ((j+\nu(v_0)) \bmod 2) - 2.$$

Therefore the maximum distance between vertices of respective weights j and $j+1$ is $2(\binom{n-1}{j-1} + \binom{n-1}{j} + \binom{n-1}{j+1}) - 1$. The maximum value, approximately $3 \cdot 2^n/\sqrt{2\pi n}$, occurs when j is approximately $n/2$. [This is only about three times the smallest value achievable in *any* ordering of the vertices, which is $\sum_{j=0}^{n-1}\binom{j}{\lfloor j/2 \rfloor}$ by exercise 7.10–00.]

75. There are only five essentially distinct solutions, all of which turn out in fact to be Gray *cycles*. The delta sequences are

$$0123012421032101210321040123012\,(1)$$
$$0123012421032101301230141032103\,(1)$$
$$0123012421032102032103242301230\,(2)$$
$$0123012423012302012301242301230\,(2)$$
$$0123410121030143210301410123410\,(3)$$

76. If $v_0, \ldots, v_{2^n-1}$ is trend-free, so is the $(n+1)$-bit cycle $0v_0$, $1v_0$, $1v_1$, $0v_1$, $0v_2$, $1v_2$, $\ldots$, $1v_{2^n-1}$, $0v_{2^n-1}$. Figure 14(g) shows a somewhat more interesting construction, which generalizes the first solution of exercise 75 to an $(n+2)$-bit cycle

$$00\Gamma''^R,\ 01\Gamma'^R,\ 11\Gamma',\ 10\Gamma'',\ 10\Gamma,\ 11\Gamma''',\ 01\Gamma'''^R,\ 00\Gamma^R$$

where Γ is the n-bit sequence $g(1), \ldots, g(2^{n-1})$ and $\Gamma' = \Gamma \oplus g(1)$, $\Gamma'' = \Gamma \oplus g(2^{n-1})$, $\Gamma''' = \Gamma \oplus g(2^{n-1}+1)$. [An n-bit trend-free design that is *almost* a Gray code, having just four steps in which $\nu(v_k \oplus v_{k+1}) = 2$, was found for all $n \geq 3$ by C. S. Cheng, *Proc. Berkeley Conf. Neyman and Kiefer* **2** (Hayward, Calif.: Inst. of Math. Statistics, 1985), 619–633.]

77. Replace the array $(o_{n-1}, \ldots, o_0)$ by an array of sentinel values $(s_{n-1}, \ldots, s_0)$, with $s_j \leftarrow m_j - 1$ in step H1. Set $a_j \leftarrow (a_j + 1) \bmod m_j$ in step H4. If $a_j = s_j$ in step H5, set $s_j \leftarrow (s_j - 1) \bmod m_j$, $f_j \leftarrow f_{j+1}$, $f_{j+1} \leftarrow j + 1$.

78. For (50), notice that B_{j+1} is the number of times reflection has occurred in coordinate j, because we bypass coordinate j on steps that are multiples of $m_j \ldots m_0$. Hence, if $b_j < m_j$, an increase of b_j by 1 causes a_j to increase or decrease by 1 as appropriate. Furthermore, if $b_i = m_i - 1$ for $0 \leq i < j$, changing all these b_i to 0 when incrementing b_j will increase each of $B_0, \ldots, B_j$ by 1, thereby leaving the values $a_0, \ldots, a_{j-1}$ unchanged in (50).

For (51), note that $B_j = m_j B_{j+1} + b_j \equiv m_j B_{j+1} + a_j + (m_j - 1)B_{j+1} \equiv a_j + B_{j+1}$ (modulo 2); hence $B_j \equiv a_j + a_{j+1} + \cdots$, and (51) is obviously equivalent to (50).

In the modular Gray code for general radices $(m_{n-1}, \ldots, m_0)$, let

$$\bar{g}(k) = \begin{bmatrix} a_{n-1}, & \ldots, & a_2, & a_1, & a_0 \\ m_{n-1}, & \ldots, & m_2, & m_1, & m_0 \end{bmatrix}$$

when k is given by (46). Then $a_j = (b_j - B_{j+1}) \bmod m_j$, because coordinate j has increased modulo m_j exactly $B_j - B_{j+1}$ times if we start at $(0, \ldots, 0)$. The inverse function, which determines the b's from the modular Gray a's, is $b_j = (a_j + a_{j+1} + a_{j+2} + \cdots) \bmod m_j$ in the special case that each m_j is a divisor of m_{j+1} (for example, if all m_j are equal). But the inverse has no simple form in general; it can be computed by using the recurrences $b_j = (a_j + B_{j+1}) \bmod m_j$, $B_j = m_j B_{j+1} + b_j$ for $j = n-1$, $\ldots$, 0, starting with $B_n = 0$.

[Reflected Gray codes for radix $m > 2$ were introduced by Ivan Flores in *IRE Trans.* **EC-5** (1956), 79–82; he derived (50) and (51) in the case that all m_j are equal. Modular Gray codes with general mixed radices were implicitly discussed by Joseph Rosenbaum in *AMM* **45** (1938), 694–696, but without the conversion formulas; conversion formulas when all m_j have a common value m were published by Martin Cohn, *Info. and Control* **6** (1963), 70–78.]

79. (a) The last n-tuple always has $a_{n-1} = m_{n-1} - 1$, so it is one step from $(0, \ldots, 0)$ only if $m_{n-1} = 2$. And this condition suffices to make the final n-tuple $(1, 0, \ldots, 0)$.

[Similarly, the final subforest output by Algorithm K is adjacent to the initial one if and only if the leftmost tree is an isolated vertex.]

(b) The last n-tuple is $(m_{n-1}-1, 0, \ldots, 0)$ if and only if $m_{n-1} \ldots m_{j+1} \bmod m_j = 0$ for $0 \le j < n-1$, because $b_j = m_j - 1$ and $B_j = m_{n-1} \ldots m_j - 1$.

80. Run through $p_1^{a_1} \ldots p_t^{a_t}$ using reflected Gray code with radices $m_j = e_j + 1$.

81. The first cycle contains the edge from (x, y) to $\big(x, (y+1) \bmod m\big)$ if and only if $(x+y) \bmod m \neq m-1$ if and only if the second cycle contains the edge from (x, y) to $\big((x+1) \bmod m, y\big)$.

82. There are two 4-bit Gray cycles $(u_0, \ldots, u_{15})$ and $(v_0, \ldots, v_{15})$ that cover all edges of the 4-cube. (Indeed, the non-edges of classes A, B, D, H, and I in exercise 56 form Gray cycles, in the same classes as their complements.) Therefore with 16-ary modular Gray code we can form the four desired cycles $(u_0u_0, u_0u_1, \ldots, u_0u_{15}, u_1u_{15}, \ldots, u_{15}u_0)$, $(u_0u_0, u_1u_0, \ldots, u_{15}u_0, u_{15}u_1, \ldots, u_0u_{15})$, $(v_0v_0, \ldots, v_{15}v_0)$, $(v_0v_0, \ldots, v_0v_{15})$.

In a similar way we can show that $n/2$ edge-disjoint n-bit Gray cycles exist when n is 16, 32, 64, etc. [*Abhandlungen Math. Sem. Hamburg* **20** (1956), 13–16.] J. Aubert and B. Schneider [*Discrete Math.* **38** (1982), 7–16] have proved that the same property holds for *all* even values of $n \ge 4$, but no simple construction is known.

83. Mark Cooke found the following totally unsymmetric solution in December, 2002:

(1) 27374650573202656123165467436105251060520424163143721451014217372506246064173213107351607103156205713172463452102434643207054702414735614673762504735074513062065641507312373142737643256124026430167354675324025246374752176402707360651052151060735754632531056710141503047313570615453627623241426465272021632075363710750740315767476154565275651045102402310735342465123040654530621371053726205017524534067034373435315026024630456276741527524060216104343016735467532402524637475217640270736065105215106073575463253105;

(2) 06167134172321751716715404602471647424732025316216735316327360526710141503047313570615453627623241426465272021632075363710750740315767476154565275651045102402310735342465123040654530621371053726205017524534067034373435315026024630456276741527524060216104343016735467532402524637475217640270736065105215106073575463253105;

(3) 37010637515071312362437657351030120423537472074104736216172473246505132565057121565024570473247421427640231034362703262764130574056062034174561315175631470272172520561321260405350626046017364267176417435134012453602417306365450615630274145356764326257450516701410150304731357061545362762324142646527202163207536371075074031576747615456527565104510240231073534246512304065453062137105372620501752453406703437343531502602463045627674152752406021610434;

(4) 67065464356721472362104054320545107374051705321454316364305046734560621206416201320742373627204506473140171020514126107452343672132045275235341051542637060136356730710542016315121053506173123642725371656172175425107602154623754526742570373464036473762716571.

(Each of these delta sequences should start from the same vertex of the cube.) Is there a symmetrical way to do the job?

84. Calling the initial position $(2, 2)$, the 8-step solution in Fig. A-1 shows how the sequence progresses down to $(0, 0)$. In the first move, for example, the front half of the cord passes around and behind the right comb, then through the large right loop. The middle line should be read from right to left. The generalization to n pairs of loops would, similarly, take $3^n - 1$ steps.

[The origin of this delightful puzzle is obscure. *The Book of Ingenious & Diabolical Puzzles* by Jerry Slocum and Jack Botermans (1994) shows a 2-loop version carved from horn, probably made in China about 1850 [page 101], and a modern 6-loop version made in Malaysia about 1988 [page 93]. Slocum also owns a 4-loop version made from bamboo in England about 1884. He has found it listed in Henry Novra's *Catalogue of*

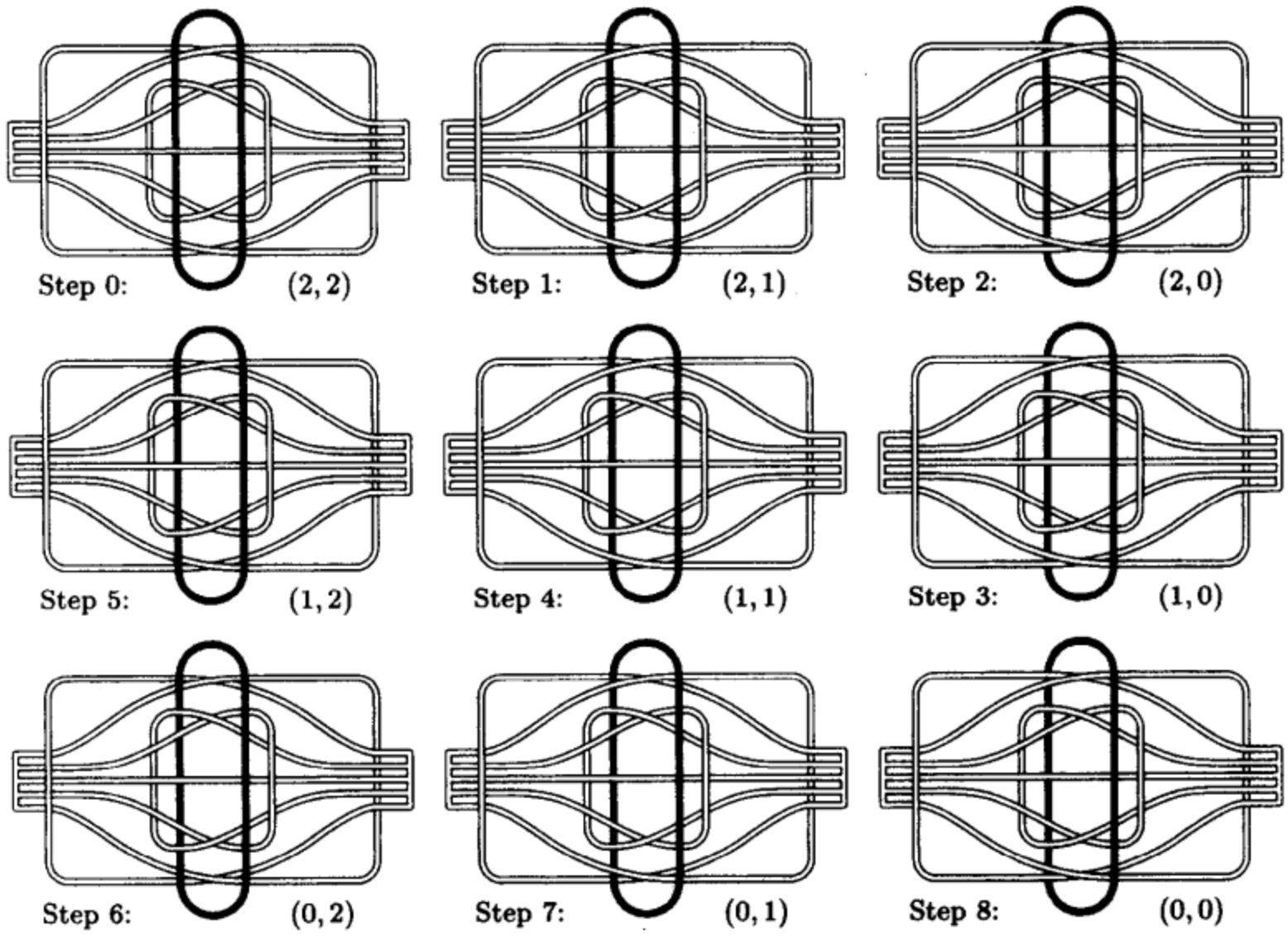


Fig. A-1.

Conjuring Tricks and Puzzles (1858 or 1859) and W. H. Cremer's *Games, Amusements, Pastimes and Magic* (1867), as well as in Hamley's catalog of 1895, under the name "Marvellous Canoe Puzzle." See also *U.S. Patents 2091191* (1937), *D172310* (1954), *3758114* (1973), *D406866* (1999). Dyckman noted its connection to reflected Gray ternary in a letter to Martin Gardner, dated 2 August 1972.]

85. By (50), element $\left[\begin{smallmatrix} b, & b' \\ t, & t' \end{smallmatrix}\right]$ of $\Gamma \boxtimes \Gamma'$ is $\alpha_a \alpha'_{a'}$ if $\hat{g}(\left[\begin{smallmatrix} b, & b' \\ t, & t' \end{smallmatrix}\right]) = \left[\begin{smallmatrix} a, & a' \\ t, & t' \end{smallmatrix}\right]$ in the reflected Gray code for radices (t, t'). We can now show that element $\left[\begin{smallmatrix} b, & b', & b'' \\ t, & t', & t'' \end{smallmatrix}\right]$ of both $(\Gamma \boxtimes \Gamma') \boxtimes \Gamma''$ and $\Gamma \boxtimes (\Gamma' \boxtimes \Gamma'')$ is $\alpha_a \alpha'_{a'} \alpha''_{a''}$ if $\hat{g}(\left[\begin{smallmatrix} b, & b', & b'' \\ t, & t', & t'' \end{smallmatrix}\right]) = \left[\begin{smallmatrix} a, & a', & a'' \\ t, & t', & t'' \end{smallmatrix}\right]$ in the reflected Gray code for radices (t, t', t''). See exercise 4.1–10, and note also the mixed-radix law

$$m_1 \ldots m_n - 1 - \begin{bmatrix} x_1, & \ldots, & x_n \\ m_1, & \ldots, & m_n \end{bmatrix} = \begin{bmatrix} m_1 - 1 - x_1, & \ldots, & m_n - 1 - x_n \\ m_1, & \ldots, & m_n \end{bmatrix}.$$

In general, the reflected Gray code for radices $(m_1, \ldots, m_n)$ is $(0, \ldots, m_1 - 1) \boxtimes \cdots \boxtimes (0, \ldots, m_n - 1)$. [*Information Processing Letters* **22** (1986), 201–205.]

86. Let Γ_{mn} be the reflected m-ary Gray code, which can be defined by $\Gamma_{m0} = \epsilon$ and

$$\Gamma_{m(n+1)} = (0, 1, \ldots, m-1) \boxtimes \Gamma_{mn}, \qquad n \geq 0.$$

This path runs from $(0, 0, \ldots, 0)$ to $(m-1, 0, \ldots, 0)$ when m is even. Consider the Gray path Π_{mn} defined by $\Pi_{m0} = \emptyset$ and

$$\Pi_{m(n+1)} = \begin{cases} (0, 1, \ldots, m-1) \boxtimes \Pi_{mn},\ m\Gamma^R_{(m+1)n}, & \text{if } m \text{ is odd;} \\ (0, 1, \ldots, m) \boxtimes \Pi_{mn},\ m\Gamma^R_{mn}, & \text{if } m \text{ is even.} \end{cases}$$

This path traverses all of the $(m+1)^n - m^n$ nonnegative integer n-tuples for which $\max(a_1, \ldots, a_n) = m$, starting with $(0, \ldots, 0, m)$ and ending with $(m, 0, \ldots, 0)$. The desired infinite Gray path is Π_{0n}, Π_{1n}^R, Π_{2n}, Π_{3n}^R, ...

87. This is impossible when n is odd, because the n-tuples with $\max(|a_1|, \ldots, |a_n|) = 1$ include $\frac{1}{2}(3^n + 1)$ with odd parity and $\frac{1}{2}(3^n - 3)$ with even parity. When $n = 2$ we can use a spiral Σ_0, Σ_1, Σ_2, ..., where Σ_m winds counterclockwise from $(m, 1-m)$ to $(m, -m)$ when $m > 0$. For even values of $n \geq 2$, if T_m is a path of n-tuples from $(m, 1-m, m-1, 1-m, \ldots, m-1, 1-m)$ to $(m, -m, m, -m, \ldots, m, -m)$, we can use $\Sigma_m \boxtimes (T_0, \ldots, T_{m-1}), (\Sigma_0, \ldots, \Sigma_m)^R \mathbin{\bar{\boxtimes}} T_m$ for $(n+2)$-tuples with the same property, where $\bar{\boxtimes}$ is the dual operation

$$\Gamma \mathbin{\bar{\boxtimes}} \Gamma' = (\alpha_0\alpha'_0, \ldots, \alpha_{t-1}\alpha'_0, \alpha_{t-1}\alpha'_1, \ldots, \alpha_0\alpha'_1, \alpha_0\alpha'_2, \ldots, \alpha_{t-1}\alpha'_2, \alpha_{t-1}\alpha'_3, \ldots).$$

[Infinite n-dimensional Gray codes *without* the magnitude constraint were first constructed by E. Vázsonyi, *Acta Litterarum ac Scientiarum*, sectio Scientiarum Mathematicarum **9** (Szeged: 1938), 163–173.]

88. It would visit all the subforests again, but in reverse order, ending with $(0, \ldots, 0)$ and returning to the state it had after the initialization step K1. (This reflection principle is, in fact, the key to understanding how Algorithm K works.)

89. (a) Let $M_0 = \epsilon$, $M_1 = \bullet$, and $M_{n+2} = \bullet M_{n+1}^R$, $-M_n^R$. This construction works because the last element of M_{n+1}^R is the first element of M_{n+1}, namely a dot followed by the first element of M_n^R.

(b) Given a string $d_1 \ldots d_l$ where each d_j is $\bullet$ or $-$, we can find its successor by letting $k = l - [d_l = \bullet]$ and proceeding as follows: If k is odd and $d_k = \bullet$, change $d_k d_{k+1}$ to $-$; if k is even and $d_k = -$, change d_k to $\bullet\bullet$; otherwise decrease k by 1 and repeat until either making a change or reaching $k = 0$. The successor of the given word is $\bullet - - \bullet\bullet\bullet - \bullet - \bullet$.

90. A cycle can exist only when the number of code words is even, since the number of dashes changes by ± 1 at each step. Thus we must have $n \bmod 3 = 2$. The Gray paths M_n of exercise 89 are not suitable; they begin with $(\bullet -)^{\lfloor n/3 \rfloor} \bullet^{n \bmod 3}$ and end with $(- \bullet)^{\lfloor n/3 \rfloor} \bullet^{[n \bmod 3 = 1]} -^{[n \bmod 3 = 2]}$. But $M_{3k+1}\bullet$, $M_{3k}^R -$ is a Hamiltonian cycle in the Morse code graph when $n = 3k + 2$.

91. Equivalently, the n-tuples $a_1\bar{a}_2a_3\bar{a}_4\ldots$ have no two consecutive 1s. Such n-tuples correspond to Morse code sequences of length $n+1$, if we append 0 and then represent $\bullet$ and $-$ respectively by 0 and 10. Under this correspondence we can convert the path M_{n+1} of exercise 89 into a procedure like Algorithm K, with the fringe containing the indices where each dot or dash begins (except for a final dot):

Q1. [Initialize.] Set $a_j \leftarrow \lfloor ((j-1) \bmod 6)/3 \rfloor$ and $f_j \leftarrow j$ for $1 \leq j \leq n$. Also set $f_0 \leftarrow 0$, $r_0 \leftarrow 1$, $l_1 \leftarrow 0$, $r_j \leftarrow j + (j \bmod 3)$ and $l_{j+(j \bmod 3)} \leftarrow j$ for $1 \leq j \leq n$, except if $j + (j \bmod 3) > n$ set $r_j \leftarrow 0$ and $l_0 \leftarrow j$. (The "fringe" now contains 1, 2, 4, 5, 7, 8, ...)

Q2. [Visit.] Visit the n-tuple $(a_1, \ldots, a_n)$.

Q3. [Choose p.] Set $q \leftarrow l_0$, $p \leftarrow f_q$, $f_q \leftarrow q$.

Q4. [Check a_p.] Terminate the algorithm if $p = 0$. Otherwise set $a_p \leftarrow 1 - a_p$ and go to Q6 if $a_p + p$ is now even.

Q5. [Insert $p+1$.] If $p < n$, set $q \leftarrow r_p$, $l_q \leftarrow p+1$, $r_{p+1} \leftarrow q$, $r_p \leftarrow p+1$, $l_{p+1} \leftarrow p$. Go to Q7.

Q6. [Delete $p+1$.] If $p < n$, set $q \leftarrow r_{p+1}$, $r_p \leftarrow q$, $l_q \leftarrow p$.

Q7. [Make p passive.] Set $f_p \leftarrow f_{l_p}$ and $f_{l_p} \leftarrow l_p$. Return to Q2. ∎

This algorithm can also be derived as a special case of a considerably more general method due to Gang Li, Frank Ruskey, and D. E. Knuth, which extends Algorithm K by allowing the user to specify either $a_p \geq a_q$ or $a_p \leq a_q$ for each (parent, child) pair (p, q). [See Knuth and Ruskey, *Lecture Notes in Computer Science* **2635** (2004), 183–204.] A generalization in another direction, which produces all strings of length n that do not contain certain substrings, has been discovered by M. B. Squire, *Electronic J. Combinatorics* **3** (1996), #R17, 1–29.

Incidentally, it is amusing to note that the mapping $k \mapsto g(k)/2$ is a one-to-one correspondence between all binary n-tuples with no odd-length runs of 1s and all binary n-tuples with no two consecutive 1s.

92. Yes, because the digraph of all $(n-1)$-tuples $(x_1, \ldots, x_{n-1})$ with $x_1, \ldots, x_{n-1} \leq m$ and with arcs $(x_1, \ldots, x_{n-1}) \to (x_2, \ldots, x_n)$ whenever $\max(x_1, \ldots, x_n) = m$ is connected and balanced; see Theorem 2.3.4.2G. Indeed, we get such a sequence from Algorithm F if we note that the final k^n elements of the prime strings of length dividing n, when subtracted from $m-1$, are the same for all $m \geq k$. When $n = 4$, for example, the first 81 digits of the sequence Φ_4 are $2 - \alpha^R = 0\,0001\,01\,0011\ldots$, where α is the string (62). [There also are infinite m-ary sequences whose first m^n elements are de Bruijn cycles for all n, given any fixed $m \geq 3$. See L. J. Cummings and D. Wiedemann, *Cong. Numerantium* **53** (1986), 155–160.]

93. The cycle generated by $f()$ is a cyclic permutation of $\alpha 1$, where α has length $m^n - 1$ and ends with 1^{n-1}. The cycle generated by Algorithm R is a cyclic permutation of $\gamma = c_0 \ldots c_{m^{n+1}-1}$, where $c_k = (c_0 + b_0 + \cdots + b_{k-1}) \bmod m$ and $b_0 \ldots b_{m^{n+1}-1} = \beta = \alpha^m 1^m$.

If $x_0 \ldots x_n$ occurs in γ, say $x_j = c_{k+j}$ for $0 \leq j \leq n$, then $y_j = b_{k+j}$ for $0 \leq j < n$, where $y_j = (x_{j+1} - x_j) \bmod m$. [This is the connection with modular m-ary Gray code; see exercise 78.] Now if $y_0 \ldots y_{n-1} = 1^n$ we have $m^{n+1} - m - n < k \leq m^{n+1} - n$; otherwise there is an index k' such that $-n < k' < m^n - n$ and $y_0 \ldots y_{n-1}$ occurs in β at positions $k = (k' + r(m^n - 1)) \bmod m^{n+1}$ for $0 \leq r < m$. In both cases the m choices of k have different values of x_0, because the sum of all elements in α is $m-1$ (modulo m) when $n \geq 2$. [Algorithm R is valid also for $n = 1$ if $m \bmod 4 \neq 2$, because $m \perp \sum \alpha$ in that case.]

94. 0010203041121314223243344. (The underlined digits are effectively inserted into the interleaving of 00112234 with 34. Algorithm D can be used in general when $n = 1$ and $r = m - 2 \geq 0$; but it is pointless to do so, in view of (54).)

95. (a) Let $c_0 c_1 c_2 \ldots$ have period r. If r is odd we have $p = q = r$, so $r = pq$ only in the trivial case when $p = q = 1$ and $a_0 = b_0$. Otherwise $r/2 = \operatorname{lcm}(p, q) = pq/\gcd(p, q)$ by 4.5.2–(10), hence $\gcd(p, q) = 2$. In the latter case the $2n$-tuples $c_l c_{l+1} \ldots c_{l+2n-1}$ that occur are $a_j b_k \ldots a_{j+n-1} b_{k+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \equiv k$ (modulo 2), and $b_k a_j \ldots b_{k+n-1} a_{j+n-1}$ for $0 \leq j < p$, $0 \leq k < q$, $j \not\equiv k$ (modulo 2).

(b) The output would interleave two sequences $a_0 a_1 \ldots$ and $b_0 b_1 \ldots$ whose periods are respectively $m^n + r$ and $m^n - r$; the a's are the cycle of $f()$ with x^n changed to x^{n+1} and the b's are the cycle of $f'()$ with x^n changed to x^{n-1}, for $0 \leq x < r$. By (58) and part (a), the period length is $m^{2n} - r^2$, and every $2n$-tuple occurs with the exception of $(xy)^n$ for $0 \leq x, y < r$.

(c) The real step D6 alters the behavior of (b) by going to D3 when $t \geq n$, $t' = n$, and $0 \leq x' = x < r$; this change emits an extra x at the time when x^{2n-1} has just been output and b is about to be emitted, where b is the digit following x^n in the cycle. D6 also allows control to pass to D7 and then D3 with $t' = n$ in the case that $t \geq n$ and $x < x' < r$; this behavior emits an extra $x'x$ at the time when $(xx')^{n-1}x$ has just been output and b will be next. These r^2 extra digits provide the r^2 missing $2n$-tuples of (b).

96. (a) The recurrences $S_2 = 1$, $S_{2n+1} = S_{2n} = 2S_n$, $R_2 = 0$, $R_{2n+1} = 1 + R_{2n}$, $R_{2n} = 2R_n$, $D_2 = 0$, $D_{2n+1} = D_{2n} = 1 + 2D_n$ have the solution $S_n = 2^{\lfloor \lg n \rfloor - 1}$, $R_n = n - 2S_n$, $D_n = S_n - 1$. Thus $S_n + R_n + D_n = n - 1$.

(b) Each top-level output usually involves $\lfloor \lg n \rfloor - 1$ D-activations and $\nu(n) - 1$ R-activations, plus one basic activation at the bottom level. But there are exceptions: Algorithm R might invoke its $f()$ twice, if the first activation completed a sequence 1^n; and sometimes Algorithm R doesn't need to invoke $f()$ at all. Algorithm D might invoke its $f'()$ twice, if the first activation completed a sequence $(x')^n$; but sometimes Algorithm D doesn't need to invoke either $f()$ or $f'()$.

Algorithm R completes a sequence x^{n+1} if and only if its child $f()$ has just completed a sequence 0^n. Algorithm D completes a sequence x^{2n} for $x < r$ if and only if it has just jumped from D6 to D3 without invoking any child.

From these observations we can conclude that at most $\lfloor \lg n \rfloor + \nu(n) + 1$ activations are possible per top-level output, if $r > 1$; such a case happens when Algorithm D for $n = 6$ goes from D6 to D4. But when $r = 1$ we can have as many as $2\lfloor \lg n \rfloor + 3$ activations, for example when Algorithm R for $n = 25$ goes from R4 to R2.

97. (a) (0011), (00011101), (0000101001111011), and (0000011000101101111 1001110101001). Thus $j_2 = 2$, $j_3 = 3$, $j_4 = 9$, $j_5 = 15$.

(b) We obviously have $f_{n+1}(k) = \Sigma f_n(k) \bmod 2$ for $0 \leq k < j_n + n$. The next value, $f_{n+1}(j_n + n)$, depends on whether step R4 jumps to R2 after computing $y = f_n(j_n + n - 1)$. If it does (namely, if $f_{n+1}(j_n + n - 1) \neq 0$), we have $f_{n+1}(k) \equiv 1 + \Sigma(k+1)$ for $j_n + n \leq k < 2^n + j_n + n$; otherwise we have $f_{n+1}(k) \equiv 1 + \Sigma(k-1)$ for those values of k. In particular, $f_{n+1}(k) = 1$ when $2^n \leq k + \delta_n \leq 2^n + n$. The stated formula, which has simpler ranges for the index k, holds because $1 + \Sigma(k \pm 1) \equiv \Sigma(k)$ when $j_n < k < j_n + n$ or $2^n + j_n < k < 2^n + j_n + n$.

(c) The interleaved cycle has $c_n(2k) = f_n^+(k)$ and $c_n(2k+1) = f_n^-(k)$, where

$$f_n^+(k) = \begin{cases} f_n(k-1), & \text{if } 0 < k \leq j_n + 1; \\ f_n(k-2), & \text{if } j_n + 1 < k \leq 2^n + 2; \end{cases} \qquad f_n^-(k) = \begin{cases} f_n(k+1), & \text{if } 0 \leq k < j_n; \\ f_n(k+2), & \text{if } j_n \leq k < 2^n - 2; \end{cases}$$

$f_n^+(k) = f_n^+(k \bmod (2^n + 2))$, $f_n^-(k) = f_n^-(k \bmod (2^n - 2))$. Therefore the subsequence 1^{2n-1} begins at position $k_n = (2^{n-1} - 2)(2^n + 2) + 2j_n + 2$ in the c_n cycle; this will make j_{2n} odd. The subsequence $(01)^{n-1}0$ begins at position $l_n = (2^{n-1} + 1)(j_n - 1)$ if $j_n \bmod 4 = 1$, at $l_n = (2^{n-1} + 1)(2^n + j_n - 3)$ if $j_n \bmod 4 = 3$. Also $k_2 = 6$, $l_2 = 2$.

(d) Algorithm D inserts four elements into the c_n cycle; hence

when $j_n \bmod 4 < 3$ $(l_n < k_n)$: when $j_n \bmod 4 = 3$ $(k_n < l_n)$:

$$f_{2n}(k) = \begin{cases} c_n(k-1), & \text{if } 0 < k \leq l_n + 2; \\ c_n(k-3), & \text{if } l_n + 2 < k \leq k_n + 3; \\ c_n(k-4), & \text{if } k_n + 3 < k \leq 2^{2n}; \end{cases} = \begin{cases} c_n(k-1), & \text{if } 0 < k \leq k_n + 1; \\ c_n(k-2), & \text{if } k_n + 1 < k \leq l_n + 3; \\ c_n(k-4), & \text{if } l_n + 3 < k \leq 2^{2n}. \end{cases}$$

(e) Consequently $j_{2n} = k_n + 1 + 2[j_n \bmod 4 < 3]$. Indeed, the elements preceding 1^{2n} consist of $2^{n-2} - 1$ complete periods of $f_n^+()$ interleaved with 2^{n-2} complete periods of $f_n^-()$, with one 0 inserted and also with 10 inserted if $l_n < k_n$, followed

by $f_n(1)f_n(1)f_n(2)f_n(2)\ldots f_n(j_n-1)f_n(j_n-1)$. The sum of all these elements is odd, unless $l_n < k_n$; therefore $\delta_{2n} = 1 - 2[j_n \bmod 4 = 3]$.

Let $n = 2^t q$, where q is odd and $n > 2$. The recurrences imply that, if $q = 1$, we have $j_n = 2^{n-1} + b_t$ where $b_t = 2^t/3 - (-1)^t/3$. And if $q > 1$ we have $j_n = 2^{n-1} \pm b_{t+2}$, where the + sign is chosen if and only if $\lfloor \lg q \rfloor + [\lfloor 4q/2^{\lfloor \lg q \rfloor} \rfloor = 5]$ is even.

98. If $f(k) = g(k)$ when k lies in a certain range, there's a constant C such that $\Sigma f(k) = C + \Sigma g(k)$ for k in that range. We can therefore continue almost mindlessly to derive additional recurrences: If $n > 1$ we have

$$\Sigma f_{2n}(k),\ \text{when } j_n \bmod 4 < 3\ (l_n < k_n): \qquad \text{when } j_n \bmod 4 = 3\ (k_n < l_n):$$
$$\equiv \begin{cases} \Sigma c_n(k-1), & \text{if } 0 < k \le l_n+2; \\ 1+\Sigma c_n(k-3), & \text{if } l_n+2 < k \le k_n+3; \\ \Sigma c_n(k-4), & \text{if } k_n+3 < k \le 2^{2n}; \end{cases} \quad \equiv \begin{cases} \Sigma c_n(k-1), & \text{if } 0 < k \le k_n+1; \\ 1+\Sigma c_n(k-2), & \text{if } k_n+1 < k \le l_n+3; \\ \Sigma c_n(k-4), & \text{if } l_n+3 < k \le 2^{2n}. \end{cases}$$

$$\Sigma c_n(k) \equiv \Sigma f_n^+(\lceil k/2 \rceil) + \Sigma f_n^-(\lfloor k/2 \rfloor).$$

$$\Sigma f_n^+(k) \equiv \begin{cases} \Sigma f_n(k-1), & \text{if } 0 < k \le j_n+1; \\ 1+\Sigma f_n(k-2), & \text{if } j_n+1 < k \le 2^n+2; \end{cases} \quad \Sigma f_n^-(k) \equiv \begin{cases} \Sigma f_n(k+1), & \text{if } 0 \le k < j_n; \\ 1+\Sigma f_n(k+2), & \text{if } j_n \le k < 2^n-2; \end{cases}$$

$$\Sigma f_n^\pm(k) \equiv \lfloor k/(2^n \pm 2) \rfloor + \Sigma f_n^\pm(k \bmod (2^n \pm 2)); \quad \Sigma f_n(k) = \Sigma f_n(k \bmod 2^n).$$

$$\Sigma f_{2n+1}(k) \equiv \begin{cases} \Sigma\Sigma f_{2n}(k), & \text{if } 0 < k \le j_{2n} \text{ or } 2^{2n} + j_{2n} < k \le 2^{2n+1}; \\ 1 + k + \Sigma\Sigma f_{2n}(k + \delta_{2n}), & \text{if } j_{2n} < k \le 2^{2n} + j_{2n}. \end{cases}$$

$$\Sigma\Sigma f_{2n}(k),\ \text{when } j_n \bmod 4 < 3\ (l_n < k_n): \qquad \text{when } j_n \bmod 4 = 3\ (k_n < l_n):$$
$$\equiv \begin{cases} \Sigma\Sigma c_n(k-1), & \text{if } 0 < k \le l_n+2; \\ 1+k+\Sigma\Sigma c_n(k-3), & \text{if } l_n+2 < k \le k_n+3; \\ \Sigma\Sigma c_n(k-4), & \text{if } k_n+3 < k \le 2^{2n}; \end{cases} \quad \equiv \begin{cases} \Sigma\Sigma c_n(k-1), & \text{if } 0 < k \le k_n+1; \\ 1+k+\Sigma c_n(k-2), & \text{if } k_n+1 < k \le l_n+3; \\ 1+\Sigma\Sigma c_n(k-4), & \text{if } l_n+3 < k \le 2^{2n}. \end{cases}$$

$$\Sigma\Sigma f_{2n}(k) \equiv [j_n \bmod 4 < 3]\lfloor k/2^{2n} \rfloor + \Sigma\Sigma f_{2n}(k \bmod 2^{2n}).$$

And then, aha, there is closure:

$$\Sigma\Sigma c_n(2k) = \Sigma f_n^+(k), \qquad \Sigma\Sigma c_n(2k+1) = \Sigma f_n^-(k).$$

If $n = 2^t q$ where q is odd, the running time to evaluate $f_n(k)$ by this system of recursive formulas is $O(t + S(q))$, where $S(1) = 1$, $S(2k) = 1 + 2S(k)$, and $S(2k+1) = 1 + S(k)$. Clearly $S(k) < 2k$, so the evaluations involve at most $O(n)$ simple operations on n-bit numbers. In fact, the method is often significantly faster: If we average $S(k)$ over all k with $\lfloor \lg k \rfloor = s$ we get $(3^{s+1} - 2^{s+1})/2^s$, which is less than $3k^{\lg(3/2)} < 3k^{0.59}$. (Incidentally, if $k = 2^{s+1} - 1 - (2^{s-e_1} + 2^{s-e_2} + \cdots + 2^{s-e_t})$ we have $S(k) = s + 1 + e_t + 2e_{t-1} + 4e_{t-2} + \cdots + 2^t e_1$.)

99. A string that starts at position k in $f_n()$ starts at position $k^+ = k + 1 + [k > j_n]$ in $f_n^+()$ and at position $k^- = k - 1 - [k > j_n]$ in $f_n^-()$, except that 0^n and 1^n occur twice in $f_n^+()$ but not at all in $f_n^-()$.

To find $\gamma = a_0 b_0 \ldots a_{n-1} b_{n-1}$ in the cycle $f_{2n}()$, let $\alpha = a_0 \ldots a_{n-1}$ and $\beta = b_0 \ldots b_{n-1}$. Suppose α starts at position j and β at position k in $f_n()$, and assume that neither α nor β is 0^n or 1^n. If $j^+ \equiv k^+$ (modulo 2), let $l/2$ be a solution to the equation $j^+ + (2^n+2)x = k^- + (2^n-2)y$; we may take $l/2 = k + (2^n-2)(2^{n-3}(j-k) \bmod (2^{n-1}+1))$ if $j \ge k$, otherwise $l/2 = j + (2^n+2)(2^{n-3}(k-j) \bmod (2^{n-1}-1))$. Otherwise let $(l-1)/2 = k^+ + (2^n+2)x = j^- + (2^n-2)y$. Then γ starts at position l in the cycle $c_n()$; hence it starts at position $l + 1 + [l \ge k_n] + 2[l \ge l_n]$ in the cycle $f_{2n}()$.

Similar formulas hold when $\alpha \in \{0^n, 1^n\}$ or $\beta \in \{0^n, 1^n\}$ (but not both). Finally, 0^{2n}, 1^{2n}, $(01)^n$, and $(10)^n$ start respectively in positions 0, j_{2n}, $l_n + 1 + [k_n < l_n]$, and $l_n + 2 + [k_n < l_n]$.

To find $\beta = b_0 b_1 \dots b_n$ in $f_{n+1}()$ when n is even, suppose that the n-bit string $(b_0 \oplus b_1) \dots (b_{n-1} \oplus b_n)$ starts at position j in $f_n()$. Then β starts at position $k = j - \delta_n[j \geq j_n] + 2^n[j = j_n][\delta_n = 1]$ if $f_{n+1}(k) = b_0$, otherwise at position $k + (2^n - \delta_n, \delta_n, 2^n + \delta_n)$ according as $(j<j_n, j=j_n, j>j_n)$.

The running time of this recursion satisfies $T(n) = O(n) + 2T(\lfloor n/2 \rfloor)$, so it is $O(n \log n)$. [Exercises 97–99 are based on the work of J. Tuliani, who also has developed methods for certain larger values of m; see *Discrete Math.* **226** (2001), 313–336.]

100. No obvious defects are apparent, but extensive testing should be done before any sequence can be recommended. By contrast, the de Bruijn cycle produced implicitly by Algorithm F is a terrible source of supposedly random bits, even though it is n-distributed in the sense of Definition 3.5D, because 0s predominate at the beginning. Indeed, when n is prime, bits $tn + 1$ of that sequence are zero for $0 \leq t < (2^n - 2)/n$.

101. (a) Let β be a proper suffix of $\lambda\lambda'$ with $\beta \leq \lambda\lambda'$. Either β is a suffix of λ', whence $\lambda < \lambda' \leq \beta$, or $\beta = \alpha\lambda'$ and we have $\lambda < \alpha < \beta$.

Now $\lambda < \beta \leq \lambda\lambda'$ implies that $\beta = \lambda\gamma$ for some $\gamma \leq \lambda'$. But γ is a suffix of β with $1 \leq |\gamma| = |\beta| - |\lambda| < |\lambda'|$; hence γ is a proper suffix of λ', and $\lambda' < \gamma$. Contradiction.

(b) Any string of length 1 is prime. Combine adjacent primes by (a), in any order, until no further combination is possible. [See the more general results of M. P. Schützenberger, *Proc. Amer. Math. Soc.* **16** (1965), 21–24.]

(c) If $t \neq 0$, let λ be the smallest suffix of $\lambda_1 \dots \lambda_t$. Then λ is prime by definition, and it has the form $\beta\gamma$ where β is a nonempty suffix of some λ_j. Therefore $\lambda_t \leq \lambda_j \leq \beta \leq \beta\gamma = \lambda \leq \lambda_t$, so we must have $\lambda = \lambda_t$. Remove λ_t and repeat until $t = 0$.

(d) True. For if we had $\alpha = \lambda\beta$ for some prime λ with $|\lambda| > |\lambda_1|$, we could append the factors of β to obtain another factorization of α.

(e) $3 \cdot 14159265358979323846264338327 95 \cdot 02884197$. (An efficient algorithm appears in exercise 106. Knowing more digits of π would not change the first two factors. The infinite decimal expansion of any number that is "normal" in the sense of Borel (see Section 3.5) factors into primes of finite length.)

102. We must have $1/(1 - mz) = 1/\prod_{n=1}^{\infty}(1 - z^n)^{L_m(n)}$. This implies (6o) as in exercise 4.6.2–4.

103. When $n = p$ is prime, (59) tells us that $L_m(1) + pL_m(p) = m^p$, and we also have $L_m(1) = m$. [This combinatorial proof provides an interesting contrast to the traditional algebraic proof of Theorem 1.2.4F.]

104. The 4483 nonprimes are **abaca**, **agora**, **ahead**, ...; the 1274 primes are ..., **rusts**, **rusty**, **rutty**. (Since **prime** isn't prime, we should perhaps call prime strings **lowly**.)

105. (a) Let α' be α with its last letter increased, and suppose $\alpha' = \beta\gamma'$ where $\alpha = \beta\gamma$ and $\beta \neq \epsilon$, $\gamma \neq \epsilon$. Let θ be the prefix of α with $|\theta| = |\gamma|$. By hypothesis there is a string ω such that $\alpha\omega$ is prime; hence $\theta \leq \alpha\omega < \gamma\omega$, so we must have $\theta \leq \gamma$. Consequently $\theta < \gamma'$, and we have $\alpha' < \gamma'$.

(b) Let $\alpha = \lambda_1\beta = a_1 \dots a_n$ where $\lambda_1\beta\omega$ is prime. The condition $\lambda_1\beta\omega < \beta\omega$ implies that $a_j \leq a_{j+r}$ for $1 \leq j \leq n-r$, where $r = |\lambda_1|$. But we cannot have $a_j < a_{j+r}$; otherwise α would begin with a prime longer than λ_1, contradicting exercise 101(d).

(c) If α is the n-extension of both λ and λ', where $|\lambda| > |\lambda'|$, we must have $\lambda = (\lambda')^q\theta$ where θ is a nonempty prefix of λ'. But then $\theta \leq \lambda' < \lambda < \theta$.

106. **B1.** [Initialize.] Set $a_1 \leftarrow \cdots \leftarrow a_n \leftarrow m-1$, $a_{n+1} \leftarrow -1$, and $j \leftarrow 1$.

B2. [Visit.] Visit $(a_1, \ldots, a_n)$ with index j.

B3. [Subtract one.] Terminate if $a_j = 0$. Otherwise set $a_j \leftarrow a_j - 1$, and $a_k \leftarrow m-1$ for $j < k \le n$.

B4. [Prepare to factor.] (According to exercise 105(b), we now want to find the first prime factor λ_1 of $a_1 \ldots a_n$.) Set $j \leftarrow 1$ and $k \leftarrow 2$.

B5. [Find the new j.] (Now $a_1 \ldots a_{k-1}$ is the $(k-1)$-extension of the prime $a_1 \ldots a_j$.) If $a_{k-j} > a_k$, return to B2. Otherwise, if $a_{k-j} < a_k$, set $j \leftarrow k$. Then increase k by 1 and repeat this step. ▮

The efficient factoring algorithm in steps B4 and B5 is due to J. P. Duval, *J. Algorithms* **4** (1983), 363–381. For further information, see Cattell, Ruskey, Sawada, Serra, and Miers, *J. Algorithms* **37** (2000), 267–282.

107. The number of n-tuples visited is $P_m(n) = \sum_{j=1}^{n} L_m(j)$. Since $L_m(n) = \frac{1}{n}m^n + O(m^{n/2}/n)$, we have $P_m(n) = Q(m,n) + O(Q(\sqrt{m}, n))$, where

$$Q(m,n) = \sum_{k=1}^{n} \frac{m^k}{k} = \frac{m^n}{n} R(m,n);$$

$$R(m,n) = \sum_{k=0}^{n-1} \frac{m^{-k}}{1-k/n} = \sum_{k=0}^{n/2} \frac{m^{-k}}{1-k/n} + O(nm^{-n/2})$$

$$= \frac{m}{m-1} \sum_{j=0}^{t-1} \frac{1}{n^j} \sum_{l} \left\{ {j \atop l} \right\} \frac{l!}{(m-1)^l} + O(n^{-t}).$$

Thus $P_m(n) \sim m^{n+1}/((m-1)n)$. The main contributions to the running time come from the loops in steps F3 and F5, which cost $n-j$ for each prime of length j, hence a total of $nP_m(n) - \sum_{j=1}^{n} jL_m(j) = m^{n+1}(1/((m-1)^2 n) + O(1/(mn^2)))$. This is less than the time needed to output the m^n individual digits of the de Bruijn cycle.

108. (a) If $\alpha \neq 9 \ldots 9$, we have $\lambda_{k+1} \le \beta 9^{|\alpha|}$, because the latter is prime.

(b) We can assume that β is not all 0s, since $9^j 0^{n-j}$ is a substring of $\lambda_{t-1}\lambda_t\lambda_1\lambda_2 = 89^n 0^n 1$. Let k be minimal with $\beta \le \lambda_k$; then $\lambda_k \le \beta\alpha$, so β is a prefix of λ_k. Since β is a preprime, it is the $|\beta|$-extension of some prime $\beta' \le \beta$. The preprime visited by Algorithm F just before β' is $(\beta'-1)9^{n-|\beta'|}$, by exercise 106, where $\beta'-1$ denotes the decimal number that is one less than β'. Thus, if β' is not λ_{k-1}, the hint (which also follows from exercise 106) implies that λ_{k-1} ends with at least $n - |\beta'| \ge n - |\beta|$ 9s, and α is a suffix of λ_{k-1}. On the other hand if $\beta' = \lambda_{k-1}$, α is a suffix of λ_{k-2}, and β is a prefix of $\lambda_{k-1}\lambda_k$.

(c) If $\alpha \neq 9 \ldots 9$, we have $\lambda_{k+1} \le (\beta\alpha)^{d-1}\beta 9^{|\alpha|}$, because the latter is prime. Otherwise λ_{k-1} ends with at least $(d-1)|\beta\alpha|$ 9s, and $\lambda_{k+1} \le (\beta\alpha)^{d-1}9^{|\beta\alpha|}$, so $(\alpha\beta)^d$ is a substring of $\lambda_{k-1}\lambda_k\lambda_{k+1}$.

(d) Within the primes 135899 135914, 787899 787979, 129999 13 131314, 09 090911, 089999 09 090911, 118999 119 119122.

(e) Yes: In all cases, the position of $a_1 \ldots a_n$ precedes the position of the substring $a_1 \ldots a_{n-1}(a_n+1)$, if $0 \le a_n < 9$ (and if we assume that strings like $9^j 0^{n-j}$ occur at the beginning). Furthermore $9^j 0^{n-j-1}$ occurs only after $9^{j-1}0^{n-j}a$ has appeared for $1 \le a \le 9$, so we must not place 0 after $9^j 0^{n-j-1}$.

109. Suppose we want to locate the submatrix

$$\begin{pmatrix} (w_{n-1}\dots w_1w_0)_2 & (x_{n-1}\dots x_1x_0)_2 \\ (y_{n-1}\dots y_1y_0)_2 & (z_{n-1}\dots z_1z_0)_2 \end{pmatrix}.$$

The binary case $n=1$ is the given example, and if $n>1$ we can assume by induction that we only need to determine the leading bits a_{2n-1}, a_{2n-2}, b_{2n-1}, and b_{2n-2}. The case $n=3$ is typical: We must solve

$$\begin{aligned}
b_5 &= w_2, & b_4 &= x_2, & a_5\oplus b_5 &= y_2, & a_4\oplus b_4 &= z_2, & &\text{if } a_0=0,\ b_0=0;\\
b_4 &= w_2, & b'_5 &= x_2, & a_4\oplus b_4 &= y_2, & a_5\oplus b'_5 &= z_2, & &\text{if } a_0=0,\ b_0=1;\\
a_5\oplus b_5 &= w_2, & a_4\oplus b_4 &= x_2, & b_5 &= y_2, & b_4 &= z_2, & &\text{if } a_0=1,\ b_0=0;\\
a_4\oplus b_4 &= w_2, & a_5\oplus b'_5 &= x_2, & b_4 &= y_2, & b'_5 &= z_2, & &\text{if } a_0=1,\ b_0=1;
\end{aligned}$$

here $b'_5=b_5\oplus b_4b_3b_2b_1$ takes account of carrying when j becomes $j+1$.

110. Let $a_0a_1\dots a_{m^2-1}$ be an m-ary de Bruijn cycle, such as the first m^2 elements of (54). If m is odd, let $d_{ij}=a_j$ when i is even, $d_{ij}=a_{(j+(i-1)/2)\bmod m^2}$ when i is odd. [The first of many people to discover this construction seems to have been John C. Cock, who also constructed de Bruijn toruses of other shapes and sizes in *Discrete Math.* **70** (1988), 209–210.]

If $m=m'm''$ where $m'\perp m''$, we use the Chinese remainder theorem to define

$$d_{ij}\equiv d'_{ij}\ (\text{modulo } m') \qquad \text{and} \qquad d_{ij}\equiv d''_{ij}\ (\text{modulo } m'')$$

in terms of matrices that solve the problem for m' and m''. Thus the previous exercise leads to a solution for arbitrary m.

Another interesting solution for even values of m was found by Zoltán Tóth [*2nd Conf. Automata, Languages, and Programming Systems* (1988), 165–172; see also Hurlbert and Isaak, *Contemp. Math.* **178** (1994), 153–160]. The first m^2 elements a_j of the infinite sequence

$$0011\,02133120322304152435534251405445\,0617263746577564\dots07667\,08\dots$$

define a de Bruijn cycle with the property that the distance between the appearances of ab and ba is always even. Then we can let $d_{ij}=a_j$ if $i+j$ is even, $d_{ij}=a_i$ if $i+j$ is odd. For example, when $m=4$ we have

$$\begin{pmatrix}
0\,0\,1\,0\,0\,2\,1\,2\,2\,0\,3\,0\,2\,2\,3\,2\\
0\,0\,0\,1\,0\,2\,0\,3\,2\,0\,2\,1\,2\,2\,2\,3\\
0\,1\,1\,1\,0\,3\,1\,3\,2\,1\,3\,1\,2\,3\,3\,3\\
1\,0\,1\,1\,1\,2\,1\,3\,3\,0\,3\,1\,3\,2\,3\,3\\
0\,0\,1\,0\,0\,2\,1\,2\,2\,0\,3\,0\,2\,2\,3\,2\\
0\,2\,0\,3\,0\,0\,0\,1\,2\,2\,2\,3\,2\,0\,2\,1\\
0\,1\,1\,1\,0\,3\,1\,3\,2\,1\,3\,1\,2\,3\,3\,3\\
1\,2\,1\,3\,1\,0\,1\,1\,3\,2\,3\,3\,3\,0\,3\,1\\
0\,0\,1\,0\,0\,2\,1\,2\,2\,0\,3\,0\,2\,2\,3\,2\\
2\,0\,2\,1\,2\,2\,2\,3\,0\,0\,0\,1\,0\,2\,0\,3\\
0\,1\,1\,1\,0\,3\,1\,3\,2\,1\,3\,1\,2\,3\,3\,3\\
3\,0\,3\,1\,3\,2\,3\,3\,1\,0\,1\,1\,1\,2\,1\,3\\
0\,0\,1\,0\,0\,2\,1\,2\,2\,0\,3\,0\,2\,2\,3\,2\\
2\,2\,2\,3\,2\,0\,2\,1\,0\,2\,0\,3\,0\,0\,0\,1\\
0\,1\,1\,1\,0\,3\,1\,3\,2\,1\,3\,1\,2\,3\,3\,3\\
3\,2\,3\,3\,3\,0\,3\,1\,1\,2\,1\,3\,1\,0\,1\,1
\end{pmatrix}$$ (exercise 109);

$$\begin{pmatrix}
0\,0\,1\,0\,0\,0\,1\,0\,3\,0\,2\,0\,3\,0\,2\,0\\
0\,0\,0\,1\,0\,2\,0\,3\,0\,1\,0\,0\,0\,2\,0\,3\\
0\,1\,1\,1\,0\,1\,1\,1\,3\,1\,2\,1\,3\,1\,2\,1\\
1\,0\,1\,1\,1\,2\,1\,3\,1\,1\,1\,0\,1\,2\,1\,3\\
0\,0\,1\,0\,0\,0\,1\,0\,3\,0\,2\,0\,3\,0\,2\,0\\
2\,0\,2\,1\,2\,2\,2\,3\,2\,1\,2\,0\,2\,2\,2\,3\\
0\,1\,1\,1\,0\,1\,1\,1\,3\,1\,2\,1\,3\,1\,2\,1\\
3\,0\,3\,1\,3\,2\,3\,3\,3\,1\,3\,0\,3\,2\,3\,3\\
0\,3\,1\,3\,0\,3\,1\,3\,3\,3\,2\,3\,3\,3\,2\,3\\
1\,0\,1\,1\,1\,2\,1\,3\,1\,1\,1\,0\,1\,2\,1\,3\\
0\,2\,1\,2\,0\,2\,1\,2\,3\,2\,2\,2\,3\,2\,2\,2\\
0\,0\,0\,1\,0\,2\,0\,3\,0\,1\,0\,0\,0\,2\,0\,3\\
0\,3\,1\,3\,0\,3\,1\,3\,3\,3\,2\,3\,3\,3\,2\,3\\
2\,0\,2\,1\,2\,2\,2\,3\,2\,1\,2\,0\,2\,2\,2\,3\\
0\,2\,1\,2\,0\,2\,1\,2\,3\,2\,2\,2\,3\,2\,2\,2\\
3\,0\,3\,1\,3\,2\,3\,3\,3\,1\,3\,0\,3\,2\,3\,3
\end{pmatrix}$$ (Tóth).

111. (a) Let $d_j = j$ and $0 \le a_j < 3$ for $1 \le j \le 9$, $a_9 \neq 0$. Form sequences s_j, t_j by the rules $s_1 = 0$, $t_1 = d_1$; $t_{j+1} = d_{j+1} + 10t_j[a_j = 0]$ for $1 \le j < 9$; $s_{j+1} = s_j + (0, t_j, -t_j)$ for $a_j = (0, 1, 2)$ and $1 \le j \le 9$. Then s_{10} is a possible result; we need only remember the smallish values that occur. More than half the work is saved by disallowing $a_k = 2$ when $s_k = 0$, then using $|s_{10}|$ instead of s_{10}. Since fewer than $3^8 = 6561$ possibilities need to be tried, brute force via the ternary version of Algorithm M works well; fewer than 24,000 mems and 1600 multiplications are needed to deduce that all integers less than 211 are representable, but 211 is not.

Another approach, using Gray code to vary the signs after breaking the digits into blocks in 2^8 possible ways, reduces the number of multiplications to 255, but at the cost of about 500 additional mems. Therefore Gray code is not advantageous in this application.

(b) Now (with 73,000 mems and 4900 multiplications) we can reach all numbers less than 241, but not 241. There are 46 ways to represent 100, including the remarkable $9 - 87 + 6 + 5 - 43 + 210$.

[H. E. Dudeney introduced his "century" problem in *The Weekly Dispatch* (4 and 18 June 1899). See also *The Numerology of Dr. Matrix* by Martin Gardner, Chapter 6; Steven Kahan, *J. Recreational Math.* **23** (1991), 19–25.]

112. The method of exercise 111 now needs more than 167 million mems and 10 million multiplications, because 3^{16} is so much larger than 3^8. We can do much better (10.4 million mems, 1100 mults) by first tabulating the possibilities obtainable from the first k and last k digits, for $1 \le k < 9$, then considering all blocks of digits that use the 9. There are 60,318 ways to represent 100, and the first unreachable number is 16,040.

SECTION 7.2.1.2

1. [J. P. N. Phillips, *Comp. J.* **10** (1967), 311.] Assuming that $n \geq 3$, we can replace steps L2–L4 by:

L2′. [Easiest case?] Set $y \leftarrow a_{n-1}$ and $z \leftarrow a_n$. If $y < z$, set $a_{n-1} \leftarrow z$, $a_n \leftarrow y$, and return to L1.

L2.1′. [Next easiest case?] Set $x \leftarrow a_{n-2}$. If $x \geq y$, go on to step L2.2′. Otherwise set $(a_{n-2}, a_{n-1}, a_n) \leftarrow (z, x, y)$ if $x < z$, (y, z, x) if $x \geq z$. Return to L1.

L2.2′. [Find j.] Set $j \leftarrow n-3$ and $y \leftarrow a_j$. If $y \geq x$, set $j \leftarrow j-1$, $x \leftarrow y$, $y \leftarrow a_j$, and repeat until $y < x$. Terminate if $j = 0$.

L3′. [Easy increase?] If $y < z$, set $a_j \leftarrow z$, $a_{j+1} \leftarrow y$, $a_n \leftarrow x$, and go to L4.1′.

L3.1′. [Increase a_j.] Set $l \leftarrow n-1$; if $y \geq a_l$, repeatedly decrease l by 1 until $y < a_l$. Then set $a_j \leftarrow a_l$ and $a_l \leftarrow y$.

L4′. [Begin to reverse.] Set $a_n \leftarrow a_{j+1}$ and $a_{j+1} \leftarrow z$.

L4.1′. [Reverse $a_{j+2} \ldots a_{n-1}$.] Set $k \leftarrow j+2$, $l \leftarrow n-1$. Then, if $k < l$, interchange $a_k \leftrightarrow a_l$, set $k \leftarrow k+1$, $l \leftarrow l-1$, and repeat until $k \geq l$. Return to L1. ▮

The program might run still faster if a_t is stored in memory location $\mathtt{A}[n-t]$ for $0 \leq t \leq n$, or if reverse colex order is used as in the following exercise.

2. Again we assume that $a_1 \leq a_2 \leq \cdots \leq a_n$ initially; the permutations generated from $\{1, 2, 2, 3\}$ will, however, be 1223, 2123, 2213, ..., 2321, 3221. Let a_{n+1} be an auxiliary element, *larger* than a_n.

L1. [Visit.] Visit the permutation $a_1 a_2 \ldots a_n$.

L2. [Find j.] Set $j \leftarrow 2$. If $a_{j-1} \geq a_j$, increase j by 1 until $a_{j-1} < a_j$. Terminate if $j > n$.

L3. [Decrease a_j.] Set $l \leftarrow 1$. If $a_l \geq a_j$, increase l until $a_l < a_j$. Then swap $a_l \leftrightarrow a_j$.

L4. [Reverse $a_1 \ldots a_{j-1}$.] Set $k \leftarrow 1$ and $l \leftarrow j-1$. Then, if $k < l$, swap $a_k \leftrightarrow a_l$, set $k \leftarrow k+1$, $l \leftarrow l-1$, and repeat until $k \geq l$. Return to L1. ▮

3. Let $C_1 \ldots C_n = c_{a_1} \ldots c_{a_n}$ be the inversion table, as in exercise 5.1.1–7. Then $\operatorname{rank}(a_1 \ldots a_n)$ is the mixed-radix number $\left[\begin{smallmatrix} C_1, & \ldots, & C_{n-1}, & C_n \\ n, & \ldots, & 2, & 1 \end{smallmatrix}\right]$. [See H. A. Rothe, *Sammlung combinatorisch-analytischer Abhandlungen* **2** (1800), 263–264; and see also the pioneering work of Nārāyaṇa cited in Section 7.2.1.7.] For example, 314592687 has rank $\left[\begin{smallmatrix} 2, & 0, & 1, & 1, & 4, & 0, & 0, & 1, & 0 \\ 9, & 8, & 7, & 6, & 5, & 4, & 3, & 2, & 1 \end{smallmatrix}\right] = 2 \cdot 8! + 6! + 5! + 4 \cdot 4! + 1! = 81577$; this is the factorial number system featured in Eq. 4.1–(10).

4. Use the recurrence $\operatorname{rank}(a_1 \ldots a_n) = \frac{1}{n} \sum_{j=1}^{t} n_j [x_j < a_1] \binom{n}{n_1, \ldots, n_t} + \operatorname{rank}(a_2 \ldots a_n)$. For example, rank(314159265) is

$$\tfrac{3}{9}\tbinom{9}{2,1,1,1,2,1,1} + 0 + \tfrac{2}{7}\tbinom{7}{1,1,1,2,1,1} + 0 + \tfrac{1}{5}\tbinom{5}{1,2,1,1} + \tfrac{3}{4}\tbinom{4}{1,1,1,1} + 0 + \tfrac{1}{2}\tbinom{2}{1,1} = 30991.$$

5. (a) Step L2 is performed $n!$ times. The probability that exactly k comparisons are made is $q_k - q_{k+1}$, where q_t is the probability that $a_{n-t+1} > \cdots > a_n$, namely $[t \leq n]/t!$. Therefore the mean is $\sum k(q_k - q_{k+1}) = q_1 + \cdots + q_n = \lfloor n!\, e \rfloor / n! - 1 \approx e - 1 \approx 1.718$, and the variance is

$$\sum k^2(q_k - q_{k+1}) - \text{mean}^2 = q_1 + 3q_2 + \cdots + (2n-1)q_n - (q_1 + \cdots + q_n)^2 \approx e(3-e) \approx 0.766.$$

[For higher moments, see R. Kemp, *Acta Informatica* **35** (1998), 17–89, Theorem 4.]

Incidentally, the average number of interchange operations in step L4 is therefore $\sum\lfloor k/2\rfloor(q_k - q_{k+1}) = q_2 + q_4 + \cdots \approx \cosh 1 - 1 = (e + e^{-1} - 2)/2 \approx 0.543$, a result due to R. J. Ord-Smith [*Comp. J.* **13** (1970), 152–155].

(b) Step L3 is performed only $n! - 1$ times, but we will assume for convenience that it occurs once more (with 0 comparisons). Then the probability that exactly k comparisons are made is $\sum_{j=k+1}^{n} 1/j!$ for $1 \le k < n$ and $1/n!$ for $k = 0$. Hence the mean is $\frac{1}{2}\sum_{j=0}^{n-2} 1/j! \approx e/2 \approx 1.359$; exercise 1 reduces this number by $\frac{2}{3}$. The variance is $\frac{1}{3}\sum_{j=0}^{n-3} 1/j! + \frac{1}{2}\sum_{j=0}^{n-2} 1/j! - \text{mean}^2 \approx \frac{5}{6}e - \frac{1}{4}e^2 \approx 0.418$.

6. (a) Let $e_n(z) = \sum_{k=0}^{n} z^k/k!$; then the number of different prefixes $a_1 \ldots a_j$ is $j!\,[z^j]\,e_{n_1}(z)\ldots e_{n_t}(z)$. This is $N = \binom{n}{n_1,\ldots,n_t}$ times the probability q_{n-j} that at least $n-j$ comparisons are made in step L2. Therefore the mean is $\frac{1}{N}w(e_{n_1}(z)\ldots e_{n_t}(z)) - 1$, where $w(\sum x_k z^k/k!) = \sum x_k$. In the binary case the mean is $M/\binom{n}{s} - 1$, where $M = \sum_{l=0}^{s}\sum_{k=l}^{n-s+l}\binom{k}{l} = \sum_{l=0}^{s}\binom{n-s+l+1}{l+1} = \binom{n+2}{s+1} - 1 = \binom{n}{s}(2 + \frac{s}{n-s+1} + \frac{n-s}{s+1}) - 1$.

(b) If $\{a_1, \ldots, a_j\} = \{n_1' \cdot x_1, \ldots, n_t' \cdot x_t\}$, the prefix $a_1 \ldots a_j$ contributes altogether $\sum_{1\le k<l\le t}(n_k - n_k')[n_l > n_l']$ to the total number of comparisons made in step L3. Thus the mean is $\frac{1}{N}\sum_{1\le k<l\le t} w(f_{kl}(z))$, where

$$f_{kl}(z) = \Bigg(\prod_{\substack{1\le m\le t\\ m\ne k,\ m\ne l}} e_{n_m}(z)\Bigg)\Bigg(\sum_{r=0}^{n_k}(n_k - r)\frac{z^r}{r!}\Bigg)e_{n_l-1}(z)$$

$$= e_{n_1}(z)\ldots e_{n_t}(z)(n_k - z r_k(z))r_l(z), \qquad \text{where } r_k(z) = \frac{e_{n_k-1}(z)}{e_{n_k}(z)}.$$

In the two-valued case this formula reduces to $\frac{1}{N}w((se_s(z) - ze_{s-1}(z))e_{n-s-1}(z)) = \frac{s}{N}(\binom{n+1}{s+1} - 1) - \frac{1}{N}(\binom{n+1}{s+1}(s - \frac{s+1}{n-s+1}) + 1) = \frac{1}{N}(-s - 1 + \binom{n+1}{s}) = \frac{n+1}{n-s+1} - \frac{s+1}{N}$.

7. In the notation of the previous answer, the quantity $\frac{1}{N}w(e_{n_1}(z)\ldots e_{n_t}(z)) - 1$ is

$$\frac{n_1 + \cdots + n_t}{n} + \frac{(n_1n_2 + n_1n_3 + \cdots + n_{t-1}n_t) + n_1(n_1-1) + \cdots + n_t(n_t-1)}{n(n-1)} + \cdots.$$

One can show using Eq. 1.2.9–(38) that the limit is $-1 + \exp\sum_{k\ge1} r_k/k$, where $r_k = \lim_{t\to\infty}(n_1^k + \cdots + n_t^k)/(n_1 + \cdots + n_t)^k$. In cases (a) and (b) we have $r_k = [k=1]$, so the limit is $e - 1 \approx 1.71828$. In case (c) we have $r_k = 1/(2^k - 1)$, so the limit is $-1 + \exp\sum_{k\ge1} 1/(k(2^k - 1)) \approx 2.46275$.

8. Assume that j is initially zero, and change step L1 to

L1′. [Visit.] Visit the variation $a_1 \ldots a_j$. If $j < n$, set $j \leftarrow j + 1$ and repeat this step. ▮

This algorithm is due to L. J. Fischer and K. C. Krause, *Lehrbuch der Combinationslehre und der Arithmetik* (Dresden: 1812), 55–57.

Incidentally, the total number of variations is $w(e_{n_1}(z)\ldots e_{n_t}(z))$ in the notation of answer 6. This counting problem was first treated by James Bernoulli in *Ars Conjectandi* (1713), Part 2, Chapter 9.

9. **R1.** [Visit.] Visit the variation $a_1 \ldots a_r$. (At this point $a_{r+1} \le \cdots \le a_n$.)

R2. [Easy case?] If $a_r < a_n$, interchange $a_r \leftrightarrow a_j$ where j is the smallest subscript such that $j > r$ and $a_j > a_r$, and return to R1.

R3. [Reverse.] Set $(a_{r+1}, \ldots, a_n) \leftarrow (a_n, \ldots, a_{r+1})$ as in step L4.

R4. [Find j.] Set $j \leftarrow r-1$. If $a_j \geq a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate if $j = 0$.

R5. [Increase a_j.] Set $l \leftarrow n$. If $a_j \geq a_l$, decrease l by 1 repeatedly until $a_j < a_l$. Then interchange $a_j \leftrightarrow a_l$.

R6. [Reverse again.] Set $(a_{j+1}, \ldots, a_n) \leftarrow (a_n, \ldots, a_{j+1})$ as in step L4, and return to R1. ▮

The number of outputs is $r![z^r]\, e_{n_1}(z) \ldots e_{n_t}(z)$; this is, of course, n^r when the elements are distinct.

10. $a_1 a_2 \ldots a_n = 213 \ldots n$, $c_1 c_2 \ldots c_n = 010 \ldots 0$, $o_1 o_2 \ldots o_n = 1(-1)1 \ldots 1$, if $n \geq 2$.

11. Step (P1, ..., P7) is performed $(1, n!, n!, n! + x_n, n!, (x_n+3)/2, x_n)$ times, where $x_n = \sum_{k=1}^{n-1} k!$, because P7 is performed $(j-1)!$ times when $2 \leq j \leq n$.

12. We want the permutation of rank 999999. The answers are (a) 2783915460, by exercise 3; (b) 8750426319, because the reflected mixed-radix number corresponding to $\left[\begin{smallmatrix} 0, & 0, & 1, & 2, & 3, & 0, & 2, & 7, & 0, & 9 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10 \end{smallmatrix}\right]$ is $\left[\begin{smallmatrix} 0, & 0, & 1, & 3-2, & 3, & 5-0, & 2, & 7, & 8-0, & 9-9 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10 \end{smallmatrix}\right]$ by 7.2.1.1–(50); (c) the product $(0\,1\,\ldots\,9)^9 (0\,1\,\ldots\,8)^0 (0\,1\,\ldots\,7)^7 (0\,1\,\ldots\,6)^2 \ldots (0\,1\,2)^1$, namely 9703156248.

13. The first statement is true for all $n \geq 2$. But when 2 crosses 1, namely when c_2 changes from 0 to 1, we have $c_3 = 2$, $c_4 = 3$, $c_5 = \cdots = c_n = 0$, and the next permutation when $n \geq 5$ is $432156 \ldots n$. [See *Time Travel* (1988), page 74.]

14. True at the beginning of steps P4, P5, and P6, because exactly $j-1-c_j+s$ elements lie to the left of x_j, namely $j-1-c_j$ from $\{x_1, \ldots, x_{j-1}\}$ and s from $\{x_{j+1}, \ldots, x_n\}$. (In a sense, this formula is the main point of Algorithm P.)

15. If $\left[\begin{smallmatrix} b_{n-1}, & \ldots, & b_0 \\ 1, & \ldots, & n \end{smallmatrix}\right]$ corresponds to the reflected Gray code $\left[\begin{smallmatrix} c_1, & \ldots, & c_n \\ 1, & \ldots, & n \end{smallmatrix}\right]$, we get to step P6 if and only if $b_k = k-1$ for $j \leq k \leq n$ and B_{n-j+1} is even, by 7.2.1.1–(50). But $b_{n-k} = k-1$ for $j \leq k \leq n$ implies that B_{n-k} is odd for $j < k \leq m$. Therefore $s = [c_{j+1} = j] + [c_{j+2} = j+1] = [o_{j+1} < 0] + [o_{j+2} < 0]$ in step P5. [See *Math. Comp.* **17** (1963), 282–285.]

16. **P1′.** [Initialize.] Set $c_j \leftarrow j$ and $o_j \leftarrow -1$ for $1 \leq j < n$; also set $z \leftarrow a_n$.

P2′. [Visit.] Visit $a_1 \ldots a_n$. Then go to P3.5′ if $a_1 = z$.

P3′. [Hunt down.] For $j \leftarrow n-1, n-2, \ldots, 1$ (in this order), set $a_{j+1} \leftarrow a_j$, $a_j \leftarrow z$, and visit $a_1 \ldots a_n$. Then set $j \leftarrow n-1$, $s \leftarrow 1$, and go to P4′.

P3.5′. [Hunt up.] For $j \leftarrow 1, 2, \ldots, n-1$ (in this order), set $a_j \leftarrow a_{j+1}$, $a_{j+1} \leftarrow z$, and visit $a_1 \ldots a_n$. Then set $j \leftarrow n-1$, $s \leftarrow 0$.

P4′. [Ready to change?] Set $q \leftarrow c_j + o_j$. If $q = 0$, go to P6′; if $q > j$, go to P7′.

P5′. [Change.] Interchange $a_{c_j+s} \leftrightarrow a_{q+s}$. Then set $c_j \leftarrow q$ and return to P2′.

P6′. [Increase s.] Terminate if $j = 1$; otherwise set $s \leftarrow s+1$.

P7′. [Switch direction.] Set $o_j \leftarrow -o_j$, $j \leftarrow j-1$, and go back to P4′. ▮

17. Initially $a_j \leftarrow a'_j \leftarrow j$ for $1 \leq j \leq n$. Step P5 should now set $t \leftarrow j - c_j + s$, $u \leftarrow j - q + s$, $v \leftarrow a_u$, $a_t \leftarrow v$, $a'_v \leftarrow t$, $a_u \leftarrow j$, $a'_j \leftarrow u$, $c_j \leftarrow q$. (See exercise 14.)

But with the inverse required and available we can actually simplify the algorithm significantly, avoiding the offset variable s and letting the control table $c_1 \ldots c_n$ count only downwards, as noted by G. Ehrlich [*JACM* **20** (1973), 505–506]:

Q1. [Initialize.] Set $a_j \leftarrow a'_j \leftarrow j$, $c_j \leftarrow j-1$, and $o_j \leftarrow -1$ for $1 \leq j \leq n$. Also set $c_0 = -1$.

Q2. [Visit.] Visit the permutation $a_1 \dots a_n$ and its inverse $a'_1 \dots a'_n$.

Q3. [Find k.] Set $k \leftarrow n$. Then if $c_k = 0$, set $c_k \leftarrow k-1$, $o_k \leftarrow -o_k$, $k \leftarrow k-1$, and repeat until $c_k \neq 0$. Terminate if $k = 0$.

Q4. [Change.] Set $c_k \leftarrow c_k - 1$, $j \leftarrow a'_k$, and $i = j + o_k$. Then set $t \leftarrow a_i$, $a_i \leftarrow k$, $a_j \leftarrow t$, $a'_t \leftarrow j$, $a'_k \leftarrow i$, and return to Q2. ∎

18. Set $a_n \leftarrow n$, and use $(n-1)!/2$ iterations of Algorithm P to generate all permutations of $\{1, \dots, n-1\}$ such that 1 precedes 2. [M. K. Roy, *CACM* **16** (1973), 312–313; see also exercise 13.]

19. For example, we can use the idea of Algorithm P, with the n-tuples $c_1 \dots c_n$ changing as in Algorithm 7.2.1.1H with respect to the radices $(1, 2, \dots, n)$. That algorithm maintains the directions correctly, although it numbers subscripts differently. The offset s needed by Algorithm P can be computed as in the answer to exercise 15, or the inverse permutation can be maintained as in exercise 17. [See G. Ehrlich, *CACM* **16** (1973), 690–691.] Other algorithms, like that of Heap, can also be implemented looplessly.

(*Note:* In most applications of permutation generation we are interested in minimizing the *total* running time, not the maximum time between successive visits; from this standpoint looplessness is usually undesirable, except on a parallel computer. Yet there's something intellectually satisfying about the fact that a loopless algorithm exists, whether practical or not.)

20. For example, when $n = 3$ we can begin 123, 132, 312, $\bar{3}12$, $1\bar{3}2$, $12\bar{3}$, $21\bar{3}$, ..., 213, $\bar{2}13$, ... If the delta sequence for n is $(\delta_1 \delta_2 \dots \delta_{2^n n!})$, the corresponding sequence for $n+1$ is $(\Delta_n \delta_1 \Delta_n \delta_2 \dots \Delta_n \delta_{2^n n!})$, where Δ_n is the sequence of $2n-1$ operations $n\ n{-}1\ \dots\ 1\ -\ 1\ \dots\ n{-}1\ n$; here $\delta_k = j$ means $a_j \leftrightarrow a_{j+1}$ and $\delta_k = -$ means $a_1 \leftarrow -a_1$.

(Signed permutations appear in another guise in exercises 5.1.4–43 and 44. The set of all signed permutations is called the octahedral group.)

21. Clearly $\mathtt{M} = 1$, hence $\mathtt{O}$ must be 0 and $\mathtt{S}$ must be $b-1$. Then $\mathtt{N} = \mathtt{E}+1$, $\mathtt{R} = b-2$, and $\mathtt{D} + \mathtt{E} = b + \mathtt{Y}$. This leaves exactly $\max(0, b-7-k)$ choices for $\mathtt{E}$ when $\mathtt{Y} = k \geq 2$, hence a total of $\sum_{k=2}^{b-7}(b-7-k) = \binom{b-8}{2}$ solutions when $b \geq 8$. [*Math. Mag.* **45** (1972), 48–49. Incidentally, D. Eppstein has proved that the task of solving alphametics with a given radix is NP-complete; see *SIGACT News* **18**, 3 (1987), 38–40.]

22. $(\mathtt{XY})_b + (\mathtt{XX})_b = (\mathtt{XYX})_b$ is solvable only when $b = 2$.

23. Almost true, because the number of solutions will be even, ***unless*** $[j \in F] \neq [k \in F]$. (Consider the ternary alphametic $\mathtt{X} + (\mathtt{XX})_3 + (\mathtt{YY})_3 + (\mathtt{XZ})_3 = (\mathtt{XYX})_3$.)

24. (a) $9283 + 7 + 473 + 1062 = 10825$. (b) $698392 + 3192 = 701584$. (c) $63952 + 69275 = 133227$. (d) $653924 + 653924 = 1307848$. (e) $5718 + 3 + 98741 = 104462$. (f) $127503 + 502351 + 3947539 + 46578 = 4623971$. (g) $67432 + 704 + 8046 + 97364 = 173546$. (h) $59 + 577404251698 + 69342491650 + 49869442698 + 1504 + 40614 + 82591 + 344 + 41 + 741425 = 5216367650 + 691400684974$. [All solutions are unique. References for (b)–(g): *J. Recreational Math.* **10** (1977), 115; **5** (1972), 296; **10** (1977), 41; **10** (1978), 274; **12** (1979), 133–134; **9** (1977), 207.]

(i) In this case there are $\frac{8}{10}10! = 2903040$ solutions, because *every* permutation of $\{0, 1, \dots, 9\}$ works except those that assign $\mathtt{H}$ or $\mathtt{N}$ to 0. (A well-written general additive alphametic solver will be careful to reduce the amount of output in such cases.)

25. We may assume that $s_1 \leq \dots \leq s_{10}$. Let i be the least index $\notin F$, and set $a_i \leftarrow 0$; then set the remaining elements a_j in order of increasing j. A proof like that

of Theorem 6.1S shows that this procedure maximizes $a \cdot s$. A similar procedure yields the minimum, because $\min(a \cdot s) = -\max(a \cdot (-s))$.

26. $400739 + 63930 - 2379 - 1252630 + 53430 - 1390 + 738300$.

27. Readers can probably improve upon the following examples: BLOOD + SWEAT + TEARS = LATER; EARTH + WATER + WRATH = HELLO + WORLD; AWAIT + ROBOT + ERROR = SOBER + WORDS; CHILD + THEME + PEACE + ETHIC = IDEAL + ALPHA + METIC. (This exercise was inspired by WHERE + SEDGE + GRASS + GROWS = MARSH [A. W. Johnson, Jr., *J. Recr. Math.* **15** (1982), 51], which would be marvelously pure except that D and O have the same signature.)

28. (a) $11 = 3+3+2+2+1$, $20 = 11+3+3+3$, $20 = 11+3+3+2+1$, $20 = 11+3+3+1+1+1$, $20 = 8+8+2+1+1$, $20 = 7+7+6$, $20 = 7+7+2+2+2$, $20 = 7+7+2+1+1+1+1$, $20 = 7+5+5+2+1$, $20 = 7+5+2+2+2+1+1$, $20 = 7+5+2+2+1+1+1+1$, $20 = 7+3+3+2+2+1+1+1$, $20 = 7+3+3+1+1+1+1+1+1+1$, $20 = 5+3+3+3+3+3$. [These fourteen solutions were first computed by Roy Childs in 1999. The next doubly partitionable values of n are 30 (in 20 ways), then 40 (in 94 ways), 41 (in 67), 42 (in 57), 50 (in 190 ways, including $50 = 2+2+\cdots+2$), etc.]

(b) $51 = 20+15+14+2$, $51 = 15+14+10+9+3$, $61 = 19+16+11+9+6$, $65 = 17+16+15+9+7+1$, $66 = 20+19+16+6+5$, $69 = 18+17+16+10+8$, $70 = 30+20+10+7+3$, $70 = 20+16+12+9+7+6$, $70 = 20+15+12+11+7+5$, $80 = 50+20+9+1$, $90 = 50+12+11+9+5+2+1$, $91 = 45+19+11+10+5+1$. [The two 51s are due to Steven Kahan; see his book *Have Some Sums To Solve* (Farmingdale, New York: Baywood, 1978), 36–37, 84, 112. Amazing examples with seventeen distinct terms in Italian and fifty-eight distinct terms in Roman numerals have been found by Giulio Cesare, *J. Recr. Math.* **30** (1999), 63.]

Notes: The beautiful example THREE = TWO+ONE+ZERO [Richard L. Breisch, *Recreational Math. Magazine* **12** (December 1962), 24] is unfortunately ruled out by our conventions. The total number of doubly true partitions into distinct parts is probably finite, in English, although nomenclature for arbitrarily large integers is not standard. Is there an example bigger than NINETYNINENONILLIONNINETYNINESEXTILLIONSIXTYONE = NINETYNINENONILLIONNINETYNINESEXTILLIONNINETEEN+SIXTEEN+ELEVEN+NINE+SIX (suggested by G. González-Morris)?

29. $10+7+1 = 9+6+3$, $11+10 = 8+7+6$, $12+7+6+5 = 11+10+9$, ..., $19+10+3 = 14+13+4+1$ (31 examples in all).

30. (a) $567^2 = 321489$, $807^2 = 651249$, or $854^2 = 729316$. (b) $958^2 = 917764$. (c) $96 \times 7^2 = 4704$. (d) $51304/61904 = 7260/8760$. (e) $328509^2 = 4761^3$. [*Strand* **78** (1929), 91, 208; *J. Recr. Math* **3** (1970), 43; **13** (1981), 212; **27** (1995), 137; **31** (2003), 133. The solutions to (b), (c), (d), and (e) are unique. With a right-to-left approach based on Algorithm X, the answers are found in (14, 13, 11, 3423, 42) kilomems, respectively. Nob also noticed that NORTH/SOUTH = WEST/EAST has the unique solution $67104/27504 = 9320/3820$.]

31. $5/34 + 7/68 + 9/12$(!). One can verify uniqueness with Algorithm X using the side condition A < D < G, in about 265 Kμ. [*Quark Visual Science Magazine*, No. 136 (Tokyo: Kodansha, October 1993).] Curiously, a very similar puzzle also has a unique solution: $1/(3 \times 6) + 5/(8 \times 9) + 7/(2 \times 4) = 1$; see Scot Morris, *Omni* **17**, 4 (January 1995), 97.

32. There are eleven ways, of which the most surprising is $3 + 69258/714$. [See *The Weekly Dispatch* (9 and 23 June 1901); *Amusements in Mathematics* (1917), 158–159.]

33. (a) 1, 2, 3, 4, 15, 18, 118, 146. (b) 6, 9, 16, 20, 27, 126, 127, 129, 136, 145. [*The Weekly Dispatch* (11 and 30 November, 1902); *Amusements in Math.* (1917), 159.]

In this case one suitable strategy is to find all variations where $a_k \ldots a_{l-1}/a_l \ldots a_9$ is an integer, then to record solutions for all permutations of $a_1 \ldots a_{k-1}$. There are exactly 164959 integers with a unique solution, the largest being 9876533. There are solutions for all years in the 21st century except 2091. The most solutions (125) occur when $n = 6443$; the longest stretch of representable n's is $5109 < n < 7060$. Dudeney was able to get the correct answers by hand for small n by "casting out nines."

34. (a) $x = 10^5$, 7378+155+92467 = 7178+355+92467 = 1016+733+98251 = 100000. (b) $x = 4^7$, $3036 + 455 + 12893 = 16384$ is unique. The fastest way to resolve this problem is probably to start with a list of the 2529 primes that consist of five distinct digits (namely 10243, 10247, ..., 98731) and to permute the five remaining digits.

Incidentally, the unrestricted alphametic `EVEN + ODD = PRIME` has ten solutions; both `ODD` and `PRIME` are prime in just one of them. [See M. Arisawa, *J. Recr. Math.* **8** (1975), 153.]

35. In general, if $s_k = |S_k|$ for $1 \le k < n$, there are $s_1 \ldots s_{k-1}$ ways to choose each of the nonidentity elements of S_k. Hence the answer is $\prod_{k=1}^{n-1}(\prod_{j=1}^{k-1} s_j^{s_k-1})$, which in this case is $2^2 \cdot 6^3 \cdot 24^{15} = 436196692474023836123136$.

(But if the vertices are renumbered, the s_k values may change. For example, if vertices $(0,3,5)$ of (12) are interchanged with $(\mathtt{e},\mathtt{d},\mathtt{c})$, we have $s_{14} = 1$, $s_{13} = 6$, $s_{12} = 4$, $s_{11} = 1$, and $4^5 \cdot 24^{15}$ Sims tables.)

36. Since each of $\{0,3,5,6,9,\mathtt{a},\mathtt{c},\mathtt{f}\}$ lies on three lines, but every other element lies on only two, it is clear that we may let $S_\mathtt{f} = \{(), \sigma, \sigma^2, \sigma^3, \alpha, \alpha\sigma, \alpha\sigma^2, \alpha\sigma^3\}$, where $\sigma = (\mathtt{03fc})(\mathtt{17e4})(\mathtt{2bd4})(\mathtt{56a9})$ is a 90° rotation and $\alpha = (\mathtt{05})(\mathtt{14})(\mathtt{27})(\mathtt{36})(\mathtt{8d})(\mathtt{9c})(\mathtt{af})(\mathtt{be})$ is an inside-out twist. Also $S_\mathtt{e} = \{(), \beta, \gamma, \beta\gamma\}$, where $\beta = (\mathtt{14})(\mathtt{28})(\mathtt{3c})(\mathtt{69})(\mathtt{be})$ is a transposition and $\gamma = (\mathtt{12})(\mathtt{48})(\mathtt{5a})(\mathtt{69})(\mathtt{7b})(\mathtt{de})$ is another twist; $S_\mathtt{d} = \cdots = S_1 = \{()\}$. (There are $4^7 - 1$ alternative answers.)

37. The set S_k can be chosen in $k!^{k-1}$ ways (see exercise 35), and its nonidentity elements can be assigned to $\sigma(k,1), \ldots, \sigma(k,k)$ in $k!$ further ways. So the answer is $A_n = \prod_{k=1}^{n-1} k!^k = n!^{\binom{n}{2}} / \prod_{k=1}^{n} k^{\binom{k}{2}}$. For example, $A_{10} \approx 6.256 \times 10^{148}$. We have

$$\sum_{k=1}^{n-1} \binom{k}{2} \ln k = \frac{1}{2}\int_1^n x(x-1)\ln x\,dx + O(n^2 \log n) = \frac{1}{6}n^3 \ln n + O(n^3)$$

by Euler's summation formula; thus $\ln A_n = \frac{1}{3}n^3 \ln n + O(n^3)$.

38. The probability that $\phi(k)$ is needed in step G4 is $1/k! - 1/(k+1)!$, for $1 \le k < n$; the probability is $1/n!$ that we don't get to step G4 at all. Since $\phi(k)$ does $\lceil k/2 \rceil$ transpositions, the average is $\sum_{k=1}^{n-1}(1/k! - 1/(k+1)!)\lceil k/2 \rceil = \sum_{k=1}^{n-1}(\lceil k/2 \rceil - \lceil (k-1)/2 \rceil)/k! - \lceil (n-1)/2 \rceil/n! = \sum_{k\,\mathrm{odd}} 1/k! + O(1/(n-1)!)$.

39. (a) 0123, 1023, 2013, 0213, 1203, 2103, 3012, 0312, 1302, 3102, 0132, 1032, 2301, 3201, 0231, 2031, 3021, 0321, 1230, 2130, 3120, 1320, 2310, 3210; (b) 0123, 1023, 2013, 0213, 1203, 2103, 3102, 1302, 0312, 3012, 1032, 0132, 0231, 2031, 3021, 0321, 2301, 3201, 3210, 2310, 1320, 3120, 2130, 1230.

40. By induction we find $\sigma(1,1) = (0\ 1)$, $\sigma(2,2) = (0\ 1\ 2)$,

$$\sigma(k,k) = \begin{cases} (0\ k)(k{-}1\ k{-}2\ \ldots\ 1), & \text{if } k \ge 3 \text{ is odd,} \\ (0\ k{-}1\ k{-}2\ 1\ \ldots\ k{-}3\ k), & \text{if } k \ge 4 \text{ is even;} \end{cases}$$

also $\omega(k) = (0\ k)$ when k is even, $\omega(k) = (0\ k{-}2\ \ldots\ 1\ k{-}1\ k)$ when $k \geq 3$ is odd. Thus when $k \geq 3$ is odd, $\sigma(k,1) = (k\ k{-}1\ 0)$ and $\sigma(k,j)$ takes $k \mapsto j-1$ for $1 < j < k$; when $k \geq 4$ is even, $\sigma(k,j) = (0\ k\ k{-}3\ \ldots\ 1\ k{-}2\ k{-}1)^j$ for $1 \leq j \leq k$.

Notes: The first scheme that causes Algorithm G to generate all permutations by single transpositions was devised by Mark Wells [*Math. Comp.* **15** (1961), 192–195], but it was considerably more complicated. W. Lipski, Jr., studied such schemes in general and found a variety of additional methods [*Computing* **23** (1979), 357–365].

41. We may assume that $r < n$. Algorithm G will generate r-variations for any Sims table if we simply change '$k \leftarrow 1$' to '$k \leftarrow n - r$' in step G3, provided that we redefine $\omega(k)$ to be $\sigma(n-r, n-r)\ldots\sigma(k,k)$ instead of using (16).

If $n-r$ is odd, the method of (27) is still valid, although the formulas in answer 40 need to be revised when $k < n-r+2$. The new formulas are $\sigma(k,j) = (k\ j{-}1\ \ldots\ 1\ 0)$ and $\omega(k) = (k\ \ldots\ 1\ 0)$ when $k = n-r$; $\sigma(k,j) = (k\ \ldots\ 1\ 0)^j$ when $k = n-r+1$.

If $n-r$ is even, we can use (27) with even and odd reversed, if $r \leq 3$. But when $r \geq 4$ a more complex scheme is needed, because a fixed transposition like $(k\ 0)$ can be used for odd k only if $\omega(k-1)$ is a k-cycle, which means that $\omega(k-1)$ must be an even permutation; but $\omega(k)$ is odd for $k \geq n-r+2$.

The following scheme works when $n-r$ is even: Let $\tau(k,j)\omega(k-1)^- = (k\ k{-}j)$ for $1 \leq j \leq k = n-r$, and use (27) when $k > n-r$. Then, when $k = n-r+1$, we have $\omega(k-1) = (0\ 1\ \ldots\ k{-}1)$, hence $\sigma(k,j)$ takes $k \mapsto (2j-1) \bmod k$ for $1 \leq j \leq k$, and $\sigma(k,k) = (k\ k{-}1\ k{-}3\ \ldots\ 0\ k{-}2\ \ldots\ 1)$, $\omega(k) = (k\ \ldots\ 1\ 0)$, $\sigma(k{+}1,j) = (k{+}1\ \ldots\ 0)^j$.

42. If $\sigma(k,j) = (k\ j{-}1)$ we have $\tau(k,1) = (k\ 0)$ and $\tau(k,j) = (k\ j{-}1)(k\ j{-}2) = (k\ j{-}1\ j{-}2)$ for $2 \leq j \leq k$.

43. Of course $\omega(1) = \sigma(1,1) = \tau(1,1) = (0\ 1)$. The following construction makes $\omega(k) = (k{-}2\ k{-}1\ k)$ for all $k \geq 2$: Let $\alpha(k,j) = \tau(k,j)\omega(k{-}1)^-$, where $\alpha(2,1) = (2\ 0)$, $\alpha(2,2) = (2\ 0\ 1)$, $\alpha(3,1) = \alpha(3,3) = (3\ 1)$, $\alpha(3,2) = (3\ 1\ 0)$; this makes $\sigma(2,2) = (0\ 2)$, $\sigma(3,3) = (0\ 3\ 1)$. Then for $k \geq 4$, let $\alpha(k,1) = (k\ k{-}4)$, $\alpha(k,j) = (k\ k{-}3{-}j\ k{-}2{-}j)$ for $1 < j < k-2$, and

	$k \bmod 3 = 0$		$k \bmod 3 = 1$		$k \bmod 3 = 2$
$\alpha(k,k{-}2) =$	$(k\ k{-}2\ 0)$	or	$(k\ k{-}3\ 0)$	or	$(k\ k{-}1\ 0)$,
$\alpha(k,k{-}1) =$	$(k\ k{-}2\ k{-}3)$	or	$(k\ k{-}3)$	or	$(k\ k{-}1\ k{-}3)$,
$\alpha(k,k) =$	$(k\ k{-}2)$	or	$(k\ k{-}3\ k{-}2)$	or	$(k\ k{-}2)$;

this makes $\sigma(k,k) = (k{-}3\ k\ k{-}2)$ as required.

44. No, because $\tau(k,j)$ is a $(k+1)$-cycle, not a transposition. (See (19) and (24).)

45. (a) 202280070, since $u_k = \max\big(\{0,1,\ldots,a_k-1\} \setminus \{a_1,\ldots,a_{k-1}\}\big)$. (Actually u_n is never set by the algorithm, but we can assume that it is zero.) (b) 425368917.

46. True (assuming that $u_n = 0$). If either $u_k > u_{k+1}$ or $a_k > a_{k+1}$ we must have $a_k > u_k \geq a_{k+1} > u_{k+1}$.

47. Steps (X1, X2, ..., X6) are performed respectively $(1, A, B, A{-}1, B{-}N_n, A)$ times, where $A = N_0 + \cdots + N_{n-1}$ and $B = nN_0 + (n-1)N_1 + \cdots + 1N_{n-1}$.

48. Steps (X2, X3, X4, X5, X6) are performed respectively $A_n + (1, n!, 0, 0, 1)$ times, where $A_n = \sum_{k=1}^{n-1} n^{\underline{k}} = n! \sum_{k=1}^{n-1} 1/k! \approx n!\,(e-1)$. Assuming that they cost respectively $(1,1,3,1,3)$ mems, for operations involving a_j, l_j, or u_j, the total cost is about $9e - 8 \approx 16.46$ mems per permutation.

Algorithm L uses approximately $(e, 2 + e/2, 2e + 2e^{-1} - 4)$ mems per permutation in steps (L2, L3, L4), for a total of $3.5e + 2e^{-1} - 2 \approx 8.25$ (see exercise 5).

Algorithm X could be tuned up for this case by streamlining the code when k is near n. But so can Algorithm L, as shown in exercise 1.

49. Order the signatures so that $|s_0| \geq \cdots \geq |s_9|$; also prepare tables $w_0 \ldots w_9$, $x_0 \ldots x_9$, $y_0 \ldots y_9$, so that the signatures $\{s_k, \ldots, s_9\}$ are $w_{x_k} \leq \cdots \leq w_{y_k}$. For example, when SEND + MORE = MONEY we have $(s_0, \ldots, s_9) = (-9000, 1000, -900, 91, -90, 10, 1, -1, 0, 0)$ for the respective letters (M, S, O, E, N, R, D, Y, A, B); also $(w_0, \ldots, w_9) = (-9000, -900, -90, -1, 0, 0, 1, 10, 91, 1000)$, and $x_0 \ldots x_9 = 0112233344$, $y_0 \ldots y_9 = 9988776554$. Yet another table $f_0 \ldots f_9$ has $f_j = 1$ if the digit corresponding to w_j cannot be zero; in this case $f_0 \ldots f_9 = 1000000001$. These tables make it easy to compute the largest and smallest values of

$$s_k a_k + \cdots + s_9 a_9$$

over all choices $a_k \ldots a_9$ of the remaining digits, using the method of exercise 25, since the links l_j tell us those digits in increasing order.

This method requires a rather expensive computation at each node of the search tree, but it often succeeds in keeping that tree small. For example, it solves the first eight alphametics of exercise 24 with costs of only 7, 13, 7, 9, 5, 343, 44, and 89 kilomems; this is a substantial improvement in cases (a), (b), (e), and (h), although case (f) comes out significantly worse. Another bad case is the 'CHILD' example of answer 27, where left-to-right needs 2947 kilomems compared to 588 for the right-to-left approach. Left-to-right does, however, fare better on BLOOD + SWEAT + TEARS (73 versus 360) and HELLO + WORLD (340 versus 410).

50. If α is in a permutation group, so are all its powers $\alpha^2, \alpha^3, \ldots$, including $\alpha^{m-1} = \alpha^-$, where m is the order of α (the least common multiple of its cycle lengths). And (32) is equivalent to $\alpha^- = \sigma_1 \sigma_2 \ldots \sigma_{n-1}$.

51. False. For example, $\sigma(k,i)^-$ and $\sigma(k,j)^-$ might both take $k \mapsto 0$.

52. $\tau(k,j) = (k{-}j\ k{-}j{+}1)$ is an adjacent interchange, and

$$\omega(k) = (n{-}1\ \ldots\ 0)(n{-}2\ \ldots\ 0)\ldots(k\ \ldots\ 0) = \phi(n-1)\phi(k-1)$$

is a k-flip followed by an n-flip. The permutation corresponding to control table $c_0 \ldots c_{n-1}$ in Algorithm H has c_j elements to the right of j that are less than j, for $0 \leq j < n$; so it is the same as the permutation corresponding to $c_1 \ldots c_n$ in Algorithm P, except that subscripts are shifted by 1.

The only essential difference between Algorithm P and this version of Algorithm H is that Algorithm P uses a reflected Gray code to run through all possibilities of its control table, while Algorithm H runs through those mixed-radix numbers in ascending (lexicographic) order.

Indeed, Gray code can be used with any Sims table, by modifying either Algorithm G or Algorithm H. Then all transitions are by $\tau(k,j)$ or by $\tau(k,j)^-$, and the permutations $\omega(k)$ are irrelevant.

53. The text's proof that $n! - 1$ transpositions cannot be achieved for $n = 4$ also shows that we can reduce the problem from n to $n-2$ at the cost of a single transposition $(n{-}1\ n{-}2)$, which was called '$(3\,c)$' in the notation of that proof.

Thus we can generate all permutations by making the following transformation in step H4: If $k = n-1$ or $k = n-2$, transpose $a_{j \bmod n} \leftrightarrow a_{(j-1) \bmod n}$, where $j = c_{n-1} - 1$. If $k = n-3$ or $k = n-4$, transpose $a_{n-1} \leftrightarrow a_{n-2}$ and also $a_{j \bmod (n-2)} \leftrightarrow a_{(j-1) \bmod (n-2)}$, where $j = c_{n-3} - 1$. And in general if $k = n-2t-1$ or $k = n-2t-2$,

transpose $a_{n-2i+1} \leftrightarrow a_{n-2i}$ for $1 \le i \le t$ and also $a_{j \bmod (n-2t)} \leftrightarrow a_{(j-1) \bmod (n-2t)}$, where $j = c_{n-2t-1} - 1$. [See *CACM* **19** (1976), 68–72.]

The corresponding Sims table permutations can be written down as follows, although they don't appear explicitly in the algorithm itself:

$$\sigma(k,j)^- = \begin{cases} (0\ 1\ \ldots\ j{-}1\ k), & \text{if } n-k \text{ is odd;} \\ (0\ 1\ \ldots\ k)^j, & \text{if } n-k \text{ is even.} \end{cases}$$

The value of $a_{j \bmod (n-2t)}$ will be $n-2t-1$ after the interchange. For efficiency we can also use the fact that k usually equals $n-1$. The total number of transpositions is $\sum_{t=0}^{\lfloor n/2 \rfloor} (n-2t)! - \lfloor n/2 \rfloor - 1$.

54. Yes; the transformation can be any k-cycle on positions $\{1, \ldots, k\}$.

55. (a) Since $\rho_!(m) = \rho_!(m \bmod n!)$ when $n > \rho_!(m)$, we have $\rho_!(n! + m) = \rho_!(m)$ for $0 < m < n \cdot n! = (n+1)! - n!$. Therefore $\beta_{n!+m} = \sigma_{\rho_!(n!+m)} \cdots \sigma_{\rho_!(n!+1)} \beta_{n!} = \sigma_{\rho_!(m)} \cdots \sigma_{\rho_!(1)} \beta_{n!} = \beta_m \beta_{n!}$ for $0 \le m < n \cdot n!$, and we have in particular

$$\beta_{(n+1)!} = \sigma_{n+1} \beta_{(n+1)!-1} = \sigma_{n+1} \beta_{n!-1} \beta_{n!}^n = \sigma_{n+1} \sigma_n^- \beta_{n!}^{n+1}.$$

Similarly $\alpha_{n!+m} = \beta_{n!}^- \alpha_m \beta_{n!} \alpha_{n!}$ for $0 \le m < n \cdot n!$.

Since $\beta_{n!}$ commutes with τ_n and τ_{n+1} we find $\alpha_{n!} = \tau_n \alpha_{n!-1}$, and

$$\begin{aligned} \alpha_{(n+1)!} = \tau_{n+1} \alpha_{(n+1)!-1} &= \tau_{n+1} \beta_{n!}^- \alpha_{(n+1)!-1-n} \beta_{n!} \alpha_{n!} = \cdots \\ &= \tau_{n+1} \beta_{n!}^{-n} \alpha_{n!-1} (\beta_{n!} \alpha_{n!})^n \\ &= \beta_{n!}^{-n-1} \tau_{n+1} \tau_n^- (\beta_{n!} \alpha_{n!})^{n+1} \\ &= \beta_{(n+1)!}^- \sigma_{n+1} \sigma_n^- \tau_{n+1} \tau_n^- (\beta_{n!} \alpha_{n!})^{n+1}. \end{aligned}$$

(b) In this case $\sigma_{n+1} \sigma_n^- = (n\ n{-}1\ \ldots\ 1)$ and $\tau_{n+1} \tau_n^- = (n{+}1\ n\ 0)$, and we have $\beta_{(n+1)!} \alpha_{(n+1)!} = (n{+}1\ n\ \ldots\ 0)$ by induction. Therefore $\alpha_{jn!+m} = \beta_{n!}^{-j} \alpha_m (n\ \ldots\ 0)^j$ for $0 \le j \le n$ and $0 \le m < n!$. All permutations of $\{0, \ldots, n\}$ are achieved because $\beta_{n!}^{-j} \alpha_m$ fixes n and $(n\ \ldots\ 0)^j$ takes $n \mapsto n - j$.

56. If we set $\sigma_k = (k{-}1\ k{-}2)(k{-}3\ k{-}4)\ldots$ in the previous exercise, we find by induction that $\beta_{n!} \alpha_{n!}$ is the $(n+1)$-cycle $(0\ n\ n{-}1\ n{-}3\ \ldots\ (2 \text{ or } 1)\ (1 \text{ or } 2)\ \ldots\ n{-}4\ n{-}2)$.

57. Arguing as in answer 5, we obtain $\sum_{k=2}^{n-1} [k \text{ odd}]/k! - (\lfloor n/2 \rfloor - 1)/n! = \sinh 1 - 1 - O(1/(n-1)!)$.

58. True. By the formulas of exercise 55 we have $\alpha_{n!-1} = (0\ n) \beta_{n!}^- (n\ \ldots\ 0)$, and this takes $0 \mapsto n-1$ because $\beta_{n!}$ fixes n. (Consequently Algorithm E will define a Hamiltonian *cycle* on the graph of exercise 66 if and only if $\beta_{n!} = (n{-}1\ \ldots\ 2\ 1)$, and this holds if and only if the length of every cycle of $\beta_{(n-1)!}$ is a divisor of n. The latter is true for $n = 2$, 3, 4, 6, 12, 20, and 40, but for no other $n \le 250{,}000$.)

59. The Cayley graph with generators $(\alpha_1, \ldots, \alpha_k)$ in the text's definition is isomorphic to the Cayley graph with generators $(\alpha_1^-, \ldots, \alpha_k^-)$ in the alternative definition, since $\pi \to \alpha_j \pi$ in the former if and only if $\pi^- \to \pi^- \alpha_j^-$ in the latter.

60. There are 88 delta sequences, which reduce to four classes: $P = (32131231)^3$ (plain changes, represented by 8 different delta sequences); $Q = (32121232)^3$ (a doubly Gray variant of plain changes, with 8 representatives); $R = (121232321232)^2$ (a doubly Gray code with 24 representatives); $S = 2\alpha 3 \alpha^R$, $\alpha = 12321312121$ (48 representatives). Classes P and Q are cyclic shifts of their complements; classes P, Q, and S are shifts of their reversals; class R is a shifted reversal of its complement. [See A. L. Leigh Silver, *Math. Gazette* **48** (1964), 1–16.]

61. There are respectively (26, 36, 20, 26, 28, 40, 40, 20, 26, 28, 28, 26) such paths ending at (1243, 1324, 1432, 2134, 2341, 2413, 3142, 3214, 3421, 4123, 4231, 4312).

62. There are only two paths when $n = 3$, ending respectively at 132 and 213. But when $n \geq 4$ there are Gray codes leading from $12\ldots n$ to any odd permutation $a_1 a_2 \ldots a_n$. Exercise 61 establishes this when $n = 4$, and we can prove it by induction for $n > 4$ as follows.

Let $A(j)$ be the set of all permutations that begin with j, and let $A(j,k)$ be those that begin with jk. If $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ are any odd permutations such that $\alpha_j \in A(x_j, x_{j+1})$, then $(1\,2)\alpha_j$ is an even permutation in $A(x_{j+1}, x_j)$. Consequently, if $x_1 x_2 \ldots x_n$ is a permutation of $\{1, 2, \ldots, n\}$, there is at least one Hamiltonian path of the form

$$(1\,2)\alpha_0 \text{ — } \cdots \text{ — } \alpha_1 \text{ — } (1\,2)\alpha_1 \text{ — } \cdots \text{ — } \alpha_2 \text{ — } \cdots \text{ — } (1\,2)\alpha_{n-1} \text{ — } \cdots \text{ — } \alpha_n;$$

the subpath from $(1\,2)\alpha_{j-1}$ to α_j includes all elements of $A(x_j)$.

This construction solves the problem in at least $(n-2)!^n/2^{n-1}$ distinct ways when $a_1 \neq 1$, because we can take $\alpha_0 = 2\,1\ldots n$ and $\alpha_n = a_1 a_2 \ldots a_n$; there are $(n-2)!$ ways to choose $x_2 \ldots x_{n-1}$, and $(n-2)!/2$ ways to choose each of $\alpha_1, \ldots, \alpha_{n-1}$.

Finally, if $a_1 = 1$, take any path $12\ldots n$ — $\cdots$ — $a_1 a_2 \ldots a_n$ that runs through all of $A(1)$, and choose any step α — α' with $\alpha \in A(1,j)$ and $\alpha' \in A(1,j')$ for some $j \neq j'$. Replace that step by

$$\alpha \text{ — } (1\,2)\alpha_1 \text{ — } \cdots \text{ — } \alpha_2 \text{ — } \cdots \text{ — } (1\,2)\alpha_{n-1} \text{ — } \cdots \text{ — } \alpha_n \text{ — } \alpha',$$

using a construction like the Hamiltonian path above but now with $\alpha_1 = \alpha$, $\alpha_n = (1\,2)\alpha'$, $x_1 = 1$, $x_2 = j$, $x_n = j'$, and $x_{n+1} = 1$. (In this case the permutations $\alpha_1, \ldots, \alpha_n$ might all be even.)

63. Monte Carlo estimates using the techniques of Section 7.2.3 suggest that the total number of equivalence classes will be roughly 1.2×10^{21}; most of those classes will contain 480 Gray cycles.

64. Exactly 2,005,200 delta sequences have the doubly Gray property; they belong to 4206 equivalence classes under cyclic shift, reversal, and/or complementation. Nine classes, such as the code $2\alpha 2\alpha^R$ where

$$\alpha = 123432343212321212323212321212343432121234321234321212321321,$$

are shifts of their reversal; 48 classes are composed of repeated 60-cycles. One of the most interesting of the latter type is $\alpha\alpha$ where

$$\alpha = \beta 2 \beta 4 \beta 4 \beta 4 \beta 4, \qquad \beta = 32121232123.$$

65. Such a path exists for any given $N \leq n!$: Let the Nth permutation be $\alpha = a_1 \ldots a_n$, and let $j = a_1$. Also let Π_k be the set of all permutations $\beta = b_1 \ldots b_n$ for which $b_1 = k$ and $\beta \leq \alpha$. By induction on N there is a Gray path P_1 for Π_j. We can then construct Gray paths P_k for $\Pi_j \cup \Pi_1 \cup \cdots \cup \Pi_{k-1}$ for $2 \leq k \leq j$, successively combining P_{k-1} with a Gray cycle for Π_{k-1}. (See the "absorption" construction of answer 62. In fact, P_j will be a Gray *cycle* when N is a multiple of 6.)

66. Defining the delta sequence by the rule $\pi_{(k+1) \bmod n!} = (1\ \delta_k)\pi_k$, we find exactly 36 such sequences, all of which are cyclic shifts of a pattern like $(xyzyzyxzyzyz)^2$. (The next case, $n = 5$, probably has about 10^{18} solutions that are inequivalent with respect to cyclic shifting, reversal, and permutation of coordinates, thus about 6×10^{21} different

delta sequences.) Incidentally, Igor Pak has shown that the Cayley graph generated by star transpositions is an $(n-2)$-dimensional torus in general.

67. If we let π be equivalent to $\pi(12345)$, we get a reduced graph on 24 vertices that has 40768 Hamiltonian cycles, 240 of which lead to delta sequences of the form α^5 in which α uses each transposition 6 times (for example, $\alpha = 354232534234532454352452$). The total number of solutions to this problem is probably about 10^{16}.

68. If A isn't connected, neither is G. If A is connected, we can assume that it is a free tree. Moreover, in this case we can prove a generalization of the result in exercise 62: For $n \geq 4$ there is a Hamiltonian path in G from the identity permutation to any odd permutation. For we can assume without loss of generality that A contains the edge $1 — 2$ where 1 is a leaf of the tree, and a proof like that of exercise 62 applies.

[This elegant construction is due to M. Tchuente, *Ars Combinatoria* **14** (1982), 115–122. Extensive generalizations have been discussed by Ruskey and Savage in *SIAM J. Discrete Math.* **6** (1993), 152–166. See also the original Russian publication in *Kibernetika* **11**, 3 (1975), 17–25; English translation, *Cybernetics* **11** (1975), 362–366.]

69. Following the hint, the modified algorithm behaves like this when $n = 5$:

1234	1243	1423	4123	4132	1432	1342	1324	3124	3142	3412	4312
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑
54321	24351	24153	54123	14523	14325	24315	24513	54213	14253	14352	54312
12345	15342	35142	32145	32541	52341	51342	31542	31245	35241	25341	21345
15342	12435	32415	35412←31452		51432	52431	32451←35421		31425	21435	25431
23451	53421	51423	21453→25413		23415	13425	15423→12453		52413	53412	13452
21543	51243	53241	23541	23145	25143	15243	13245	13542	53142	52143	12543
34512	34215	14235	14532	54132	34152	34251	54231	24531	24135	34125	34521
32154→35124		15324→12354		52314	32514←31524		51324	21354→25314		35214→31254	
45123←42153		42351←45321		41325	41523→42513		42315	45312←41352		41253←45213	
43215	43512←41532		41235	45231→43251		43152→45132		42135	42531←43521		43125
51234	21534→23514		53214	13254←15234		25134←23154		53124	13524→12534		52134
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑

Here the columns represent sets of permutations that are cyclically rotated and/or reflected in all $2n$ ways; therefore each column contains exactly one "rosary permutation" (exercise 18). We can use Algorithm P to run through the rosary permutations systematically, knowing that the pair xy will occur before yx in its column, at which time τ' instead of ρ' will move us to the right or to the left. Step Z2 omits the interchange $a_1 \leftrightarrow a_2$, thereby causing the permutations $a_1 \ldots a_{n-1}$ to repeat themselves going backwards. (We implicitly use the fact that $t[k] = t[n! - k]$ in the output of Algorithm T.)

Now if we replace $1 \ldots n$ by $24 \ldots 31$ and change $A_1 \ldots A_n$ to $A_1 A_n A_2 A_{n-1} \ldots$, we get the unmodified algorithm whose results are shown in Fig. 22(b).

This method was inspired by a (nonconstructive) theorem of E. S. Rapoport, *Scripta Math.* **24** (1959), 51–58. It illustrates a more general fact observed by Carla Savage in 1989, namely that the Cayley graph for *any* group generated by three involutions ρ, σ, τ has a Hamiltonian cycle when $\rho\tau = \tau\rho$ [see I. Pak and R. Radoičić, "Hamiltonian paths in Cayley graphs," to appear].

70. No; the longest cycle in that digraph has length 358. But there do exist pairs of disjoint 180-cycles from which a Hamiltonian path of length 720 can be derived. For

example, consider the cycles $\alpha\sigma\beta\sigma$ and $\gamma\sigma\sigma$ where

$$\alpha = \tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^1;$$

$$\beta = \sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^4;$$

$$\gamma = \sigma\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2$$
$$\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^2.$$

If we start with 134526 and follow $\alpha\sigma\beta\tau$ we reach 163452; then follow $\gamma\sigma\tau$ and reach 126345; then follow $\sigma\gamma\tau$ and reach 152634; then follow $\beta\sigma\alpha$, ending at 415263.

71. Brendan McKay and Frank Ruskey have found such cycles by computer when $n = 7$, 9, and 11, but no nice structure was apparent.

72. Any Hamiltonian path includes $(n-1)!$ vertices that take $y \mapsto x$, each of which (if not the last) is followed by a vertex that takes $x \mapsto x$. So one must be last; otherwise $(n-1)!+1$ vertices would take $x \mapsto x$.

73. (a) Assume first that β is the identity permutation (). Then every cycle of α that contains an element of A lies entirely within A. Hence the cycles of σ are obtained by omitting all cycles of α that contain no element of A. All remaining cycles have odd length, so σ is an even permutation.

If β is not the identity, we apply this argument to $\alpha' = \alpha\beta^-$, $\beta' = ()$, and $\sigma' = \sigma\beta^-$, concluding that σ' is an even permutation; thus σ and β have the same sign.

Similarly, σ and α have the same sign, because $\beta\alpha^- = (\alpha\beta^-)^-$ has the same order as $\alpha\beta^-$.

(b) Let X be the vertices of the Cayley graph in Theorem R, and let α be the permutation of X that takes a vertex π into $\alpha\pi$; this permutation has g/a cycles of length a. Define the permutation β similarly. Then $\alpha\beta^-$ has g/c cycles of length c. If c is odd, any Hamiltonian cycle in the graph defines a cycle σ that contains all the vertices and satisfies the hypotheses of (a). Therefore α and β have an odd number of cycles, because the sign of a permutation on n elements with r cycles is $(-1)^{n-r}$ (see exercise 5.2.2–2).

[This proof, which shows that X cannot be the union of any odd number of cycles, was presented by Rankin in *Proc. Cambridge Phil. Soc.* **62** (1966), 15–16.]

74. The representation $\beta^j\gamma^k$ is unique if we require $0 \le j < g/c$ and $0 \le k < c$. For if we had $\beta^j = \gamma^k$ for some j with $0 < j < g/c$, the group would have at most jc elements. It follows that $\beta^{g/c} = \gamma^t$ for some t.

Let σ be a Hamiltonian cycle, as in the previous answer. If $\pi\sigma = \pi\alpha$ then $\pi\gamma\sigma$ must be $\pi\gamma\alpha$, because $\pi\gamma\beta = \pi\alpha$. And if $\pi\sigma = \pi\beta$ then $\pi\gamma\sigma$ cannot be $\pi\gamma\alpha$, because that would imply $\pi\gamma^c\sigma = \pi\gamma^c\alpha$. Thus the elements $\pi\gamma^k$ all have equivalent behavior with respect to their successors in σ.

Notice that if $j \ge 0$ there is a $k \le j$ such that $\pi\sigma^j = \pi\alpha^k\beta^{j-k} = \pi\beta^j\gamma^k$. Therefore $\pi\sigma^{g/c} = \pi\gamma^{t+k}$ is equivalent to π, and the same behavior will repeat. We return to π for the first time in g steps if and only if $t+k$ is relatively prime to c.

75. Apply the previous exercise with $g = mn$, $a = m$, $b = n$, $c = mn/d$. The number t satisfies $t \equiv 0$ (modulo m), $t + d \equiv 0$ (modulo n); and it follows that $k + t \perp c$ if and only if $(d-k)m/d \perp kn/d$.

Notes: The modular Gray code of exercise 7.2.1.1–78 is a Hamiltonian path from $(0,0)$ to $(m-1, (-m) \bmod n)$, so it is a Hamiltonian cycle if and only if m is a multiple of n. It is natural to conjecture (falsely) that at least one Hamiltonian cycle exists whenever $d > 1$. But P. Erdős and W. T. Trotter have observed [*J. Graph Theory* **2**

(1978), 137–142] that if p and $2p+1$ are odd prime numbers, no suitable k exists when $m = p(2p+1)(3p+1)$ and $n = (3p+1)\prod_{q=1}^{3p} q^{[q \text{ is prime}][q\neq p][q\neq 2p+1]}$.

See J. A. Gallian, *Mathematical Intelligencer* **13**, 3 (Summer 1991), 40–43, for interesting facts about other kinds of cycles in $O_m \times O_n$.

76. We may assume that the tour begins in the lower left corner. There are no solutions when m and n are both divisible by 3, because 2/3 of the cells are unreachable in that case. Otherwise, letting $d = \gcd(m,n)$ and arguing as in the previous exercise but with $(x,y)\alpha = ((x+2) \bmod m, (y+1) \bmod n)$ and $(x,y)\beta = ((x+1) \bmod m, (y+2) \bmod n)$, we find the answer

$$\sum_{k=0}^{d}\binom{d}{k}[\gcd((2d-k)m,(k+d)n) = d \text{ or } (mn \perp 3 \text{ and } \gcd((2d-k)m,(k+d)n) = 3d)].$$

77.
```
01  * Permutation generator \'a la Heap
02  N     IS    10              The value of n (3 or more, not large)
03  t     IS    $255
04  j     IS    $0              8j
05  k     IS    $1              8k
06  ak    IS    $2
07  aj    IS    $3
08        LOC   Data_Segment
09  a     GREG  @               Base address for a_0 ... a_{n-1}
10  A0    IS    @
11  A1    IS    @+8
12  A2    IS    @+16
13        LOC   @+8*N           Space for a_0 ... a_{n-1}
14  c     GREG  @-8*3           Location of 8c_0
15        LOC   @-8*3+8*N       8c_3 ... 8c_{n-1}, initially zero
16        OCTA  -1              8c_n = -1, a convenient sentinel
17  u     GREG  0               Contents of a_0, except in inner loop
18  v     GREG  0               Contents of a_1, except in inner loop
19  w     GREG  0               Contents of a_2, except in inner loop
20        LOC   #100
21  1H    STCO  0,c,k           B-A   c_k ← 0.
22        INCL  k,8             B-A   k ← k+1.
23  0H    LDO   j,c,k             B   j ← c_k.
24        CMP   t,j,k             B
25        BZ    t,1B              B   Loop if c_k = k.
26        BN    j,Done            A   Terminate if c_k < 0 (k = n).
27        LDO   ak,a,k          A-1   Fetch a_k.
28        ADD   t,j,8           A-1
29        STO   t,c,k           A-1   c_k ← j+1.
30        AND   t,k,#8          A-1
31        CSZ   j,t,0           A-1   Set j ← 0 if k is even.
32        LDO   aj,a,j          A-1   Fetch a_j.
33        STO   ak,a,j          A-1   Replace it by a_k.
34        CSZ   u,j,ak          A-1   Set u ← a_k if j = 0.
35        SUB   j,j,8           A-1   j ← j-1.
36        CSZ   v,j,ak          A-1   Set v ← a_k if j = 0.
```

```
37          SUB     j,j,8          A-1   j ← j − 1.
38          CSZ     w,j,ak         A-1   Set w ← a_k if j = 0.
39          STO     aj,a,k         A-1   Replace a_k by what was a_j.
40    Inner PUSHJ   0,Visit          A
            ...                          (See (42))
55          PUSHJ   0,Visit        A
56          SET     t,u            A     Swap u ↔ w.
57          SET     u,w            A
58          SET     w,t            A
59          SET     k,8*3          A     k ← 3.
60          JMP     0B             A
61    Main  LDO     u,A0           1
62          LDO     v,A1           1
63          LDO     w,A2           1
64          JMP     Inner          1     ∎
```

78. Lines 31–38 become $2r-1$ instructions, lines 61–63 become r, and lines 56–58 become $3+(r-2)[r \text{ even}]$ instructions (see $\omega(r-1)$ in answer 40). The total running time is therefore $\big((2r!+2)A+2B+r-5\big)\mu+\big((2r!+2r+7+(r-2)[r\,\text{even}])A+7B-r-4\big)\upsilon$, where $A=n!/r!$ and $B=n!(1/r!+\cdots+1/n!)$.

79. `SLU u,[#f],t; SLU t,a,4; XOR t,t,a; AND t,t,u; SRU u,t,4; OR t,t,u; XOR a,a,t`; here, as in the answer to exercise 1.3.1′–34, the notation '`[#f]`' denotes a register that contains the constant value `#f`.

80. `SLU u,a,t; MXOR u,[#8844221188442211],u; AND u,u,[#ff000000]; SRU u,u,t; XOR a,a,u`. This cheats, since it transforms `#12345678` to `#13245678` when $t=4$, but (45) still works.

Even faster and trickier would be a routine analogous to (42): Consider

```
PUSHJ 0,Visit; MXOR a,a,c1; PUSHJ 0,Visit; ... MXOR a,a,c5; PUSHJ 0,Visit
```

where `c1`, ..., `c5` are constants that would cause `#12345678` to become successively `#12783456`, `#12567834`, `#12563478`, `#12785634`, `#12347856`. Other instructions, executed only 1/6 or 1/24 as often, can take care of shuffling nybbles within and between bytes. Very clever, but it doesn't beat (46) in view of the `PUSHJ/POP` overhead.

```
81. t  IS   $255 ;k IS $0 ;kk IS $1 ;c IS $2 ;d IS $3
       SET  k,1       k ← 1.
    3H SRU  d,a,60    d ← leftmost nybble.
       SLU  a,a,4     a ← 16a mod 16^16.
       CMP  c,d,k
       SLU  kk,k,2
       SLU  d,d,kk
       OR   t,t,d     t ← t + 16^k d.
       PBNZ c,1B      Return to main loop if d ≠ k.
       INCL k,1       k ← k + 1.
       PBNZ a,3B      Return to second loop if k < n.   ∎
```

82. $\mu+(5n!+11A-(n-1)!+6)\upsilon=\big((5+10/n)\upsilon+O(n^{-2})\big)n!$, plus the visiting time, where $A=\sum_{k=1}^{n-1}k!$ is the number of times the loop at `3H` is used.

83. With suitable initialization and a 13-octabyte table, only about a dozen `MMIX` instructions are needed:

```
magic  GREG #8844221188442211
0H     ⟨Visit register a⟩
       PBN  c,Sigma
Tau    MXOR t,magic,a; ANDNL t,#ffff; JMP 1F
Sigma  SRU  t,a,20; SLU a,a,4; ANDNML a,#f00
1H     XOR  a,a,t; SLU c,c,1
2H     PBNZ c,0B; INCL p,8
3H     LDOU c,p,0; PBNZ c,0B                    ∎
```

84. Assuming that the processors all have essentially the same speed, we can let the kth processor generate all permutations of rank r for $(k-1)n!/p \le r < kn!/p$, using any method based on control tables $c_1 \ldots c_n$. The starting and ending control tables are easily computed by converting their ranks to mixed-radix notation (exercise 12).

85. We can use a technique like that of Algorithm 3.4.2P: To compute $k = r(\alpha)$, first set $a'_{a_j} \leftarrow j$ for $1 \le j \le n$ (the inverse permutation). Then set $k \leftarrow 0$, and for $j = n$, $n-1$, ..., 2 (in this order) set $t \leftarrow a'_j$, $k \leftarrow kj + t - 1$, $a_t \leftarrow a_j$, $a'_{a_j} \leftarrow t$. To compute $r^{[-1]}(k)$, start with $a_1 \leftarrow 1$. Then for $j = 2, \ldots, n-1, n$ (in this order) set $t \leftarrow (k \bmod j) + 1$, $a_j \leftarrow a_t$, $a_t \leftarrow j$, $k \leftarrow \lfloor k/j \rfloor$. [See S. Pleszczyński, *Inf. Proc. Letters* **3** (1975), 180–183; W. Myrvold and F. Ruskey, *Inf. Proc. Letters* **79** (2001), 281–284.]

Another method is preferable if we want to rank and unrank only the $n^{\underline{m}}$ *variations* $a_1 \ldots a_m$ of $\{1, \ldots, n\}$: To compute $k = r(a_1 \ldots a_m)$, start with $b_1 \ldots b_n \leftarrow b'_1 \ldots b'_n \leftarrow 1 \ldots n$; then for $j = 1, \ldots, m$ (in this order) set $t \leftarrow b'_{a_j}$, $b_t \leftarrow b_{n+1-j}$, and $b'_{b_t} \leftarrow t$; finally set $k \leftarrow 0$ and for $j = m, \ldots, 1$ (in this order) set $k \leftarrow k \times (n+1-j) + b'_{a_j} - 1$. To compute $r^{[-1]}(k)$, start with $b_1 \ldots b_n \leftarrow 1 \ldots n$; then for $j = 1, \ldots, m$ (in this order) set $t \leftarrow (k \bmod (n+1-j)) + 1$, $a_j \leftarrow b_t$, $b_t \leftarrow b_{n+1-j}$, $k \leftarrow \lfloor k/(n+1-j) \rfloor$. (See exercise 3.4.2–15 for cases with large n and small m.)

86. If $x \prec y$ and $y \prec z$, the algorithm will never move y to the left of x, nor z to the left of y, so it will never test x versus z.

87. They appear in lexicographic order; Algorithm P used a reflected Gray order.

88. Generate inverse permutations with $a'_0 < a'_1 < a'_2$, $a'_3 < a'_4 < a'_5$, $a'_6 < a'_7$, $a'_8 < a'_9$, $a'_0 < a'_3$, $a'_6 < a'_8$.

89. (a) Let $d_k = \max\{j \mid 0 \le j \le k \text{ and } j \text{ is nontrivial}\}$, where 0 is considered nontrivial. This table is easily precomputed, because j is trivial if and only if it must follow $\{1, \ldots, j-1\}$. Set $k \leftarrow d_n$ in step V2 and $k \leftarrow d_{k-1}$ in step V5. (Assume $d_n > 0$.)

(b) Now $M = \sum_{j=1}^{n} t_j [j \text{ is nontrivial}]$.

(c) There are at least two topological sorts $a_j \ldots a_k$ of the set $\{j, \ldots, k\}$, and either of them can be placed after any topological sort $a_1 \ldots a_{j-1}$ of $\{1, \ldots, j-1\}$.

(d) Algorithm 2.2.3T repeatedly outputs minimal elements (elements with no predecessors), removing them from the relation graph. We use it in reverse, repeatedly removing and giving the highest labels to *maximal* elements (elements with no successors). If only one maximal element exists, it is trivial. If k and l are both maximal, they both are output before any element x with $x \prec k$ or $x \prec l$, because steps T5 and T7 keep maximal elements in a queue (not a stack). Thus if k is nontrivial and output first, element l might become trivial, but the next nontrivial element j will not be output before l; and k is unrelated to l.

(e) Let the nontrivial t's be $s_1 < s_2 < \cdots < s_r = N$. Then we have $s_j \ge 2s_{j-2}$, by (c). Consequently $M = s_2 + \cdots + s_r \le s_r(1 + \frac{1}{2} + \frac{1}{4} + \cdots) + s_{r-1}(1 + \frac{1}{2} + \frac{1}{4} + \cdots) < 4s_r$.

(A sharper estimate is in fact true, as observed by M. Peczarski: Let $s_0 = 1$, let the nontrivial indices be $0 = k_1 < k_2 < \cdots < k_r$, and let $k'_j = \max\{k \mid 1 \le k < k_j, k \not\prec k_j\}$ for $j \ge 1$. Then $k'_j \ge k_{j-1}$. There are s_j topological sorts of $\{1, \ldots, k_{j+1}\}$ that end with k_{j+1}; and there are at least s_{j-1} that end with k'_{j+1}, since each of the s_{j-1} topological sorts of $\{1, \ldots, k_j - 1\}$ can be extended. Hence

$$s_{j+1} \;\ge\; s_j + s_{j-1} \qquad \text{for } 1 \le j < r.$$

Now let $y_0 = 0$, $y_1 = F_2 + \cdots + F_r$, and $y_j = y_{j-2} + y_{j-1} - F_{r+1}$ for $1 < j < r$. Then

$$F_{r+1}(s_1 + \cdots + s_r) + \sum_{j+1}^{r-1} y_j (s_{r+1-j} - s_{r-j} - s_{r-1-j}) \;=\; (F_2 + \cdots + F_{r+1}) s_r,$$

and each $y_j = F_{r+1} - 2F_j + (-1)^j F_{r+1-j}$ is nonnegative. Hence $s_1 + \cdots + s_r \le ((F_2 + \cdots + F_{r+1})/F_{r+1}) s_r \approx 2.6 s_r$. The following exercise shows that this bound is best possible.)

90. The number N of such permutations is F_{n+1} by exercise 5.2.1–25. Therefore $M = F_{n+1} + \cdots + F_2 = F_{n+3} - 2 \approx \phi^2 N$. Notice incidentally that all such permutations satisfy $a_1 \ldots a_n = a'_1 \ldots a'_n$. They can be arranged in a Gray path (exercise 7.2.1.1–89).

91. Since $t_j = (j-1)(j-3)\ldots(2 \text{ or } 1)$, we find $M = (1 + 2/\sqrt{\pi n} + O(1/n))N$.

Note: The inversion tables $c_1 \ldots c_{2n}$ for permutations satisfying (49) are characterized by the conditions $c_1 = 0$, $0 \le c_{2k} \le c_{2k-1}$, $0 \le c_{2k+1} \le c_{2k-1} + 1$.

92. The total number of pairs (R, S), where R is a partial ordering and S is a linear ordering that includes R, is equal to P_n times the expected number of topological sorts; it is also Q_n times $n!$. So the answer is $n!\,Q_n/P_n$.

We will discuss the computation of P_n and Q_n in Section 7.2.3. For $1 \le n \le 12$ the expectation turns out to be approximately

$$(1,\ 1.33,\ 2.21,\ 4.38,\ 10.1,\ 26.7,\ 79.3,\ 262,\ 950,\ 3760,\ 16200,\ 74800).$$

Asymptotic values as $n \to \infty$ have been deduced by Brightwell, Prömel, and Steger [*J. Combinatorial Theory* **A73** (1996), 193–206], but the limiting behavior is quite different from what happens when n is in a practical range. The values of Q_n were first determined for $n \le 5$ by S. P. Avann [*Æquationes Math.* **8** (1972), 95–102].

93. The basic idea is to introduce dummy elements $n+1$ and $n+2$ with $j \prec n+1$ and $j \prec n+2$ for $1 \le j \le n$, and to find all topological sorts of such an extended relation via adjacent interchanges; then take every *second* permutation, suppressing the dummy elements. An algorithm similar to Algorithm V can be used, but with a recursion that reduces n to $n-2$ by inserting $n-1$ and n among $a_1 \ldots a_{n-2}$ in all possible ways, assuming that $n - 1 \not\prec n$, occasionally swapping $n+1$ with $n+2$. [See G. Pruesse and F. Ruskey, *SICOMP* **23** (1994), 373–386. A loopless implementation has been described by Canfield and Williamson, *Order* **12** (1995), 57–75.]

94. The case $n = 3$ illustrates the general idea of a pattern that begins with $1 \ldots (2n)$ and ends with $1(2n)2(2n-1)\ldots n(n+1)$: 123456, 123546, 123645, 132645, 132546, 132456, 142356, 142536, 142635, 152634, 152436, 152346, 162345, 162435, 162534.

Matchings can also be regarded as involutions of $\{1, \ldots, 2n\}$ that have n cycles. With that representation this pattern involves two transpositions per step.

Notice that the C inversion tables of the permutations just listed are respectively 000000, 000100, 000200, 010200, 010100, 010000, 020000, 020100, 020200, 030200, 030100, 030000, 040000, 040100, 040200. In general, $C_1 = C_3 = \cdots = C_{2n-1} = 0$

and the n-tuples $(C_2, C_4, \ldots, C_{2n})$ run through a reflected Gray code on the radices $(2n-1, 2n-3, \ldots, 1)$. Thus the generation process can easily be made loopless if desired. [See Timothy Walsh, *J. Combinatorial Math. and Combinatorial Computing* **36** (2001), 95–118, Section 1.]

Note: Algorithms to generate all matchings go back to J. F. Pfaff [*Abhandlungen Akad. Wissenschaften* (Berlin: 1814–1815), 124–125], who described two such procedures: His first method was lexicographic, which also corresponds to lexicographic order of the C inversion tables; his second method corresponds to *colex* order of those tables. Even and odd permutations alternate in both cases.

95. Generate inverse permutations with $a'_1 < a'_n > a'_2 < a'_{n-1} > \cdots$, using Algorithm V. (See exercise 5.1.4–23 for the number of solutions.)

96. For example, we can start with $a_1 \ldots a_{n-1}a_n = 2 \ldots n1$ and $b_1 b_2 \ldots b_n b_{n+1} = 12 \ldots n1$, and use Algorithm P to generate the $(n-1)!$ permutations $b_2 \ldots b_n$ of $\{2, \ldots, n\}$. Just after that algorithm swaps $b_i \leftrightarrow b_{i+1}$, we set $a_{b_{i-1}} \leftarrow b_i$, $a_{b_i} \leftarrow b_{i+1}$, $a_{b_{i+1}} \leftarrow b_{i+2}$, and visit $a_1 \ldots a_n$.

97. Use Algorithm X, with $t_k(a_1, \ldots, a_k) = \text{`}a_k \neq k\text{'}$.

98. Using the notation of exercise 47, we have $N_k = \sum \binom{k}{j}(-1)^j (n-j)^{\underline{k-j}}$ by the method of inclusion and exclusion (exercise 1.3.3–26). If $k = O(\log n)$ then $N_{n-k} = (n!\,e^{-1}/k!)(1 + O(\log n)^2/n)$; hence $A/n! \approx (e-1)/e$ and $B/n! \approx 1$. The number of memory references, under the assumptions of answer 48, is therefore $\approx A + B + 3A + B - N_n + 3A \approx n!\,(9 - \frac{8}{e}) \approx 6.06n!$, about 16.5 per derangement. [See S. G. Akl, *BIT* **20** (1980), 2–7, for a similar method.]

99. Suppose L_n generates $D_n \cup D_{n-1}$, beginning with (1 2 … n), then (2 1 … n), and ending with (1 … n−1); for example, $L_3 = (1\,2\,3), (2\,1\,3), (1\,2)$. Then we can generate D_{n+1} as $K_{nn}, \ldots, K_{n2}, K_{n1}$, where $K_{nk} = (1\ 2\ \ldots\ n)^{-k}(n\ n{+}1)L_n(1\ 2\ \ldots\ n)^k$; for example, D_4 is

(1 2 3 4), (2 1 3 4), (1 2)(3 4), (3 1 2 4), (1 3 2 4), (3 1)(2 4), (2 3 1 4), (3 2 1 4), (2 3)(1 4).

Notice that K_{nk} begins with the cycle $(k{+}1\ \ldots\ n\ 1\ \ldots\ k\ n{+}1)$ and ends with $(k{+}1\ \ldots\ n\ 1\ \ldots\ k{-}1)(k\ n{+}1)$; so premultiplication by $(k{-}1\ k)$ takes us from K_{nk} to $K_{n(k-1)}$. Also, premultiplication by $(1\ n)$ will return from the last element of D_{n+1} to the first. Premultiplication by $(1\ 2\ n{+}1)$ takes us from the last element of D_{n+1} to $(2\ 1\ 3\ \ldots\ n)$, from which we can return to $(1\ 2\ \ldots\ n)$ by following the cycle for D_n backwards, thereby completing the list L_{n+1} as desired.

100. Use Algorithm X, with $t_k(a_1, \ldots, a_k) = \text{`}p > 0$ or $l[q] \neq k+1\text{'}$.

Notes: The number of indecomposable permutations is $[z^n]\,(1 - 1/\sum_{k=0}^{\infty} k!\,z^k)$; see L. Comtet, *Comptes Rendus Acad. Sci.* **A275** (Paris, 1972), 569–572. It appears likely that the indecomposable permutations can be generated by adjacent transpositions; for example, when $n = 4$ they are 3142, 3412, 3421, 3241, 2341, 2431, 4231, 4321, 4312, 4132, 4123, 4213, 2413.

101. Here is a lexicographic involution generator analogous to Algorithm X.

Y1. [Initialize.] Set $a_k \leftarrow k$ and $l_{k-1} \leftarrow k$ for $1 \le k \le n$. Then set $l_n \leftarrow 0$, $k \leftarrow 1$.

Y2. [Enter level k.] If $k > n$, visit $a_1 \ldots a_n$ and go to Y3. Otherwise set $p \leftarrow l_0$, $u_k \leftarrow p$, $l_0 \leftarrow l_p$, $k \leftarrow k+1$, and repeat this step. (We have decided to let $a_p = p$.)

Y3. [Decrease k.] Set $k \leftarrow k-1$, and terminate if $k=0$. Otherwise set $q \leftarrow u_k$ and $p \leftarrow a_q$. If $p=q$, set $l_0 \leftarrow q$, $q \leftarrow 0$, $r \leftarrow l_p$, and $k \leftarrow k+1$ (preparing to make $a_p > p$). Otherwise set $l_{u_{k-1}} \leftarrow q$, $r \leftarrow l_q$ (preparing to make $a_p > q$).

Y4. [Increase a_p.] If $r=0$ go to Y5. Otherwise set $l_q \leftarrow l_r$, $u_{k-1} \leftarrow q$, $u_k \leftarrow r$, $a_p \leftarrow r$, $a_q \leftarrow q$, $a_r \leftarrow p$, $k \leftarrow k+1$, and go to Y2.

Y5. [Restore a_p.] Set $l_0 \leftarrow p$, $a_p \leftarrow p$, $a_q \leftarrow q$, $k \leftarrow k-1$, and return to Y3. ∎

Let $t_{n+1} = t_n + nt_{n-1}$, $a_{n+1} = 1 + a_n + na_{n-1}$, $t_0 = t_1 = 1$, $a_0 = 0$, $a_1 = 1$. (See Eq. 5.1.4–(40).) Step Y2 is performed t_n times with $k > n$ and a_n times with $k \le n$. Step Y3 is performed a_n times with $p = q$ and $a_n + t_n$ times altogether. Step Y4 is performed $t_n - 1$ times; step Y5, a_n times. The total number of mems for all t_n outputs is therefore approximately $11a_n + 12t_n$, where $a_n < 1.25331414t_n$. (Optimizations are clearly possible if speed is essential.)

102. We construct a list L_n that begins with () and ends with $(n{-}1\ n)$, starting with $L_3 = (), (1\ 2), (1\ 3), (2\ 3)$. If n is odd, L_{n+1} is L_n, K_{n1}^R, K_{n2}, …, K_{nn}^R, where $K_{nk} = (k\ \ldots\ n)^- L_{n-1}(k\ \ldots n)(k\ n{+}1)$. For example,

$$L_4 = (), (1\ 2), (1\ 3), (2\ 3), (2\ 3)(1\ 4), (1\ 4), (2\ 4), (1\ 3)(2\ 4), (1\ 2)(3\ 4), (3\ 4).$$

If n is even, L_{n+1} is L_n, $K_{n(n-1)}$, $K_{n(n-2)}^R$, …, K_{n1}, $(1\ n{-}2)L_{n-1}^R(1\ n{-}2)(n\ n{+}1)$.

For further developments, see the article by Walsh cited in answer 94.

103. The following elegant solution by Carla Savage needs only $n-2$ different operations ρ_j, for $1 < j < n$, where ρ_j replaces $a_{j-1}a_ja_{j+1}$ by $a_{j+1}a_{j-1}a_j$ when j is even, $a_ja_{j+1}a_{j-1}$ when j is odd. We may assume that $n \ge 4$; let $A_4 = (\rho_3\rho_2\rho_2\rho_3)^3$. In general A_n will begin and end with ρ_{n-1}, and it will contain $2n-2$ occurrences of ρ_{n-1} altogether. To get A_{n+1}, replace the kth ρ_{n-1} of A_n by $\rho_nA_n'\rho_n$, where $k = 1, 2, 4, \ldots, 2n-2$ if n is even and $k = 1, 3, \ldots, 2n-3, 2n-2$ if n is odd, and where A_n' is A_n with its first or last element deleted. Then, if we begin with $a_1 \ldots a_n = 1 \ldots n$, the operations ρ_{n-1} of A_n will cause position a_n to run through the successive values $n \to p_1 \to n \to p_2 \to \cdots \to p_{n-1} \to n$, where $p_1 \ldots p_{n-1} = (n{-}1 - [n\,\text{even}]) \ldots 4213 \ldots (n{-}1 - [n\,\text{odd}])$; the final permutation will again be $1 \ldots n$.

104. (a) A well-balanced permutation has $\sum_{k=1}^n ka_k = n(n+1)^2/4$, an integer.

(b) Replace k by a_k when summing over k.

(c) A fairly fast way to count, when n is not too large, can be based on the streamlined plain-change algorithm of exercise 16, because the quantity $\sum ka_k$ changes in a simple way with each adjacent interchange, and because $n-1$ of every n steps are "hunts" that can be done rapidly. We can save half the work by considering only permutations in which 1 precedes 2. The values for $1 \le n \le 15$ are 0, 0, 0, 2, 6, 0, 184, 936, 6688, 0, 420480, 4298664, 44405142, 0, 6732621476.

105. (a) For each permutation $a_1 \ldots a_n$, insert $\prec$ between a_j and a_{j+1} if $a_j > a_{j+1}$; insert either $\equiv$ or $\prec$ between them if $a_j < a_{j+1}$. (A permutation with k "ascents" therefore yields 2^k weak orders. Weak orders are sometimes called "preferential arrangements; exercise 5.3.1–4 shows that there are approximately $n!/(2(\ln 2)^{n+1})$ of them. A Gray code for weak orders, in which each step changes $\prec \leftrightarrow \equiv$ and/or $a_j \leftrightarrow a_{j+1}$, can be obtained by combining Algorithm P with Gray binary code at the ascents.

(b) Start with $a_1 \ldots a_na_{n+1} = 0 \ldots 00$ and $a_0 = -1$. Perform Algorithm L until it stops with $j = 0$. Find k such that $a_1 > \cdots > a_k = a_{k+1}$, and terminate if $k = n$. Otherwise set $a_l \leftarrow a_{k+1} + 1$ for $1 \le l \le k$ and go to step L4. [See M. Mor

and A. S. Fraenkel, *Discrete Math.* **48** (1984), 101–112. Weak ordering sequences are characterized by the property that, if k appears and $k > 0$, then $k-1$ also appears.]

106. All weak ordering sequences can be obtained by a sequence of elementary operations $a_i \leftrightarrow a_j$ or $a_i \leftarrow a_j$. (Perhaps one could actually restrict the transformations further, allowing only $a_j \leftrightarrow a_{j+1}$ or $a_j \leftarrow a_{j+1}$ for $1 \le j < n$.)

107. Every step increases the quantity $\sum_{k=1}^{n} 2^k[a_k = k]$, as noted by H. S. Wilf, so the game must terminate. At least three approaches to the solution are plausible: one bad, one good, and one better.

The bad one is to play the game on all 13! shuffles and to record the longest. This method does produce the correct answer; but 13! is 6,227,020,800, and the average game lasts ≈ 8.728 steps.

The good one [A. Pepperdine, *Math. Gazette* **73** (1989), 131–133] is to play backwards, starting with the final position $1*\ldots*$ where $*$ denotes a card that is face down; we will turn a card up only when its value becomes relevant. To move backward from a given position $a_1 \ldots a_n$, consider all $k > 1$ such that either $a_k = k$ or $a_k = *$ and k has not yet turned up. Thus the next-to-last positions are $21*\ldots*$, $3*1*\ldots*$, $\ldots$, $n*\ldots*1$. Some positions (like $6{*}{*}213$ for $n = 6$) have no predecessors, even though we haven't turned all the cards up. It is easy to explore the tree of potential backwards games systematically, and one can in fact show that the number of nodes with t $*$'s is exactly $(n-1)!/t!$. Hence the total number of nodes considered is exactly $\lfloor (n-1)!\,e \rfloor$. When $n = 13$ this is 1,302,061,345.

The better one is to play forwards, starting with initial position $*\ldots*$ and turning over the top card when it is face down, running through all $(n-1)!$ permutations of $\{2,\ldots,n\}$ as cards are turned. If the bottom $n-m$ cards are known to be equal to $(m+1)(m+2)\ldots n$, in that order, at most $f(m)$ further moves are possible; thus we need not pursue a line of play any further if it cannot last long enough to be interesting. A permutation generator like Algorithm X allows us to share the computation for all permutations with the same prefix and to reject unimportant prefixes. The card in position j need not take the value j when it is turned. When $n = 13$ this method needs to consider only respectively $(1, 11, 940, 6960, 44745, 245083, 1118216, 4112676, 11798207, 26541611, 44380227, 37417359)$ branches at levels $(1, 2, \ldots, 12)$ and to make a total of only 482,663,902 forward moves. Although it repeats some lines of play, the early cutoffs of unprofitable branches make it run more than 11 times faster than the backward method when $n = 13$.

The unique way to attain length 80 is to start with 2 9 4 5 11 12 10 1 8 13 3 6 7.

108. This result holds for any game in which

$$a_1 \ldots a_n \to a_k a_{p(k,2)} \ldots a_{p(k,k-1)} a_1 a_{k+1} \ldots a_n$$

when $a_1 = k$, where $p(k,2)\ldots p(k,k-1)$ is an arbitrary permutation of $\{2,\ldots,k-1\}$. Suppose a_1 takes on exactly m distinct values $d(1) < \cdots < d(m)$ during a play of the game; we will prove that at most F_{m+1} permutations occur, including the initial shuffle. This assertion is obvious when $m = 1$.

Let $d(j)$ be the initial value of $a_{d(m)}$, where $j < m$, and suppose $a_{d(m)}$ changes on step r. If $d(j) = 1$, the number of permutations is $r + 1 \le F_m + 1 \le F_{m+1}$. Otherwise $r \le F_{m-1}$, and at most F_m further permutations follow step r. [*SIAM Review* **19** (1977), 239–241.]

The values of $f(n)$ for $1 \le n \le 16$ are (0, 1, 2, 4, 7, 10, 16, 22, 30, 38, 51, 65, 80, 101, 113, 139), and they are attainable in respectively (1, 1, 2, 2, 1, 5, 2, 1, 1, 1, 1, 1,

1, 4, 6, 1) ways. The unique longest-winded permutation for $n = 16$ is

$$9\ 12\ 6\ 7\ 2\ 14\ 8\ 1\ 11\ 13\ 5\ 4\ 15\ 16\ 10\ 3.$$

109. The forward method of answer 107 suggests that $f(n)$ probably grows at least as fast as $n \log n$ (by comparison with coupon collecting).

110. For $0 \le j \le 9$ construct the bit vectors $A_j = [a_j \in S_1] \ldots [a_j \in S_m]$ and $B_j = [j \in S_1] \ldots [j \in S_m]$. Then the number of j such that $A_j = v$ must equal the number of k such that $B_k = v$, for all bit vectors v. And if so, the values $\{a_j \mid A_j = v\}$ should be assigned to permutations of $\{k \mid B_k = v\}$ in all possible ways.

For example, the bit vectors in the given problem are

$$(A_0, \ldots, A_9) = (9, 6, 8, \mathtt{b}, 5, 4, 0, \mathtt{a}, 2, 0), \qquad (B_0, \ldots, B_9) = (5, 0, 8, 6, 2, \mathtt{a}, 4, \mathtt{b}, 9, 0),$$

in hexadecimal notation; hence $a_0 \ldots a_9 = 8327061549$ or 8327069541.

In a larger problem we would keep the bit vectors in a hash table. It would be better to give the answer in terms of equivalence classes, not permutations; indeed, this problem has comparatively little to do with permutations.

111. In the directed graph with $n!/2$ vertices $a_1 \ldots a_{n-2}$ and $n!$ arcs $a_1 \ldots a_{n-2} \to a_2 \ldots a_{n-1}$ (one for each permutation $a_1 \ldots a_n$), each vertex has in-degree 2 and out-degree 2. Furthermore, from paths like $a_1 \ldots a_{n-2} \to a_2 \ldots a_{n-1} \to a_3 \ldots a_n \to a_4 \ldots a_n a_2 \to a_5 \ldots a_n a_2 a_1 \to \cdots \to a_2 a_1 a_3 \ldots a_{n-2}$, we can see that any vertex is reachable from any other. Therefore an Eulerian trail exists by Theorem 2.3.4.2D, and such a trail clearly is equivalent to a universal cycle of permutations. The lexicographically smallest example when $n = 4$ is (123124132134214324314234).

[G. Hurlbert and G. Isaak, in *Discrete Math.* **149** (1996), 123–129, have suggested another appealing approach: Let's say that a *modular universal cycle of permutations* is a cycle of $n!$ digits $\{0, \ldots, n\}$ with the property that each permutation $a_1 \ldots a_n$ of $\{1, \ldots, n\}$ arises from consecutive digits $u_1 \ldots u_n$ by letting $a_j = (u_j - c) \bmod (n+1)$, where c is the "missing" digit in $\{u_1, \ldots, u_n\}$. For example, the modular universal cycle (012032) is essentially unique for $n = 3$; and the lexicographically smallest for $n = 4$ is (012301420132014321430243). If vertices $a_1 \ldots a_{n-2}$ and $a'_1 \ldots a'_{n-2}$ in the digraph of the previous paragraph are considered equivalent when $a_1 - a'_1 \equiv \cdots \equiv a_{n-2} - a'_{n-2}$ (modulo n), we get a digraph of $(n-1)!/2$ vertices whose Eulerian trials correspond to the modular universal cycles of permutations for $\{1, \ldots, n-1\}$.]

112. By exercise 2.3.4.2–22 it suffices to count the oriented trees rooted at $12 \ldots (n-2)$, in the digraph of the preceding answer; and those trees can be counted by exercise 2.3.4.2–19. For $n \le 6$ the numbers U_n turn out to be tantalizingly simple: $U_2 = 1$, $U_3 = 3$, $U_4 = 2^7 \cdot 3$, $U_5 = 2^{33} \cdot 3^8 \cdot 5^3$, $U_6 = 2^{190} \cdot 3^{49} \cdot 5^{33}$. (Here we consider (121323) to be the same cycle as (213231), but different from (131232).)

Mark Cooke has discovered the following instructive way to compute these values efficiently: Notice first that a universal cycle of permutations is also equivalent to a *Hamiltonian* cycle on the Cayley graph with generators $\sigma = (1\ 2\ \ldots\ n)$ and $\rho = (1\ 2\ \ldots\ n{-}1)$. For example, the cycle in the previous answer for $n = 4$ corresponds to the cycle $\sigma^3\rho^2\sigma\rho\sigma^2\rho^2\sigma^3\rho\sigma^2\rho^2\sigma\rho\sigma^2\rho$.

Now consider the $n! \times n!$ matrix $M = 2I - R - S$, where $R_{\pi\pi'} = [\pi' = \pi\rho]$ and $S_{\pi\pi'} = [\pi' = \pi\sigma]$. There is a matrix H such that $H^- RH$ and $H^- SH$ each have block diagonal form consisting of k_λ copies of $k_\lambda \times k_\lambda$ matrices R_λ and S_λ, for each partition λ of n, where k_λ is $n!$ divided by the product of the hook lengths of shape λ (Theorem 5.1.4H), and where R_λ and S_λ are matrix representations of ρ and σ based on Young

tableaux. [A proof can be found in Bruce Sagan, *The Symmetric Group* (Pacific Grove, Calif.: Wadsworth & Brooks/Cole, 1991).] For example, when $n = 3$ we have

$$R = \begin{pmatrix} 0&0&0&1&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0\\ 1&0&0&0&0&0\\ 0&0&1&0&0&0\\ 0&1&0&0&0&0 \end{pmatrix},\quad S = \begin{pmatrix} 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 0&0&0&1&0&0 \end{pmatrix},\quad H = \begin{pmatrix} 1&1&1&-1&1&0\\ 1&1&-1&0&0&-1\\ 1&1&0&1&-1&1\\ 1&-1&-1&1&0&1\\ 1&-1&1&0&1&-1\\ 1&-1&0&-1&-1&0 \end{pmatrix},$$

$$H^- RH = \begin{pmatrix} 1&0&0&0&0&0\\ 0&-1&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&1&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0 \end{pmatrix},\qquad H^- SH = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&-1&0&0\\ 0&0&1&-1&0&0\\ 0&0&0&0&0&-1\\ 0&0&0&0&1&-1 \end{pmatrix}$$

when rows and columns are indexed by the respective permutations 1, σ, σ^2, ρ, $\rho\sigma$, $\rho\sigma^2$; here $k_3 = k_{111} = 1$ and $k_{21} = 2$. Therefore the eigenvalues of M are the union, over λ, of k_λ-fold repeated eigenvalues of the $k_\lambda \times k_\lambda$ matrices $2I - R_\lambda - S_\lambda$. In the example, the eigenvalues of (0), (2), and $\left(\begin{smallmatrix} 2 & 0 \\ -2 & 3 \end{smallmatrix}\right)$ twice are $\{0\}$, $\{2\}$, and $\{2,3\}$ twice.

The eigenvalues of M are directly related to those of the matrix A in exercise 2.3.4.2–19. Indeed, each eigenvector of A yields an eigenvector of M, if we equate the components for permutations π and $\pi\rho\sigma^-$, because rows π and $\pi\rho\sigma^-$ of $R + S$ are equal. For example,

$A = \begin{pmatrix} 2&-1&-1\\ -1&2&-1\\ -1&-1&2 \end{pmatrix}$ has eigenvectors $\begin{pmatrix} 1\\1\\1 \end{pmatrix}, \begin{pmatrix} 1\\-1\\0 \end{pmatrix}, \begin{pmatrix} 1\\0\\-1 \end{pmatrix}$ for eigenvalues 0, 3, 3,

yielding the eigenvectors $(1,1,1,1,1,1)^T$, $(1,-1,0,0,-1,1)^T$, $(1,0,-1,-1,0,1)^T$ of M for the same eigenvalues. And M has $n!/2$ additional eigenvectors, with all components zero except those indexed by π and $\pi\sigma^-\rho$ for some π, because only rows $\pi\rho^-$ and $\pi\sigma^-$ of $R+S$ have nonzero entries in columns π and $\pi\sigma^-\rho$; such vectors yield $n!/2$ additional eigenvalues, all equal to 2.

Therefore U_n, which is $2/n!$ times the product of the nonzero eigenvalues of A, is $2^{1-n!/2}/n!$ times the product of the nonzero eigenvalues of M.

Unfortunately the small-prime-factor phenomenon does not continue; U_7 equals $2^{1217}3^{123}5^{119}7^5 11^{28}43^{35}73^{20}79^{21}109^{35}$, and U_9 is divisible by 592290131963333^{168}.

At least one of these cycles must almost surely be easy to describe and to compute, as we did for de Bruijn cycles in Section 7.2.1.1. But no simple construction has yet been found.

PDG

计算机程序设计艺术

第4卷　第2册

生成所有元组和排列

译 者 序

金秋华夏喜频传，神舟太空威尽现。在结束了第1卷第1册的翻译工作之后，我又立即投入了高德纳的第4卷第2、3册（现在已出版的即是这两册）的翻译工作。时间也从炎夏进入盛秋，而金秋时节的和风丽日，加上华夏大地频频传来的喜讯，给我的工作带来了强大的动力。

高德纳的这部巨著，始于1968年，而且很快，他就把前三卷完成了。人们原来以为，接下来出版的自然应该是第4卷了，然后顺理成章的是第5卷、第6卷，直到第7卷。然而，事实并非如此。近40年过去了，他对第1卷做过多次修改并再版。第2、3卷，也都分别至少做了一次修改并再版。而且如他所宣称的那样，每次修改，不论是哪一卷，都涉及几乎所有页面。工作量之大，作者为此付出之匠心，实在令人叹为观止。在此期间，他又发表了大量的论著，特别是完成了METAFONT和T_EX，为计算机排版技术做出了巨大贡献。完成了著名的《公理与外壳》和《具体数学》等著述。但是对于第4卷，却迟迟未闻有任何信息。只是后来才听说，第4卷将分为三个分卷出版，但也未见任何一卷问世。直到此前，人们才终于见到我现在译出的这第2册和第3册。至于后边还有多少分册，仍不得而知。因此，它的出场，真可谓是“犹抱琵琶半遮面”了。

但是，也正因为是经过几十年才出来的东西，因此它就不同凡响，是名副其实的专著和经典之作。作者在前言写道：“我怀着非常喜悦的心情来写这部分内容，就像许多年前写第2卷时我感觉到的激动那样。如同在第2卷中那样，在那里我高兴地看到，初等概率论和数论的基本原理很自然地出现在关于随机数生成和算术运算的研究中。而在准备7.2.1节的过程中我了解到，当我们研究组合生成的算法时，初等组合学的基本原理也自然地出现，并且具有高度启发性。”他感到，这里有着许多美丽的故事，等待他来讲述。

他说的确实是真的。我想，在这方面，至少有三个方面值得我们从中学习。我们不仅要学习其中的这些内容，还要学习作者的治学精神。

1. 首先是关于生成基本的组合模式的问题。

乍一看去，无论是生成所有n元组，还是生成排列，这些都不过是很简单的问题，并没有什么高深的理论，一说谁都知道，属于已经解决了的问题，再无什么遗留问题需要研究。然而，经过作者点拨，才发现事实上问题绝非我们所想像得那么简单。作者在提出对于这个问题的研究动机时指出，没有一个聪明人愿意通过以几千张纸来打印出$\{0, 1, \cdots, 9\}$的所有$10! = 3\ 628\ 800$个排列的清单，甚至也没有人愿意在一个计算机文件中把它们全写出来。但是，我们需要的是，当确实用得着它们时，在顷刻之间就以某种数据结构把它们提供出来，使得一个程序可以一次一个地来考察每个排列。正是出于这个目的，人们就还要考虑这样一个问题 。

而且就这一点而言，它确实是远未解决的问题。

作者在研究中所体现的严谨性表现在，不仅考虑二进制，还考虑十进制和混合进制；不仅考虑集合的元素，还考虑集合的子集、多重集合等。因此，它就体现为既有问题的深度，又有问题的广度。

2. 关于组合模式同一些问题的关系。

前边所说的深度，指的是如何以最快速度和最少内存访问来生成所需元组或排列。这样一个问题，竟是同图论有关，同哈密顿通路或循环有关，也同树形的遍历有关。因而作者在这里向我们揭示了这种关系，使我们懂得原来组合模式的问题还有这种背景，或者说可以从那些问题的求解中获得解决问题的理论基础。

记得几十年前当人工智能在我国掀起一股热潮时，一些人曾经从人工智能的角度研究过九链环（又称九龙环）的问题。然而，就译者所知，并没有任何一个中国学者，深入地去研究九链环问题的历史以及它的发展过程，更没有人去探讨它和其他问题的联系。但在这本书里，作者告诉我们，早在1872年法国人刘易斯·格罗斯在一本题为《步行理论》的小册子中，就揭示了九链环同二进制数之间的关系，比如我们以0表示环与杆分离的状态，而以1表示环在杆上的状态，则原来环锁在杆上的状态就是9个1的状态。问题是要通过从右边开始（或从左边开始），每次改换一位，最终使9个1变成为9个0。因此作者说，格罗斯才是格雷二进制码真正的发明者，也是九链环问题真正透彻的最初研究者。

所以，我们对于几十年前在我国人工智能学界曾经有过的研究九链环的热潮，不得不作一些反思，那就是我们对于问题的研究，是否确实应更着重于深度和广度。为什么不是我们，而是外国人来发现原本是我们提出的问题的理论基础，从而对它给出真正的解呢？

高德纳还列举了组合模式生成与欧洲教堂的洪钟鸣响模式的联系，揭示了各种钟鸣模式与生成排列的关系，这同样给人以巨大的启示。

3. 关于组合模式同字母算术或密码数学的关系。

在国外，有些人或者生来就喜欢标新立异、独出心裁，或者有闲情逸致来这样做。因而很早以前就有人提出字母算术的概念。如亨利·厄思尼斯特·达德尼在1924年就提出这样一个著名问题：如果每个字母代表着不同的十进制数字，问要使

$$\begin{array}{r} \text{SEND} \\ +\text{MORE} \\ \hline \text{MONEY} \end{array}$$

表示一个正确的求和，则每个字母应当分别表示什么数？

这看似纯粹游戏的题目，却引发了人们的思考。假若要传送的是数字信息，用字母来对它们加密，这不就成了密码了吗？因而在1931年，西蒙·瓦特利宽特就给它起了另一个名称“密码数学”。

在他们的开创下，字母算术或密码数学就蓬勃发展起来。开头，人们关注于

具有唯一解的问题。而后，人们开始考虑它们能有的各种解，乃至研究它有多少解。有唯一解的情况，称为纯的字母算术或纯的密码数学。而有的问题，不仅是字母上有意义，而且以数值解代入后仍正确。例如

VIOLIN + VIOLIN + VIOLA = TRIO + SUNATA

（小提琴 + 小提琴 + 中提琴 = 三重 + 合奏）

ZEROES + ONES = BINARY

（诸0 + 诸1 = 二进制）

COUPLE + COUPLE = QUARTET

（两个 + 两个 = 四个）

等等。这些通过加法给出的字母算术，称为加法性字母算术。还可以有乘法性字母算术，例如

TWO × TWO = SQUARE

PI × R × R = AREA

等等。依照这些问题，我们也可以提出，如何代入数字，使

工业化 + 农业化 + 科学化 = 现代化

和

立党为公 + 执政为民 = 为人民服务

以及

更快 × 更高 × 更强 = 奥林匹克精神

成立?

除了字母算术外，也还有像在0与9这些数字之间，如何加上适当的加减乘除号，使得结果成为一个数，如100。在这方面的研究，不仅仅在于给出正确答案，还要研究它们可以有多少个解，如对于123456789，可以有12种方法来插入+和−，使其和为100，如100 = 1 + 23 − 4 + 5 + 6 + 78 − 9 = 123 − 45 − 67 + 89 = − 1 + 2 − 3 + 4 + 5 + 6 + 78 + 9。又如，在123456789876543 21中，插入+和 − 的一个方式为100 = − 1234 − 5 − 6 + 7898 − 7 − 6543 − 2 − 1。

为了鼓励这方面的研究，国外出版了多种杂志，出版的专门书籍也不少。我们所知道的杂志就有*J. Recreational Math*（娱乐数学杂志）、*Recreational Math. Magazine*（娱乐数学期刊）等。这样，也就使这方面的研究长盛不衰，一直延续下来。它也推动了人们特别是青少年对于科学的兴趣和追求。

我们说，现代化的科学技术源自于基础理论。有了基础理论的深厚功底和深刻探索，才带来了现代科学技术的发展，最雄辩的例子是爱因斯坦的理论，该理论乃是现代科学技术用之不竭的思想源泉。有一项研究表明，现代科学技术中直接利用爱因斯坦的理论成果的就达2300多项。因此我们可以斩钉截铁地说，没有

基础数学的研究成果，也就没有今天的计算机科学和经济学等的辉煌成果。我国今天取得的航天领域的成就，也是我们在基础理论和综合科学技术方面的成就。然而，在我国仍然存在忽视基础理论研究的倾向，或者更确切地说，我们在基础理论研究上的投入和从事这方面研究的人数，同我们这样一个大国，是不大相称的。为了我国今后科学的持续发展，特别是为了使我国早日成为不仅是经济强国、军事强国，更是科学强国，我们希望上至党中央国务院，下至各地方党委政府，还有我国人民，都能从振兴中华，从民族的未来发展上重视这个问题。如果我们能引导青少年一代，首先重视自己的思想、素质的成长，然后是去专注于包括娱乐数学、娱乐物理这种难题的研究，从而激发对科学本身的兴趣，而不是沉溺于网上那些无聊的活动，那我们民族的科学素质就会大为改观。而到那时，科学大家的出现，诺贝尔奖、图灵奖的获得，都是指日可待的。

因此，我想，我们出版高德纳的书，其深远意义也应包括这些，而不仅仅在于计算机的那些算法本身。

我们等待着祖国科学辉煌的明天。

译　者

2006年2月于广西梧州

前　言

我对所有朋友表示感谢，并且在这里对那些一段时间之后不再问我：
“书写得怎样了”的朋友，表示最特殊的感谢。

——彼得·约·戈姆斯（Peter J. Gomes），
The Good Book（1996）

本书是《计算机程序设计艺术 第4卷 组合算法》的第2册。如同在第1卷第1册的前言中所说的那样，我采用这种预备性的形式来公布这些内容，因为我知道，完成第4卷的任务将花费多年时间。我迫不及待地等着人们来阅读我已写完的内容并向我提供宝贵的反馈意见。

第4卷将有第1册，甚至还有第0册，但我先写完第2册。有经验的程序员都懂得，在程序的主体完成之前，程序的初始化通常不能被适当地写出。

这一册包含了关于组合查找的一个极冗长的一章的7.2.1.1节和7.2.1.2节。假定我能保持自己的健康状态，则第7章将最终归入三卷中（即卷4A、4B和4C）。它将以图论的一个简短复习开始，重点在于突出The Stanford GraphBase一书中的某些重要图形，从中我将引出许多例子，然后开始7.1节。它将涉及按进位操作并讨论有关布尔函数的算法。7.2节是关于生成所有可能性的，而它由7.2.1节“生成基本的组合模式”开始，这就设置了本册的主要内容的框架，即7.2.1.1节（在这里我通过讨论n元组的生成来开始）和7.2.1.2节（该节把这个思想扩充到排列）。然后将是7.2.1.3节（关于组合）和7.2.1.4节（关于整数分划），以及7.2.1.5节（关于集合分划），这3节全在第3册中。7.2.1.6节（关于树）以及7.2.1.7节（关于组合生成的历史）将组成第4册。7.2.2节一般地讨论回溯，然后如果一切顺利，它将继续下去。关于当前设想的整个第7章的轮廓如同在`taocp`的网页所见到的那样，在封底中对它作了引述。

我怀着非常喜悦的心情来写这部分内容，就像许多年前写第2卷时我所感觉到的激动那样。如同在第2卷中那样，在那里我高兴地看到，初等概率论和数论的基本原理很自然地出现在关于随机数生成和算术运算的研究中。而在准备7.2.1节的过程中我了解到，当我们研究组合生成的算法时，初等组合学的基本原理也自然地出现，并且具有高度启发。因此，我再次发现，一个美丽的故事就在那里等候着被讲述。

我本打算以少得多的篇幅来讲述这一课题，但当我看到对于组合研究来说，那些思想是何等基本，我就知道了，除非我十分透彻地讲述它们，否则我将绝不可能满意。因此我正尽我所能地构筑理论和实践的思想的坚固基础，它们将支持许多种类的可靠的超级结构。

厌恶平方根或代数的普通男孩很喜欢玩相同原理的智力游戏，
而且可能导致他去学习一门课程，而这将以使家庭成员
惊异的方式发展数学和创造发明的才能。

——萨姆尔·洛伊德（Sam Loyd），
The World of Puzzledom（1896）

对于在本册中所出现的每个错误，若是头一个向我报告的，我将很高兴为发现者支付2.56美元。无论这个错误是印刷上的、技术上的还是历史上的。对于我忘了放入索引中的条目，同样的报酬也适用。而每一个对于正文的改进建议值32美分。（而且，如果你找到一个习题的更好的解，我将通过在最后出版的书中发表你的名字，而不仅仅是钱，来给你以永久的荣誉的奖励。）

我愿在此对播口阳一表示感谢，是他帮助我构造和重新构造撰写本书的计算机，我还要感谢弗朗克·拉斯基（Frank Ruskey），因为他大胆地把本书的早期草稿介绍给大学生们并且向我讲述了他的授课经验。

这里所用的记号以及未加说明的那些记号可在第1、2、3卷末尾的记号索引中找到。那些索引指出可获得进一步信息的位置。当然，第4卷将在某一天包含它本身的记号索引。

在《计算机程序设计艺术》所有未来版本中的机器语言例子，都将以MMIX计算机为基础，它在第1卷第1册的1.3.1'节中定义。对于现有小册子的1.3.1'节、1.3.2'节、1.4.1'节、1.4.2'节以及1.4.3'节的交叉参考，请参阅相关分册。

对于尚未完成的内容的交叉参考有时在随后页中以“00”表示；这个不可能的值是日后要加以提供的实际号码的位置表示。

祝阅读愉快!

D. E. K

加利福尼亚，斯坦福

2004年12月

第7章

组合查找

说明：这一章的开始部分将出现在计划于2006年和2007年出版的第4卷第0册和第4卷第1册中。

7.2 生成所有可能性

全都到齐或计算在内了，长官。
——传统的美国军队的说法
全都到齐和正确，长官。
——传统的英国军队的说法

7.2.1 生成基本的组合模式

在这一节里，我们的目标是研究在某个组合世界中，跑遍所有可能性的方法，因为我们经常面对这样一些问题，其中有必要并希望对所有情况作穷尽的考察。例如，我们可能要考察一个给定集合的所有排列。

有些作者把这个叫做枚举所有可能性的任务；但这并非十分贴切，因为“枚举”经常意味着我们只要计算全部情况的个数，而不是真正要全都考察它们。如果某个人要你枚举{1, 2, 3}的排列，你回答3! = 6是十分正当的，你不需要给出更完备的答案{123, 132, 213, 231, 312, 321}。

其他的作者谈及列举所有可能性；但它也不是这么大的字眼。没有一个聪明人愿意通过以几千张纸来打印出{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}的10! = 3 628 800个排列的清单，甚至也没有人愿意在一个计算机文件中把它们全写出来。我们真正要做的是以某种数据结构来以可记忆的方式给出它们，使得一个程序可以一次一个地考察每个排列。

因此我们谈及生成我们所需要的所有组合对象，并且依次地访问每个对象，就如同在2.3.1节中研究树的遍历的算法那样。在那里，我们的目标是访问一棵树的每一个节点，现在我们转到系统地遍历可能性的一个组合空间的若干算法。

他已经在表中得到m字——
他已经在表中得到m字；
而且他们将不会丢失m——
他们将不会丢失m。
——威廉斯·S. 吉尔伯特（William S. Gilbert），
The Mikado（1885）

7.2.1.1 生成所有n元组

我们由简单的开始，首先考虑如何跑遍由n个二进制数字组成的所有2^n个字符串。同样地，我们要访问所有n元组$(a_1, \cdots, a_n)$，其中每个a_j为0或1。实际上，这个任务也等价于考察一个给定集合$\{x_1, \cdots, x_n\}$的所有子集。因为我们可以说，x_j在

子集中当且仅当$a_j = 1$。

当然，这样一个问题有一个极其简单的解。所要做的只是以二进制数$(0\cdots00)_2 = 0$开始，并重复地加1，直到达到$(1\cdots11)_2 = 2^n - 1$为止。然而我们将看到，当我们更深入地考察它时，即便对于这个完全平凡的问题，也有令人惊讶的感兴趣之点。因此，当我们转去生成更困难的种类的模式时，我们对于n元组的研究将是有益的。 1

首先，我们可以看到，二进制记号的技巧可扩展到其他种类的n元组。例如，如果我们要来生成所有的$(a_1, \cdots, a_n)$，其中每个a_j是十进数字{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}之一，我们可以在十进制数字系统中简单地从$(0\cdots00)_{10} = 0$开始计算到$(9\cdots99)_{10} = 10^n - 1$。而如果我们要跑遍所有情况，其中

$$0 \leqslant a_j < m_j \qquad 1 \leqslant j \leqslant n \tag{1}$$

其中的上限m_j对于向量$(a_1, \cdots, a_n)$的不同分量，可能是不同的。这个任务实质上同在一个混合进制数字系统中重复地加1到数

$$\begin{bmatrix} a_1, a_2, \cdots, a_n \\ m_1, m_2, \cdots, m_n \end{bmatrix} \tag{2}$$

中是一样的。参见等式4.1-(9) 和习题4.3.1-9。

我们可能要暂时停下来，更形式地来描述这一过程。

算法M（混合进制的生成） 通过重复地往在（2）中表示的混合进制的数加1，直到出现溢出为止，这一算法访问满足（1）的所有n元组。这里为了方便起见引进辅助变量a_0和m_0。

M1. [初始化。] 对于$0 \leqslant j \leqslant n$，置$a_j \leftarrow 0$，并置$m_0 \leftarrow 2$。

M2. [访问。] 访问n元组$(a_1, \cdots, a_n)$。（要来考察所有n元组的这个程序现在就做这件事）。

M3. [准备来加1。] 置$j \leftarrow n$。

M4. [必要时进位。] 如果$a_j = m_j - 1$，则置$a_j \leftarrow 0$，$j \leftarrow j - 1$，并且重复本步骤。

M5. [增加，除非已完成。] 如果$j = 0$，则算法终止，否则$a_j \leftarrow a_j + 1$并转回步骤M2。 ∎

算法M是简单和直截了当的，但我们不应忘记，当n是很小的常数时，嵌套的循环甚至更简单。例如当$n = 4$时，我们可以写出以下结构:

```
For  a1 = 0, 1, ..., m1 - 1 (按此顺序) 做下列:
  For  a2 = 0, 1, ..., m2 - 1 (按此顺序) 做下列:
    For  a3 = 0, 1, ..., m3 - 1 (按此顺序) 做下列:
      For  a4 = 0, 1, ..., m4 - 1 (按此顺序) 做下列:
        访问 (a1, a2, a3, a4)。
```

(3)

这些指令等价于算法M，而且在任何程序设计语言中，它们很容易表达。

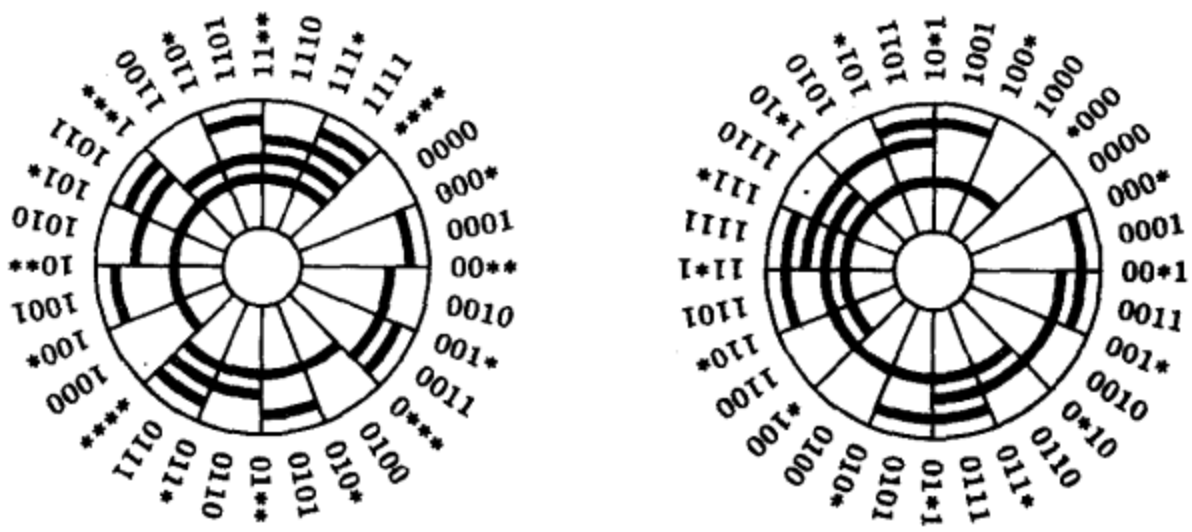
格雷二进制码。 如同在一本词典中那样，算法M以词汇字顺序跑遍所有$(a_1, \cdots,$

a_n)。但还有许多情况，我们情愿以某种其他顺序来访问那些n元组。最著名的可供选择的安排是所谓的格雷（Gray）二进制码，它以这样一种方式来列出所有2^n个n位的串，即它以一种简单和规则的方式，每次仅改变一个二进位。例如，对于$n = 4$，格雷二进制码是

2

$$0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,$$
$$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000 \tag{4}$$

在模拟信息被转换成数字信息或者反过来的情况下，这样的编码特别重要。例如，假设我们要来标识在一个转动的磁盘上我们当前的位置，这个磁盘使用区分白与黑的4个传感器，已经被划分成16个扇区。如果我们像在图10a中那样，从0000到1111使用词汇字顺序来标记磁道的话，则在扇区与扇区之间的边界处，就会出现惊人的不精确的度量，但是在图10b中的编码绝不会给出差的读数来。



a) 词典字顺序下的二进制编码　　b) 格雷二进制编码

图 10

格雷二进制码可以以许多等价的方式来加以定义。例如，如果Γ_n表示n位字符串的格雷二进制序列，我们可以通过两个规则

$$\Gamma_0 = \varepsilon$$
$$\Gamma_{n+1} = 0\,\Gamma_n,\ 1\Gamma_n^R \tag{5}$$

递归地定义Γ_n。这里ε表示空串，$0\Gamma_n$表示以0加到每个串前缀的序列Γ_n，而$1\Gamma_n^R$表示以1加到每个串前缀，而本身反序的序列Γ_n。由于Γ_n的最后一个串等于Γ_n^R的头一个串，由(5)显然可知，如果Γ_n满足相同性质，则在Γ_{n+1}的每一步恰有一个二进制位改变。

定义序列$\Gamma_n = g(0), g(1), \cdots, g(2^n - 1)$的另一个方法是对于它的个别元素$g(k)$给出一个明显的公式。其实，由于$\Gamma_{n+1}$以$0\Gamma_n$开始，如果我们把0和1的每个串看做带有任选的前导位0的一个二进制整数的话，则无穷序列

$$\begin{aligned}\Gamma_\infty &= g(0), g(1), g(2), g(3), g(4), \cdots \\ &= (0)_2, (1)_2, (11)_2, (10)_2, (110)_2, \cdots\end{aligned} \tag{6}$$

是所有非负整数的一个排列。于是Γ_n由 (6) 的头2^n个元素组成，通过在必要时把0

插入左边而转换成n位的串。 3

当$k = 2^n + r$，其中$0 \leqslant r < 2^n$时，关系式 (5) 告诉我们$g(k)$ 等于$2^n + g(2^n - 1 - r)$。因此我们可以通过对n的归纳法说明，其二进制表示为$(\cdots b_2 b_1 b_0)_2$的整数k有其表示为$(\cdots a_2 a_1 a_0)_2$的一个格雷二进制等价$g(k)$，其中对于$j \geqslant 0$，

$$a_j = b_j \oplus b_{j+1} \tag{7}$$

(参见习题6)。例如$g((111001000011)_2) = (100101100010)_2$。反之，如果给定$g(k) = (\cdots a_2 a_1 a_0)_2$，通过逆转等式 (7) 的系统，我们可求出$k = (\cdots b_2 b_1 b_0)_2$，得到对于$j \geqslant 0$，

$$b_j = a_j \oplus a_{j+1} \oplus a_{j+2} \oplus \cdots \tag{8}$$

这个无穷和实际上是有限的，因为对于所有很大的t，$a_{j+t} = 0$。

等式 (7) 的许多令人高兴的结果之一是，通过按位进行的算术，可以很容易地计算出$g(k)$:

$$g(k) = k \oplus \lfloor k/2 \rfloor \tag{9}$$

类似地，等式 (8) 中的逆函数满足

$$g^{[-1]}(l) = l \oplus \lfloor l/2 \rfloor \oplus \lfloor l/4 \rfloor \oplus \cdots \tag{10}$$

然而这个函数要求更多的计算（参见习题7.1-00)。我们也可以从等式 (7) 推导出，如果k和k'是任何非负整数，则

$$g(k \oplus k') = g(k) \oplus g(k') \tag{11}$$

还有另一个结果是（$n + 1$）位的格雷二进制编码可以被写成为

$$\Gamma_{n+1} = 0\Gamma_n, (0\Gamma_n) \oplus 110\cdots0$$

这个模式是显然的，例如在(4)中。同(5)相比较，我们看到，逆转格雷二进制编码的顺序等价于对头一个二进制位求补:

$$\Gamma_n^R = \Gamma_n \oplus \overbrace{10\cdots0}^{n-1} \tag{12}$$

以下习题证明，等式 (7) 中所定义的函数$g(k)$，以及在等式 (8) 中定义的它的逆$g^{[-1]}$，有许多进一步的性质和有趣的应用。有时我们把这些当做函数，其中取二进制串作为自变量，取二进制串作为值，而在其他时间，我们把它们看做是通过二进制记号，和不相关的前导零，由整数到整数的函数。

格雷码由于弗朗克·格雷（Frank Gray）而得名，他是一名物理学家，因帮助设计长期可兼容的彩色电视的广播所使用的方法而闻名 [*Bell System Tech. J.*（贝尔系统技术杂志）13(1934)，464-515]。他发明Γ_n以供脉冲编码调制使用，这是用于数字信号的模拟转换的一个方法 [参见*Bell System Tech. J.*30(1951)，38-40；美国专利2632058（1953年3月17日)；威·拉·贝内特（W. R. Bennett)，*Introduction to Signal Transmission*（信号传输导论）(1971)，238-240]。但是，“格雷二进制编码”的思想远在他从事此项工作之前就已为人所知了。例如，它出现在乔治·斯

蒂比兹（George Stibitz）的美国专利2307868中（1943年1月12日）。更重要的是，Γ_5于1878年就被埃米里·鲍多特（Émile Baudot）用在所演示的一台电报机器上。
4 正是为了纪念他，后来使用了“波特”（baud）这个术语。大约在同一时间，由奥托·沙费勒尔（Otto Schäffler）独立地设计出了用于电报的类似的但比较不系统的编码[参见*Journal Télégraphique*（电报杂志）4(1878)，252-253；*Annales Télégraphiques*（电报年鉴）6(1879)，361，382-383]。⊖

事实上，在一个经典的玩具上就体现出格雷二进制编码的原理，它已经吸引人们好几个世纪了，现在英文中叫做“中国环难题”（译者注：即九龙环），英国人习惯上叫它为“厌烦的铁”。图11示出了7个环的例子，其挑战性是要从杆上把环取下来，而这些环以这样一种方式相互锁住，即仅有两种基本类型的移动是可能的（尽管从图中看可能并不明显）：

a) 最右边的环任何时候都可以卸下或替代；

b) 其他的任何环可被拆卸或替代，当且仅当它右边的环在杆上，且这个右边的环右边的所有环都卸下了。

我们可以以二进制记号来表示这个难题的当前状态，如果一个环在杆上就写1，而如果它不在就写0。因此，图11示出环的状态为1011000。（左数第二个环就编码为0，因为它完全在杆外面。）

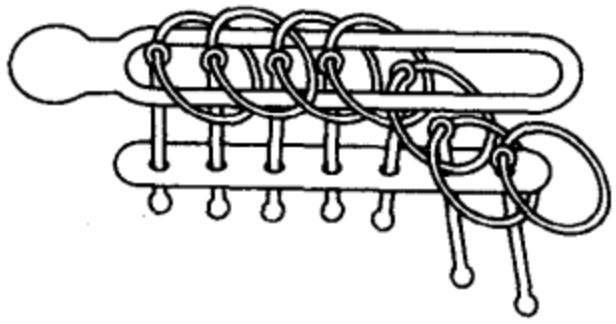


图11 中国环难题

法国一位名为路易斯·格罗斯（Louis Gros）的地方行政官在叫做“步行、短途旅游的理论”（*Théorie du Baguenodier*[*sic*]）的小册子（Lyon: Aimé Vingtrinier，1872）里，演示了中国环和二进数之间的一个明显关系，该书是匿名发表的。如果这些环处于状态$a_{n-1}\cdots a_0$，而且如果我们以等式(8)来定义二进数$k=(b_{n-1}\cdots b_0)_2$，他证明了，解决这个难题的充分必要条件是恰好需要k步。因此格罗斯才是格雷码真正的发明者。

肯定地，如果没有这个引人入胜的历史
和有教益的难题，就找不到发源地。
——亨利·欧·杜迪尼（Henry E. Dudeney）（1901）

⊖ 有些作者断言，格雷码是由埃利沙·格雷（Elisha Gray）发明的，他在和鲍多特（Bandot）及沙费勒尔（Schäffler）相同的时期里研制了打印电报机，但这样的断言是不成立的，尽管埃利沙得到了关于发明电话的优先级的原始思想[参见罗·威·泰勒（L. W. Taylor），*Amer. Physics Teacher*（美国物理教师）5(1937), 243-251]。

当环处于不同于00…0或10…0的任何状态时，恰好两步是可能的，即类型(a)之一和类型(b)之一，只有这些移动之一才朝向所希望的目标前进；而另一步则是朝后走，因而将需要返工。类型(a)把k变成为$k \oplus 1$；因此当k是奇数时，我们要做它，因为这将减少k。类型(b)的移动从对于$1 \leqslant j < n$以在$(10^{j-1})_2$处结尾的位置移动，把k变成为$k \oplus (1^{j+1})_2 = k \oplus (2^{j+1}-1)$。当$k$为偶数时，我们要$k \oplus (2^{j+1}-1)$等于$k-1$， 5
这意味着k必须是2^j的一个倍数但不是2^{j+1}的一个倍数；换言之，

$$j = \rho(k) \tag{13}$$

其中ρ是等式7.1-(oo)的“直尺函数”。因此当这个难题真正获解时，这些环遵循一个很好的模式：如果我们以$0, 1, \cdots, n-1$来对它们进行编号（由自由端开始），则进入和离开杆的环的移动序列是以$\cdots, \rho(4), \rho(3), \rho(2), \rho(1)$结尾的数的序列。

往后进行，逐个地把环放上或放出直到我们达到最终状态10…0（如同1693年约翰·沃里斯（John Wallis）所发现的那样，它比起设想的更难的状态11…1来更难以达到）。产生出用于在格雷二进制码中进行统计的一个算法：

算法G（格雷二进制码的生成） 由$(0, \cdots, 0, 0)$开始，而且一次只改变一个二进制位，这个算法访问所有的n元组$(a_{n-1}, \cdots, a_1, a_0)$，而且还维持一个奇偶位$a_\infty$使得

$$a_\infty = a_{n-1}, \oplus \cdots \oplus, a_1 \oplus a_0 \tag{14}$$

它逐次地求补二进制位$\rho(1), \rho(2), \rho(3), \cdots, \rho(2^n-1)$而后停止。

G1. [初始化。] 对于$0 < j < n$，置$a_j \leftarrow 0$；也置$a_\infty \leftarrow 0$。

G2. [访问。] 访问n元组$(a_{n-1}, \cdots, a_1, a_0)$。

G3. [改变奇偶性。] 置$a_\infty \leftarrow 1 - a_\infty$。

G4. [选择j。] 如果$a_\infty = 1$，则置$j \leftarrow 0$，否则令$j > 1$是使得$a_{j-1} = 1$的极小者。（在我们执行这步k次后，$j = \rho(k)$。）

G5. [对坐标j求补。] 如果$j = n$则终止；否则置$a_j \leftarrow 1 - a_j$，并返回G2。 ∎

如果我们计算像

$$X_{000} - X_{001} - X_{010} + X_{011} - X_{100} + X_{101} + X_{110} - X_{111}$$

或

$$X_\emptyset - X_a - X_b + X_{ab} - X_c + X_{ac} + X_{bc} - X_{abc}$$

这样一个和，则奇偶位a_∞很容易出来，其中符号依赖于一个二进制串的奇偶性或者一个子集中的元素的数目。这样的和经常出现在如等式1.3.3-(29)那样的“容斥”公式中。为了有效性，奇偶位也是必要的：如果没有它，我们就不能容易地从确定j的两个方法中选择一个，这两种方法对应于在中国环难题中实施类型(a)还是类型(b)的移动。但是算法G最重要的特征是步骤G5只进行一个坐标变化。因此对于我们正在求和的项X，或者当我们访问的每个n元组，对于我们关心的无论什么结

构，通常只有一个简单的变化是必要的。

> 当然，除了像据说爱尔兰铁路所使用的移开每列火车的最后一节车厢这样的方案外，不可能在最底位数字上删除所有二义性。因为它太容易引进冲突的损坏了。
>
> ——乔·罗·斯蒂比兹（G. R. Stibitz）和朱·阿·拉尔里威（J. A. Larrivee），*Mathematics and Computers* (1957)

6

约·伦·沃尔斯（J. L. Walsh）在同现在称为沃尔斯函数的重要函数序列相关联中，发现了格雷二进制码的另一个重要性质［参见*Amer. J. Math.*（美国数学学报）45 (1923)，5-24］。对于所有实数x，令$w_0(x)=1$，而且对于$k>0$

$$w_k(x)=(-1)^{\lfloor 2x\rfloor\lceil k/2\rceil}w_{\lfloor k/2\rfloor}(2x) \tag{15}$$

例如，每当x是一个整数或一个整数加1/2时，$w_1(x)=(-1)^{\lfloor 2x\rfloor}$改变符号。由此得出，对于所有$k$，$w_k(x)=w_k(x+1)$，以及对于所有$x$，$w_k(x)=\pm 1$。更重要的是$w_k(0)=1$和$w_k(x)$在区间（0..1）中恰有$k$个符号变化，使得当$x$从左边逼近于1时，它趋近$(-1)^k$。因此$w_k(x)$的特性稍微像$\cos k\pi x$或$\sin k\pi x$这样一个三角函数，我们可以非常像传统中把它们表示为傅里叶级数那样，把其他函数表示为沃尔斯函数的线性组合。这一事实，连同$w_k(x)$的简单离散性质，使沃尔斯函数在同信息传输、图像处理以及许多其他应用相关联的计算机计算中极其有用。

图12示出头8个沃尔斯函数以及它们的三角函数近亲。与$\cos k\pi x$和$\sin k\pi x$频率为$k/2$这一事实相类似，工程师们普遍地把$w_k(x)$称为频率为k的沃尔斯函数。［例如，参见*Sequency Theory: Foundation and Applications*（序列理论：基础和应用）(New York:Academic Press, 1977)。作者为亨·弗·哈尔穆特（H. F. Harmuth）。］

7

尽管乍看起来，等式(15)显得挺复杂，但实际上它提供了一个容易的方法，通过归纳法来看出为什么$w_k(x)$恰如所断言的那样有k个符号变化。如果k为偶数，比如说$k=2l$，则对于$0\leqslant x<1/2$，我们有$w_{2l}(x)=w_l(2x)$，其效果只不过是压缩函数$w_l(x)$到一半的空间中，所以$w_{2l}(x)$累计了迄今的l个符号变化。于是在$1/2\leqslant x<1$的范围内$w_{2l}(x)=(-1)^l w_l(2x)=(-1)^l w_l(2x-1)$；这就连接了$w_l(2x)$的另一个副本，如果需要的话使符号触发以避免在$x=1/2$处符号发生变化。函数$w_{2l+1}(x)$是类似的，但当$x=1/2$时它迫使符号发生变化。

那么对于格雷二进制码这有什么要做的呢？沃尔斯发现，他的函数几乎全都能借助于称之为拉德曼彻函数［汉斯·拉德曼彻（Hans Rademacher），*Math. Annalen* 87(1922)，112-138］的较简单函数来表达

$$r_k(x)=(-1)^{\lfloor 2^k x\rfloor} \tag{16}$$

当$(\cdots c_2c_1c_0.c_{-1}c_{-2}\cdots)_2$是$x$的二进制表示时，它取$(-1)^{c_{-k}}$的值。确实，我们有$w_1(x)=r_1(x)$，$w_2(x)=r_1(x)r_2(x)$，$w_3(x)=r_2(x)$，而且一般地当$k=(b_{n-1}\cdots b_1b_0)_2$时

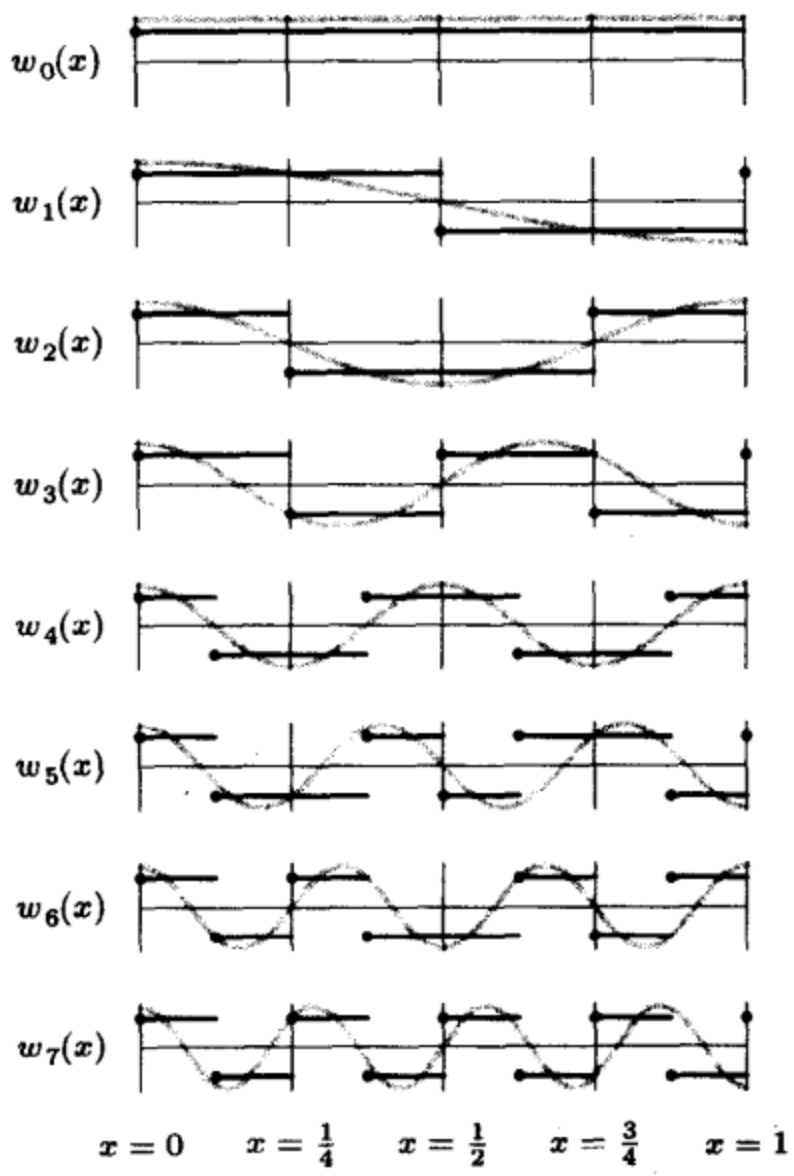


图12 对于$0 \leqslant k < 8$的沃尔斯函数$w_k(x)$，而且为了比较以灰色显示类似的三角函数$\sqrt{2}\cos k\pi x$

$$w_k(x) = \prod_{j>0} r_{j+1}(x)^{b_j \oplus b_{j+1}} \tag{17}$$

（参见习题33。）于是按照(7)，在$w_k(x)$ 中$r_{j+1}(x)$ 的指数是格雷二进制数$g(k)$ 的第j个二进制位，因此我们有，对于$k>0$，

$$w_k(x) = r_{\rho(k)+1}(x)\, w_{k-1}(x) \tag{18}$$

等式(17)意味着一个使用方便的公式

$$w_k(x) w_{k'}(x) = w_{k \oplus k'}(x) \tag{19}$$

它比起正弦与余弦对应的乘积公式要简单得多。这个恒等式很容易得出，因为对于所有的j和x，$r_j(x)^2 = 1$。因此$r_j(x)^{a \oplus b} = r_j(x)^{a+b}$。当$k \neq k'$时，在$w_k(x)\, w_{k'}(x)$ 的平均值为零的意义下，它特别意味着$w_k(x)$ 和$w_{k'}(x)$ 是正交的。对于像1/2或13/8这样的k的分数值，我们也可以使用等式 (17) 来定义$w_k(x)$。

2^n个数 $(X_0, \cdots, X_{2^n-1})$ 的沃尔斯变换是由等式 $(x_0, \cdots, x_{2^n-1})^T = W_n (X_0, \cdots, X_{2^n-1})^T$ 定义的向量，其中W_n是$2^n \times 2^n$的矩阵，对于$0 \leqslant j, k < 2^n$，在j行k列处有$w_j(k/2^n)$。例如，图12告诉我们当$n=3$时，沃尔斯变换是

$$\begin{pmatrix} x_{000}\\ x_{001}\\ x_{010}\\ x_{011}\\ x_{100}\\ x_{101}\\ x_{110}\\ x_{111}\end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 1&1&1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}\\ 1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}&1&1\\ 1&1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}\\ 1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}&1\\ 1&\bar{1}&\bar{1}&1&\bar{1}&1&1&\bar{1}\\ 1&\bar{1}&1&\bar{1}&\bar{1}&1&\bar{1}&1\\ 1&\bar{1}&1&\bar{1}&1&\bar{1}&1&\bar{1}\end{pmatrix} \begin{pmatrix} X_{000}\\ X_{001}\\ X_{010}\\ X_{011}\\ X_{100}\\ X_{101}\\ X_{110}\\ X_{111}\end{pmatrix} \tag{20}$$

8

这里$\bar{1}$代表-1，下标方便地当作二进制串000-111，而不是整数（0~7）。哈达玛变换（Hadamard transform）可类似地定义，但使用矩阵H_n来代替W_n，其中H_n在j行k列处有$(-1)^{j\cdot k}$，这里“$j\cdot k$”为二进制表示$j=(a_{n-1}\cdots a_0)_2$和$k=(b_{n-1}\cdots b_0)_2$的点积$a_{n-1}b_{n-1}+\cdots+a_0b_0$。例如，对于$n=3$的哈达玛变换是

$$\begin{pmatrix} x'_{000}\\ x'_{001}\\ x'_{010}\\ x'_{011}\\ x'_{100}\\ x'_{101}\\ x'_{110}\\ x'_{111}\end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 1&\bar{1}&1&\bar{1}&1&\bar{1}&1&\bar{1}\\ 1&1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}\\ 1&\bar{1}&\bar{1}&1&1&\bar{1}&\bar{1}&1\\ 1&1&1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}\\ 1&\bar{1}&1&\bar{1}&\bar{1}&1&\bar{1}&1\\ 1&1&\bar{1}&\bar{1}&\bar{1}&\bar{1}&1&1\\ 1&\bar{1}&\bar{1}&1&\bar{1}&1&1&\bar{1}\end{pmatrix} \begin{pmatrix} X_{000}\\ X_{001}\\ X_{010}\\ X_{011}\\ X_{100}\\ X_{101}\\ X_{110}\\ X_{111}\end{pmatrix} \tag{21}$$

这和n维立体的离散傅里叶变换，即等式4.6.4-(38) 是一样的。因此通过采用4.6.4节中讨论的雅特斯（Yates）方法，我们可以迅速地“就地”计算它:

给定	第一步	第二步	第三步
X_{000}	$X_{000}+X_{001}$	$X_{000}+X_{001}+X_{010}+X_{011}$	$X_{000}+X_{001}+X_{010}+X_{011}+X_{100}+X_{101}+X_{110}+X_{111}$
X_{001}	$X_{000}-X_{001}$	$X_{000}-X_{001}+X_{010}-X_{011}$	$X_{000}-X_{001}+X_{010}-X_{011}+X_{100}-X_{101}+X_{110}-X_{111}$
X_{010}	$X_{010}+X_{011}$	$X_{000}+X_{001}-X_{010}-X_{011}$	$X_{000}+X_{001}-X_{010}-X_{011}+X_{100}+X_{101}-X_{110}-X_{111}$
X_{011}	$X_{010}-X_{011}$	$X_{000}-X_{001}-X_{010}+X_{011}$	$X_{000}-X_{001}-X_{010}+X_{011}+X_{100}-X_{101}-X_{110}+X_{111}$
X_{100}	$X_{100}+X_{101}$	$X_{100}+X_{101}+X_{110}+X_{111}$	$X_{000}+X_{001}+X_{010}+X_{011}-X_{100}-X_{101}-X_{110}-X_{111}$
X_{101}	$X_{100}-X_{101}$	$X_{100}-X_{101}+X_{110}-X_{111}$	$X_{000}-X_{001}+X_{010}-X_{011}-X_{100}+X_{101}-X_{110}+X_{111}$
X_{110}	$X_{110}+X_{111}$	$X_{100}+X_{101}-X_{110}-X_{111}$	$X_{000}+X_{001}-X_{010}-X_{011}-X_{100}-X_{101}+X_{110}+X_{111}$
X_{111}	$X_{110}-X_{111}$	$X_{100}-X_{101}-X_{110}+X_{111}$	$X_{000}-X_{001}-X_{010}+X_{011}-X_{100}+X_{101}+X_{110}-X_{111}$

注意H_3的行是W_3的行的一个排列。一般来说，这为真，所以通过排列哈达玛变换的元素，我们可以得到沃尔斯变换。习题36讨论这些细节。

更快地进行。当我们要跑遍2^n个可能性时，我们通常要尽可能多地减少计算时间。算法G对于对$(a_{n-1},\cdots,a_0)$的每一个访问只需要对一个二进制位a_j求补，但当选择j的一个适当的值时，它在步骤G4中循环。吉迪安·厄尔里茨（Gideon Ehrlich）[*JACM* 20(1973)，500-513]提出了另一个方法，他介绍了无循环组合生成的思想。通过一个无循环的算法，预先要求在逐次的访问之间所执行的操作次数是有界的。因此在一个新的模式被生成之前决没有一个长的等待。

在7.1节中，我们学习了确定在一个二进制数中前导或尾部0的个数的一些快速方法的技巧。这些方法可在步骤G4中用来使算法G成为无循环的，假定n不是大得不合理的话。但厄尔里茨方法与众不同，而且也更加通用，所以它在我们进行有

效计算的技术库中向我们提供了新的武器。这里是关于他的方法如何可用来生成二进制n元组的一个介绍[参见毕特纳（Bitner）、厄尔里茨和赖因戈德（Reingold），*CACM* 19(1976)，517-521]:

9

算法L（*无循环的格雷二进制码的生成*） 和算法G一样，这个算法以格雷二进制码的顺序访问所有二进制的n元组$(a_{n-1}, \cdots, a_0)$。但不同于维持一个奇偶位，它使用一个“焦点指针”数组$(f_n, \cdots, f_0)$。下面会讨论它的意义。

L1. [初始化。] 对于$0 \leqslant j < n$，置$a_j \leftarrow 0$, $f_j \leftarrow j$，并置$f_n \leftarrow n$。（允许一个无循环算法在它的初始化步骤里有循环，只要初始化步骤是合理地有效的；毕竟，每个程序都需要被加载和启动。）

L2. [访问。] 访问n元组$(a_{n-1}, \cdots, a_1, a_0)$。

L3. [选择j。] 置$j \leftarrow f_0$，$f_0 \leftarrow 0$。（如果这是我们执行本步骤的第k次，则j现在等于$\rho(k)$。）如果$j = n$，则终止；否则置$f_j \leftarrow f_{j+1}$，以及$f_{j+1} \leftarrow j + 1$。

L4. [坐标j取补。] 置$a_j \leftarrow 1 - a_j$，并返回步骤L2。 ∎

例如，当$n = 4$时，这个计算如下进行。如果在二进制串$b_3\ b_2\ b_1\ b_0$中对应的二进制位b_j为1，使得$a_3\ a_2\ a_1\ a_0 = g(b_3\ b_2\ b_1\ b_0)$，则在这个表中诸元素$a_j$已经画了下划线。

a_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
a_2	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
a_1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
a_0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
f_3	3	3	3	3	3	3	3	3	4	4	4	4	3	3	3	3
f_2	2	2	2	2	3	3	2	2	2	2	2	2	4	4	2	2
f_1	1	1	2	1	1	1	3	1	1	1	2	1	1	1	4	1
f_0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

尽管在算法L中二进制数$k = (b_{n-1}\cdots b_0)_2$从未明显出现，但焦点指针$f_j$以一个巧妙的方式含蓄地表示它，使得如同我们应该做的那样，通过对二进位$a_{\rho(k)}$求补可以重复地形成$g(k) = (a_{n-1}\cdots a_0)_2$。我们说当$a_j$划下划线的时候它是*被动的*，否则它是*活动的*。于是焦点指针满足下列的不变量关系:

1) 如果a_j是被动的且a_{j-1}是活动的，则f_j是使得$a_{j'}$是活动的且有$j' > j$的最小下标。（二进制位a_n和a_{-1}由于此目的而被认为是活动的，尽管它们实际上不出现于本算法中。）

2) 否则$f_j = j$。

因此有递减下标的被动元素的一个块$a_{i-1}\cdots a_{j+1}a_j$的最右元素a_j有一个焦点f_j，它恰好指向这个块左边的元素a_i。所有其他元素a_j都有指向它们本身的f_j。

在这些术语中，在步骤L3中的头两个操作“$j \leftarrow f_0, f_0 \leftarrow 0$”等价于说“置$j$成为最右活动元素的下标，而且激活$a_j$右边的所有元素。”注意如果$f_0 = 0$，则操作$f_0 \leftarrow 0$是多余的，但它不造成任何损害。至于L3的其余两个操作“$f_j \leftarrow f_{j+1}, f_{j+1} \leftarrow j + 1$”等价于说“使$a_j$成为被动的”，因为我们知道在计算这个点时，$a_j$和$a_{j-1}$都是活动的。

10

（再次，操作$f_{j+1}\leftarrow j+1$可以是无害地多余的。）因此，活动和被动的纯效果等价于在二进制记号中的计数，如同在算法M中那样，以二进制位1表示被动和二进制位0表示活动。

算法L几乎是令人眼花缭乱地快，因为在对一个生成的n元组进行的每个访问之间它只做5个赋值操作和1个结束的检验。但我们甚至可做得更好些。为了看看它有多快，考虑在消遣的语言学中的一个应用。鲁道夫·卡斯汤（Rudolph Castown）说明 [见*Word Ways*（文字方式）1(1968), 165-169]，把sins的字母同fate的对应字母相互混合起来的16种方式中，产生出充分大英语词典才能找到的词：sine, sits, site等。而除了三个词（即fane, fite和sats）以外，其他词对于标准英语来说毫无疑问十分常见。因此很自然地对于5个字母的词来问类似的问题。当以所有32种可能的方式来交换对应位置时，5个字母的两个串在斯坦福图形库中将产生极大个数的词是哪些？

为了回答这个问题，我们无需考察全部$\binom{26}{2}^5=3\ 625\ 908\ 203\ 125$个实际上不同的字符串对；假定至少那些对之一至少产生17个词，就只要考察在图形库中所有$\binom{5757}{2}=16\ 568\ 646$对的词就够了，因为可以从两个5字母的串得到的17个或更多个字母的词的集合必须包含两个作为“对跖点”（两者恰好相反）（即无相同的对应字母）。对于每个对跖点对，我们要尽可能快地确定32个可能的子集交换是否能产生有意义的英文词的数量。

使用5个二进位表示每个字母，每5个字母可以表示为25个二进位的数，从“a”=00000到“z”=11001为止。一个2^{25}个二进制位或字节的表将快速地确定给定的五字母串是否为一个字。因此，通过混合两个给定词的字母，并在这个表中查阅这些模式，问题被简化为生成所得到的32个可能的词的二进位模式。对于每对25个二进位词w和w'，我们可进行如下：

W1. [检查差别。] 置$z\leftarrow w\oplus w'$，如果$((z-m)\oplus z\oplus m)\ \&\ m'\neq 0$，其中$m=2^{20}+2^{15}+2^{10}+2^5+1$和$m'=2^5m$，则就拒绝词对$(w,w')$。这个检验消除了在某些位置$w$和$w'$有相同字母的情况。（参见7.1-(oo)。结果是，16 568 646个词对中有10 614 085个没有这样公共的字母。）

W2. [形成个别的屏蔽。] 为准备下一步，置$m_0\leftarrow z\ \&\ (2^5-1)$, $m_1\leftarrow z\ \&\ (2^{10}-2^5)$, $m_2\leftarrow z\ \&\ (2^{15}-2^{10})$, $m_3\leftarrow z\ \&\ (2^{20}-2^{15})$和$m_4\leftarrow z\ \&\ (2^{25}-2^{20})$。

W3. [计算词数。] 置$l\leftarrow 1$, $A_0\leftarrow w$；变量l将计算迄今为止我们已找到的以w开始的词有多少，然后执行由下面定义的操作$swap(4)$。

W4. [打印一个设置记录的解。] 如果l超过或等于当前的极大值，则对于$0\leqslant j<l$打印A_j。 ∎

11 这一高速方法的核心是$swap(4)$操作序列，它应在线地（例如和一个宏处理器一起）加以扩充来消除所有不必要的开销。它借助于基本操作

$sw(j)$: 置$w\leftarrow w\oplus m_j$，则如果w是一个词，置$A_l\leftarrow w$和$l\leftarrow l+1$。

而被定义。

给定$sw(j)$，它翻转在位置j处的字母，我们定义

$$\begin{aligned} swap(0) &= sw(0) \\ swap(1) &= swap(0), sw(1), swap(0) \\ swap(2) &= swap(1), sw(2), swap(1) \\ swap(3) &= swap(2), sw(3), swap(2) \\ swap(4) &= swap(3), sw(4), swap(3) \end{aligned} \tag{22}$$

于是，$swap(4)$ 扩展成31步的序列$sw(0), sw(1), sw(0), sw(2), \cdots, sw(0) = sw(\rho(1)), sw(\rho(2)), \cdots, sw(\rho(31))$；这些步骤将被使用1000万次。通过把直尺函数$\rho(k)$ 的值直接嵌入到我们的程序中，而不是对于每个词对通过算法M，G或L重新计算它们，我们显然可获得速度。

获胜的词对生成21个词的一个集合，即

```
ducks  ducky  duces  dunes  dunks  dinks  dinky
dines  dices  dicey  dicky  dicks  picks  picky
pines  piney  pinky  pinks  punks  punky  pucks
```
(23)

例如，如果w = ducks且w' = piney，则m_0 = s ⊕ y，所以头一个操作$sw(0)$ 把ducks改变成ducky，它可看出是一个词。下一个操作$sw(1)$ 应用到m_1，它是在倒数第二个位置处的k ⊕ e，所以它产生非词ducey。$sw(0)$ 的另一个应用把ducey变成duces（这一般是词tecum所跟随的一个合法的词），等等。在至多几秒钟时间内通过这个方法能够处理所有词对。

进一步地流水线化也是可能的。例如，一旦我们已经找到产生出k个词的一对，只要它们生成$33 - k$个非词，我们就能拒绝后边的词对。但是我们已经讨论过的方法已经十分快了，因而它揭示了这样一个事实，即甚至无循环的算法L也可被打败。

当然，喜欢算法L的人们会抱怨说，我们仅仅在小的特殊情况$n = 5$下加速这个过程，而算法L对于一般的n解决生成的问题。然而，一个类似的思想对于$n > 5$的一般的值也有效：我们可以扩展出一个程序使得它和上述一样，快速地生成最右二进位$a_4\ a_3\ a_2\ a_1\ a_0$的所有32个设置。在每32步之后我们可以应用算法L，使用它来生成对于其他二进位$a_{n-1}\cdots a_5$的逐次的改变。这个方法所减少的由算法L完成的不必要的工作量大约是32的一个因子那么多。

其他的二进制格雷码。这个格雷二进制编码$g(0), g(1), \cdots, g(2^n - 1)$ 是遍历所有n个二进位的串，而在每一步当中仅改变一位的许多方法之一。我们说，一般地，在二进n元组上的一个“格雷循环”是任何这样的序列 $(v_0, v_1, \cdots, v_{2^n-1})$，它包括每一个$n$元组，而且有这样的性质，即$v_k$恰在一个二进位的位置上不同于$v_{(k+1) \bmod 2^n}$。因此，在图论的术语中，格雷循环是一个在$n$立体中的有向哈密顿循环。我们可以 12
假定，下标已被选定，使得$v_0 = 0\cdots0$。

如果我们把诸v想像成二进数，则有整数$\delta_0\cdots\delta_{2^n-1}$，使得对于$0 < k < 2^n$，

$$v_{(k+1)} \bmod 2^n = v_k \oplus 2^{\delta_k} \tag{24}$$

这所谓的δ序列是描述一个格雷循环的另一个方法。例如，当$n = 3$时，标准的格雷二进制码的δ序列是01020102，它实际上是(13)的直尺函数$\delta_k = \rho(k + 1)$，但最后值δ_{2^n-1}是$n - 1$而不是n，使得该循环封闭。个别的元素δ_k总处于范围$0 \leqslant \delta_k < n$中，因此把它们叫做“坐标”。

令$d(n)$为定义一个n个二进位的格雷循环的不同δ序列的个数，并令$c(n)$为“规范”δ序列的个数，其中每个坐标k出现在$k + 1$的头一个出现之前。于是$d(n) = n!c(n)$，因为在一个δ序列中坐标数的每一个排列显然地产生出另一个δ序列。对于$n \leqslant 3$，容易看出仅可能有的规范δ序列是

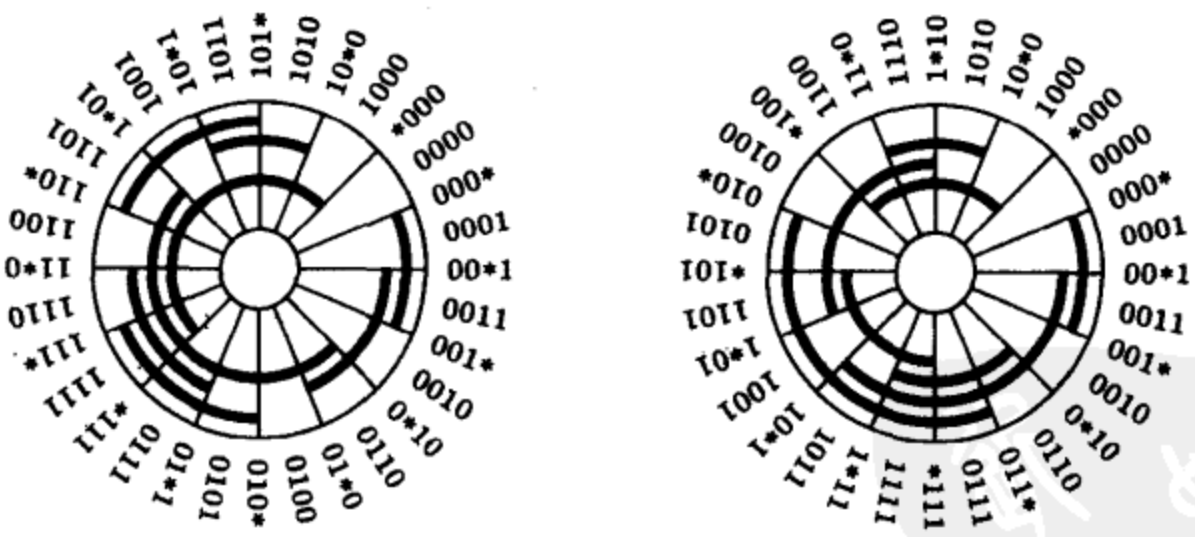
$$00;\ 0101;\ 01020102和01210121 \tag{25}$$

因此$c(1) = c(2) = 1, c(3) = 2;\ d(1) = 1, d(2) = 2$以及$d(3) = 12$。使用我们稍后将要研究的哈密顿循环的枚举技术，一个直截了当的计算机计算确立以下一些值：

$$\begin{aligned} &c(4) = 112; && d(4) = 2688; \\ &c(5) = 15\ 109\ 096; && d(5) = 1\ 813\ 091\ 520 \end{aligned} \tag{26}$$

没有明显的简单模式，而且这些数非常快速地增长（参见习题45）。因此一个相当保险的打赌是，无人将知道$c(8)$和$d(8)$的精确值。

由于可能性的个数是如此巨大，因此鼓励人们来寻找有另外的有用性质的格雷循环。例如，图13a示出一个4个二进位的格雷循环，其中每一个串$a_3\ a_2\ a_1\ a_0$都和它的补码$\bar{a}_3\bar{a}_2\bar{a}_1\bar{a}_0$在直径上相对立。每当二进位的个数是偶数时，这样的编码方案就是可能的（参见习题49）。



a) 互补的格雷码　　b) 平衡的格雷码

13

图 13

图13b所示是由杰·科·图特梯尔（G. C. Tootill）发现的甚至更为有趣的格雷循环 [*Proc. IEE* 103. Part B Supplement (1956), 435] 。这个循环在四个坐标的轨迹的每个当中，都有相同数目的改变。因此所有坐标同样地享有这些活动。以类似方式平衡的格雷码通过使用以下的灵活多样的方法，把n个二进位的循环推广到$n + 2$位的循环。事实上，可以对所有较大的n值来加以构造。

定理D 设$\alpha_1 j_1 \alpha_2 j_2 \cdots \alpha_l j_l$是$n$个二进位的格雷循环的一个$\delta$序列，其中每个$j_k$是一个坐标，每个$\alpha_k$是可能为空的一个坐标序列，而且$l$为奇数。则

$$\begin{aligned}&\alpha_1(n+1)a_1^R n\alpha_1\\ &\quad j_1\alpha_2 n\alpha_2^R(n+1)\alpha_2\ j_2\alpha_3(n+1)\alpha_3^R n\alpha_3\cdots j_{l-1}\alpha_l(n+1)\alpha_l^R n\alpha_l\\ &\qquad (n+1)\alpha_l^R j_{l-1}\alpha_{l-1}^R\cdots\alpha_2^R j_1\alpha_1^R n\end{aligned}\tag{27}$$

是一个（$n+2$）个二进位的格雷循环的δ序列。

例如，如果对于$n=3$，我们以序列01020102开始，并令三个划底元素为j_1, j_2, j_3，则对于一个5个二进位的循环，新序列(27)是

$$0141030102013102420104340102 0103\tag{28}$$

证明 设α_k有长度m_k，设v_{kt}是如果我们以$0\cdots0$开始并且应用坐标变化$\alpha_1 j_1\cdots\alpha_{k-1}j_{k-1}$和$\alpha_k$的头一个$t$，所达到的顶点。我们需要证明，对于$1\leqslant k\leqslant l$和$0\leqslant t<m_k$，当使用(27)时，所有顶点$00v_{kt}$, $01v_{kt}$, $10v_{kt}$以及$11v_{kt}$都出现。（最左坐标是$n+1$。）

由$000\cdots0=00v_{10}$开始，我们往下进行来得到顶点

$$00v_{11},\cdots,00v_{1m_1},10v_{1m_1},\cdots,10v_{10},11v_{10},\cdots,11v_{1m_1}$$

然后j_1产生$11v_{20}$，它由

$$11v_{21},\cdots,11v_{2m_2},10v_{2m_2},\cdots,10v_{20},00v_{20},\cdots,00v_{2m_2}$$

所跟随；然后是$00v_{30}$，等等，而最后我们终于达到$11v_{lm_l}$。光荣的最后者使用(27)的第3行来生成所有未出现的顶点$01v_{lm_l},\cdots,01v_{10}$并把我们带回到$000\cdots0$。 ∎

通过令c_j为$\delta_k=j$的次数，可定义一个δ序列的**转换计数**（$c_0,\cdots,c_{n-1}$）。例如，(28)有转换计数（12, 8, 4, 4, 4），而且它是从有转换计数（4, 2, 2）的一个序列引进的。如果我们仔细选择原来的δ序列，并且对于适当的元素j_k画下划线，我们可以得到尽可能地相等的转换计数。

推论B 对于所有$n\geqslant1$，有满足条件：对于$0\leqslant j<k<n$，

$$|c_j-c_k|\leqslant2\tag{29}$$

的转换计数（$c_0, c_1, \cdots, c_{n-1}$）的一个$n$个二进位的格雷循环。（这是最好可能的平衡条件，因为每个c_j必须是一个偶数，因此我们必须有$c_0+c_1+\cdots+c_{n-1}=2^n$。其实，当且仅当计数的$n-r$等于$2q$和$r$等于$2q+2$，其中$q=\lfloor 2^{n-1}/n\rfloor$和$r=2^{n-1}\bmod n$时，条件(29)成立。）

14

证明 对于一个n个二进位的格雷循环且其转换计数为（$c_0,\cdots,c_{n-1}$），给定一个δ序列，则通过从值$(c_0',\cdots,c_{n-1}',c_n',c_{n+1}')=(4c_0,\cdots,4c_{n-1},l+1,l+1)$开始、并且对于$1\leqslant k<l$，从$c_{jk}'$减去2，和从$c_{jl}'$减去4，可得到循环(27)的计数。例如，当$n=3$时，我们可以得到有转换计数$(8-2,16-10,8,6,6)=(6,6,8,6,6)$的一个平衡的5位的格雷循环——如果我们应用定理D到δ序列01210121的话。习题51给出对于

n的其他值的细节。 ▮

当我们考虑*运程长度*，即相同δ值的连续出现的距离时，出现了其中每个坐标轨迹有相同责任的另一个重要的n个二进位格雷循环类。标准格雷循环二进制编码在最低有效位置上有2的运程长度。当需要精确的测量时，这可能导致精度的损失[例如，参见乔·默·劳伦斯（G. M. Lawrence）和威·爱·麦克林托克（W. E. McClintock）的讨论，*Proc. SPIE* **2831**(1996)，104-111]。但是在其δ序列为

$$(0123042103210423)^2 \tag{30}$$

的著名的5个二进位的格雷循环中，所有运程都有4或以上的长度。

令$r(n)$是使得其中能找到所有运程长度$\geq r$的一个n个二进位的格雷循环的极大值r。显然$r(1)=1$，$r(2)=r(3)=r(4)=2$；而且容易看到当$n>2$时$r(n)$必定小于n，因此(30)证明$r(5)=4$。穷尽的计算机查找确定$r(6)=4$和$r(7)=5$。其实，一个相当直截了当的对于$n=7$的计算需要仅大约6000万个节点的一个树形来确定$r(7)<6$。而习题61(a)构造了没有短于5的运程的一个7个二进位的循环。对于$n\geq 8$，$r(n)$的精确值还未被求出，但是几乎肯定地，$r(10)$是8，而且我们还知道了有趣的构造，通过这些构造，我们可以证明当$n\to\infty$时$r(n)=n-O(\log n)$。（参见习题60–64。）

***二进制格雷通路**。我们已经把一个n位的格雷循环定义为把所有二进的n元组安排到一个序列$(v_0, v_1, \cdots, v_{2^n-1})$的方式，而且使它具有这样一个性质，即对于$0\leq k<2^n$的$n$立体中，$v_k$相邻于$v_{k+1}$，并且使得$v_{2^n-1}$相邻于$v_0$。这个循环的性质很好，但不总是必要的；而且有时没有它我们还可以做得更好。因此我们说一个n位的*格雷通路*，通常也叫做一个*格雷码*，是满足一个格雷循环的条件的任何序列，但最后的元素不必相邻于头一个元素。换言之，一个格雷循环是在n个立体顶点上的一个哈密顿循环。但一个格雷码只不过是在该图上的一个哈密顿通路。

最重要的非格雷循环的二进制格雷通路是在下列意义下n位*单调*的序列$(v_0, v_1, \cdots, v_{2^n-1})$，即对于$0\leq k<2^n-2$，

$$\nu(v_k)\leq \nu(v_{k+2}) \tag{31}$$

（这里，和在别处一样，我们使用ν来表示一个二进制串的“权”或“边路和”，即
15 在它当中1的个数。）反复试验表明，对于 $n\leq 4$实质上仅有两个单调的n位格雷编码，其一以0^n开始而另一个以$0^{n-1}1$开始。对于$n=3$，这两个是

$$000, 001, 011, 010, 110, 100, 101, 111 \tag{32}$$

$$001, 000, 010, 110, 100, 101, 111, 011 \tag{33}$$

对于$n=4$，这两个不那么明显，但实际上也不难发现。

由于每当v_k和v_{k+1}是相邻的时，$\nu(v_{k+1})=\nu(v_k)\pm 1$，因此我们显然不能加强(31)而来要求所有$n$元组都按权来严格排序。但是关系式(31)足够地强，只要给定k和v_0的权，就可确定每个v_k的权，因为可以确切地知道恰好有$\binom{n}{j}$个n元组的权是j。

图14通过作成经过256个所有可能的8位字节的模式的盛大的旅程，示出无限

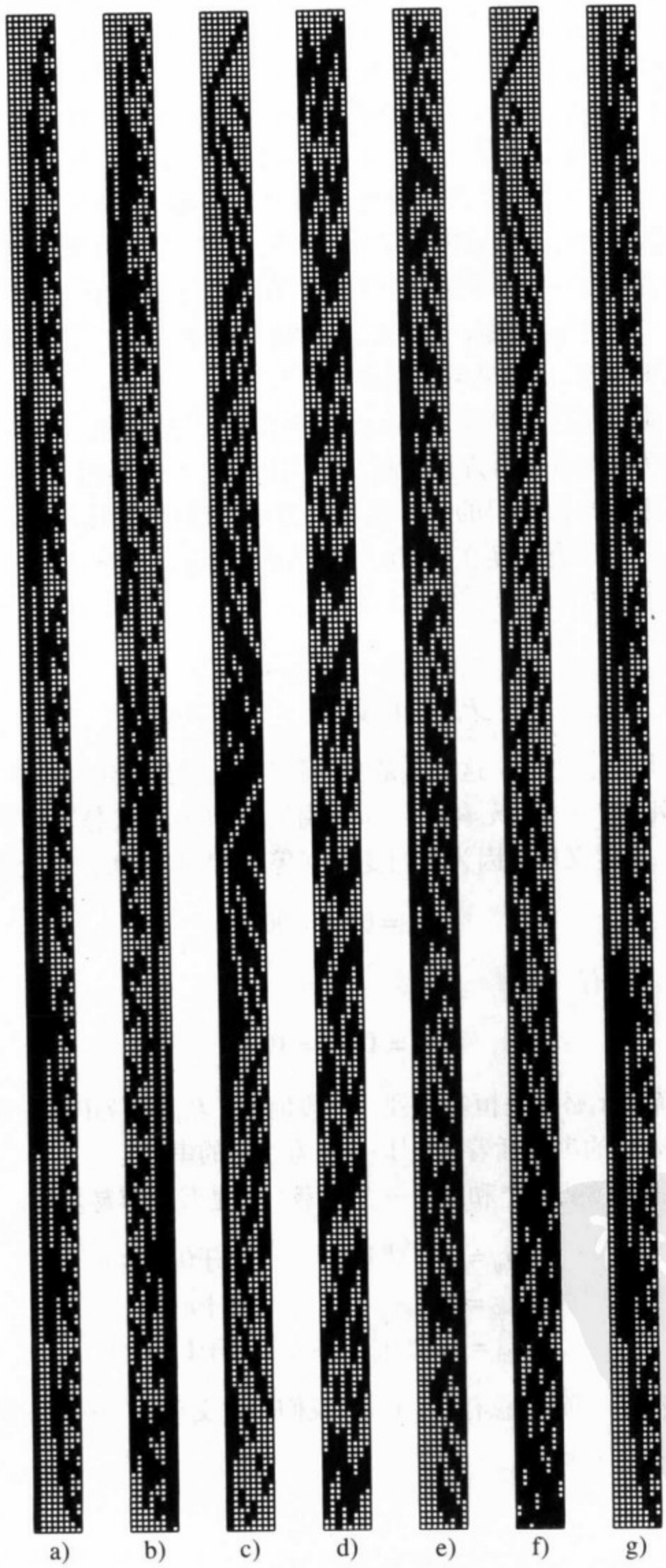


图14 8位格雷码示例：a) 标准的；b) 平衡的；c) 互补的；d) 长运程的；e) 非局部的；f) 单调的；g) 无趋势的

大数目的格雷编码中的7个。黑方格表示1，白方格表示0。图14a是标准的格雷二进制编码，而图14b是平衡的格雷编码，而且在每个坐标位置中恰有256/8＝32个转换。图14c是类似于图13a的一个格雷编码，其中下边的128个编码是顶上的128个编码的反码。在图14d中，在每个坐标位置中的转换决不出现比5步更近些；换言之，所有运程长度至少是5。在图14e中的循环在习题59的意义下是*非局部的*。图14f示出对于$n = 8$为单调的一个通路；注意在靠近底部处它变得多黑。最后，图14g示出一个完全非单调的格雷编码。可见黑方格的重心恰好位于每列的半途的点处。标准的格雷二进制编码在7个位置中有这个性质，但是图14g在所有8个位置中实现完美的黑白权的平衡，这样的码叫做*无趋势的码*，它们在农业的设计和其他实验中是很重要的。（参见习题75和习题76。）

卡尔拉·萨维吉（Carla Savage）和彼得·温格尔（Peter Winkler）[*J. Combinatorial Theory*（组合理论学报）A70(1995), 230-248] 发现了对于所有$n>0$，构造单调的二进制格雷编码的一个优雅的方法。这样的通路有必要从子通路P_{nj}构造出来，在P_{nj}中，所有转换在权j和$j+1$的n元组之间进行，萨维吉和温格尔通过令$P_{10}=0,1$和对于所有$n>0$，令

$$P_{(n+1)j} = 1P^{\pi_n}_{n(j-1)}, 0P_{nj} \tag{34}$$

$$P_{nj} = \varnothing \quad \text{如果} j<0 \text{或} j \geqslant n \tag{35}$$

递归地定义适当的子通路。这里π_n是我们稍后将予描述的坐标的一个排列，而记号P^{π}指的是序列P的每一个元素$a_{n-1}\cdots a_1a_0$由$b_{n-1}\cdots b_1b_0$所代替，其中$b_{j\pi} = a_j$。（我们不通过令$b_j = a_{j\pi}$来定义P^{π}，因为我们要$(2^j)^{\pi}$等于$2^{j\pi}$。）例如，由此得出

$$P_{20} = 0P_{10} = 00, 01 \tag{36}$$

16 ~ 17

因为$P_{1(-1)}$为空；还有

$$P_{21} = 1P^{\pi_1}_{10} = 10, 11 \tag{37}$$

因为P_{11}为空，而且π_1必定是恒等排列。一般地说，P_{nj}是n位串的一序列，该串恰好包含$\binom{n-1}{j}$个权为j的串穿插着$\binom{n-1}{j}$个权为$j+1$的串。

令α_{nj}和ω_{nj}是P_{nj}的头一个和最后一个元素，于是我们容易发现：

$$\omega_{nj} = 0^{n-j-1}\,1^{j+1}, \qquad \text{对于} 0 \leqslant j < n \tag{38}$$

$$\alpha_{n0} = 0^n, \qquad \text{对于} n>0 \tag{39}$$

$$\alpha_{nj} = 1\alpha^{\pi_{n-1}}_{(n-1)(j-1)}, \qquad \text{对于} 1 \leqslant j < n \tag{40}$$

特别是，α_{nj}总有权j，而ω_{nj}总有权$j+1$。我们将定义$\{0, 1, \cdots, n-1\}$的排列π_n，使得对于$n = 1, 2, 3, \cdots$，序列

$$P_{n0}, P^R_{n1}, P_{n2}, P^R_{n3}, \cdots \tag{41}$$

$$P^R_{n0}, P_{n1}, P^R_{n2}, P_{n3}, \cdots \tag{42}$$

都是单调的格雷通路。事实上，单调性是显然的；因此有疑问的仅仅是格雷性；

而且序列(41)和(42)很好地链接，因为不管排列π_n如何，相邻性

$$\alpha_{n0}——\alpha_{n1}——\cdots——\alpha_{n(n-1)}, \qquad \omega_{n0}——\omega_{n1}——\cdots——\omega_{n(n-1)} \tag{43}$$

立即从(34)得出。因此，关键点在于在式(34)中逗号处的转换，它使$P_{(n+1)j}$成为一个格雷通路，当且仅当对于$0<j<n$，

$$\omega_{n(j-1)}^{\pi_n} = \alpha_{nj} \tag{44}$$

例如，当$n=2$和$j=1$时，由(38)~(40)，我们需要$(01)^{\pi_2}=\alpha_{21}=10$，因此$\pi_2$必定把坐标0和1转置。一般公式（参见习题71）结果是

$$\pi_n = \sigma_n \pi_{n-1}^2 \tag{45}$$

其中σ_n是n循环$(n-1\cdots10)$。因此最初的一些情况是

$$\pi_1=(0), \qquad \pi_4=(03), \qquad \pi_2=(01), \qquad \pi_5=(04321), \qquad \pi_3=(021), \qquad \pi_6=(052413)$$

对于魔术排列π_n，不存在简单的“封闭形式”是显然的。习题73证明萨维吉-温格尔编码可以有效地生成。

非二进制格雷编码。我们已经很详细地研究了二进制n元组的情况，因为它是最简单的、最典型的、最可应用的，也是这一课题中探索最透彻的部分。当然它还有许多其他的应用，其中，如同在算法M中一样，我们要来生成具有更一般范围$0\le a_j<m_j$的坐标的$(a_1,\cdots,a_n)$。格雷码也非常适用于这个情况。

例如，考虑十进制数字，其中对于每个j，我们要$0\le a_j<10$。是否也有可解释为类似于格雷二进制编码的十进制方式，即一次只有一个数字在改变？回答是肯定的。事实上，有两种自然的方案可以利用。第一种情况，称为*反射的格雷十进制编码*，带有3个数字的串可计算到一千，这个序列有以下形式 18

$$000,001,\cdots,009,019,018,\cdots,011,010,$$
$$020,021,\cdots,091,090,190,191,\cdots,900$$

其中每个坐标交替地从0变化成9，而后倒过来从9变成0。第二种，叫做*取模的格雷十进编码*，数字总是加1 mod 10，因此它们从9到0“包在一起”：

$$000,001,\cdots,009,019,010,\cdots,017,018,$$
$$028,029,\cdots,099,090,190,191,\cdots,900$$

在两种情况下，在第k步上改变的数字由十进制直尺函数$\rho_{10}(k)$来确定，即整除k的10的最大次幂。因此数字的每个n元组恰出现一次：对于$1\le j<n$，在对任何其他数字进行改变之前，我们生成10^j个最右j个数字的不同设置。

一般地说，在任何混合进制系统中的反射的格雷编码可以看作是非负整数的一个排列，即把一个通常的混合进制的数

$$k = \begin{bmatrix} b_{n-1}, \cdots, b_1, b_0 \\ m_{n-1}, \cdots, m_1, m_0 \end{bmatrix} = b_{n-1}m_{n-2}\cdots m_1 m_0 + \cdots + b_1 m_0 + b_0 \tag{46}$$

映射成它的反射的格雷等价值

$$\hat{g}(k) = \begin{bmatrix} a_{n-1}, \cdots, a_1, a_0 \\ m_{n-1}, \cdots, m_1, m_0 \end{bmatrix} = a_{n-1}m_{n-2}\cdots m_1 m_0 + \cdots + a_1 m_0 + a_0 \tag{47}$$

就如同等式 (7) 在二进制数的特殊情况中来做这件事一样。令

$$A_j = \begin{bmatrix} a_{n-1}, \cdots, a_j \\ m_{n-1}, \cdots, m_j \end{bmatrix}, \qquad B_j = \begin{bmatrix} b_{n-1}, \cdots, b_j \\ m_{n-1}, \cdots, m_j \end{bmatrix} \tag{48}$$

且$A_n = B_n = 0$，使得当$0 \leqslant j < n$时，我们有

$$A_j = m_j A_{j+1} + a_j 和 B_j = m_j B_{j+1} + b_j \tag{49}$$

通过对于$n - j$施行归纳法，不难导出联系诸a和诸b的规则:

$$a_j = \begin{cases} b_j, & 如果B_{j+1}为偶 \\ m_j - 1 - b_j, & 如果B_{j+1}为奇 \end{cases} \tag{50}$$

(这里我们从右至左地对n元组 $(a_{n-1}, \cdots, a_1, a_0)$ 和 $(b_{n-1}, \cdots, b_1, b_0)$ 的坐标编号，以便同等式 (7) 和在等式4.1-(9)中混合进制记号的约定相一致。更喜欢像 $(a_1, \cdots, a_n)$ 这样的记号的读者，如果愿意，可以在所有公式中把j改成$n - j$。) 从另一方面，我们有

$$b_j = \begin{cases} a_j, & 如果a_{j+1} + a_{j+2} + \cdots 为偶 \\ m_j - 1 - a_j, & 如果a_{j+1} + a_{j+2} + \cdots 为奇 \end{cases} \tag{51}$$

奇怪的是，当所有进制m_j为奇数时，规则(50)和它在(51)中的逆完全相同。例如，当$m_0 = m_1 = \cdots = 3$时，在格雷二进制编码中，我们有 $\hat{g}((10010211012)_3) =$ $(12210211010)_3$还有 $\hat{g}((12210211010)_3) = (10010211012)_3$。习题78证明(50)和(51)，并且讨论在取模情况下成立的类似的公式。

19

事实上，我们能无循环地生成这样的格雷序列，并推广算法M和L。

算法H（*无循环的反射混合进制格雷编码的生成*） 这个算法访问所有n元组 $(a_{n-1}, \cdots, a_0)$ 使得对于$0 \leqslant j < n$，有$0 \leqslant a_j < m_j$，而且在每一步中仅有一个坐标发生 ± 1的改变。它维持焦点指针数组 $(f_n, \cdots, f_0)$ 来控制如同在算法L中的那些动作，连同一个方向数组 $(o_{n-1}, \cdots, o_0)$ 一起。我们假定每个进制$m_j \geqslant 2$。

H1. [初始化。] 对于$0 \leqslant j < n$，置$a_j \leftarrow 0, f_j \leftarrow j$，以及$o_j \leftarrow 1$，还置$f_n \leftarrow n$。

H2. [访问。] 访问n元组 $(a_{n-1}, \cdots, a_1, a_0)$。

H3. [选择j。] 置 $j \leftarrow f_0$和$f_0 \leftarrow 0$。(如同在算法L中那样，j是最右边的活动坐标，它右边的所有元素现在都已被重新激活。)

H4. [改变坐标j。] 如果$j = n$，则终止；否则$a_j \leftarrow a_j + o_j$。

H5. [反射？] 如果$a_j = 0$或$a_j = m_j - 1$，则置$o_j \leftarrow -o_j$, $f_j \leftarrow f_{j+1}$, 而且$f_{j+1} \leftarrow j + 1$。（这样坐标j已变成被动的。）返回H2。 ∎

一个类似的算法生成取模的格雷方案（参见习题77）。

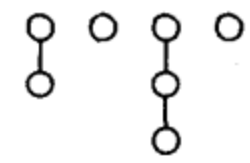
***子森林**。由黄田保宪（Y. Koda）和弗·拉斯基（F. Ruskey）发现的算法H的一个有趣的和有教益的推广[*J. Algorithms* 15(1993), 324-340]，进一步发展了格雷编码和无循环生成这一课题。假设有n个节点的一个森林，而我们要访问所有它的“主要的子森林”，即节点S的所有这样的子集，即使得如果x在S中，而且x不是一个根，则x的双亲也在S中。例如，7个节点的森林 有33个这样的子集合，对应于下列33个框图中的黑节点。



(52)

注意，如果我们从左到右地读该顶部一行，从右到左地读中间一行，和从左到右地读底下一行，则恰好有一个节点的状态在每步中改变。

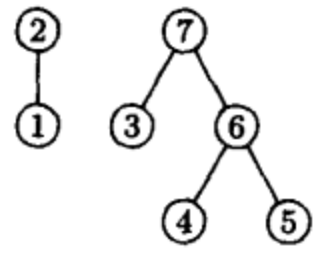
如果给定的森林由退化的非分支的树组成，则主要的子森林等价于混合进制数。例如，像



这样一个森林有$3 \times 2 \times 4 \times 2$个主子森林，对应于4元组$(x_1, x_2, x_3, x_4)$，使得$0 \leqslant x_1 < 3, 0 \leqslant x_2 < 2, 0 \leqslant x_3 < 4$及$0 \leqslant x_4 < 2$；$x_j$的值是在第$j$个树中选择的节点数。当把黄田保宪和拉斯基的算法应用到这样一个森林时，它将以像在进制$(3, 2, 4, 2)$上的反射格雷编码那样的顺序来访问这些子森林。 20

算法K（*无循环反射子森林的生成*） 给定一个森林，当以后根顺序安排时，其节点为$(1, \cdots, n)$，这个算法访问所有二进制的n元组$(a_1, \cdots, a_n)$，使得每当p是q的一个双亲时$a_p \geqslant a_q$。（因此，$a_p = 1$意味着p是在当前子森林中的一个节点。）在一个访问与下一个访问之间恰有一个二进位a_j改变。类似于算法L的焦点指针$(f_0, f_1, \cdots, f_n)$和另外的指针数组$(l_0, l_1, \cdots, l_n)$及$(r_0, r_1, \cdots, r_n)$一起使用，后两者表示称为“当前边缘”的双重链接表。当前边缘包含当前子森林以及它们的子女的所有节点；r_0指向它的最左节点，而l_0指向它的最右节点。

一个辅助的数组$(c_0, c_1, \cdots, c_n)$定义森林如下：如果p没有子女，则$c_p = 0$；否则c_p是p的最左（最小）的子女。而且c_0是森林本身的最左的根。当算法开始时，每当p和q是同一个家族的连续的子女时，我们假定$r_p = q$和$l_q = p$。例如，因此(52)中的森林有后根顺序的编号



因此我们将有 $(c_0, \cdots, c_7) = (2, 0, 1, 0, 0, 0, 4, 3)$ 以及在此情况下，在步骤K1开始时 $r_2 = 7, l_7 = 2, r_3 = 6, l_6 = 3, r_4 = 5$以及$l_5 = 4$。

K1. [初始化。] 对于$1 \leqslant j \leqslant n$置$a_j \leftarrow 0$和$f_j \leftarrow j$，因此使初始化的子森林为空而且所有节点为活动的。置$f_0 \leftarrow 0, l_0 \leftarrow n, r_n \leftarrow 0, r_0 \leftarrow c_0$以及$l_{c_0} \leftarrow 0$，因此把所有根放进当前的边缘中。

K2. [访问。] 访问由 $(a_1, \cdots, a_n)$ 所定义的子森林。

K3. [选择p。] 置$q \leftarrow l_0, p \leftarrow f_q$。（现在$p$是边缘最右的活动节点。）还置$f_q \leftarrow q$（由此把$p$右边的所有节点都激活）。

K4. [校验a_p。] 如果$p = 0$，则终止此算法，否则如果$a_p = 1$则转到K6。

K5. [插入p的子女。] 置$a_p \leftarrow 1$。然后如果$c_p \neq 0$，则置$q \leftarrow r_p, l_q \leftarrow p-1, r_{p-1} \leftarrow q, r_p \leftarrow c_p, l_{c_p} \leftarrow p$（由此把$p$的子女放到边缘中$p$的右边）。转到K7。

K6. [删去p的子女。] 置$a_p \leftarrow 0$。然后，如果$c_p \neq 0$，则置$q \leftarrow r_{p-1}, r_p \leftarrow q, l_q \leftarrow p$（由此从边缘中删除$p$的子女）。

K7. [使p成为被动的。]（此时我们知道p是活动的。）置$f_p \leftarrow f_{l_p}$和$f_{l_p} \leftarrow l_p$，返回到K2。 ∎

为了理解恰好在正确的时刻边缘增长和收缩的优美机制，鼓励读者对于像(52) 21 这样的例子来运行本算法。

***移位寄存器序列**。生成m进制数字的所有n元组还有可能有一个完全不同的方法：我们可以一次生成一位数字，并且重复地对n个最近生成的数字进行工作，由此通过移动右边的一个适当的新数字，就从一个n元组 $(x_0, x_1, \cdots, x_{n-1})$ 过渡到另一个 $(x_1, \cdots, x_{n-1}, x_n)$。例如，图15示出，在长度为32的一个确定的循环模式下，如何得到5个连续二进位的模块的所有5个二进位的数。这个常规思想已在2.3.4.2节和3.2.2节的某些习题中讨论过，现在我们已作好准备来进一步剖析它。

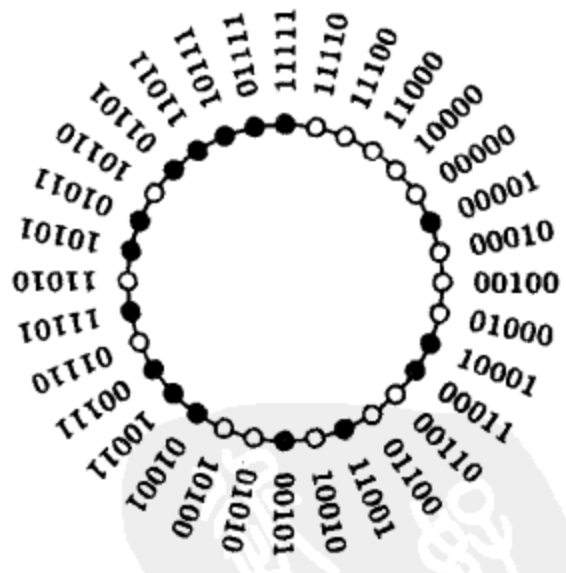


图15 5个二进位的德·布鲁因循环

算法S（普通的移位寄存器生成） 假定在步骤S3中使用一个适当的函数f，本算法访问所有n元组 $(a_1, \cdots, a_n)$，使得对于$1 \leqslant j \leqslant n$，有$0 \leqslant a_j < m$。

S1. [初始化。] 对于$-n < j \leqslant 0$，置$a_j \leftarrow 0$且$k \leftarrow 1$。

S2. [访问。] 访问n元组 $(a_{k-n}, \cdots, a_{k-1})$。如果$k = m^n$，则结束。

S3. [前进。] 置$a_k \leftarrow f(a_{k-n}, \cdots, a_{k-1})$，$k \leftarrow k+1$，并返回S2。 ∎

使算法S有效的每个函数f对应于m^n个m进制数字的一个循环，使得n个数字的每个组合在循环中都连续地出现。例如，图15中所示的情况$m = 2$和$n = 5$对应于二进制循环

$$00000100011001010011101011011111 \tag{53}$$

而且无穷序列的头m^2个数字

$$00110212203132330414243 44\cdots \tag{54}$$

生成对于$n = 2$和任意的m的一个适当循环。这样的循环普通称为*m进制的德·布鲁因循环*，因为尼·戈·德·布鲁因（N. G. de Bruijn）在[*Indagationes Mathematicæ* 8 (1946), 461-467] 中讨论了对于任意的n二进制的情况。

习题2.3.4.2-23证明恰有$m!^{m^{n-1}}/m^n$个函数f有所要求的性质。这是一个巨大的数字，但这些函数中仅有少数几个是有效地可计算的。我们将讨论三种f，它们看起来是最有用的。 22

当m是一个素数，而且f是几乎线性递归时，出现头一个重要情况，

$$f(x_1,\cdots,x_n)=\begin{cases}c_1, \text{如果 } (x_1,x_2,\cdots,x_n)=(0,0,\cdots,0)\\ 0, \text{ 如果 } (x_1,x_2,\cdots,x_n)=(1,0,\cdots,0)\\ (c_1x_1+c_2x_2+\cdots+c_nx_n)\bmod m, \text{否则}\end{cases} \tag{55}$$

这里系数$(c_1,\cdots,c_n)$必须使得在等式3.2.2-(9)后边讨论的意义下

$$x^n-c_nx^{n-1}-\cdots-c_2x-c_1 \tag{56}$$

是modulo m本原多项式。这样的多项式的个数是$\varphi(m^n-1)/n$，这足够大以允许我们找出其中仅有少数c为非零的一个来。[这个构造可追溯到威廉·曼特尔（Willem Mantel）的先驱性论文，*Nieuw Archief voor Wiskunde* (2) 1 (1897), 172-184。]

例如，假设$m = 2$，我们可以以一个非常简单的无循环过程来生成二进制n元组：

算法A（*几乎线性的二进位移位生成*） 如同在表1中所见，通过使用一个特殊的偏离s[情况1]或两个特殊的偏离s和t [情况2]，本算法访问所有n个二进位的向量。

A1. [初始化。] 置$(x_0, x_1, \cdots, x_{n-1})\leftarrow(1, 0, \cdots, 0)$和$k\leftarrow0$, $j\leftarrow s$。在情况2中，也置$i\leftarrow t$和$h\leftarrow s + t$。

A2. [访问。] 访问n元组$(x_{k-1}, \cdots, x_0, x_{n-1}, \cdots, x_{k+1}, x_k)$。

A3. [测试是否结束。] 如果$x_k\neq0$，则置$r\leftarrow0$；否则置$r\leftarrow r + 1$，而且如果$r = n - 1$则转到A6。

A4. [移位。] 置$k\leftarrow(k - 1)\bmod n$和$j\leftarrow(j - 1)\bmod n$。在情况2中，也置$i\leftarrow(i - 1)\bmod n$和$h\leftarrow(h - 1)\bmod n$。

A5. [计算一个新的二进位。] 置$x_k\leftarrow x_k\oplus x_j$[情况1]或$x_k\leftarrow x_k\oplus x_j\oplus x_i\oplus x_h$ [情况

2]。返回A2。

A6. [结束。] 访问 $(0, \cdots, 0)$ 并终止。 ∎

表1 算法A的参数

3:1	8:1, 5	13:1, 3	18:7	23:5	28:3
4:1	9:4	14:1, 11	19:1, 5	24:1, 3	29:2
5:2	10:3	15:1	20:3	25:3	30:1, 15
6:1	11:2	16:2, 3	21:2	26:1, 7	31:3
7:1	12:3.4	17:3	22:1, 7	27:1, 7	32:1, 27

上述条目“n:s”或“n:s, t”指的是多项式$x^n + x^s + 1$或$x^n + (x^s + 1)(x^t + 1)$是modulo 2本原的。韦·斯坦克（W. Stahnke）已经对高达$n = 168$的附加的值造出表来，见*Math. Comp.* 27(1973), 977-980。

对于所有的n，几乎肯定地存在适当的偏离参数s和可能还有的t，因为本原多项式极为繁多；例如，当$n = 32$时，(s, t)的8个不同选择都将有效，而表1仅仅列举了最小的。然而，在所有情况下，存在性的严格证明超出了数学知识的当前状态。

23

在(55)中，我们关于德·布鲁因循环的头一个构造是代数的，其正确性依赖于有限域理论。当m不是质数时，有效的类似方法可参见习题3.2.2-21。反之，我们的下一个构造，将纯粹是组合学的。事实上，它强烈地同取模的格雷m进制编码有关。

算法R（*递归的德·布鲁因循环的生成*） 假设$f()$是一个共行程序，当它被重复地调用时，由输出n个0开始，它将输出长度为m^n的一个m进制的德·布鲁因循环的逐个的数字。假定$n \geqslant 2$，这个算法是输出长度为$m^n + 1$的一个循环的类似的共行程序。它维持三个私有变量x，y和t；变量x开始时应为零。

R1. [输出。] 输出x。如果$x \neq 0$且$t \geqslant n$，则转到R3。

R2. [调用f。] 置$y \leftarrow f()$。

R3. [计1。] 如果$y = 1$则置$t \leftarrow t + 1$；否则置$t \leftarrow 0$。

R4. [跳1?] 如果$t = n$并且$x \neq 0$，则转回到R2。

R5. [调整x。] 置$x \leftarrow (x + y) \bmod m$，并返回R1。 ∎

例如，令$m = 3$和$n = 2$，如果$f()$产生无穷的9循环

$$001102122 \quad 001102122 \; 0\cdots \tag{57}$$

则算法R在步骤R1将产生以下的无穷27循环:

$$\begin{aligned} y &= 001021220011110212200102122\ 001\cdots \\ t &= 001001000012340010000100100\ 001\cdots \\ x &= 000110102220120020211122121\ 0001\cdots \end{aligned}$$

关于算法R正确地工作的证明是有趣和有教益的（参见习题93）。下一个算法把窗口大小n加倍，其正确性证明甚至更有趣和更有教益（参见习题95）。

算法D（加倍的递归德·布鲁因循环生成） 假设$f()$和$f'()$都是共行程序，以n个零开始，当重复地调用它们时，它们每一个都将输出长度为m^n的一个m进制德·布鲁因循环的逐次的数字。（两个循环是相同的，但它们必须由独立的共行程序生成，因为我们将以不同的速度消耗它们的值。）这个算法是一个类似的共行程序，它将输出长度为m^{2n}的一个循环。它维持6个私有变量x，y，t，x'，y'和t'；变量x和x'开始时应都为m。

特殊的参数r必须被置为一个常数值，使得

$$0 \leqslant r \leqslant m \text{和} \gcd(m^n - r, m^n + r) = 2 \tag{58}$$

当m为奇数时，最好的选择通常是$r = 1$，而当m为偶数时，$r = 2$。

D1. [可能调用f。] 如果$t \neq n$，或$x \geqslant r$，则置$y \leftarrow f()$。

D2. [计算重复。] 如果$x \neq y$，则置$x \leftarrow y$和$t \leftarrow 1$，否则置$t \leftarrow t + 1$。

D3. [从f输出。] 输出x的当前值。 24

D4. [调用f'。] 置$y' \leftarrow f'()$。

D5. [计算重复。] 如果$x' \neq y'$，则置$x' \leftarrow y'$以及$t' \leftarrow 1$，否则置$t' \leftarrow t' + 1$。

D6. [可能拒绝f'。] 如果$t' = n$和$x' < r$，以及$t < n$或$x' < x$，则转到D4。如果$t' = n$和$x' < r$且$x' = x$，则转到D3。

D7. [从f'输出。] 输出x'的当前值。如果$t' = n$和$x' < r$，则返回D3。否则返回D1。 ▮

算法D的基本思想是交替地从$f()$和$f'()$作输出，并且当两个序列的任一个对于$x < r$，生成n个连续的x时，进行特殊的调整。例如，当$f()$和$f'()$产生9循环的(57)时，我们取$r = 1$并得到

在步骤D2中的t: 12 31211112 12312111 12123121 11121231 21111212…

在步骤D3中的x: 00001102122 00011021 22000110 21220001 102122000…

在步骤D6中的t': 12121111212121111212121111212121111212121111212121…

在步骤D7中的x': 0 11021220 11021220 11021220 11021220 11021220 1…

使得在步骤D3和D7中产生的81循环是**00001011012…222200001…**。

算法R的$m = 2$的情况是由阿布拉罕·廉培尔（Abraham Lempel）发现的[*IEEE Trans.* C-19(1970), 1204-1209]。过了25年之后，算法D才被发现[克·约·米切尔（C. J. Mitchell）、图·埃特津（T. Etzion）和肯·格·佩特森（K. G. Paterson）, *IEEE Trans.* IT-42(1996), 1472-1478]。通过一起使用它们两个，并且基于(54)，由对于$n = 2$的简单共行程序开始，我们可以构造相互协作的共行程序的有趣的族，对于每位数字的输出，仅仅使用$O(\log n)$个简单的计算，对任何所希望的$m \geqslant 2$和$n \geqslant 2$，它们将生成长度为m^n的一个德·布鲁因循环。（参见习题96。）而且，在最简单的$m = 2$的情况下，这个组合的“R和D方法”有这样一个性质，即通过对于n个二进位的数做$O(n \log n)$次简单的计算，它的第k个输出作为k的一个函数可以直接计算出来。反之，给定任何n个二进位的模式β，β在循环中的位置也可$O(n \log n)$步之内计算出来。（参见习题97～99。）现在还不知道有无其他的二进制的德·布

鲁因循环有后边所述性质。

我们对于德·布鲁因循环的第三个构造是以素串理论为基础的。当我们研究第9章中的模式匹配时，此理论对于我们将极为重要。假设$\gamma = \alpha\beta$是两个串的连接；我们说α是γ的一个前缀，而β是一个后缀。γ的前缀或后缀说是真的，如果它的长度为正但小于γ的长度。因此，β是$\alpha\beta$的一个真后缀当且仅当$\alpha \neq \varepsilon$且$\beta \neq \varepsilon$。

定义P 一个串为素的，如果它非空而且（按词典顺序）小于所有它的真后缀。

例如，01101不是素的，因为它大于01；但01102是素的，因为它小于1102，102，02和2。（我们假定串是由字母、数字或来自一个线性有序的字符表的其他符号组成，词典顺序是比较串的通常方法。因此当按词典顺序α小于β时，我们写
25 $\alpha < \beta$并说α小于β。特别是，我们总有$\alpha \leqslant \alpha\beta$，以及$\alpha < \alpha\beta$当且仅当$\beta \neq \varepsilon$时。）

素串通常称为林顿字，因为它们是由罗·科·林顿（R. C. Lyndon）引入的[*Trans. Amer. Math. Soc.* 77(1954), 202-215]。林顿称它们为“标准序列”。基于后面习题101中的基本因式分解定理，使用术语“素数”（prime）来描述素串很恰当，也很简单。然而，我们通常通过使用λ来表示素串，继续尊重林顿的含蓄性。

陈国才、福克（Fox）和林顿在关于群论的一篇重要论文中推导了关于素串的许多最重要的性质[*Annals of Math.* 68(1958), 81-95]，其中包括以下容易的但很基本的结果。

定理P 小于其所有循环移位的一个非空串是素串。（$(a_1 \cdots a_n)$ 的循环移位是 $a_2 \cdots a_n a_1$，$a_3 \cdots a_n a_1 a_2$，$\cdots$，$a_n a_1 \cdots a_{n-1}$。）

证明 假设$\gamma = \alpha\beta$不是素串，因为$\alpha \neq \varepsilon$和$\gamma \geqslant \beta \neq \varepsilon$。但假设$\gamma$也小于它的循环移位$\beta\alpha$，则条件$\beta \leqslant \gamma < \beta\alpha$意味着对于某个串$\theta < \alpha$，有$\gamma = \beta\theta$。因此如果$\gamma$也小于它的循环移位$\theta\beta$，则我们有$\theta < \alpha < \alpha\beta < \theta\beta$。但这是不可能的，因为$\alpha$和$\theta$有相同长度。 ∎

设$L_m(n)$ 是长度为n的m进制素串的个数。每个串$a_1 \cdots a_n$，连同它的循环移位，对于n的某个因子d，产生d个不同的串，它并且恰好对应于长度为d的一个素串。例如，从010010通过循环移位，也得到100100和001001，而且周期部分{010, 100, 001}的最小者是素串001，因此我们必定有，对于所有$m, n \geqslant 1$，

$$\sum_{d \backslash n} d\, L_m(d) = m^n \tag{59}$$

使用习题4.5.3-28(*a*) 可以对$L_m(n)$ 求解这个方程族，而且我们得到

$$L_m(n) = \frac{1}{n} \sum_{d \backslash n} \mu(d) m^{n/d} \tag{60}$$

在20世纪70年代，哈罗德·弗雷德里克森（Harold Fredricksen）和詹姆斯·梅奥兰纳（James Maiorana）发现了以递增顺序生成长度为n或更小的所有m进制素串的一个漂亮而简单的方法[*Discrete Math.* 23(1978), 207-210]。在我们准备好来理解他们的算法之前，我们需要来考虑一个非空串λ的n-扩充，即无穷串$\lambda\lambda\lambda\cdots$

的头n个字符。例如123的10扩充是1231231231。一般地说，如果$|\lambda| = k$，它的n扩充是$\lambda^{\lfloor n/k \rfloor}\lambda'$，其中$\lambda'$是长度为$n \bmod k$的$\lambda$的前缀。

定义Q 在某个字符集上，一个串是预素的如果它是一个素串的非空前缀。

定理Q 一个长度$n>0$的串为预素的当且仅当它是长度$k \leq n$的一个素串λ的n扩充。这个素串是惟一确定的。

证明 参见习题105。 ∎ 26

定理Q实际上指出，在长度$\leq n$的素串和长度为n的预素串之间有一个一一对应。下列算法以递增顺序生成所有m进制的实例。

算法F（素串和预素串的生成） 这个算法访问所有m进制的n元组$(a_1, \cdots, a_n)$使得串$a_1 \cdots a_n$是预素串。它也标识下标j使得$a_1 \cdots a_n$是素串$a_1 \cdots a_j$的n扩充。

F1. [初始化。] 置$a_1 \leftarrow \cdots \leftarrow a_n \leftarrow 0$及$j \leftarrow 1$；也置$a_0 \leftarrow -1$。

F2. [访问。] 以下标j访问$(a_1, \cdots, a_n)$。

F3. [准备增加。] 置$j \leftarrow n$。然后如果$a_j = m-1$，则减小j直到找到$a_j < m-1$为止。

F4. [加1。] 如果$j = 0$则终止。否则置$a_j \leftarrow a_j + 1$。（现在由习题105(a)，$a_1 \cdots a_j$为素串。）

F5. [作n-扩充。] 对于$k \leftarrow j+1, \cdots, n$（以此顺序），置$a_k \leftarrow a_{k-j}$，返回F2。 ∎

例如，当$m = 3$和$n = 4$时，算法F访问32个三进制的预素串：

$$\begin{array}{cccccccc} 0\hat{0}00 & 0\hat{0}11 & 0\hat{0}22 & 0\hat{1}11 & 0\hat{1}22 & 0\hat{2}12 & 1\hat{1}11 & 1\hat{2}12 \\ 0\hat{0}01 & 0\hat{0}12 & 0\hat{1}01 & 0\hat{1}12 & 0\hat{2}02 & 0\hat{2}20 & 1\hat{1}12 & 1\hat{2}21 \\ 0\hat{0}02 & 0\hat{0}20 & 0\hat{1}02 & 0\hat{1}20 & 0\hat{2}10 & 0\hat{2}21 & 1\hat{1}21 & 1\hat{2}22 \\ 0\hat{0}10 & 0\hat{0}21 & 0\hat{1}10 & 0\hat{1}21 & 0\hat{2}11 & 0\hat{2}22 & 1\hat{1}22 & 2\hat{2}22 \end{array} \tag{61}$$

（在“^”之前的数字是素串0, 0001, 0002, 001, 0011, …, 2。）

定理Q说明为什么这个算法是正确的，因为步骤F3和F4明显地找出超出以前的预素串$a_1 \cdots a_n$的长度$\leq n$的最小m进制素串。注意在a_1从0增到1之后，本算法就来访问所有$m-1$进制的素串和预素串，以$1 \cdots 1$来递增。

算法F是很优美的，但对于德·布鲁因循环它需要做什么呢？以下是关键：如果每当j是n的一个因子时，我们在步骤F2输出数字$a_1, \cdots, a_j$，则所有这样的数字的序列形成一个德·布鲁因循环！例如，在$m = 3$和$n = 4$的情况下，以下的81个数字是输出：

$$\begin{aligned} &0\ 0001\ 0002\ 0011\ 0012\ 0021\ 0022\ 01\ 0102\ 0111\ 0112 \\ &\qquad 0121\ 0122\ 02\ 0211\ 0212\ 0221\ 0222\ 1\ 1112\ 1122\ 12\ 1222\ 2 \end{aligned} \tag{62}$$

（我们省略(61)的素串001, 002, 011, …, 122，因为它们的长度不整除4。）构成这个几乎是魔术般的性质的原因在习题108中剖析。注意由(59)，这个循环有正确的

长度。

有一种意义，其中这个过程的输出实际上等价于对于所有m和n都有效的所有德·布鲁因构造的“祖父”，即由蒙·哈·马丁（M. H. Martin）在[*Bull Amer. Math. Soc*（美国数学学会会刊）40(1934)，859-864] 上最先发表的构造：对于$m = 3$和$n = 4$，马丁的原始循环是2222122202211…10000，这是(62)的2的补码。事实上，弗雷德里克森和梅奥兰纳在寻找马丁序列的一个简单方法时，几乎是碰巧地发现算法F的。在他们的算法和预素串之间的明显联系一直到许多年之后才被发现，那就是当拉斯基、萨维吉和王珉懿对运行时间进行仔细分析时[*J. Algorithms* 13(1992)，414-430]。该分析的主要结果出现在习题107中，即：

27

1) 在步骤F3和F5中$n - j$的平均值近似为$1/(m - 1)$。

2) 产生像(62)那样的一个德·布鲁因循环的总运行时间是$O(m^n)$。

习 题

1. [*10*] 给定对于每个坐标的下限l_j和上限u_j（假定$l_j \leq u_j$）。说明如何生成所有n元组$(a_1, \cdots, a_n)$，其中$l_j \leq a_j \leq u_j$。

2. [*15*] 如果$n = 10$且对于$1 \leq j \leq n$，$m_j = j$，由算法M所访问的第1 000 000个n元组是什么？

提示：$\begin{bmatrix} 0,0,1,2,3,0,2,7,1,0 \\ 1,2,3,4,5,6,7,8,9,10 \end{bmatrix} = 1\,000\,000$。

▶ 3. [*M20*] 算法M执行步骤M4多少次？

▶ 4. [*18*] 在大多数计算机上下数到0比上数到m要更快些。试修改算法M使得它从$(m_1 - 1, \cdots, m_n - 1)$开始，以相反的顺序访问所有$n$元组，并以$(0, \cdots, 0)$结束。

▶ 5. [*20*] 如同“快速傅里叶变换”（参见习题4.6.4-14）这样的算法经常以二进位反射顺序下的答案数组来结束，而且有$A[(b_0 \cdots b_{n-1})_2]$在希望有$A[(b_{n-1} \cdots b_0)_2]$的位置上。重新把答案安排成适当顺序的一个好方法是怎样的？（提示：反射算法M。）

6. [*M17*] 证明(7)，即格雷二进制编码的基本公式。

7. [*20*] 图10b示出对于分成为16个扇区的一个盘的格雷二进制编码。如果扇区数是12或60（对于一个钟上的小时数或分数），或360（对于一个圆周的度数），一个好的类似格雷二进制编码将是怎样的？

8. [*15*] 跑遍偶校验的所有n个二进位的串，而且在每步中只改变两个二进位的一个容易的方法是怎样的？

9. [*16*] 当求解中国环难题时，在图11之后的下一步应是什么？

▶ 10. [*M21*] 求出在删除n个中国环的最短过程中，一个环a) 被删去或b) 被替代的步骤总次数A_n或B_n的一个简单公式。例如$A_3 = 4$及$B_3 = 1$。

11. [*M22*]（亨·约·帕基斯（H. J. Purkiss），1865。）中国环难题的两个最小的环实际上可同时地放到杆上或从杆上卸下。当允许这样加快的动作时，问求解此难题需要多少步？

▶ 12. [*25*] n的组成是总和为n的正整数的序列。例如，4的组成是1111, 112, 121, 13, 211, 22, 31和4。一个整数n恰有2^{n-1}个组成，对应于点$\{1, \cdots, n - 1\}$所有这样的子集，即它们可

被用来把区间（0…n）分成整数大小的子区间。

a) 试设计一个无循环算法来生成n的所有组成，并且把每个组成表示为整数$s_1 s_2 \cdots s_j$的一个顺序数组。

b) 类似地，试设计在指针$q_0 q_1 \cdots q_t$的数组中含蓄地表示组成的一个无循环算法，其中组成的元素为$(q_0 - q_1)(q_1 - q_2)\cdots(q_{t-1} - q_t)$，而且我们有$q_0 = n$，$q_t = 0$。例如，在这个方案之下组成211将由指针$q_0 = 4$，$q_1 = 2$，$q_2 = 1$，$q_3 = 0$表示且$t = 3$。 28

13. [*21*] 继续上一道题，也计算多项式系数$C = \binom{n}{s_1, \cdots, s_j}$供在组成$s_1 \cdots s_j$被访问时使用。

14. [*20*] 试设计生成所有串$a_1 \cdots a_j$的一个算法，使得在词典顺序下，有$0 \leqslant j \leqslant n$且对于$1 \leqslant i \leqslant j$有$0 \leqslant a_i < m_i$。例如，如果$m_1 = m_2 = n = 2$时，你的算法应逐次访问$\varepsilon$，0，00，01，1，10，11。

▶15. [*25*] 试设计生成上道题的串的一个无循环算法。所有相同长度的串都应和上题一样以词典顺序来访问，但是不同长度的串可以以任何方便的方法混合起来。例如，当$m_1 = m_2 = n = 2$时，0，00，01，ε，10，11，1是可接受的顺序。

16. [*23*] 一个无循环算法明显地不能以词典顺序来生成所有二进制向量$(a_1, \cdots, a_n)$，因为在逐次的访问之间需要加以改变的坐标a_j的个数是不受限的。然而，试证明，如果使用链接表示以代替顺序表示的话，无循环的词典顺序生成确实是可能的。假设有$2n + 1$个节点$\{0, 1, \cdots, 2n\}$，每一个含一个链接字段LINK。通过令

$$\begin{aligned} &\texttt{LINK}(0) = 1 + na_1 \\ &\texttt{LINK}(j - 1 + na_{j-1}) = j + na_j, \qquad 1 < j \leqslant n \\ &\texttt{LINK}(n + na_n) = 0 \end{aligned}$$

来表示二进制n元组$(a_1, \cdots, a_n)$，其他n个LINK字段可以有任何方便的值。

17. [*20*] 一个叫做卡诺图的著名构造[莫·卡诺（M. Karnaugh），*Amer. Inst. Elect. Eng. Trans.* **72**, part I(1953)，593-599]使用二维的格雷二进制编码，来显示在一个4×4的圆环体中所有4个二进位的数。

0000	0001	0011	0010
0100	0101	0111	0110
1100	1101	1111	1110
1000	1001	1011	1010

（一个圆环体的条目在左边和右边包起来，而且还在顶点和底部也包起来，就如同它们是瓦片那样，在一个平面上无穷地重复。）试证明，类似地，所有6个二进位的数可以被安排在一个8×8的圆环体中，使得当我们从任何点向北、向南、向东或向西时，只有一个坐标在改变。

▶18. [*20*] 一个向量$u = (u_1, \cdots, u_n)$，其中每个分量满足$0 \leqslant u_j < m_j$，它的李权被定义为

$$\nu_L(u) = \sum_{j=1}^{n} \min(u_j, m_j - u_j)$$

并且两个这样的向量u和v之间的李距离是

$$d_L(u, v) = \nu_L(u - v),\quad \text{其中} u - v = ((u_1 - v_1) \bmod m_1, \cdots, (u_n - v_n) \bmod m_n)$$

(如果在每一步中我们通过 ± 1(modulo m_j) 来调整某些分量u_j，则这是为改变u和v所需要的极小步骤数。)

对于$1 \leqslant j \leqslant n$，一个四进向量有$m_j = 4$，而一个二进向量全有$m_j = 2$。试求四进向量$u = (u_1, \cdots, u_n)$和二进向量$u' = (u'_1, \cdots, u'_{2n})$之间简单的一对一对应，且具有$\nu_L(u) = \nu(u')$和$d_L(u, v) = \nu(u' \oplus v')$的性质。

29

19. [21] (八字码)。令$g(x) = x^3 + 2x^2 + x - 1$。

a) 用本小节中的算法之一来计算对于$0 \leqslant v_0, v_1, v_2, v_3 < 4$，所有256个多项式$(v_0 + v_1x + v_2x^2 + v_3x^3)\ g(x) \bmod 4 = u_0 + u_1x + u_2x^2 + u_3x^3 + u_4x^4 + u_5x^5 + u_6x^6$之和$\sum z_{u_0}z_{u_1}z_{u_2}z_{u_3}z_{u_4}z_{u_5}z_{u_6}z_{u_\infty}$，其中$u_\infty$是如此选定的，即使$0 \leqslant u_\infty < 4$且$(u_0 + u_1 + u_2 + u_3 + u_4 + u_5 + u_6 + u_\infty) \bmod 4 = 0$。

b) 试构造256个16位的二进数的一个集合，这些数在至少6个不同的位置上彼此不同。(这样一个集合，首先是由诺德斯特洛姆（Nordstrom）和罗宾森（Robinson）发现的[*Information and Control*（信息和控制）11(1967), 613-616]。它实质上是惟一的。)

20. [*M36*] 上题中的16个二进位的码字可以用来传送信息位，并且如果任何一位或两位出错，则允许校正传输错误。而且，如果有三位被不正确地接收，错误将被探测出来（但不必是可校正的)。试设计一个算法，它或者找出最接近于一个给定的16个二进位数u'的码字，或者确定u'至少有3个二进位是错误的。你的算法如何对数$(1100100100001111)_2$进行译码？[提示：使用$x^7 \equiv 1$(modulo $g(x)$ 和4) 以及次数<3的每一四进制多项式对于某个j，$k \in \{0, 1, 2, 3, 4, 5, 6, \infty\}$同余于$x^j + 2x^k$(modulo $g(x)$ 和4) 这两个事实，其中$x^\infty = 0$。]

21. [*M30*] 一个n立体的t子立体可以通过像**10**0*这样的串来表示，它包含t个星号和$n - t$个确定的二进位。如果以词典顺序写出所有2^n个二进制的n元组，属于这样一个子立体的元素出现于连续的条目的$2^{t'}$从集中，其中t'是位于最右边确定的二进位左边的星号个数。(在给定的例子中，$n = 8$，$t = 5$及$t' = 4$。) 但如果以格雷二进制编码的顺序来写出n元组，则从集的个数可被减少。例如，当使用格雷二进制编码顺序时，$(n - 1)$子立体*…*0和*…*1仅分别在$2^{n-2} + 1$和2^{n-2}个从集中出现，而不是2^{n-1}个从集。

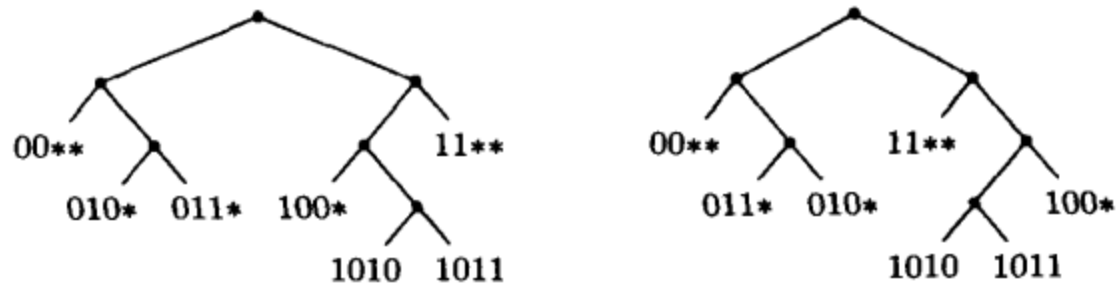
a) 说明如何来计算由星号、0和1组成的一个给定的串α定义的子立体的格雷二进制编码从集的个数$C(\alpha)$，什么是$C(**10**0*)$?

b) 证明$C(\alpha)$总位于$2^{t'-1}$和$2^{t'}$（含）之间。

c) 在所有可能的$2^{n-t}\binom{n}{t}$个t子立体中，什么是$C(\alpha)$的平均值？

▶ 22. [22] 一个“右子立体”是如同0110**这样的一个子立体，其中所有的星号都出现在所有确定的数字之后。任何二进制的检索结构（6.3节)，如同在图16a中那样，都可以被看作是把一个立体分划成为不相交的右子立体的一个方法。如果我们把每一个右子检索结构的左和右子检索结构交换，并且从根开始往下进行，则如同在图16b中那样，我们得到一个格雷二进制检索结构。

证明如果格雷二进制检索结构的“叶”按从左到右的顺序被遍历，连续的叶对应于相邻子立体。(若子立体包含相邻顶点，则子立体是相邻的。例如，00**与011*相邻，因为前者包含0010而后者包含0110；但011*与10**不相邻。)



a) 通常的二进制检索结构　　b) 格雷二进制检索结构

图 16

30

23. [20] 假设$g(k) \oplus 2^j = g(l)$。给定j和k，什么是求l的简单方法？

24. [M21] 试考虑把格雷二进制函数g扩充成所有二进的整数（参见习题4.1-31）。什么是对应的逆函数$g^{[-1]}$？

▶25. [M25] 试证明，如果$g(k)$和$g(l)$在$t>0$位上不同，而且如果$0 \leq k, l < 2^n$，则$\lceil 2^t/3 \rceil \leq |k-l| \leq 2^n - \lceil 2^t/3 \rceil$。

26. [25]（弗朗克·拉斯基。）对于哪些整数N，能以这样一种方式来生成所有小于N的非负整数，即对于它们的二进制表示，在每步中，只有一个二进位发生改变？

▶27. [20] 令$S_0 = \{1\}$和$S_{n+1} = 1/(2+S_n) \cup 1/(2-S_n)$；于是，例如

$$S_2 = \left\{ \frac{1}{2+\frac{1}{2+1}}, \frac{1}{2+\frac{1}{2-1}}, \frac{1}{2-\frac{1}{2+1}}, \frac{1}{2-\frac{1}{2-1}} \right\} = \left\{ \frac{3}{7}, \frac{1}{3}, \frac{3}{5}, 1 \right\}$$

而且S_n有位于1/3和1之间的2^n个元素。试计算S_{100}的第10^{10}个最小元素。

28. [M27] n个二进位的串$\{\alpha_1, \cdots, \alpha_t\}$，其中$\alpha_k$有二进表示$\alpha_k = a_{k(n-1)} \cdots a_{k0}$，它的均值是这样一个串$\hat{\alpha} = a_{n-1} \cdots a_0$，对于$0 \leq j < n$，其二进位$a_j$同对于$1 \leq k \leq t$的二进位$a_{kj}$的大多数相一致。（如果$t$是偶数，而且二进位$\alpha_{kj}$有一半为0和一半为1，则均值二进位$a_j$既可为0也可为1。）例如，串{0010, 0100, 0101, 1110}有两个均值，0100和0110，我们可以用01*0来表示之。

a) 求当$0 < t \leq 2^n$时，描述$G_t = \{g(0), \cdots, g(t-1)\}$即头$t$个格雷二进制串的均值的一个简单方法。

b) 证明如果$\alpha = a_{n-1} \cdots a_0$是这样一个均值，而且如果$2^{n-1} < t < 2^n$，则由$\alpha$通过对于任何二进位$a_j$求补得到的串$\beta$也是$G_t$的一个元素。

29. [M24] 假定k的所有值是同样可能的，则如果把整数k作为n个二进位的格雷二进制编码$g(k)$加以传输，而且以一个二进位模式$p = (p_{n-1} \cdots p_0)_2$描述的误差而被接受，则平均数值误差是

$$\frac{1}{2^n} \sum_{k=0}^{2^n-1} |(g^{[-1]}(k) \oplus p) - k|$$

试证明这个和等于$\sum_{k=0}^{2^n-1} |(k \oplus p) - k| / 2^n$，就如同不使用格雷二进制编码而明显地计算它那样。

▶30. [M27]（格雷排列）。试仅使用一个固定数量的辅助存储，设计一遍扫描算法。通过$(X_{g(0)}, X_{g(1)}, X_{g(2)}, \cdots, X_{g(2^n-1)})$来代替数组元素$(X_0, X_1, X_2, \cdots, X_{2^n-1})$。提示：考虑函数$g(n)$为所有非负整数的一个排列，证明集合

$$L = \{0, 1, (10)_2, (100)_2, (100*)_2, (100*0)_2, (100*0*)_2, \cdots\}$$

是循环前导（循环的最小元素）的集合。

31. [*HM35*]（格雷域）。令$f_n(x) = g(r_n(x))$表示如同在习题5中那样反射一个n位二进制串的位的操作，而后把它转换成格雷二进制编码。例如，操作$f_3(x)$使$(001)_2 \mapsto (110)_2 \mapsto (010)_2 \mapsto (011)_2 \mapsto (101)_2 \mapsto (111)_2 \mapsto (100)_2 \mapsto (001)_2$，因此所有非零可能性出现于单个循环中。所以我们可以使用f_3来定义8个元素的一个域，并且以⊕作为加法操作符和用以下规则

31

$$f_3^{[j]}(1) \times f_3^{[k]}(1) = f_3^{[j+k]}(1) = f_3^{[j]}(f_3^{[k]}(1))$$

定义乘法。函数f_2，f_5和f_6有同样美妙的性质。但f_4则不然，因为$f_4((1011)_2) = (1011)_2$。

求出所有$n < 100$，使得f_n定义2^n个元素的一个域。

32. [*M20*] 真或假：沃尔斯函数满足$w_k(-x) = (-1)^k w_k(x)$。

▶33. [*M20*] 证明拉德曼彻-到-沃尔斯的定律(17)。

34. [*M21*] 由$p_0(x) = 1$和$p_k(x) = (-1)^{\lfloor 2x \rfloor k} p_{\lfloor k/2 \rfloor}(2x)$定义帕利（Paley）函数$p_k(x)$。试证明，类似于(17)，借助于拉德曼彻函数，$p_k(x)$有一个简单的表达式，并且把帕利函数同沃尔斯函数关联起来。

35. [*HM23*] 就如同沃尔斯矩阵W_n可从沃尔斯函数得到（参见(20)）那样，$2^n \times 2^n$的帕利矩阵P_n也可从帕利函数得到。试求P_n，W_n以及哈达玛矩阵H_n之间的有趣关系。证明所有这三个矩阵都是对称的。

36. [*21*] 给出计算一个给定向量$(X_0, \cdots, X_{2^n-1})$的沃尔斯转换$(x_0, \cdots, x_{2^n-1})$的有效算法的细节。

37. [*HM23*] 对于$1 \leq l \leq k$和$0 < z_{kl} < 1$，设z_{kl}是在$w_k(x)$中第l个符号变化的位置。试证明$|z_{kl} - l/(k+1)| = O((\log k)/k)$。

▶38. [*M25*] 设计沃尔斯函数的三进制推广。

▶39. [*HM30*]（詹·约·希尔维斯特（J. J. Sylvester）。）$\left(\begin{smallmatrix} a & b \\ b & -a \end{smallmatrix}\right)$的行是彼此正交的，并且有相同的数量；因此矩阵恒等式

$$\begin{aligned}(A\ \ B)\begin{pmatrix} a^2+b^2 & 0 \\ 0 & a^2+b^2 \end{pmatrix}\begin{pmatrix} A \\ B \end{pmatrix} &= (A\ \ B)\begin{pmatrix} a & b \\ b & -a \end{pmatrix}\begin{pmatrix} a & b \\ b & -a \end{pmatrix}\begin{pmatrix} A \\ B \end{pmatrix} \\ &= (Aa+Bb\ \ \ Ab-Ba)\begin{pmatrix} aA+bB \\ bA-aB \end{pmatrix}\end{aligned}$$

意味着两个平方和恒等式$(a^2+b^2)(A^2+B^2) = (aA+bB)^2 + (bA-aB)^2$。类似地，矩阵

$$\begin{pmatrix} a & b & c & d \\ b & -a & d & -c \\ d & c & -b & -a \\ c & -d & -a & b \end{pmatrix}$$

导致四平方之和恒等式

$$\begin{aligned}(a^2+b^2+c^2+d^2)(A^2+B^2+C^2+D^2) &= (aA+bB+cC+dD)^2 + (bA-aB+dC-cD)^2 \\ &\quad + (dA+cB-bC-aD)^2 + (cA-dB-aC+bD)^2\end{aligned}$$

a) 把(21)中矩阵H_3的符号附加到符号$\{a, b, c, d, e, f, g, h\}$上，以得到具有正交行的一个矩阵和八个平方之和的恒等式。

b) 推广到H_4和更高阶的矩阵。

▶40. [*21*] 如果对于$0 \leqslant j < 5$，在步骤W2中的屏蔽作为$m_j = x \ \& \ (2^{5j} - 1)$来予以计算，则文本的5个字母的词的计算方案也将产生正确答案吗？ 32

41. [*25*]如果我们把5个字母的词的问题限定到最普通的3000个词——由此从(23)中消除 `ducky`, `duces`, `dunks`, `dinks`, `dinky`, `dices`, `dicey`, `dicky`, `dicks`, `picky`, `pinky`, `punky`和`pucks`——则从单个对的词仍可以产生多少有效的词来？

42. [*35*]（迈·劳·弗雷德曼（M.L.Fredman）。）当判定下一次格雷二进制编码的哪一个二进位a_j应该求补时，算法L使用$\Theta(n \log n)$个辅助存储的二进位作为焦点指针。在L3的每步中，它考察辅助二进位的$\Theta(\log n)$个，而且有时它改变它们中的$\Omega(\log n)$个。

试证明，从理论的观点说我们可以做得更好些：在访问之间通过至多改变2个辅助的二进位，就可生成n个二进位的格雷二进制编码。（我们仍将允许在每步考察$O(\log n)$个辅助的二进位，使得我们知道它们中的哪些位应加以改变。）

43. [*47*] 确定6个二进位的格雷循环的个数$d(6)$。

44. [*M37*] 试证明对于$n-1$或$n-2$个二进位的格雷循环的任意δ序列，可以用来构造n个二进位的格雷循环的大量δ序列，而且它们具有如下性质，即恰好有坐标名的(a)一个或(b)两个仅出现两次。

45. [*M25*] 试证明，序列$d(n)$有双倍的指数增长，即有一个常数$A > 1$，使得$d(n) = \Omega(A^{2^n})$。

46. [*HM48*] 确定当$n \to \infty$时，$d(n)^{1/2^n}$的渐近特性。

47. [*M46*]（希尔弗曼（Silverman）、韦克斯（Vickers）和萨姆普森（Sampson）。）令$S_k = \{g(0), \cdots, g(k-1)\}$是标准格雷二进制编码的头$k$个元素，并令$H(k, v)$是以0开始和以$v$结尾的$S_k$中哈密顿通路的个数。证明或否定，对于所有同$g(k)$相邻的$v \in S_k$，$H(k, v) \leqslant H(k, g(k-1))$。

48. [*36*] 如果上题中的猜测为真，试证明$d(n) \leqslant 4(n/2)^{2^n}$。[提示：令$d(n, k)$是以$g(0) \cdots g(k-1)$开始的$n$个二进位的格雷循环的个数；这个猜测意味着$d(n) \leqslant c_{n1} \cdots c_{n(k-1)} d(n, k)$，其中$c_{nk}$是在$n$立体中而非在$S_k$中相邻于$g(k-1)$的顶点个数。]

49. [*20*] 试证明，对于所有的$n \geqslant 1$，有$2n$个二进位的格雷循环，其中对于所有$k \geqslant 0$，$v_{k+2^{2n-1}}$是v_k的补。

▶50. [*21*] 求类似于定理D的但有偶数的l的一个构造。

51. [*M24*] 完成对于定理D的推论B的证明。

52. [*M20*] 试证明，如果一个n个二进位的格雷循环的转换计数满足$c_0 \leqslant c_1 \leqslant \cdots \leqslant c_{n-1}$，我们必有$c_0 + \cdots + c_{j-1} \geqslant 2^j$，而且当$j = n$时等式成立。

53. [*M46*] 如果数$(c_0, \cdots, c_{n-1})$是偶数且满足上题的条件，是否总有具有这些转移计数的一个n进制格雷循环？

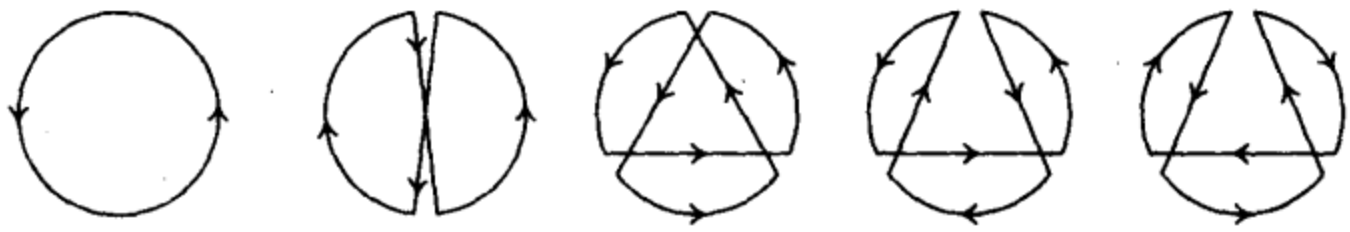
54. [*M20*]（哈·塞·萨皮罗（H.S.Shapiro），1953。）证明如果一个整数序列$(a_1, \cdots, a_{2^n})$仅含n个不同的值，则对于某个$0 \leqslant k < l \leqslant 2^n$，存在一个子序列，其乘积$a_{k+1} a_{k+2} \cdots a_l$是

一个完全平方。但如果我们不允许$l = 2^n$的情况，则这个结论可能不成立。

55. [47]（弗·拉斯基和卡·萨维吉，1993。）如果$(v_0, \cdots, v_{2^n-1})$是一个n个二进位的格雷循环，则对偶$\{\{v_{2k}, v_{2k+1}\} | 0 \leqslant k < 2^{n-1}\}$形成在$n$立体中偶校验和奇校验顶点间的一个完全匹配。反之，每个这样的完全匹配是否作为某个n个二进位的格雷循环的“一半”而出现？

56. [M30]（埃·尼·吉尔伯特（E. N. Gilbert），1958。）如果两个格雷循环的δ序列可
33 以通过排列坐标名，或通过颠倒循环和/或通过在一个不同的位置处驱动这个循环而作成相等，就说这两个格雷循环是等价的。试证明2688个不同的4个二进位格雷循环刚好落入9个等价类中。

57. [32] 考虑这样一个图，其顶点是2688种可能的4个二进位的格雷循环，其中两个这样的循环是相邻的——如果通过下列简单的转换之一而相关：



（类型1在循环被分成两部分之后且其中一部分被颠倒来重新装配后出现，类型2、3和4在这个循环被分成三个部分并且颠倒0、1或2个部分再重新装配之后出现，这些部分不必有相同的大小。哈密顿循环的这样的变换通常是可能的。）

试写出一个程序来找出哪些4个二进位的格雷循环是彼此可转换的，通过寻找这个图的连通分量来做。限定一次只考虑四种类型之一。

▶58. [21] 设α是一个n个二进位的格雷循环的δ序列，并且通过把0的q个出现变成为n个而从α得到β，其中q为奇数。试证明$\beta\beta$是一个$(n+1)$个二进位的格雷循环。

59. [22] 对于$1 < t < n$来说，没有2^t个连续的元素属于一个t子立体，在这样的意义下，(30)的5个二进位格雷循环是非局部的。试证明对于所有$n \geqslant 5$，非局部的n个二进位格雷循环存在。（提示：参看上一题。）

60. [20] 证明运行长度界函数满足$r(n+1) \geqslant r(n)$。

61. [M30] 如果(a) $m = 2$和$2 < r(n) < 8$；或者(b) $m < n$和$r(n) < 2^{m-3}$，试证明$r(m+n) \geqslant r(m) + r(n) - 1$。

62. [46] 是否$r(8) = 6$?

63. [30]（路易斯·戈德丁（Luis Goddyn）。）证明$r(10) \geqslant 8$。

▶64. [HM35]（路·戈德丁和帕·格沃兹德扎克（P. Gvozdjak）。）一个n个二进位的格雷流是排列$(\sigma_0, \sigma_1, \cdots, \sigma_{l-1})$的一个序列，其中每个$\sigma_k$是$n$立体的顶点的一个排列，并且取每个顶点为其邻居点之一。

a) 假设$(u_0, \cdots, u_{2^m-1})$是一个m个二进位的格雷循环，而且$(\sigma_0, \sigma_1, \cdots, \sigma_{2^m-1})$是一个$n$个二进位的格雷流，令$v_0 = 0\cdots0$和$v_{k+1} = v_k\sigma_k$，其中$\sigma_k = \sigma_{k \bmod 2^m}$若$k \geqslant 2^m$。在什么条件下，序列

$$W = (u_0v_0, u_0v_1, u_1v_1, u_1v_2, \cdots, u_{2^{m+n-1}-1}v_{2^{m+n-1}-1}, u_{2^{m+n-1}-1}v_{2^{m+n-1}})$$

作为（$m+n$）个二进位的格雷循环?

b) 试证明如果m充分大，则存在一个满足(a)的条件的n个二进位的格雷流，对于它，序列（$\nu_0, \nu_1, \cdots$）的运程长度$\geq n-2$。

c) 应用这些结果来证明$r(n) \geq n - O(\log n)$。

65. [*30*]（布里特·施蒂文斯（Brett Stevens）。）在萨缪尔·贝克特（Samuel Beckett）的戏剧作品《监狱》中，舞台开幕和空场结束；n个演员进入和一次出去一人，跑遍所有2^n种可能的子集。而且离场的演员总是上次入场最早的那个。如同在真正的戏中那样，当$n=4$时，某些子集必然重复，然而，试证明，当$n=5$时，存在有恰好2^n个进入和出去的一个完美的模式。

34

66. [*40*] 对于8个演员来说，是否有一个完美的贝克特-格雷模式?

67. [*20*] 有时在从一个步骤进行到下一个步骤时，改变尽可能多的二进位，来跑遍所有n个二进位和二进位串是所希望的。例如，当在最坏情况条件下测试一个物理线路的可靠特性时。试说明，如何以这样一种方式来遍历所有二进位的n元组，即每步交替地改变n位或$n-1$位。

68. [*21*] 鲁弗斯·昆·珀弗斯（Rufus Q. Perverse）决定来构造一个反格雷的三进制编码。其中每个n位三进制数在每个数字位置中不同于它的邻居，对于所有的n，这样一个码是否可能?

▶69. [*M25*]当$k=(\cdots b_5b_4b_3b_2b_1b_0)_2$时，通过令

$$h(k)=(\cdots(b_6\oplus b_5)\,(b_5\oplus b_4)\,(b_4\oplus b_3\oplus b_2\oplus b_0)\,(b_3\oplus b_0)\,(b_2\oplus b_1\oplus b_0)b_1)_2$$

来修改格雷二进制编码的定义(7)。

a) 证明当$n>3$时，序列$h(0), h(1), \cdots, h(2^n-1)$以这样的方式跑遍所有$n$个二进位的数，即每次恰有3个二进位发生变化。

b) 当t是奇数和$n>t$时，试推广这个规则来得到在每个步骤中恰有3个二进位发生变化的序列。

70. [*21*] 对于$n=5$和$n=6$，存在有多少单调的n个二进位的格雷码?

71. [*M22*] 试推导(45)，即定义萨维吉-温格尔排列的递归式。

72. [*20*] 什么是从00000到11111的萨维吉-温格尔码?

▶73. [*32*] 试设计一个有效的算法，来构造一个n个二进位的单调格雷编码。

74. [*M25*]（萨维吉和温格尔。）在一个单调的格雷编码中，n立体的相邻顶点可以相距多远?

75. [*32*] 在每个坐标位置j中$\sum_{k=0}^{31} k(-1)^{\nu_{kj}}=0$的意义下，求出无趋势的所有5个二进位的格雷通路$\nu_0, \cdots, \nu_{31}$。

76. [*M25*] 试证明对于所有$n>5$，存在无趋势的n个二进位的格雷编码。

77. [*21*] 试修改算法H以便在取模的格雷顺序下访问混合进制的n元组。

78. [*M26*] 试证明对于反射混合进制的格雷编码的转换公式(50)和(51)，并且对于取模的情况推导类似的公式。

▶79. [*M22*] 什么时候(a)反射的(b)取模的混合进制的格雷编码的最后n元组同头一个相邻?

80. [*M20*] 试说明，给定一个数的素因式分解 $p_1^{e_1}\cdots p_t^{e_t}$，并且在每步中重复地乘或除一个素数，如何跑遍它的所有因子？

81. [*M21*] 令$(a_0, b_0), (a_1, b_1), \cdots, (a_{m^2-1}, b_{m^2-1})$是2个数字的$m$进制取模的格雷编码。试证明，如果$m>2$，每边 (x, y)——$(x, (y+1) \bmod m)$ 和 (x, y)——$((x+1) \bmod m, y)$ 出现于两个循环

$$(a_0, b_0)——(a_1, b_1)——\cdots——(a_{m^2-1}, b_{m^2-1})——(a_0, b_0)$$
$$(b_0, a_0)——(b_1, a_1)——\cdots——(b_{m^2-1}, a_{m^2-1})——(b_0, a_0)$$

之一。

▶82. [*M25*]（杰·林吉尔（G. Ringel），1956。）试用上一题来推导出，存在四个8个二进位的格雷循环，它们一起覆盖8立体的所有边。

35 83. [*41*] 四个平衡的8个二进位的格雷循环能够覆盖8立体的所有边吗？

▶84. [*25*]（霍华德·利·迪克曼（Howard L. Dyckman）。）图17示出一个称作伦尼环（Loony Loop）或戈迪安结（Gordian Knot）的一个非常有趣的难题，其目标是从环绕着绳索的坚固的环中把一根可弯曲的绳索解下来。试证明这个难题的解固有地同反射的格雷二进制编码有关。

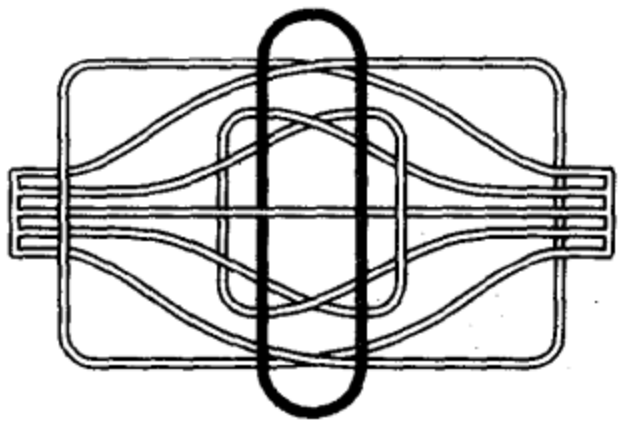


图17 伦尼环难题

▶85. [*M25*]（达纳·理查德（Dana Richards）。）如果$\Gamma = (\alpha_0, \cdots, \alpha_{t-1})$是$t$串的任何序列以及$\Gamma' = \{\alpha'_0, \cdots, \alpha'_{t'-1}\}$是$t'$串的任何序列，每行改变方向的乘积$\Gamma \boxtimes \Gamma'$是以$(\alpha_0\alpha'_0, \cdots, \alpha_0\alpha'_{t'-1}, \alpha_1\alpha'_{t'-1}, \cdots, \alpha_1\alpha'_0, \alpha_2\alpha'_0, \cdots, \alpha_2\alpha'_{t'-1}, \alpha_3\alpha'_{t'-1}, \cdots)$开始并且在$t$为偶数时，以$\alpha_{t-1}\alpha'_0$结束，而当$t$为奇数时，以$\alpha_{t-1}\alpha'_{t'-1}$结束的$tt'$串的序列。

例如，(5)中格雷二进制编码的基本定义，当$n>0$时，在这个记号下可表达为$\Gamma_n = (0,1) \boxtimes \Gamma_{n-1}$。试证明操作$\boxtimes$是结合的，因此$\Gamma_{m+n} = \Gamma_m \boxtimes \Gamma_n$。

▶86. [*26*] 试定义以这样一种方式跑遍所有可能的非负整数n元组$(a_1, \cdots, a_n)$的无穷格雷码，即当$(a_1, \cdots, a_n)$后边是$(a'_1, \cdots, a'_n)$时，则$\max(a_1, \cdots, a_n) \leqslant \max(a'_1, \cdots, a'_n)$。

87. [*27*] 继续上一道题。试定义以这样一种方式跑遍所有整数n元组$(a_1, \cdots, a_n)$的一个无穷格雷编码，即当$(a_1, \cdots, a_n)$后边是$(a'_1, \cdots, a'_n)$时，则$\max(|a_1|, \cdots, |a_n|) \leqslant \max(|a'_1|, \cdots, |a'_n|)$。

▶88. [*25*] 在算法K在步骤K4结束之后，如果我们立即在步骤K2处重新启动之，那将发生什么情况？

▶89. [*25*]（对于莫斯编码的格雷编码。）长度为n的莫斯编码字（习题4.5.3-32）是点和折线的串，其中n是点数加上两倍折线数。

a) 试证明通过逐次地把一个折线变成两个点或者反过来，有可能生成长度为n的所有莫斯码字。例如，对于$n = 3$的通路必然是•—，•••，—•，或者反过来。

b) 对于$n = 15$，在你的序列中，在•——•••——•—•之后的串是什么？

90. [26] 在习题89的基础规则下，n的什么值能把莫斯码字安排成一个循环？[提示：码字的个数是F_{n+1}。]

▶91. [34] 试设计一个无循环的算法，来这样访问所有二进制的n元组 $(a_1, \cdots, a_n)$，即使得$a_1 < a_2 > a_3 < a_4 > \cdots$。[这样的$n$元组的数目是$F_{n+2}$。]

92. [M30] 是否有这样的无穷序列Φ_n，对于所有的m，其头m^n个元素形成一个m进制的德·布鲁因循环？($n = 2$的情况在(54)中解决了。)

▶93. [M28] 试证明如同所宣告的那样，算法R输出一个德·布鲁因循环。

94. [22] 当$m = 5$，$n = 1$和$r = 3$时，如果共行程序$f()$和$f'()$生成01234 01234 01…的平凡循环，则算法D的输出是什么？ 36

▶95. [M23]假设周期为p的一个无穷序列$a_0a_1a_2\cdots$和周期为q的无穷序列$b_0b_1b_2\cdots$穿插形成无穷序列$c_0c_1c_2c_3c_4c_5\cdots = a_0b_0a_1b_1a_2b_2\cdots$。

a) 在什么情况下$c_0c_1c_2\cdots$有周期pq？(对本题而言，一个序列$a_0a_1a_2\cdots$的“周期”是对于所有$k \geq 0$，使得$a_k = a_{k+p}$的最小整数$p > 0$。)

b) 如果把步骤D6改变成只不过说“如果$t' = n$和$x' < r$，则转到D4”，则什么样的$2n$元组将作为算法D的连续的输出出现？

c) 试证明如同所宣告的那样，算法D输出一个德·布鲁因循环。

▶96. [M23] 假设递归地基于$n = 2$的基础情况的简单共行程序（如算法S），使用算法R和D，已经建立一个共行程序族，生成长度为m^n的德·布鲁因循环。

a) 将有多少个每种类型的共行程序（R_n, D_n, S_n）？

b) 为了获得一个高层次的输出数字，需要的极大个数的共行程序激活是多少？

97. [M29] 本题的目的是来分析在重要的特殊情况$m = 2$中，由算法R和D构造的德·布鲁因循环。对于$0 \leq k < n$，令$f_n(k)$是2^n循环的第$(k + 1)$位，使得$f_n(k) = 0$。对于$j_n \leq k < j_n + n$，还令j_n是使得$0 \leq j_n < 2^n$和$f_n(k) = 1$的下标。

a) 对于$n = 2$，3，4和5，写出循环$(f_n(0)\cdots f_n(2^n - 1))$。

b) 试证明，对于所有偶数的n值，存在一个数$\delta_n = \pm 1$使得我们有

$$f_{n+1}(k) = \begin{cases} \Sigma f_n(k), & \text{如果} 0 < k \leq j_n \text{或} 2^n + j_n < k \leq 2^{n+1} \\ 1 + \Sigma f_n(k + \delta_n), & \text{如果} j_n < k \leq 2^n + j_n \end{cases}$$

其中的同余是modulo 2。(在这个公式中，Σf代表求和函数 $\Sigma f(k) = \sum_{j=0}^{k-1} f(j)$。)因此当$n$为偶数时，$j_{n+1} = 2^n - \delta_n$。

c) 令$(c_n(0)c_n(1)\cdots c_n(2^{2n} - 5))$是当把习题95(b)中的算法D的简单版本应用到$f_n()$时产生的循环，在这个循环中在何处出现$(2n - 1)$元组$1^{2n-1}$和$(0\ 1)^{n-1}0$？

d) 借助于$f_n()$，使用(c)的结果来表达$f_{2n}(k)$。

e) 作为n的函数，求对于j_n的一个（稍微）简单的公式。

98. [M34] 继续上一题，给定$n \geq 2$和$k \geq 0$，试设计一个计算$f_n(k)$的有效算法。

▶99. [*M23*] 利用上题的技术，来设计一个有效的算法，它在循环 $(f_n(0)\ f_n(1)\ \cdots\ f_n(2^n-1))$ 中定址一个给定的n个二进位的串。

100. [*40*] 当n很大时，习题97的德·布鲁因循环是否提供虚拟随机二进位的有用来源？

101. [*M30*]（把串分解成非递增的素数的惟一因式分解。）

a) 试证明如果λ和λ'是素数，则如果$\lambda<\lambda'$，$\lambda\lambda'$为素数。

b) 因此，每个串α可以写成如下形式：$\alpha=\lambda_1\lambda_2\cdots\lambda_t$，$\lambda_1\geqslant\lambda_2\geqslant\cdots\geqslant\lambda_t$，其中每个$\lambda_j$为素数。

c) 事实上，只有一个这样的因式分解是可能的。提示：试证明λ_t必是α的按词典顺序最小的非空后缀。

d) 真或假：λ_1是α的最长的素前缀。

37 e) 31415926535897932384626433832795028841 97的素因子是什么？

102. [*HM28*] 从上一题的惟一因式分解定理，推导长度为n的m进素数的个数。

103. [*M20*] 利用公式(59)来证明费尔玛定理$m^p\equiv m(\text{modulo } p)$。

104. [*17*] 按照公式(60)，所有n个字母的词中大约$1/n$是素数，在5757个五个字母的图库的词中有多少是素数？它们中哪一个是最小的非素数？最大的素数呢？

105. [*M31*] 令α是在一个无穷的字母表上长度为n的一个预素数串。

a) 试证明，如果α的最后字母被增加，则得到的串是素数。

b) 如果如同在习题101中那样，α已被分解，试证明它是λ_1的n扩充。

c) 而且α不可能是两个不同素数的n扩充。

▶106. [*M30*] 通过颠倒-工程化的算法F，试设计一个算法，它以递减顺序来访问m进制的所有素数和预素数。

107. [*HM30*] 试分析算法F的运行时间。

108. [*M35*] 令$\lambda_1<\cdots<\lambda_t$是其长度整除$n$的$m$进制素串，并令$a_1\cdots a_n$是任何的$m$进制串。本题的目的是证明$a_1\cdots a_n$出现在$\lambda_1\cdots\lambda_t\lambda_1\lambda_2$中；因此$\lambda_1\cdots\lambda_t$是德布鲁因循环（因为它有长度$m^n$）。为方便起见，我们可以假定$m=10$而且这些串对应于十进数。对于任何$m\geqslant 2$，同样的论断也适用。

a) 试证明如果$a_1\cdots a_n=\alpha\beta$不同于它的所有循环移位，而且如果$\beta\alpha=\lambda_k$是素数，则$\alpha\beta$是$\lambda_k\lambda_{k+1}$的一个子串，除非对于某个$j\geqslant 1$，$\alpha=9^j$。

b) 如果$\beta\alpha$是素的而且α全由9组成，$\lambda_1\cdots\lambda_t$中$\alpha\beta$出现于何处？提示：要证明，如果对于某个$l>0$，在步骤F2中$a_{n+1-l}\cdots a_n=9^l$，而且如果j不是n的一个因式，则以前的步骤F2有$a_{n-l}\cdots a_n=9^{l+1}$。

c) 现在考虑形如$(\alpha\beta)^d$的n元组，其中$d>1$是n的一个因式而且$\beta\alpha=\lambda_k$是素数。

d) 当$n=6$时，899135，997879，913131，090909，909090和911911在何处出现？

e) $\lambda_1\cdots\lambda_t$是长度为m^n的按词典顺序最小的m进制德·布鲁因循环吗？

109. [*M22*] 对于2×2的视窗，大小为$m^2\times m^2$的一个m进制的德·布鲁因圆环体是m进制数字d_{ij}的一个矩阵，使得m^4个子矩阵的每个，对于$0\leqslant i,\ j<m^2$，有

$$\begin{pmatrix} d_{ij} & d_{i(j+1)} \\ d_{(i+1)j} & d_{(i+1)(j+1)} \end{pmatrix}$$

都不同。其中下标绕modulo m^2而包起来。因此每个可能的m进制2×2子矩阵恰出现一次。伊恩·施蒂华特（Ian Stewart）[*Game*, *Set*, *and Math*（游戏、集合和数学）（Oxford: *Blackwell*, 1989），第4章] 因此把它叫做一个m进制的外圆环体。例如

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

是一个二进制的外圆环体。其实，当$m = 2$时，除对于移位和/或转置之外，它实质上是惟一的这样的矩阵。

考虑在行$i = (\cdots a_2a_1a_0)_2$和列$j = (\cdots b_2b_1b_0)_2$中的条目为$d_{ij} = (\cdots c_2c_1c_0)_2$的无限矩阵D，其中

$$c_0 = (a_0 \oplus b_0)(a_1 \oplus b_1) \oplus b_1$$
$$c_k = (a_{2k}a_0 \oplus b_{2k})b_0 \oplus (a_{2k+1}a_0 \oplus b_{2k+1})(b_0 \oplus 1), k > 0$$

试证明对于所有$n \geqslant 0$，D的左上$2^{2n} \times 2^{2n}$子矩阵是一个2^n进制的外圆环体。 38

110. [*M25*] 继续上一题，对于所有m，试构造m进制的外圆环体。

111. [*20*] 通过插入+和−符号到序列123456789中，我们可以用12种方法得到数100；例如$100 = 1 + 23 - 4 + 5 + 6 + 78 - 9 = 123 - 45 - 67 + 89 = -1 + 2 - 3 + 4 + 5 + 6 + 78 + 9$。

a) 不能以这样一种方法表示的最小的正整数是多少？

b) 也考虑把符号插入到10个数字的序列9876543210。

▶ 112. [*25*] 继续上一题，通过把符号插入到123456789876543 21，可以产生多少结果？例如$100 = -1234 - 5 - 6 + 7898 - 7 - 6543 - 2 - 1$。

Tin tan din dan bim bam bom bo —
tan tin din dan bam bim bo bom —
tin tan dan din bim bam bom bo —
tan tin dan din bam bim bo bom —
tan dan tin bam din bo bim bom —
… Tin tan din dan bim bam bom bo.

——多萝西·赖·塞耶斯（Dorothy L. Sayers），*The Nine Tailors* (1934)

一个十进数字的排列不过是这样一个10位十进数，其中所有数字都不同。因此我们所需要做的是产生所有10位数字的数，而且选择出其数字不同的那些来。这样高速的计算使我们免除了单调的思考。这岂非十分精彩！我们只须编程$k+1 \rightarrow k$，并且考察非所希望等式的k的数字。这也给了我们在词典顺序下的排列。在第二次清醒的思考后……我们确实需要考虑别的什么。

——德·亨·莱默（1957）

7.2.1.2 生成所有排列

在n元组之后，对于组合生成，几乎每个人都希望列出的下一个最重要的项目是，访问某个给定集合或多重集合的所有排列。为解决这个问题已经设计出许多不同的方法。事实上，对于未排序所发表的许多不同的算法几乎和排序的一样多！通过以既简单又灵活的一个经典方法开始，在这一小节中我们将研究最重要的排列生成。

算法L（词典顺序的排列生成） 给定n个元素$a_1a_2\cdots a_n$的一个序列，开始时它们已排序使得

$$a_1 \leqslant a_2 \leqslant \cdots \leqslant a_n \tag{1}$$

这个算法生成$\{a_1, a_2, \cdots, a_n\}$的所有排列，以词典顺序访问它们。例如，在词典顺序下$\{1, 2, 2, 3\}$的排列是 (1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221) 为方便起见，假定一个辅助的元素a_0；a_0必须严格地小于最大元素a_n。

39 L1 [访问。] 访问排列$a_1a_2\cdots a_n$。

L2 [寻找j。] 置$j \leftarrow n-1$。如果$a_j \geqslant a_{j+1}$，就反复地对j减1直到$a_j < a_{j+1}$为止。如果$j = 0$则结束这个算法。（这时j是由$a_1\cdots a_j$开始我们已经访问过的所有排列中最小的下标，因此按词典顺序下一个排列将增加a_j的值。）

L3 [增加a_j。] 置$l \leftarrow n$。如果$a_j \geqslant a_l$，就反复地对l减1直到$a_j < a_l$为止。然后交换$a_j \leftrightarrow a_l$。（因为$a_{j+1} \geqslant \cdots \geqslant a_n$，元素$a_l$是在一个排列当中大于可以合法地跟在$a_1\cdots a_{j-1}$后边的$a_j$的最小元素。在交换之前，我们有$a_{j+1} \geqslant \cdots \geqslant a_{l-1} \geqslant a_l > a_j \geqslant a_{l+1} \geqslant \cdots \geqslant a_n$；在交换之后我们有$a_{j+1} \geqslant \cdots \geqslant a_{l-1} \geqslant a_j > a_l \geqslant a_{l+1} \geqslant \cdots \geqslant a_n$。）

L4 [颠倒$a_{j+1}\cdots a_n$。] 置$k \leftarrow j+1$和$l \leftarrow n$。则如果$k < l$，就交换$a_k \leftrightarrow a_l$，置$k \leftarrow k+1$，$l \leftarrow l-1$，并且重复直到$k \geqslant l$为止。返回到L1。 ▮

这个算法回溯到14世纪印度的纳拉雅纳·潘迪塔（Nārāyana Pandita）（参见7.2.1.7节）。它也出现在卡·弗·辛登柏格（C. F. Hindenburg）为克·弗·鲁迪吉尔（C. F. Rüdiger）所著*Specimen Analyticum de Lineis Curvis Secundi Ordinis*的序言中（Leipzig: 1784），xlvi-xlvii页。自那以后它就经常被重新发现。在步骤L2和L3中，在括弧里的注释说明为什么它有效。

一般地说，通过以下的三步过程可以得到任何组合模式$a_1\cdots a_n$在词典顺序下的后继：

1) 寻找可使a_j增加的最大的j；

2) 以最小可能的数量增加a_j；

3) 寻找把新的$a_1\cdots a_j$扩充成一个完备模式的在词典顺序下最小的方法。

如同在n元组生成情况下算法7.2.1.1M遵循其一般步骤那样，算法L在排列生成情况下也遵循这个一般步骤。当考虑其他各种类型的组合模式时，后面我们将看到许多进一步的实例。注意在步骤L4开始时，我们有$a_{j+1} \geqslant \cdots \geqslant a_n$。因此以当前

前缀$a_1\cdots a_j$开始的头一个排列是$a_1\cdots a_j a_n\cdots a_{j+1}$，而且步骤L4通过进行$\lfloor (n-j)/2\rfloor$个交换来产生它。

在实践中，当诸元素都不同时，L2在一半时间里发现$j=n-1$，因为在$n!$个排列中恰好有$n!/2$个有$a_{n-1}<a_n$。因此通过认识这个特殊情况，算法L可以加速而又不使它更加复杂化。（参见习题1。）类似地，当诸a不同时，$j<n-t$的概率仅为$1/t!$；因此在步骤L2～L4中的循环通常进行得很快。习题6一般地分析了运行时间，并且证明，即使当出现相等元素时，算法L仍相当有效，除非在多重集合$\{a_1, a_2, \cdots, a_n\}$中某些值的出现次数比其他的要多得多。

相邻交换。在7.2.1.1节中我们已经看到，对于生成n元组来说格雷编码是有优越性的，而当我们要生成排列时，类似的考虑也适用。对于一个排列来说，最简单的改变是相邻元素互换。而且从第5章我们知道，如果对于这样的交换我们构成一个适当的顺序，则任何排列都可排成顺序。（例如，在这种方式下算法5.2.2B有效。）因此，我们通过所有元素有序开始而后交换适当的相邻的对，可以向后进行并且得到任何期望的排列。

40

现在出现一个自然的问题：是否可能以这样一种方式来跑遍一个多重集合的所有排列，即在每步中只有两个相邻元素改变位置？如果可能的话，则考察所有可能的排列的整个程序通常将是较简单和较快捷的，因为它将只需要计算一个交换的效果而不是每次都重复处理一个全新的数组$a_1\cdots a_n$。

天啊，当多重集合有重复元素时，我们不能总是找到这样一个类似格雷编码的序列。例如，{1, 1, 2, 2}的6个排列通过相邻交换彼此以下列方式相连接在一起：

$$1122 \text{ — } 1212 \begin{array}{l} \diagup\ 2112\ \diagdown \\ \diagdown\ 1221\ \diagup \end{array} 2121 \text{ — } 2211 \tag{2}$$

这个图没有哈密顿通路。

但是大多数应用处理不同元素的排列，而且对于这个情况，有一个好消息：一个简单的算法使得有可能通过恰好进行$n!-1$次相邻的交换来生成所有$n!$个排列。而且，另外一个这样的交换就返回到起始点，所以我们有类似于格雷二进制编码的一个哈密顿循环。

想法是对于{1，…，$n-1$}来构造这样的序列，并且以所有的方法来把数n插入到每个排列中。例如，如果$n=4$，当把4插入到所有4个可能的位置时，则序列（123，132，312，321，231，213）导致以下数组的诸列：

$$\begin{array}{cccccc} 1234 & 1324 & 3124 & 3214 & 2314 & 2134 \\ 1243 & 1342 & 3142 & 3241 & 2341 & 2143 \\ 1423 & 1432 & 3412 & 3421 & 2431 & 2413 \\ 4123 & 4132 & 4312 & 4321 & 4231 & 4213 \end{array} \tag{3}$$

现在通过在第一列中向下读，而在第二列中向上读，在第三列中向下读……在最后一列中向上读，我们就得到所希望的序列：(1234，1243，1423，4123，4132，

1432，1342，1324，3124，3142，…，2143，2134)。

在5.1.1节中，我们研究了一个排列的逆，即在无序中的元素（不必相邻）的对。相邻元素的每一次交换就使逆的个数±1。事实上，当我们考虑习题5.1.1-7中所谓的逆表$c_1\cdots c_n$时，其中c_j是比j小的处于j右边的元素的个数，我们发现(3)中的排列有下列反演表:

$$\begin{array}{cccccc} 0000 & 0010 & 0020 & 0120 & 0110 & 0100 \\ 0001 & 0011 & 0021 & 0121 & 0111 & 0101 \\ 0002 & 0012 & 0022 & 0122 & 0112 & 0102 \\ 0003 & 0013 & 0023 & 0123 & 0113 & 0103 \end{array} \tag{4}$$

因此，如果我们像前边那样，交替地向下和向上来读这些列，则我们就如同在
41 7.2.1.1节的等式(46)~(51)那样精确地得到混合进制（1, 2, 3, 4）的反射格雷编码。如同由埃·怀·狄伊克斯特拉所说明的那样，对于所有的n，同一性质成立[*Acta Informatica* 6(1976), 357-359]。而这导致了以下的系统阐述。

算法P（*平易的改动*） 给定n个不同元素的一个序列$a_1a_2\cdots a_n$，通过重复地交换相邻的对，本算法生成它的所有排列。它使用一个辅助的数组$c_1c_2\cdots c_n$，它如同上面所述表示反演，并跑遍所有整数序列使得对于$1\leqslant j\leqslant n$，

$$0\leqslant c_j<j \tag{5}$$

另一个数组$o_1o_2\cdots o_n$支配着条目c_j改变的方向。

P1. [初始化。] 对于$1\leqslant j\leqslant n$，置$c_j\leftarrow 0$，以及$o_j\leftarrow 1$。

P2. [访问。] 访问排列$a_1a_2\cdots a_n$。

P3. [准备改动。] 置$j\leftarrow n$和$s\leftarrow 0$。(以下步骤确定c_j有待改动的坐标j，并且保持(5)成立。变量s是使得$c_k=k-1$的下标$k>j$的个数。)

P4. [准备好改动了？] 置$q\leftarrow c_j+o_j$。如果$q<0$，则转到P7；如果$q=j$，则转到P6。

P5. [改动。] 交换$a_{j-c_j+s}\leftrightarrow a_{j-q+s}$，然后置$c_j\leftarrow q$并返回P2。

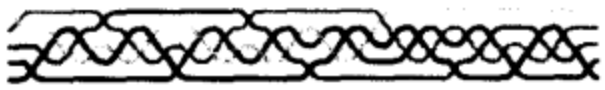
P6. [增加s。] 如果$j=1$，则结束；否则置$s\leftarrow s+1$。

P7. [转向。] 置$o_j\leftarrow -o_j$，$j\leftarrow j-1$并返回P4。 ∎

这个过程，起源于17世纪的英国，它显然对于所有$n\geqslant 1$都有效。那时敲钟者开始了以所有可能的排列敲响一组钟的好习惯。他们把*平易的改动*叫做算法P。图18a示出“剑桥48响”，这是在17世纪初期在平易改动原理揭示如何实现所有5! = 120种可能性之前，使用过的对于5个钟的一个无规则的和特殊的48个排列的顺序。算法P悠久的历史可追溯到写于大约1653年的现在保存在Bodleian图书馆的彼得·曼迪（Peter Mundy）的一篇手稿。后由欧内斯特·莫里斯（Ernest Morris）改写发表在[*The History and Art of Change Ringing*（改变钟响的历史和艺术）(1931), 29-30]中。不久之后，一本叫做*Tintinnalogia*（钟声叮当）的著名的书，匿名地发表于

1668年。不过现在已经知道它是由理查德·达克沃思（Richard Duckworth）和法比安·施蒂德曼（Fabian Stedman）写成的。该书的前60页专门来详细描述平易改动，从$n = 3$开始讲述直到任意大的n的情况。

> 许多年来，剑桥的48响，是被敲响或发明的伟大洪鸣。但是，现在，无论是48响还是100响，乃至720响，没有任何一个数可限制我们。因为我们可以作无穷的响声的改动。……对于4个钟，共有24种响声的改动，在这些钟的响声中有一个叫做振荡钟，而其他三个叫做端钟。振荡钟运动，并且连续地向上和向下追赶着……端钟的两个每次在振荡钟在它们之前或之后达到时都作一次改动。
>
> ——达克沃思和施蒂德曼，*Tintinnalogia*（1668） 42



（不完全）

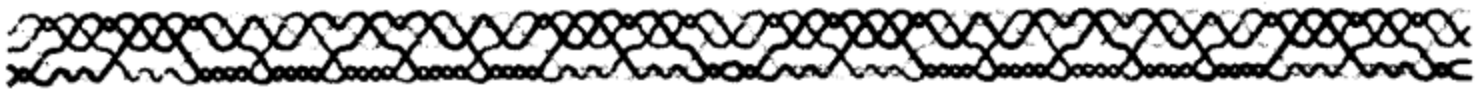
a) 剑桥48响



b) 平易的改动



c) 格兰西尔加倍



d) 施蒂德曼加倍

图18 在17世纪的英国曾使用过的对5个不同的教堂钟的响声进行排列的4种模式。模式b) 对应于算法P

英国热衷于钟声的人们很快就发展起更复杂的方案，其中两对或以上钟声改动同时发生。例如他们设计了叫做格兰西尔（Grandsire）加倍的图18c的模式，被说成为“在5个钟上发出的被创作出来的最巧妙的洪鸣”［*Tintinnalogia*，95］。从音乐的观点看，这种奇妙的方法要比算法P更有趣，但是在计算机应用中它们却不大有用，因此我们在这里不去讨论它们。感兴趣的读者们可以通过阅读威·乔·威尔森（W. G. Wilson）的书*Change Ringing*（钟声改动）（1965）学到更多东西。也可参见［阿·托·怀特（A. T. White），*AMM* 103 (1996)，771-778］。

哈·弗·特洛特（H. F. Trotter）在［*CACM* 5 (1962), 434-435］上发表了头一个平易改动的计算机实现。这个算法十分有效，特别是当它像在习题16中那样被流水线化时，因为每个n排列中的$n - 1$个不使用步骤P6和P7而被生成。相反，算法L只在大约一半时间里拥有最好的情况。

算法P在每次访问中恰做一次这一事实意味着它所生成的排列交替地为偶的和奇的（参见习题5.1.1-13）。因此我们通过简单地绕开奇数的那些就能生成全部为

偶数的排列。事实上，如同我们做过的那样，c和o表使得记住反演的当前的总数$c_1 + \cdots + c_n$变得很容易。

许多程序需要反复地生成相同的排列，因而在这次的情况下我们不能每次都跑遍算法P的诸步骤。使用以下方法，我们可以简单地准备一个适当的转换表。

算法T（*平易改动转换*） 这个算法计算一个表$t[1], t[2], \cdots, t[n!-1]$，使得对于$1 \leqslant k < n$，算法P的动作等价于$a_{t[k]} \leftrightarrow a_{t[k]+1}$逐次的交换。我们假设$n \geqslant 2$。

43 T1. [初始化。] 置$N \leftarrow n!$，$d \leftarrow N/2$，$t[d] \leftarrow 1$以及$m \leftarrow 2$。

T2. [对m进行循环。] 如果$m = n$则结束。否则置$m \leftarrow m+1$，$d \leftarrow d/m$以及$k \leftarrow 0$。（我们维持条件$d = n!/m!$。）

T3. [向下追赶。] 置$k \leftarrow k + d$和$j \leftarrow m-1$，然后当$j > 0$时，置$t[k] \leftarrow j$，$j \leftarrow j-1$以及$k \leftarrow k+d$，直到$j = 0$为止。

T4. [偏离。] 置$t[k] \leftarrow t[k] + 1$以及$k \leftarrow k + d$。

T5. [向上追赶。] 当$j < m-1$时置$j \leftarrow j+1$，$t[k] \leftarrow j$以及$k \leftarrow k+d$。如果$k < N$则返回到T3，否则返回到T2。 ∎

例如，如果$n = 4$我们得到表$(t[1], t[2], \cdots, t[23]) = (3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3)$。

字母算术。现在让我们考虑简单的一种难题，其中排列是有用的。如果每个字母代表不同的十进数字，以下模式：

$$\begin{array}{r} \texttt{SEND} \\ +\,\texttt{MORE} \\ \hline \texttt{MONEY} \end{array} \tag{6}$$

怎么能表示一个正确的和？[亨·欧·杜德尼（H. E. Dudeney），*Strand* 68(1924)，97，214。] 这样一个难题通常被叫做“字母算术”，这是由詹·阿·霍·汉特（J. A. H. Hunter）所杜撰的词[*Globe and Mail*（Toronto: 1955年10月27日），27]。另一个术语叫做“密码算术”，是由西·瓦特里宽特（S. Vatriquant）提出的[*Sphinx* 1 (1931年5月)，50]。

经典字母算术(6)可以很容易地通过手算来求解（参见习题21），但是让我们设想，我们要来处理大量的复杂的字母算术，其中有些可能无解而其他则可能有几十个解。于是通过在一台计算机上进行程序设计来尝试出匹配一个给定模式的数字的所有排列，并看看哪些排列产生出一个正确的和，那我们就可以节省时间了。[约翰·贝德勒（John Beidler）在*Creative Computing*（创造性计算）4, 6（1978年11月、12月号），110-113页上发表了解字母算术的一个计算机程序。]

我们也可以稍微地扩大我们的视野并且考虑一般的加法性字母算术，不仅处理像(6)这样简单的和，而且处理像以下这样的例子

$$\texttt{VIOLIN}+\texttt{VIOLIN}+\texttt{VIOLA}=\texttt{TRIO}+\texttt{SONATA}$$

（小提琴＋小提琴＋中提琴＝三重＋合奏）

等价地，我们要来解如同

$$2(\mathtt{VIOLIN})+\mathtt{VIOLA}-\mathtt{TRIO}-\mathtt{SONATA}=0 \tag{7}$$

这样的难题，其中给出了带有整系数的项之和，而目的是用不同的十进数字替换不同的字母来得到零。在这样一个问题中，每一个字母都有通过用1替换该字母，而用0替换其他字母得到的一个“签名”。例如，对于(7)中I的签名为

$$2(010010)+01000-0010-000000$$

即21010。如果我们把代码（1, 2, …, 10）任意地指定给字母（V, I, O, L, N, A, T, R, S, X），则对应于(7)的分别的签名为:

$$\begin{aligned} &s_1=210000 \quad s_2=21010 \quad s_3=-7901 \quad s_4=210 \quad s_5=-998\\ &s_6=-100 \quad s_7=-1010 \quad s_8=-100 \quad s_9=-100000 \quad s_{10}=0 \end{aligned} \tag{8}$$

44

（一个附加的字母X已被加上，因为我们需要10个它们。）现在的问题是找出 $\{0, 1, \cdots, 9\}$ 的所有排列，使得:

$$a\cdot s=\sum_{j=1}^{10}a_j s_j=0 \tag{9}$$

也还有一附加的条件，因为在字母算术中，不应该有零作为一个前导数字。例如，和数

$$\begin{array}{r} 7316\\ +\ 0823\\ \hline 08139 \end{array} \quad \text{以及} \quad \begin{array}{r} 5731\\ +\ 0647\\ \hline 06378 \end{array} \quad \text{以及} \quad \begin{array}{r} 6524\\ +\ 0735\\ \hline 07259 \end{array} \quad \text{以及} \quad \begin{array}{r} 2817\\ +\ 0368\\ \hline 03185 \end{array}$$

以及许多其他的都不认为是(6)的正确解。一般地说，存在头字母的一个集合F，使得我们必须有，对于所有$j\in F$，

$$a_j\neq 0 \tag{10}$$

对应于(7)和(8)的集合F是$\{1, 7, 9\}$。

处理一类加法性字母算术的一个方法是通过使用算法T开始来准备10! − 1个转换$t[k]$的一种表。然后，对于由一个签名序列（s_1, …, s_{10}）和头字母集合F所定义的每个问题，我们可以按如下步骤穷尽地寻找解:

A1. [初始化。] 置 $a_1a_2\cdots a_{10}\leftarrow 01\cdots 9$, $v\leftarrow\sum_{j=1}^{10}(j-1)s_j$, $k\leftarrow 1$，以及对于$1<j<10$，$\delta_j\leftarrow s_{j+1}-s_j$。

A2. [测试。] 如果$v=0$和如果(10)成立，则输出解$a_1\cdots a_{10}$。

A3. [交换。] 如果$k=10!$则停止，否则置$j\leftarrow t[k]$，$v\leftarrow v-(a_{j+1}-a_j)\delta_j$, $a_{j+1}\leftrightarrow a_j$, $k\leftarrow k+1$并返回A2。 ∎

步骤A3由以下事实，即a_j和a_{j+1}交换只不过使$a\cdot s$减少 $(a_{j+1}-a_j)(s_{j+1}-s_j)$ 而成立。尽管10!是3 628 800，是一个相当大的数，但在步骤A3中的操作是如此简单，因而

整个工作在一台现代计算机上只花费一秒钟的一个小数。

如果字母算术有惟一的解，则说它是纯的。不幸的是(7)不是纯的，排列1764802539和3546281970都适合(9)和(10)，因此我们有

$$176478 + 176478 + 17640 = 2576 + 368020$$

和

$$354652 + 354652 + 35468 = 1954 + 742818$$

而且在(8)中，$s_6 = s_8$。所以我们通过交换对A和R指定的数字，可以得到另外两个解。

另一方面(6)是纯的，而且我们已经描述的方法将找出求解它的两个不同排列，原因是(6)仅涉及8个不同的数字。因此通过使用两个虚拟签名$s_9 = s_{10} = 0$，我们将把它置为解。一般地说，带有m个不同字母的一个字母算术将有$10-m$个虚拟签名$s_{m+1} = \cdots = s_{10} = 0$，而且除非我们坚持比如说$a_{m+1} < \cdots < a_{10}$，否则它的每个解将被找到$(10-m)!$次。

45

一般的框架。对于生成不同对象的排列，已经提出了大量的算法，而且理解这些算法最好的方法是应用我们在1.3.3节中学习过的排列的乘法性质。为此目的，我们通过使用以0开始的下标稍微地改变我们的记号，并把$a_0a_1\cdots a_{n-1}$，写作$\{0, 1, \cdots, n-1\}$的排列，以代替写$a_1a_2\cdots a_n$表示$\{1, 2, \cdots, n\}$的排列。更重要的是，我们将考虑生成排列的方案，在这个方案中大多数的动作发生在左边，使得对于$1 \leqslant k \leqslant n$，$\{0, 1, \cdots, k-1\}$的所有排列将在头$k!$步期间被生成。例如，对于$n = 4$，一个这样的方案是:

$$\begin{aligned} &0123, 1023, 0213, 2013, 1203, 2103, 0132, 1032, 0312, 3012, 1302, 3102 \\ &0231, 2031, 0321, 3021, 2301, 3201, 1230, 2130, 1320, 3120, 2310, 3210 \end{aligned} \tag{11}$$

这称为“颠倒的协词典顺序”，因为如果我们从右到左地反射这些串，我们得到3210, 3201, 3120, …, 0123，即词典顺序的颠倒。想像(11)的另一个方法是把这些条目看做$(n-a_n)\cdots(n-a_2)\ (n-a_1)$，其中$a_1a_2\cdots a_n$按词典顺序跑遍$\{1, 2, \cdots, n\}$的排列。

回想一下1.3.3节，像$\alpha = 250143$这样的一个排列可以写成两行形式

$$\alpha = \begin{pmatrix} 012345 \\ 250143 \end{pmatrix}$$

或者写成更紧凑的循环形式

$$\alpha = (02)(153)$$

且其意义是α取$0 \mapsto 2$, $1 \mapsto 5$, $2 \mapsto 0$, $3 \mapsto 1$, $4 \mapsto 4$以及$5 \mapsto 3$；像“(4)”这样的1循环不必指出，因为(4)是这个排列的一个固定点，因此我们说“α固定4”。我们也写$0\alpha = 2$, $1\alpha = 5$，等等。表示说$j\alpha$是“在α之下的j的映像”。像α乘以β，其中$\beta =$

543210这样排列的乘法，在两行形式下很容易进行

$$\alpha\beta = \begin{pmatrix}012345\\250143\end{pmatrix}\begin{pmatrix}012345\\543210\end{pmatrix} = \begin{pmatrix}012345\\250143\end{pmatrix}\begin{pmatrix}250143\\305412\end{pmatrix} = \begin{pmatrix}012345\\305412\end{pmatrix}$$

或者在循环形式下

$$\alpha\beta = (02)(153)\cdot(05)(14)(23) = (0341)(25)$$

注意在$\alpha\beta$之下1的映像是$1(\alpha\beta) = (1\alpha)\beta = 5\beta = 0$，等等。**警告**：大约有一半的书籍是以另一种方式来处理排列乘法的（即从右到左），并且想像$\alpha\beta$为，在应用α之前先应用β。其原因是由于传统的函数记号，其中如果人们写$\alpha(1) = 5$，就使得自然会想$\alpha\beta(1)$应当意味着$\alpha(\beta(1)) = \alpha(4) = 4$。然而本书赞成另一个原理，因此我们总是从左到右地进行排列乘法。

当通过数的数组来表示排列时，则应当仔细地理解乘法的顺序。例如，如果我们“应用”反射$\beta = 543210$到排列$\alpha = 250143$，则结果341052不是$\alpha\beta$而是$\beta\alpha$。 46
一般地说，通过某个重新安排$a_{0\beta}a_{1\beta}\cdots a_{(n-1)\beta}$来代替一个排列$\alpha = a_0a_1\cdots a_{n-1}$的操作取$k \mapsto a_{k\beta} = k\beta\alpha$。通过$\beta$来排列位置对应于通过$\beta$的**预乘法**，并把$\alpha$改变成$\beta\alpha$；通过$\beta$排列值对应于通过$\beta$的**后乘法**，并把$\alpha$改成$\alpha\beta$。因此，例如交换$a_1 \leftrightarrow a_2$的排列生成程序是通过(1 2)预乘当前的排列，通过(a_1a_2)后乘它。

遵循1830年由伊瓦里斯特·伽罗华（Évariste Galois）所提的建议，排列的一个非空集合G说是形成一个**群**，如果在乘法运算下它是封闭的，即当α和β是G的元素时，乘积$\alpha\beta$在G中。[参见*Écrits et Mémoires Mathématiques d'Évariste Galois*（伽罗华数学纪念文集）(Paris: 1962)，47。] 例如，考虑如同在习题7.2.1.1-17中那样被表示为4 × 4圆环体

0	1	3	2
4	5	7	6
c	d	f	e
8	9	b	a

(12)

的4立体，并设G是保持相邻性的顶点{0, 1, …, f}的所有排列的集合：一个排列α在G中当且仅当u —— v意味着$u\alpha$ ——$v\alpha$在4立体中。（这里我们用十六进制数字(0, 1, …, f)来代表整数 (0, 1, …, 15)。(12)中的标号被选择成使得u —— v当且仅当u和v仅在一个二进位位置上不同。）这个集合G显然是一个群，而且它的元素叫做4立体的对称的或“自同构”的。

借助于由查尔斯·科·西姆斯（Charles C. Sims）[*Computational Methods in Abstract Algebra*（抽象代数中的计算方法）(Oxford: Pergamon, 1970)，169-183]所介绍的**西姆斯表**，在一台计算机内可方便地表示排列G的群，它是具有以下性质的G的子集$S_1, S_2, \cdots$的族：S_k恰含一个排列σ_{kj}，每当G包含这样一个排列时，它使$k \mapsto j$，并且固定所有大于k的元素。我们令σ_{kk}为恒等排列，它总在G中存在；但当$0 \leq j < k$时，任何适当的排列都可选择来起σ_{kj}的作用。西姆斯表的主要优点是它提供了整个群的一个方便的表示：

引理S 令$S_1, S_2, \cdots, S_{n-1}$是$\{0, 1, \cdots, n-1\}$上排列的一个群$G$的西姆斯表，则$G$的每个元素$\alpha$有惟一的表示

$$\alpha = \sigma_1\sigma_2\cdots\sigma_{n-1}，\ 其中对于1 \leqslant k < n，\ \sigma_k \in S_k \tag{13}$$

证明 如果α有这样一个表示而且如果σ_{n-1}是排列$\sigma_{(n-1)j} \in S_{n-1}$，则$\alpha$使$n-1 \mapsto j$。因为$S_1 \cup \cdots \cup S_{n-2}$的所有元素都固定$n-1$的值。反过来，如果$\alpha$使$n-1 \mapsto j$，我们有$\alpha = \alpha'\sigma_{(n-1)j}$；其中

$$\alpha' = \alpha\sigma^{-}_{(n-1)j}$$

是固定$n-1$的G的一个排列。（如同在1.3.3节中那样，σ^{-}表示σ的反演。）所有这样排列的集合G'是一个群，而且$S_1, \cdots, S_{n-2}$是G'的一个西姆斯表；因此通过对n的归
47 纳法可得出结果。 ∎

例如，一点计算就可证明对于4立体的自同构群，一个可能的西姆斯表是

$$\begin{aligned}
S_{\mathtt{f}} &= \{(), (\mathtt{01})(\mathtt{23})(\mathtt{45})(\mathtt{67})(\mathtt{89})(\mathtt{ab})(\mathtt{cd})(\mathtt{ef}), \ldots, \\
&\qquad\qquad (\mathtt{0f})(\mathtt{1e})(\mathtt{2d})(\mathtt{3c})(\mathtt{4b})(\mathtt{5a})(\mathtt{69})(\mathtt{78})\}; \\
S_{\mathtt{e}} &= \{(), (\mathtt{12})(\mathtt{56})(\mathtt{9a})(\mathtt{de}), (\mathtt{14})(\mathtt{36})(\mathtt{9c})(\mathtt{be}), (\mathtt{18})(\mathtt{3a})(\mathtt{5c})(\mathtt{7e})\}; \\
S_{\mathtt{d}} &= \{(), (\mathtt{24})(\mathtt{35})(\mathtt{ac})(\mathtt{bd}), (\mathtt{28})(\mathtt{39})(\mathtt{6c})(\mathtt{7d})\}; \\
S_{\mathtt{c}} &= \{()\}; \\
S_{\mathtt{b}} &= \{(), (\mathtt{48})(\mathtt{59})(\mathtt{6a})(\mathtt{7b})\}; \\
S_{\mathtt{a}} &= S_{\mathtt{9}} = \cdots = S_{\mathtt{1}} = \{()\};
\end{aligned} \tag{14}$$

对于$0 \leqslant j \leqslant 15$，这里$S_{\mathtt{f}}$包含16个排列$\sigma_{\mathtt{f}j}$，对于$0 \leqslant i \leqslant 15$，它分别使$i \mapsto i \oplus (15-j)$。集合$S_{\mathtt{e}}$仅含四个排列，因为固定f的一个自同构必须使e成为f的一个邻居；因此e的映像必定是e或者d或者b或者是7。集合$S_{\mathtt{c}}$只含恒等排列，因为固定f，e和d的一个自同构也必定固定c。如同在这个例子中一样，对于所有小的k值，大多数群有$S_k = \{(\,)\}$。因此西姆斯表通常只需要含相当小数目的排列，尽管群本身可能十分大。

西姆斯表示(13)使得测试一个给定的排列α是否在G中变得容易：首先我们确定$\sigma_{n-1} = \sigma_{(n-1)j}$，其中$\alpha$使$n-1 \mapsto j$，而且我们令$\alpha' = \alpha\sigma^{-}_{n-1}$；然后我们确定$\sigma_{n-2} = \sigma_{(n-2)j'}$，其中$\alpha'$使$n-2 \mapsto j'$，而且我们令$\alpha'' = \alpha'\sigma^{-}_{n-2}$；等等。如果在任何阶段所要求的$\sigma_{kj}$在$S_k$中不存在，则原来的排列$\alpha$不属于$G$。在(14)的情况下，在找到$\sigma_{\mathtt{f}}$，$\sigma_{\mathtt{e}}$，$\sigma_{\mathtt{d}}$，$\sigma_{\mathtt{c}}$和$\sigma_{\mathtt{b}}$之后，这个过程必定把$\alpha$归结为单位。

例如，令α是排列(14)(28)(3c)(69)(7d)(be)，它对应于关于其主对角线{0, 5, f, a}的转置(12)。由于α固定f，$\sigma_{\mathtt{f}}$将是恒等排列( )，而且$\alpha' = \alpha$。于是$\sigma_{\mathtt{e}}$是使e$\mapsto$b的$S_{\mathtt{e}}$的成员，即(14)(36)(9c)(be)，而且我们发现$\alpha'' = (28)(39)(6c)(7d)$。这个排列属于$S_{\mathtt{d}}$，因此$\alpha$确实是4立体的一个自同构。

反之，(13)也使得生成对应群的所有元素变得容易，我们简单地跑遍形如

$$\sigma(1, c_1)\ \sigma(2, c_2)\cdots\sigma(n-1, c_{n-1})$$

的所有排列，其中$\sigma(k, c_k)$是S_k的第(c_k+1)个元素$(0 \leqslant c_k < s_k = |S_k|$以及$1 \leqslant k < n)$。而且使用7.2.1.1节中跑遍对于分别的基数$(s_1, \cdots, s_{n-1})$的所有$(n-1)$元组$(c_1, \cdots, c_{n-1})$的任何算法来做。

使用一般的框架。我们主要关心的是在$\{0, 1, \cdots, n-1\}$上的所有排列的群。而且在此情况下，一个西姆斯表的每一个集合S_k将包含$k+1$个元素$\{\sigma(k,0), \sigma(k,1), \cdots, \sigma(k,k)\}$，其中$\sigma(k,0)$是恒等排列，而其他排列使$k$取在某个顺序下的$\{0, \cdots, k-1\}$这些值。（排列$\sigma(k,j)$不必和$\sigma_{kj}$相同，而且通常它是不同的。）按照以下的概述，每个这样的西姆斯表导致一个排列生成。 48

算法G（一般的排列生成元） 给定一个西姆斯表$(S_1, S_2, \cdots, S_{n-1})$，如同刚刚描述过的那样，这里每个$S_k$有$k+1$个元素$\sigma(k,j)$。使用一个辅助的控制表$c_n \cdots c_2 c_1$，这个算法生成$\{0, 1, \cdots, n-1\}$的所有排列$a_0 a_1 \cdots a_{n-1}$。

G1. [初始化。] 对于$0 \leqslant j < n$，置$a_j \leftarrow j$和$c_{j+1} \leftarrow 0$。

G2. [访问。]（这时混合进制数$\left[\begin{smallmatrix} c_{n-1}, \cdots, c_2, c_1 \\ n, \cdots, 3, 2 \end{smallmatrix}\right]$是迄今所访问的排列数。）访问排列$a_0 a_1 \cdots a_{n-1}$。

G3. [对$c_n \cdots c_2 c_1$加1。] 置$k \leftarrow 1$。如果$c_k = k$，置$c_k \leftarrow 0$，$k \leftarrow k+1$，并且重复直到$c_k < k$为止。如果$k = n$，则结束本算法。否则置$c_k \leftarrow c_k + 1$。

G4. [排列。] 如同下面说明的那样，应用排列$\tau(k, c_k)\ \omega(k-1)^-$到$a_0 a_1 \cdots a_{n-1}$上，并返回G2。 ∎

应用一个排列π于$a_0 a_1 \cdots a_{n-1}$上意味着对于$0 \leqslant j < n$，以$a_{j\pi}$来代替a_j；如同早先说明过的那样，这对应于通过π的预乘法。我们定义，

$$\tau(k,j) = \sigma(k,j)\ \sigma(k,j-1)^-, \text{ 对于} 1 \leqslant j \leqslant k \tag{15}$$

$$\omega(k) = \sigma(1,1) \cdots \sigma(k,k) \tag{16}$$

于是步骤G3和G4维持以下性质

$$a_0 a_1 \cdots a_{n-1} \text{是排列} \sigma(1, c_1)\ \sigma(2, c_2) \cdots \sigma(n-1, c_{n-1}) \tag{17}$$

而且引理S证明每个排列恰被访问一次。

图19中的树图示在$n = 4$情况下的算法G。按照(17)，$\{0, 1, 2, 3\}$的每一个排列$a_0 a_1 a_2 a_3$对应于三个数字的控制串$c_3 c_2 c_1$，且$0 \leqslant c_3 \leqslant 3$，$0 \leqslant c_2 \leqslant 2$，以及$0 \leqslant c_1 \leqslant 1$。这棵树的某些节点用一个数字$c_3$来标号；这些对应于被使用的西姆斯表的排列$\sigma(3, c_3)$，其他节点用两个数字$c_3 c_2$来标号，对应于排列$\sigma(2, c_2)\ \sigma(3, c_3)$。粗线连接节点$c_3$到节点$c_3 0$和连接节点$c_3 c_2$到节点$c_3 c_2 0$，因为$\sigma(2,0)$和$\sigma(1,0)$都是恒等排列，而且这些节点实质上是等价的。在步骤G3中，对混合进制数$c_3 c_2\ c_1$加1对应于在前根顺序下从图19的一个节点移动到它的后继，相应地步骤G4中的变换改变这些排列。例如，当$c_3 c_2 c_1$从121变动到200时，步骤G4以

$$\tau(3,2)\ \omega(2)^- = \tau(3,2)\ \sigma(2,2)^-\ \sigma(1,1)^- \qquad 49$$

来预乘当前的排列；以$\sigma(1,1)^-$预乘使我们从节点121变成节点12，以$\sigma(2,2)^-$预乘使我们从节点12变成节点1，而以$\tau(3,2)=\sigma(3,2)\,\sigma(3,1)^-$预乘使我们从节点1变成节点$2\equiv 200$，它是节点121的前根顺序的后继。以另一种方式来表达这一点，以$\tau(3,2)\,\omega(2)^-$预乘恰好是在保持(17)的同时，把$\sigma(1,1)\,\sigma(2,2)\,\sigma(3,1)$改变成$\sigma(1,0)\,\sigma(2,0)\,\sigma(3,2)$所需要的。

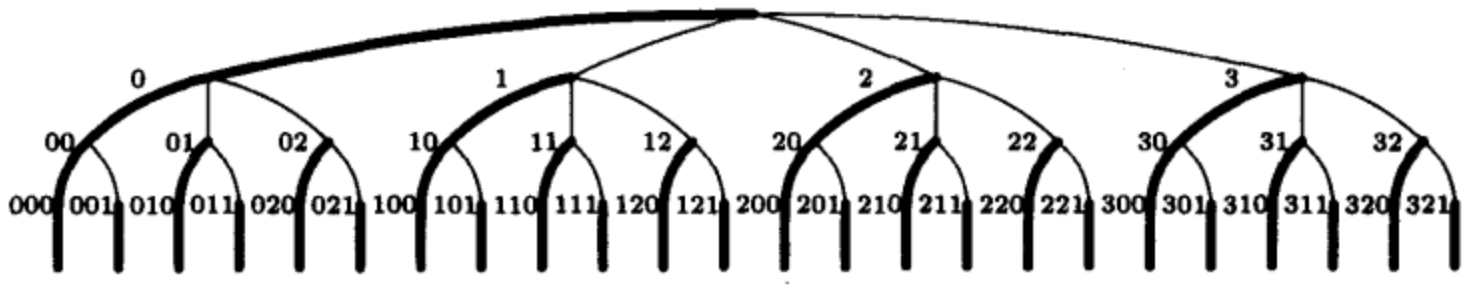


图19 当$n=4$时算法G含蓄地遍历这棵树

算法G定义了大量的排列生成元（参见习题37），所以难怪，在参考文献中出现有它的许多特殊情况。当然，它的某些变形比其他的要有效得多，因此我们要寻求这样的例子，即其操作特别适用于我们正在使用的计算机。

例如，我们通过令$\sigma(k,j)$是$(j+1)$循环

$$\sigma(k,j)=(k-j,k-j+1,\cdots,k) \tag{18}$$

可以得到算法G的特殊情况，即在颠倒的协词典顺序下的排列。原因在于对于$i\neq k$，当$c_k=j$和$c_i=0$时，$\sigma(k,j)$应该是对应于在颠倒的协词典顺序下的排列，而且这个排列$a_0a_1\cdots a_{n-1}$是$01\cdots(k-j-1)(k-j+1)\cdots(k)(k-j)(k+1)\cdots(n-1)$。例如，当$n=8$和$c_n\cdots c_1=00030000$时，对应的颠倒的协词典顺序的排列是01345267，在循环形式下它是(2345)。当$\sigma(k,j)$由(18)给出时，等式(15)和(16)导致公式

$$\tau(k,j)=(k-j\ \ k) \tag{19}$$

$$\omega(k)=(0\ 1)(0\ 1\ 2)\cdots(0\ 1\cdots k)=(0\ k)(1k-1)(2k-2)\cdots=\phi(k) \tag{20}$$

这里$\phi(k)$是把$a_0\cdots a_k$改变成$a_k\cdots a_0$的“$(k+1)$触发”。在此情况下，$\omega(k)$原来是和$\omega(k)^-$相同的，因为$\phi(k)^2=(\ )$。

在算法L和在它的颠倒的协词典顺序等价物（习题2）背后的意义下，等式(19)和(20)含蓄地存在，其中步骤L3实际上可应用于一个转置，步骤L4做一次触发，步骤G4实际上先做触发。但恒等式

$$(k-j\ k)\,\phi(k-1)=\phi(k-1)(j-1\ k) \tag{21}$$

表明，一个触发接着进行一次转置和一个（不同）的转置接着做一次触发是相同的。

事实上，等式(21)是重要的恒等式

$$\pi^-(j_1\,j_2\cdots j_t)\pi=(j_1\pi\,j_2\pi\cdots j_t\pi) \tag{22}$$

的一个特殊情况，它对于任何π和任何t循环$(j_1\,j_2\cdots j_t)$都正确。例如，在(22)的左边，我们有$j_1\pi\mapsto j_1\mapsto j_2\mapsto j_2\pi$，同右边的循环相一致。因此如果$\alpha$和$\pi$是无论什么排

列，排列$\pi^{-}\alpha\pi$（称为通过π的α的*共轭*）恰好有和α相同的循环结构；我们可简单地以来$j\pi$代替在每个循环中的每个元素j。

算法G的另一个重要的特殊情况是由理·詹·奥德-史密斯（R. J. Ord-Smith）引入的[*CACM* 10(1967), 452; 12(1969), 638; 也参见*Comp. J.* 14(1971), 136-139], 50
其算法通过置

$$\sigma(k, j) = (k \cdots 10)^{j} \tag{23}$$

得到。现在由(15)显然有

$$\tau(k, j) = (k \cdots 10) \tag{24}$$

而且再次地我们有

$$\omega(k) = (0\ k)(1\ k-1)(2\ k-2)\cdots = \phi(k) \tag{25}$$

因为$\sigma(k, k) = (0\ 1 \cdots k)$和以前一样。关于这一算法的好处在于步骤G4中所需要的排列，即$\tau(k, c_k)\omega(k-1)^{-}$不依赖于$c_k$:

$$\tau(k, j)\ \omega(k-1)^{-} = (k \cdots 1\ 0)\phi(k-1)^{-} = \phi(k) \tag{26}$$

因此，奥德-史密斯算法是算法L的特殊情况，其中步骤G4只不过交换$a_0 \leftrightarrow a_k$，$a_1 \leftrightarrow a_{k-1}$，…；这个操作通常很快，因为$k$很小，而且它节省了算法L的某些工作。（参见习题38以及在7.2.1.7节中乔·西·克鲁格尔（G. S. Klügel）的参考文献。）

通过对事情进行操纵，使得步骤G4要做的仅仅是每次进行一次转置，我们可以把事情做得更好些，这稍微有点像在算法P那样，但不必是对相邻的元素来做。许多这样的方案是可能的。而最好的大概是如同由布·理·希普（B. R. Heap）[*Comp. J.* 6(1963), 293-294]所建议的那样，令

$$\tau(k, j)\omega(k-1)^{-} = \begin{cases} (k\ \ 0), & \text{如果}k\text{为偶数} \\ (k\ \ j-1), & \text{如果}k\text{为奇数} \end{cases} \tag{27}$$

注意希普的方法总是转置$a_k \leftrightarrow a_0$，除非$k = 3, 5, \cdots$时，而且在每6步中有5步k的值是1或2。习题40证明希普的方法确实生成所有排列。

绕开不要的块。算法G值得指出的一个优点是在接触a_k之前，它跑遍$a_0 \cdots a_{k-1}$的所有排列，然后在再次改动a_k之前，它执行另一个k的循环，等等。因此，如果在任何时候，我们达到最后元素$a_k \cdots a_{n-1}$的一个设置，而它对于我们正在工作的问题并不重要，我们可以快速跳过以非所期望的后缀结尾的所有排列。更精确地说，我们可以用以下的子步骤来代替步骤G2:

G2.0 [可接受吗？] 如果$a_k \cdots a_{n-1}$不是一个可接受的后缀，则转到G2.1，否则置$k \leftarrow k-1$。然后如果$k > 0$，则重复这一步；如果$k = 0$，则进行到G2.2步。

G2.1 [跳过这后缀。] 如果$c_k = k$，则应用$\sigma(k, k)^{-}$到$a_0 \cdots a_{n-1}$，置$c_k \leftarrow 0$, $k \leftarrow k+1$，并重复直到$c_k < k$为止。如果$k = n$则结束；否则置$c_k \leftarrow c_k + 1$，应用$\tau(k,$

c_k) 到$a_0\cdots a_{n-1}$，并返回G2.0。

G2.2 [访问。] 访问排列$a_0\cdots a_{n-1}$。 ∎

步骤G1也应置$k\leftarrow n-1$。注意新步骤小心地保持条件(17)。本算法已经变得更
51 为复杂，因为除了出现在G4中的排列$\tau(k,j)\omega(k-1)^-$外，我们还需要知道排列$\tau(k,j)$和$\sigma(k,k)$。但是为额外的复杂化所付出的努力通常是值得的，因为得到的程序可能运行得快得多。

例如，图20表明，当对应于节点00, 11, 121以及2的$a_0a_1a_2a_3$的后缀不可接受时，图19的树发生什么情况。（排列$a_0\cdots a_{n-1}$的每一后缀$a_k\cdots a_{n-1}$对应于控制串$c_n\cdots c_1$的一个前缀$c_n\cdots c_k$，因为排列$\sigma(1,c_1)\cdots\sigma(k-1,c_{k-1})$不影响$a_k\cdots a_{n-1}$。）步骤G2.1以$\tau(k,j)$预乘，使得从节点$c_{n-1}\cdots c_{k+1}$ j移动到它的右兄弟$c_{n-1}\cdots c_{k+1}(j+1)$，而它通过预乘$\sigma(k,k)^-$从节点$c_{n-1}\cdots c_{k+1}$ k向上移动到它的父节点$C_{n-1}\cdots c_{k+1}$。因此，为了从被拒绝的前缀121到达它前根顺序下的后继，本算法通过预乘$\sigma(1,1)^-$、$\sigma(2,2)^-$以及$\tau(3,2)$，由此从节点121移动到12，移动到1，再移动到2。（这是稍微特殊的情况，因为仅当我们不要访问有后缀$a_1\cdots a_{n-1}$的惟一排列$a_0a_1\cdots a_{n-1}$时，才拒绝具有$k=1$的一个前缀。）在节点2被拒绝之后，$\tau(3,3)$使我们抵达节点3，等等。

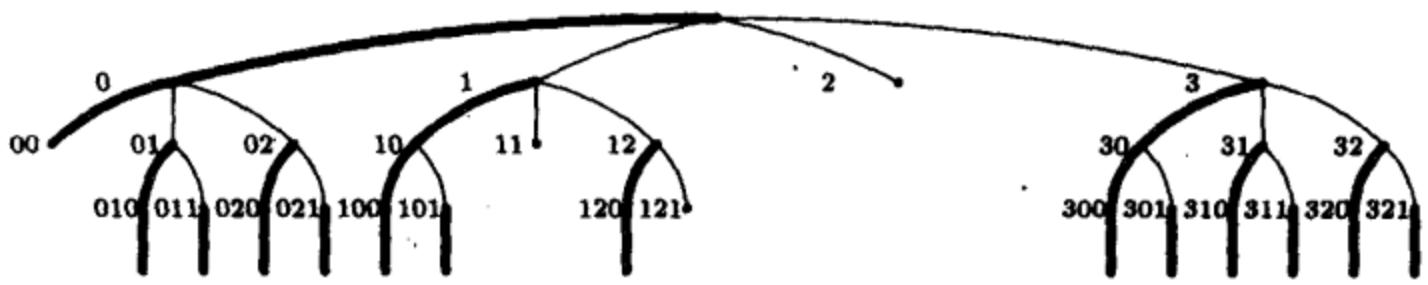


图20 如果把算法G适当地扩展，不要的分支可从图19中的树中剪去

顺便说一下，如果我们回到生成{1, …, n}的排列$a_1\cdots a_n$的思想并且在右端来进行大多数工作的话，绕开算法G的这一扩充中的后缀$a_k\cdots a_{n-1}$，实质上和绕开在我们原先的记号下的一个前缀$a_1\cdots a_j$是一样的。我们原来的记号对应于首先选择a_1，然后a_2……然后a_n；在算法G中的记号实质上首先选择a_{n-1}，然后a_{n-2}……然后a_0。算法G的约定可能看起来是落后的，但是它们使对于西姆斯表进行操作的公式简单得多。一个好的程序员很快就会毫无困难地从一个观点转到另一个观点去。

我们可以把这些思想应用到字母算术，因为比如说，很显然地，对于字母D, E和Y的大多数选择将使SEND和MORE相加成为MONEY变成不可能。在这个问题中我们需要有 (D + E − Y)mod 10 = 0，因此从考虑中可以删除许多排列。

一般地说，如果r_k是整除签名值s_k的10的极大次幂，则我们可以对字母进行排序并且指定编码{0, 1, …, 9}使得$r_0\geqslant r_1\geqslant\cdots\geqslant r_9$。例如，为了求解三奏鸣曲问题(7)，我们可以对 (S, V, A, R, I, L, T, O, N) 分别使用 (0, 1, …, 9)，并得到签名

$$s_0=0,\ s_1=-100000,\ s_2=210000,\ s_3=-100,\ s_4=-100,$$
52 $$s_5=21010,\ s_6=210,\ s_7=-1010,\ s_8=-7901,\ s_9=-998$$

因此 $(r_0,\cdots,r_9)=(\infty,5,4,2,2,1,1,1,0,0)$。现在如果我们对于一个$k$值到达步骤

G2.0且有$r_{k-1} \neq r_k$，我们就可以说，除非$a_k s_k + \cdots + a_9 s_9$是$10^{r_k - 1}$的倍数，否则后缀$a_k \cdots a_9$是不可接受的。而且，(10)告诉我们，如果$a_k = 0$和$k \in F$，$a_k \cdots a_9$是不可接受的；头字母集合$F$现在是$\{1, 2, 7\}$。

我们以前的字母算术方法连同上边的步骤A1 ~ A3使用了硬算来跑遍10!可能性。在此情况下它操作得稍微快些，因为相邻-转置方法允许它通过对于每个排列仅作6个存储访问来实现；不过，10!仍是3 628 800，所以尽管字母算术的问题解决了，但是整个过程仍然花费了几乎2200万存储访问。相反，通过希普方法和上边刚刚描述的，删除的扩充的算法G将只使用少于12万8千存储访问就找出(7)的所有4个解！因此，跳过后缀的技术比前边的方法快170多倍，它简直是飞快地胡乱一吹就奏效了。

在新的方法中12万8千内存的大多数花费在应用$\tau(k, c_k)$到步骤G2.1中。其他的存储访问主要来自于在该步骤中$\sigma(k, k)^-$的应用，但需要τ共7812次，而需要σ^-仅2162次。从图20容易理解其原因，因为在步骤G4中的“捷径移动”$\tau(k, c_k)\omega(k-1)^-$几乎不适用；在这种情况下，它仅被使用了4次，对于每个解使用一次。因此，对于此树的前根顺序的遍历几乎完全是通过向右移动的τ步和向上移动的σ^-步实现的。在类似于这个问题的一个问题里，τ步占支配地位，其中实际被访问的完备的排列很少，因为每步$\sigma(k, k)^-$前边有k步$\tau(k, 1), \tau(k, 2), \cdots, \tau(k, k)$。

这个分析揭示了希普方法——它走很长的路来优化排列$\tau(k, j)\omega(k-1)^-$，使得在步骤G4中的每个转换是一个简单的转置——除非在步骤G2.0中被拒绝的后缀相对地较少，否则对于扩充的算法G并非特别好。较为简单的颠倒的协词典顺序，对于它来说$\tau(k, j)$本身总是一个简单的转置，现在要更有吸引力（参见(19)）。确实，带有颠倒协词典顺序的算法G仅用9万7千存储访问就解决字母算术(7)。

其他的字母算术问题中也有类似的结果。比如说，如果我们应用扩充的算法G到习题24中的字母算术问题的部分(a)到(h)，计算分别涉及：

使用希普方法 (551, 110, 14, 8, 350, 84, 153, 1598)千存储访问
使用颠倒的协词典顺序 (429, 84, 10, 5, 256, 63, 117, 1189) 千存储访问 (28)

在这些例子中，同算法T的硬算相比，对于颠倒的协词典顺序的加速因子，从情况(h)的18直到情况(d)中的4200在内，平均而言它大约是80；希普方法则给出大约60的平均加速。

然而，从算法L我们得知，如果没有使用算法G的控制表$c_n \cdots c_1$的复杂性，词典顺序是易于处理的。而且对算法L进一步的观察表明，通过使用一个链接表而不是使用一个顺序的数组，当经常地跳过排列时，我们可以改进算法L的特性。改进的算法很好地适合于希望生成被限制的排列类的广泛算法。 53

算法X（*带有限制的前缀的词典顺序的排列*） 这个算法生成$\{1, 2, \cdots, n\}$的所有排列$a_1 a_2 \cdots a_n$，它通过一个给定的测试序列

$$t_1(a_1), t_2(a_1, a_2), \cdots, t_n(a_1, a_2, \cdots, a_n)$$

并且以词典顺序访问它们。它使用链接$l_0, l_1, \cdots, l_n$的一个辅助表来维持不用元素的一个循环表，使得如果当前可利用元素是：

$$\{1, \cdots, n\} \backslash \{a_1, \cdots, a_k\} = \{b_1, \cdots, b_{n-k}\}，\text{其中} b_1 < \cdots < b_{n-k} \tag{29}$$

则我们有

$$l_0 = b_1，\text{对于} 1 \leqslant j < n-k，l_{bj} = b_{j+1}，\text{以及} l_{b_{n-k}} = 0 \tag{30}$$

它也使用一个辅助表$u_1 \cdots u_n$来返工在l数组上我们已经实施的那些操作。

X1. [初始化。] 对于$0 \leqslant k < n$置$l_k \leftarrow k+1$以及$l_n \leftarrow 0$，然后置$k \leftarrow 1$。

X2. [进入级k。] 置$p \leftarrow 0$，$q \leftarrow l_0$。

X3. [测试$a_1 \cdots a_k$。] 置$a_k \leftarrow q$。如果$t_k(a_1, \cdots, a_k)$为假，则转到X5。否则，如果$k = n$，则访问$a_1 \cdots a_n$并转到X6。

X4. [增加k。] 置$u_k \leftarrow p, l_p \leftarrow l_q, k \leftarrow k+1$，并返回X2。

X5. [增加a_k。] 置$p \leftarrow q$，$q \leftarrow l_p$。如果$q \neq 0$，则返回到X3。

X6. [减小k。] 置$k \leftarrow k-1$，而且如果$k = 0$，则结束。否则置$p \leftarrow u_k$，$q \leftarrow a_k$，$l_p \leftarrow q$并返回到X5去。 ∎

这个优美算法的基本思想是由余明昭（M. C. Er）提出的[*Comp. J.*30 (1987), 282]。通过稍微改变记号，我们可以把它应用到字母算术上，得到{0, …, 9}的排列$a_0 \cdots a_9$并让l_{10}起以前l_0的作用。为求解三奏鸣曲问题(7)，得到的算法只需4万9千存储访问，而且它分别以

$$(248, 38, 4, 3, 122, 30, 55, 553)\text{ 千个存储访问} \tag{31}$$

来解习题24(a) ~ (h) 的字母算术问题。因此它比硬算方法的运行时间要快大约165倍。

应用算法X到字母算术的另一个方法通常还要更快（参见习题49）。

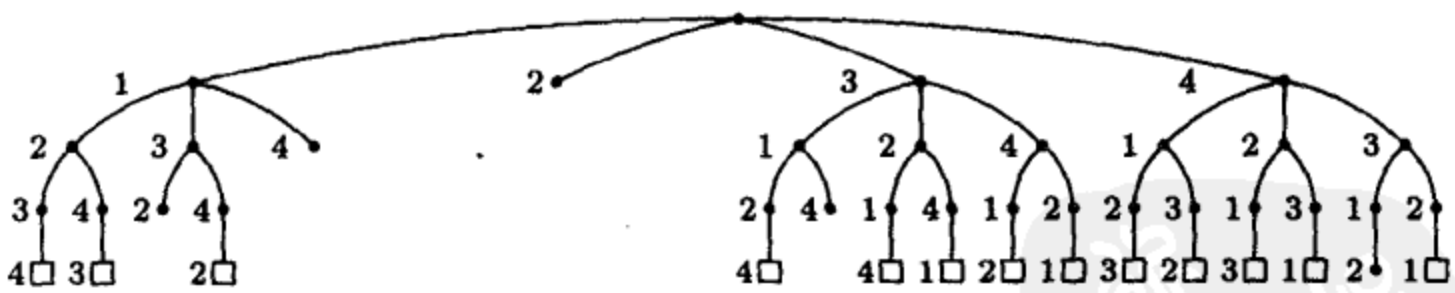


图21 当$n = 4$时，如果除了以132, 14, 2, 314或4312开始的之外，所有的排列都被访问，则由算法X含蓄地遍历的树

54

***对偶方法**。如果$S_1, \cdots, S_{n-1}$是对于排列群G的一个西姆斯表，在引理S中我们知道，G的每一个元素均可惟一地被表达为一个乘积$\sigma_1 \cdots \sigma_{n-1}$，其中$\sigma_k \in S_k$；参见(13)。习题50表明，每个元素$\alpha$也可惟一地被表达为对偶形式

$$\sigma = \sigma_{n-1}^{-} \cdots \sigma_2^{-} \sigma_1^{-}，\text{其中对于} 1 \leqslant k < n，\sigma_k \in S_k \tag{32}$$

并且这一事实得出另一个大的排列生成元族。特别是，当G是所有$n!$个排列的群时，

每个排列都可写成

$$\sigma(n-1,c_{n-1})^{-}\cdots\sigma(2,c_2)^{-}\ \sigma(1,c_1)^{-} \tag{33}$$

其中对于$1\leqslant k<n$, $0\leqslant c_k\leqslant k$，排列$\sigma(k,j)$和在算法G中相同。然而，现在我们要最快速地改变$c_{n-1}$和最慢速地改变$c_1$，因此我们得到一个不同类型的算法。

算法H（对偶排列生成程序） 如同在算法G中给定一个西姆斯表，这个算法使用一个辅助表格$c_0\cdots c_{n-1}$，生成$\{0,\cdots,n-1\}$的所有排列$a_0\cdots a_{n-1}$。

H1. [初始化。] 对于$0\leqslant j<n$置$a_j\leftarrow j$，和$c_j\leftarrow 0$。

H2. [访问。]（这时混合进制数$\begin{bmatrix}c_1,c_2,\cdots,c_{n-1}\\2,\ 3,\cdots,\ \ n\end{bmatrix}$是迄今已访问的排列个数。）访问排列$a_0a_1\cdots a_{n-1}$。

H3. [加1至$c_0c_1\cdots c_{n-1}$。] 置$k\leftarrow n-1$。如果$c_k=k$，则置$c_k\leftarrow 0$，$k\leftarrow k-1$，并且重复至$k=0$或$c_k<k$为止。如果$k=0$则结束此算法；否则置$c_k\leftarrow c_k+1$。

H4. [排列。] 如同以下所说明那样，应用排列$\tau(k,c_k)\omega(k+1)^{-}$到$a_0a_1\cdots a_{n-1}$，并返回H2。 ∎

尽管本算法看起来和算法G几乎是相同的，但是在步骤H4中它所需要的排列τ和ω与在步骤G4中所需的十分不同。代替(15)和(16)的新规则是，

$$\tau(k,j)=\sigma(k,j)^{-}\ \sigma(k,j-1),\quad 1\leqslant j\leqslant k \tag{34}$$

$$\omega(k)=\sigma(n-1,n-1)^{-}\ \sigma(n-2,n-2)^{-}\cdots\sigma(k,k)^{-} \tag{35}$$

可能性的个数就如同算法G那里的可能性那么大，所以我们将把注意力限制在有特殊利益的一些情况上。当然，要试验的一个自然情况是使算法G产生出颠倒的协词典顺序的西姆斯表，即如同在(18)中那样，

$$\sigma(k,j)=(k-j\ k-j+1\cdots k) \tag{36}$$

得到的排列生成程序结果是非常接近于和平易改动方法相同，所以我们可以说，算法L和算法P实质上彼此对偶。（参见习题52。）

另一个自然的思想是来构造一个西姆斯表，对于它而言，通过模拟在步骤G4中达到极大有效性的(27)的构造，步骤H4总是执行两个元素的一个转置。但是现在这个任务被证明是不可能的：即使当$n=4$时，我们也不能实现它。因为如果我们以恒等排列$a_0a_1a_2a_3=0123$开始，但我们从控制表$c_0c_1c_2c_3=0000$变成0001和到0002以及0003的变换必定移动3。因此，如果它们是转置，则它们必定是对于{0, 1, 2}的某个排列abc的$(3a)$，(ab)和(bc)。对应于$c_0c_1c_2c_3=0003$的排列现在是$\sigma(3,3)^{-}=(bc)(ab)(3a)=(3abc)$；而且对应于$c_0c_1c_2c_3=0010$的下一个排列将是$\sigma(2,1)^{-}$，它必定固定元素3。惟一合适的转置是$(3c)$，因此$\sigma(2,1)^{-}$必定是$(3c)\ (3abc)=(a\ b\ c)$。类似地，我们发现$\sigma(2,2)^{-}$必定是$(a\ c\ b)$，而且对应于$c_0c_1c_2c_3=0023$的排列将是$(3abc)(a\ c\ b)=(3c)$。步骤H4现在设想把这转换成排列$\sigma(1,1)^{-}$。它对应于跟随0023的控制表0100。但将把$(3\ c)$转换成为固定2和3的一个排列的惟一转置是$(3\ c)$；而且得到的排列也固定1，所以它不可能是$\sigma(1,1)^{-}$。

55

上一段的证明表明我们不能使用算法H来生成其转置个数极小的所有排列，但是它也提出了一个简单的生成方案。它非常接近于极小转置数，而且得到的算法十分有吸引力，因为它仅仅对于每$n(n-1)$步才需要做一次额外的工作。（参见习题53。）

最后，当如同在(23)中那样

$$\sigma(k,j)=(k\cdots 1\ 0)^j \tag{37}$$

时，让我们考虑奥德-史密斯方法的对偶。同样，$\tau(k,j)$的值同j无关，

$$\tau(k,j)=(0\ 1\ \cdots\ k) \tag{38}$$

而且这个事实在算法H中特别有利，因为我们无需控制表$c_0c_1\cdots c_{n-1}$。原因在于在步骤H3中$c_{n-1}=0$当且仅当$a_{n-1}=n-1$。这是由于(32)的缘故；而且确实，在步骤H3中，对于$k<j<n$，$c_j=0$时我们有$c_k=0$当且仅当$a_k=k$。因此我们可以重新叙述算法H的这个版本，如下：

算法C（通过循环移位的排列生成） 本算法访问不同元素$\{x_1,\cdots,x_n\}$的所有排列$a_1\cdots a_n$。

C1. [初始化。] 对于$1\leqslant j\leqslant n$，置$a_j\leftarrow x_j$。

C2. [访问。] 访问排列$a_1\cdots a_n$并置$k\leftarrow n$。

C3. [移位。] 以循环移位$a_2\cdots a_ka_1$代替$a_1a_2\cdots a_k$，并且如果$a_k\neq x_k$则返回C2。

C4. [减小k。] 置$k\leftarrow k-1$，并且如果$k>1$则返回C3。 ∎

例如，当$n=4$时生成的$\{1,2,3,4\}$的逐次排列为

1234, 2341, 3412, 4123, (1234),
2314, 3142, 1423, 4231, (2314),
3124, 1243, 2431, 4312, (3124), (1234),
2134, 1342, 3421, 4213, (2134),
1324, 3241, 2413, 4132, (1324),
56 3214, 2143, 1432, 4321, (3214), (2134), (1324)

连同在括弧中示出的未被访问的直接排列在内。这个算法就程序长度的极小性而言，可以说是所有排序生成程序中最简单的。它是由小格·乔·朗顿（G. G. Langdon, Jr）给出的[*CACM* 10 (1967), 298-299; 11 (1968), 392]。类似的方法以前已由查·汤普金斯（C. Tompkins）发表[*Proc. Symp. Applied Math.* 6 (1956), 202-205]且更清晰地由理·塞特兹（R. Seitz）发表过[*Unternehmensforschung*（运筹学）6 (1962), 2-15]。这个过程特别适合于其中循环移位有效的那些应用。例如，当逐次的排列是保存在机器寄存器中而不是在一个数组时。

对偶方法的主要缺点是它们通常不能很好地适应于需要跳过大量排列块的那些情况，因为带有最初的控制条目的一个给定值$c_0c_1\cdots c_{k-1}$的所有排列的集合通常

是不重要的。然而，特殊情况(36)有时是一个例外，因为在该情况下带有$c_0c_1\cdots c_{k-1}=00\cdots0$的$n!/k!$个排列恰好是那样一些$a_0a_1\cdots a_{n-1}$，其中0在1之前，1在2之前……$k-2$在$k-1$之前。

***厄尔里兹的交换方法**。吉迪安·厄尔里兹（Gideon Ehrlich）使用一个控制表$c_1\cdots c_{n-1}$的另外一种方法，发现了一个完全不同的排列生成的方法。他的方法是通过最左元素同另一个元素交换，而从其前驱获得每一个排列。

算法E（厄尔里兹交换） 本算法通过使用辅助表$b_0\cdots b_{n-1}$和$c_1\cdots c_n$，来生成不同元素$a_0\cdots a_{n-1}$的所有排列。

E1. [初始化。] 对于$0\leqslant j<n$，置$b_j\leftarrow j$和$c_{j+1}\leftarrow0$。

E2. [访问。] 访问排列$a_0\cdots a_{n-1}$。

E3. [找k。] 置$k\leftarrow1$。然后如果$c_k=k$，则置$c_k\leftarrow0$，$k\leftarrow k+1$，并且重复直到$c_k<k$为止。如果$k=n$，则结束，否则置$c_k\leftarrow c_k+1$。

E4. [交换。] 交换$a_0\leftrightarrow a_{b_k}$。

E5. [触发。] 置$j\leftarrow1$，$k\leftarrow k-1$。如果$j<k$，则交换$b_j\leftrightarrow b_k$，置$j\leftarrow j+1$，$k\leftarrow k-1$，并且重复直到$j\geqslant k$为止。返回E2。 ∎

注意，步骤E2和E3和算法G的步骤G2和G3相同。关于这个算法的最有趣之处，对此厄尔里兹曾于1987年向马丁·加德纳（Martin Gardner）通报过，在于它有效；习题55包含有一个证明。简化步骤E5的操作的一个类似方法，可以用同样的方法（参见习题56）加以证明。在步骤E5中所执行的交换的平均次数少于0.18（参见习题57）。

事实上，算法E不比我们已经见到的其他方法更快。但它有一个很棒的性质，即仅使用$n-1$个不同种类的转置，以极小的方法来改变每个排列。虽则算法P使用相邻的交换$a_{t-1}\leftrightarrow a_t$，算法E却使用头一个元素交换$a_0\leftrightarrow a_t$，也叫做*星形转置*，因为它是下标$t[1], t[2], \cdots, t[n!-1]$的某个既选的序列所致。而且如果我们反复对于相同的相当小的n值生成排列，对于算法P的下标序列如同在算法T中所做的那样，我们可以预先计算这个序列。注意星形转置相对于相邻交换有一个优点，因为我们总是从以前的交换知道a_0的值；无需从内存中来读它。 57

令E_n是使得算法E在步骤E4中把a_0同a_t进行交换的这样$n!-1$个下标t的序列。由于E_{n+1}以E_n开始，我们可以把E_n看作是无穷序列

$$E_\infty=12121321212312121321212431313213131 2\cdots \tag{39}$$

的头$n!-1$个元素。例如，如果$n=4$和$a_0a_1a_2a_3=1234$，则算法E所访问的排列为

$$\begin{gathered}1234,2134,3124,1324,2314,3214,\\4213,1243,2143,4123,1423,2413,\\3412,4312,1342,3142,4132,1432,\\2431,3421,4321,2341,3241,4231\end{gathered} \tag{40}$$

***使用较少的生成程序**。在看到算法P和E之后，我们可能自然会问，是否通过只使用两个基本操作，而不是$n-1$，就能得到所有的排列。例如，尼珍休斯（Nijenhuis）和威尔芙（Wilf）[*Combinatorial Algorithms* (1975)，习题6]注意到，对于$n=4$，如果在每步中或者以$a_2a_3\cdots a_na_1$或者以$a_2a_1a_3\cdots a_n$来代替$a_1a_2a_3\cdots a_n$，则所有排列都可被生成，而且他们想知道，对于所有的n是否存在这样一种方法。

一般地说，如果G是任何排列群，而且如果$\alpha_1,\cdots,\alpha_k$是G的元素，G的有生成元素$(\alpha_1,\cdots,\alpha_k)$的凯利（Cayley）图是其顶点为$G$的排列$\pi$和其边为从$\pi$连到$\alpha_1\pi,\cdots,\alpha_k\pi$的有向图。[阿瑟·凯利（Arthur Cayley），*American J. Math.* 1 (1878)，174-176]。尼珍休斯和威尔芙的问题等价于问，具有生成程序σ和τ（其中σ是循环排列$(1\ 2\ \cdots n)$而τ是转置(12)）的$\{1,2,\cdots,n\}$的所有排列的凯利图有无一条哈密顿通路。

由罗·阿·兰金（R. A. Rankin）给出的一个基本定理[*Proc. Cambridge Philos. Soc.* 44 (1948)，17-25]可以得出如下结论，即在许多情况下具有两个生成程序的凯利图没有一个哈密顿循环。

定理R 设G是由g个排列组成的一个群，如果有生成程序(α,β)的G的凯利图有一个哈密顿循环，而且如果排列$(\alpha,\beta,\alpha\beta^{-})$是分别有阶$(a,b,c)$的排列，则或者$c$为偶数，或者$g/a$和$g/b$是奇数。

（一个排列α的阶是使得α^a成为恒等排列的最小正整数a。）

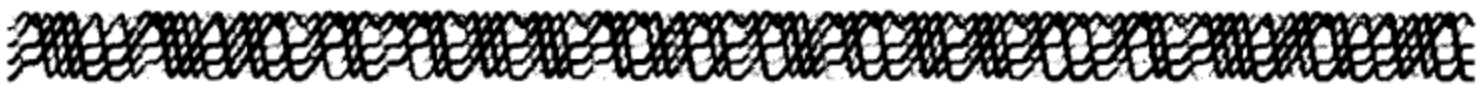
证明 参见习题73。 ∎

特别是，当如同上边那样$\alpha=\sigma$且$\beta=\tau$时，我们有$g=n!$，$a=n$，$b=2$和$c=n-1$，因为$\sigma\tau^{-}=(2,\cdots,n)$。因此我们得出结论，当$n\geqslant 4$为偶数时，不可能有哈密
58 顿循环。然而，当$n=4$时容易构造一个哈密顿通路，因为通过从2341开始并从1234跳到2134，并在4213处结束，我们可以合并多达12个循环：

$$\begin{aligned}&1234\to 2341\to 3412\to 4312\to 3124\to 1243\to 2431\to 4231\\&\qquad\to 2314\to 3142\to 1423\to 4123\to 1234\\&2134\to 1342\to 3421\to 4321\to 3214\to 2143\to 1432\to 4132\\&\qquad\to 1324\to 3241\to 2413\to 4213\to 2134\end{aligned}\tag{41}$$

拉斯基、姜明（Ming Jiang）和威斯通（Weston）[*Discrete Applied Math.* 57 (1995)，75-83]对于$n=5$的$\sigma-\tau$图进行了穷尽的查找，并且发现，它实际上有5个不同的哈密顿循环。图22a中示出其中之一（“最美丽的”）。他们也发现对于$n=6$的一个哈密顿通路；这是一个了不起的功绩，因为它是720阶段的二叉决策树的结果。不幸的是，他们发现的解没有明显的逻辑结构。在习题70中描述了一个相对不太复杂的通路，但即便是这个通路也不能称做是简单的。因此对于更大的n值，$\sigma-\tau$方法可能不具有实用意义，除非能发现一个新的构造。罗·克·坎普顿（R. C. Compton）和斯·吉·威廉森（S. G. Williamson）[*Linear and Multilinear Algebra* 35 (1993), 237-293]已经证明，如果允许三个生成程序σ、σ^{-}和τ而不是仅

仅σ和τ，则对于所有n，哈密顿循环存在。他们的循环有如下有趣的性质，即每第n个变换为τ，而且穿插的$n-1$个变换或者是全为σ，或者全为σ^{-}。但是他们的方法太复杂，因而在一个短篇幅内无法说得清楚。



a) 只使用变换(12345)和(12)



b) 只使用变换(12)(34)、(23)(45)和(34)

图22 对于5!排列的哈密顿循环

习题69描述了一个相当简单而且只需一个生成元的一般的排列算法。三个生成程序各有为2的阶。图22b示出这个方法对于$n = 5$的情况，它是由钟响声的例子所启发的。

快些，更快些。什么是生成排列的最快的方法？在计算机应用中这个问题经常被提起，因为考察$n!$种可能性的人们都想使运行时间尽可能地少。但是答案一般说来是矛盾的，因为有许多不同的方法来阐述这个问题。通过研究在MMIX计算机上如何最快地生成排列，让我们试图来理解相关的问题。

首先假设我们的目标是在n个连续的存储字（四字（奥克塔）字节）的数组中产生排列。在本小节我们已经看到的所有方法当中，做这件事最快的方法就是罗·塞奇威克（R. Sedgewick）所提议的流水线化的希普方法(27)。[*Computing Surveys* 9 (1977)，157-160。] 59

关键的思想是来优化对于步骤G2和G3的最普通情况的代码，即所有的活动出现在数组的开始的情况。如果寄存器u、v和w包含头三个字的内容，而且如果有待生成的下6个排列涉及以所有6个可能的方式来排列这些字，则我们显然可以如下来做这个工作。

```
PUSHJ 0,Visit
STO v,A0; STO u,A1; PUSHJ 0,Visit
STO w,A0; STO v,A2; PUSHJ 0,Visit
STO u,A0; STO w,A1; PUSHJ 0,Visit
STO v,A0; STO u,A2; PUSHJ 0,Visit
STO w,A0; STO v,A1; PUSHJ 0,Visit
```
(42)

（这里A0是四字（奥克塔）字节a_0的地址，等等。）用于把正确的东西放进u、v和w的一个完整的排列程序出现于习题77中，但其他指令不大重要，因为它们仅有1/6的时间要加以实施。每个排列的总的开销，不计算对于每个调用Visit时PUSHJ和POP所需要的$4v$，对于这个方法而言，近似等于$2.77\mu + 5.69v$。如果我们使用四个寄存器u，v，w，x，而且如果把(42)扩充到对Visit的24个调用，则每个排列的运行时间就降低为大约$2.19\mu + 3.07v$。而对于r个寄存器和$r!$个Visits，

习题78表明，开销是 $(2+O(1/r!))(\mu+\nu)$，这非常接近于两条STO指令的开销。

当然，后者是对于在一个顺序的数组中生成所有排列的任何方法的极小可能的时间……或者果真是吗？我们已经假定访问例程要在连续的单元中来看排列，但也许该例程有能力从不同的起点来读排列。那么我们可以安排来保持a_{n-1}固定，并且保持其他元素的两个复写在它的邻近处:

$$a_0a_1\cdots a_{n-2}a_{n-1}a_0a_1\cdots a_{n-2} \tag{43}$$

如果我们现在令$a_0a_1\cdots a_{n-2}$跑遍$(n-1)!$个排列，通过执行两个STO指令而不是一个，总是同时改变两个复写，我们可以让对于Visit的每个调用考察全都连续地出现的n个排列

$$a_0a_1\cdots a_{n-1}, a_1\cdots a_{n-1}a_0, \cdots, a_{n-1}a_0\cdots a_{n-2} \tag{44}$$

每个排列的开销现在减少到像ADD，CMP，PBNZ这样三条简单指令的开销，加上$O(1/n)$。[参见瓦洛尔（Varol）以及罗特姆（Rotem），*Comp. J.* 24 (1981), 173-176。]

其次，我们可能根本不想把时间浪费在存储在内存的排列中。例如，假设我们的目标是生成$\{0, 1, \cdots, n-1\}$的所有排列。n的值大概至多16，因为 16! = 20 922 789 888 000，而17! = 355 687 428 096 000。因此，整个的排列将装入一个四字（奥克塔）字节的16个连续的四个二进位中，我们可以在单个寄存器中来保存它。仅当访问的程序不需要来拆开个别的连续四位时这才将是有利的。但让我们假设，它不需要。在一个64位寄存器的连续四位串中，我们能多快地
60 生成排列呢？

由阿·杰·戈德斯坦（A. J. Goldstein）给出的一项技术，[美国专利3383661 (1968年5月14日)]所提议的一个思想，是使用算法T，来预先计算7个元素的平易改动转换的表 $(t[1], \cdots, t[5039])$。这些数$t[k]$位于1和6之间，所以我们可以把它们中的20个包装于一个64位的字中。对于$0 \leq j < 252$，把数 $\sum_{k=1}^{20} 2^{3k-1}t[20j+k]$ 放进一个辅助表的字j中，是方便的，且使$t[5040] = 1$。例如，这个表以码字

00|001|010|011|100|101|110|100|110|101|100|011|010|001|110|
001|010|011|100|101|110|00

开始，下列程序有效地读这样一些码:

```
Perm   〈置寄存器a为头一个排列〉
0H     LDA p,T        p←头一个码字的地址
       JMP 3F
1H     〈访问在寄存器a中的排列〉
       〈转储位于右数t位的a的四个连续数字〉
       SRU c,c,3      c←c≫3
2H     AND t,c,#1c    t←c & (11100)₂                    (45)
```

```
        PBNZ  t,1B    如果t≠0则转移
        ADD   p,p,8
3H      LDO   c,p,0   c←下个码字
        PBNZ  c,2B    〈最后的码字后边跟着0〉
        〈如果未完成，则推进前导的n-7个四个连续二进数字并返回0B〉
```

习题79说明使用在大多数计算机上的位乘法操作，如何通过7个指令〈转储四个连续位……〉。因此每个排列的开销仅仅是比10ν多一点。（取新码字的指令仅仅耗费(μ + 5ν)/20，而推进$n-7$个四个连续二进位的指令更是可忽略不计的，因为它们的开销除以5040。）注意现在不需要像在(42)中那样使用PUSHJ和POP。以前我们忽略这些指令，但它们确实耗费4ν。

然而，通过采取朗顿的循环移位方法算法C，我们甚至可以做得更好，假设我们以在词典顺序下最大的排列开始并且进行如下操作:

```
        GREG  @
0H      OCTA  #fedcba9876543210&(1<<(4*N)-1)
Perm    LDOU  a,0B              置a←#...3210
        JMP   2F
1H      SRU   a,a,4*(16-N)      a←⌊a/16^(16-n)⌋          (46)
        OR    a,a,t             a←a & t
2H      〈访问在寄存器a中的排列〉
        SRU   t,a,4*(N-1)       t←⌊a/16^(n-1)⌋
        SLU   a,a,4*(17-N)      a←16^(17-n) a mod 16^16
        PBNZ  t,1B              如果t≠0则转到1B
        〈以朗顿方法继续〉
```

再次说明，如果无须使用PUSHJ和POP，每个排列的运行时间现在仅为5ν + $O(1/n)$。关于扩充(46)成为一个完备的程序并且得到一个值得注意的短而快速的例行程序的有趣方法，参见习题81。 61

快速排列生成程序是有趣的；但在实践中，通过流水线化访问程序通常比通过加速生成程序能节省更多时间。

拓扑排序。代替对{1, …, n}的所有$n!$个排列进行工作，我们通常只需要考察遵守某种限制的一些排列。例如，我们可能仅对1居于3之前、2居于3之前以及2居于4之前这样的排列感兴趣；对于{1, 2, 3, 4}具有5个这样的排列，即

$$1234,\ 1243,\ 2134,\ 2143,\ 2413 \tag{47}$$

拓扑排序问题，作为非平凡的数据结构的头一个例子，我们在2.2.3节中研究过它，是寻找满足m个这样的条件$x_1 \prec y_1, \cdots, x_m \prec y_m$的一个排列的一般问题，其中$x \prec y$意味着在排列中$x$应居于$y$之前。在实践中这个问题经常出现，所以它有好多个不同的名称；例如，它通常被叫做*线性嵌入问题*，因为我们要在一条线上来安排

对象，同时保持某种顺序关系。它也是把一个偏序关系扩充成为全序关系的问题（参见习题2.2.3-14）。

在2.2.3节中，我们的目标是寻找满足所有关系的单个排列。但是我们现在更是要寻找所有这样的排列，所有拓扑排序。确实，我们在本小节中将假定关系是定义在1和n之间的整数上，而且每当$x \prec y$，我们有$x < y$。因此排列$12\cdots n$在拓扑上总是正确的。（如果这个简单的假定不满足，我们可以通过使用算法2.2.3T以适当重新命名这些对象来预处理这些数据。）

许多重要的排列类都属于这个拓扑排序问题的特殊情况，例如，使得

$$\{1, \cdots, 8\}\text{的}1 \prec 2, 2 \prec 3, 3 \prec 4, 6 \prec 7, 7 \prec 8$$

的排列等价于多重集合{1, 1, 1, 1, 2, 3, 3, 3}的排列。因为我们可以映射{1, 2, 3, 4}$\mapsto$1，5$\mapsto$2和{6, 7, 8}$\mapsto$3。我们知道如何使用算法L来生成一个多重集合的排列，但现在我们将学习另一个方法。

注意在一个排列$a_1\cdots a_n$中x居于y之前，当且仅当在逆排列$a'_1\cdots a'_n$中$a'_x < a'_y$。因此我们要研究的算法也将发现每当$j \prec k$时使得$a'_j < a'_k$的所有排列$a'_1\cdots a'_n$。例如，我们在5.1.4节中学过杨氏表景（Young tableau）是行和列中{1, …, n}这样一个安排，使得每行从左到右递增，而每列从顶部到底部递增。因此生成所有3 × 3杨氏表景的问题就等价于生成所有$a'_1\cdots a'_9$使得

$$\begin{gathered} a'_1 < a'_2 < a'_3, a'_4 < a'_5 < a'_6, a'_7 < a'_8 < a'_9 \\ a'_1 < a'_4 < a'_7, a'_2 < a'_5 < a'_8, a'_3 < a'_6 < a'_9 \end{gathered} \tag{48}$$

62 因而这是特殊类型的拓扑排序。

我们也可能要来寻找$2n$个元素的所有匹配，即把{1, …, $2n$}分划成n对的所有方法。一共有 $(2n-1)(2n-3)\cdots(1) = (2n)!/(2^n n!)$ 种方法来这样做，而它们对应于满足

$$a'_1 < a'_2,\ a'_3 < a'_4,\ \cdots,\ a'_{2n-1} < a'_{2n},\ a'_1 < a'_3 < \cdots < a'_{2n-1} \tag{49}$$

的排列。

对于穷尽的拓扑排序的一个优美算法是由雅·里·瓦洛尔（Varol）和多·罗特姆（Rotem）发现的[*Comp. J.* 24 (1981), 83-84]。他们认识到，可以使用一个类似于平易改动（算法P）的方法。假设我们已经发现拓扑地安排{1, …, $n-1$}的一个方法，使得$a_1\cdots a_{n-1}$满足所有不涉及n的条件。于是我们可以容易地写下插入最后元素n的所有允许的方式，而不必改变$a_1\cdots a_{n-1}$的相对顺序；我们简单地从$a_1\cdots a_{n-1}\ n$开始，然后一次向左移动n一步，直到它不能再移动为止。递归地应用这个思想产生出以下直截了当的过程。

算法V（所有拓扑的排序） 给定在{1, …, n}上的一个关系$\prec$，且它具有以下性质，即$x \prec y$意味着$x < y$。本算法生成所有排列$a_1\cdots a_n$及它们的逆$a'_1\cdots a'_n$，而且它们满足以下性质，即每当$j \prec k$时，$a'_j < a'_k$。为方便起见我们假定$a_0 = 0$且对于$1 \leqslant k \leqslant n$，$0 \prec k$。

V1. [初始化。] 对于$0 \leqslant j \leqslant n$，置$a_j \leftarrow j$且$a_j' \leftarrow j$。

V2. [访问。] 访问排列$a_1 \cdots a_n$和它的逆$a_1' \cdots a_n'$。然后置$k \leftarrow n$。

V3. [k能左移吗？] 置$j \leftarrow a_k'$，和$l \leftarrow a_{j-1}$，如果$l \prec k$，则转到V5。

V4. [能移，则移之。] 置$a_{j-1} \leftarrow k$，$a_j \leftarrow l$，$a_k' \leftarrow j-1$，而且$a_l' \leftarrow j$，转到V2。

V5. [不能移，则把k放回去。] 当$j < k$时置$l \leftarrow a_{j+1}$，$a_j \leftarrow l$，$a_l' \leftarrow j$以及$j \leftarrow j+1$。然后置$a_k \leftarrow a_k' \leftarrow k$。$k$减1，且如果$k > 0$则返回V3。 ▮

例如，定理5.1.4H告诉我们，大小为3×3的杨氏表景恰有42个。如果应用算法V到关系(48)上并以数组形式写出逆排列，

$$\begin{array}{|c|}\hline a_1'a_2'a_3' \\ a_4'a_5'a_6' \\ a_7'a_8'a_9' \\ \hline \end{array} \tag{50}$$

我们得到下列42个结果:

123 456 789	123 457 689	123 458 679	123 467 589	123 468 579	124 356 789	124 357 689	124 358 679	124 367 589	124 368 579	125 367 489	125 368 479	125 346 789	125 347 689
125 348 679	126 347 589	126 348 579	127 348 569	126 357 489	126 358 479	127 358 469	134 256 789	134 257 689	134 258 679	134 267 589	134 268 579	135 267 489	135 268 479
145 267 389	145 268 379	135 246 789	135 247 689	135 248 679	136 247 589	136 248 579	137 248 569	136 257 489	136 258 479	137 258 469	146 257 389	146 258 379	147 258 369

63

令t_r是这样的拓扑排序的个数，即对于它们而言，最后的$n-r$个元素是在它们的初始位置上，即对于$r < j \leqslant n$，$a_j = j$。等价地说，当我们忽略涉及大于r的元素的关系时，t_r是$\{1, \cdots, r\}$的拓扑排序$a_1 \cdots a_r$的个数。于是，奠定算法V的递归机制表明，步骤V2被执行N次，而步骤V3被执行M次，其中

$$M = t_n + \cdots + t_1 \quad \text{和} \quad N = t_n \tag{51}$$

还有，步骤V4和V5的循环操作被执行$N-1$次；步骤V5的剩下部分被执行$M-N+1$次。因此该算法总的运行时间是M、N和n的线性组合。

如果元素的标号选择得不好，M可能要比N大得多。例如，如果对算法V限定的输入是

$$2 \prec 3, 3 \prec 4, \cdots, n-1 \prec n \tag{52}$$

则对于$1 \leqslant j \leqslant n$，$t_j = j$，而且我们有$M = 1/2 \cdot (n^2 + n)$，$N = n$。但在元素的重命名条件下，这些限制也等价于

$$1 \prec 2, 2 \prec 3, \cdots, n-2 \prec n-1 \tag{53}$$

于是M就被归结为$2n-1 = 2N-1$。

习题89表明，一个简单的预处理步骤将发现，对算法V稍作修改，就有能力在$O(N+n)$步之内生成所有拓扑排序的元素标号。因此拓扑排序总能有效地生成。

在你做排列之前反复深思。在本小节中我们已经多次看到关于排列生成的吸引人的算法，但是我们知道的许多算法是这样的，通过它们产生出来的是对于特殊目的而言为最优的排列，它们可以不需要跑遍所有可能性而找出来。例如，定理6.1S表明，只须通过相对于某个耗费准则来对记录排序，我们就能找出在一个顺序存储中安排记录的最佳方式；而且这一过程仅仅需要$O(n \log n)$步。在7.5.2节中，我们将研究赋值问题，它涉及如何来对一个方阵的列进行排列，使得对角线元素之和为极大。这个问题可以在至多$O(n^3)$个操作内获得解，因此使用一个阶为$n!$的方法是愚蠢的，除非n极端地小。即使像在旅行的销售商问题这样的情况下，当不知道有效的算法时，我们通常还是能找到比考察每种可能情况要好得多的方法。当有很好的理由来个别地考察每个排列时，排列生成被使用得最好。

习 题

▶1. [*20*] 试说明，在j的值接近于n时如何通过流水线化其操作而使算法L运行得更快些？

2. [*20*] 试重写算法L使得它以颠倒的协词典顺序产生$a_1 \cdots a_n$的所有排列。（换言之，反射$a_n \cdots a_1$的值应当像在(11)中那样，按词典顺序为递减的。这种算法形式通常比原来的更简单且更快，因为较少的计算依赖于n的值。）

64 ▶3. [*M21*] 一个组合安排X关于一个生成算法的秩是在X之前这个算法所访问的其他安排的个数。试说明，如果$\{a_1, \cdots, a_n\} = \{1, \cdots, n\}$，如何计算一个给定的排列关于算法L的秩。314592687的秩是什么？

4. [*M23*] 试推广习题3，当$\{a_1, \cdots, a_n\}$是多重集合$\{n_1 \cdot x_1, \cdots, n_t \cdot x_t\}$时，说明如何计算$a_1 \cdots a_n$关于算法L的秩；这里$n_1 + \cdots + n_t = n$且$x_1 < \cdots < x_t$。（当然，排列的总数是多项式系数

$$\binom{n}{n_1, \cdots, n_t} = \frac{n!}{n_1! \cdots n_t!}$$

参见等式5.1.2-(3)），什么是314159265的秩？

5. [*HM25*] 当元素$\{a_1, \cdots, a_n\}$都不同时，计算(a)在步骤L2中；(b)在步骤L3中，由算法L所作比较次数的均值和方差。

6. [*HM34*] 像在习题4中那样，当$\{a_1, \cdots, a_n\}$是一般的多重集合时，试推导出(a)在步骤L2中，(b)在步骤L3中，由算法L所作的比较次数的平均数的生成函数，也给出当$\{a_1, \cdots, a_n\}$是二进制多重集合$\{s \cdot 0, (n-s) \cdot 1\}$时在封闭形式下的结果。

7. [*HM35*] 当把算法L应用到多重集合(a) $\{2 \cdot 1, 2 \cdot 2, \cdots, 2 \cdot t\}$；(b) $\{1 \cdot 1, 2 \cdot 2, \cdots, t \cdot t\}$；(c) $\{2 \cdot 1, 4 \cdot 2, \cdots, 2^t \cdot t\}$时，在步骤L2中的每个排列所作的平均比较次数，在$t \to \infty$时的极限是多少？

▶8. [*21*] 一个多重集合的变差是其所有子多重集合的排列。例如$\{1, 2, 2, 3\}$的变差是

ε, 1, 12, 122, 1223, 123, 1232, 13, 132, 1322,
2, 21, 212, 2123, 213, 2132, 22, 221, 2213, 223, 2231, 23, 231, 2312, 232, 2321,
3, 31, 312, 3122, 32, 321, 3212, 322, 3221

证明对算法L的简单修改将生成一个给定多重集合$\{a_1, a_2, \cdots, a_n\}$的所有变差。

9. [22] 继续上一题，试设计一个算法来生成一个给定多重集合$\{a_1, a_2, \cdots, a_n\}$的所有$r$变差，也叫做它的$r$排列，即它的$r$个元素子多重集合的所有排列。（例如，对于有$r$个不同字母的一个字母算术，解是$\{0, 1, \cdots, 9\}$的一个$r$变差。）

10. [20] 如果在开始时，$a_1a_2\cdots a_n = 12\cdots n$，则在算法P结束时，$a_1a_2\cdots a_n$，$c_1c_2\cdots c_n$和$o_1o_2\cdots o_n$的值是什么？

11. [M22] 在算法P被执行时，每步执行多少次？（假定$n \geq 2$。）

▶12. [M23] 如果$\{a_1, \cdots, a_n\} = \{0, \cdots, 9\}$，由(a)算法L，(b)算法P，(c)算法C，所访问的第1000000个排列是什么？提示：在混合进制记号下，我们有

$$1000000 = \left\lceil \begin{matrix} 2,6,6,2,5,1,2,2,0,0 \\ 10,9,8,7,6,5,4,3,2,1 \end{matrix} \right\rceil = \left\lceil \begin{matrix} 0,0,1,2,3,0,2,7,1,0 \\ 1,2,3,4,5,6,7,8,9,10 \end{matrix} \right\rceil$$

13. [M21]（马丁·加德纳，1974。）真或假：如果$a_1a_2\cdots a_n$开始时为$12\cdots n$，算法P通过访问$n!/2$个1居于2之前的所有排列开始，然后第二个排列是$n\cdots 21$。

14. [M22] 真或假：如果在算法P中$a_1a_2\cdots a_n$开始时为$x_1x_2\cdots x_n$，则在步骤P5开始时我们总有$a_{j-c_j+s} = x_j$。

15. [M23]（塞尔梅尔·约翰森（Selmer Johnson），1963。）证明在算法P中偏离变量s绝不超过2。

16. [21] 在j的值接近于n时，说明如何通过使它的操作流水线化，使算法P运行得更快些。（这道题类似于习题1。）

▶17. [20] 试扩充算法P，使得当在步骤P2中访问$a_1\cdots a_n$时，逆排列$a_1^{\dagger}\cdots a_n^{\dagger}$可用于处理。（这个逆满足$a_k^{\dagger} = j$当且仅当$a_j = k$。）

18. [21]（玫瑰花园排列。）试设计一个有效方法来生成表示在顶点$\{1, \cdots, n\}$上所有可能的无向循环的$(n-1)!/2$个排列，即如果生成了$a_1\cdots a_n$，则无$a_1\cdots a_n$或$a_n\cdots a_1$的循环移动可被生成。例如，当$n = 4$时可以使用（1234，1324，3124）。

19. [25] 在算法7.2.1.1L的思想下，试构造一个算法，它无循环地生成n个不同元素的所有排列。

▶20. [20] n立体有$2^n n!$个对称性，它们对应于排列和/或求补坐标的每一个方法。这样一个对称性可方便地表示为一个带符号的排列，即带有附加到元素的任选符号的一个排列。例如，$2\bar{3}\bar{1}$是通过把$x_1x_2x_3$改变成$x_2x_3\bar{x}_1$的变换3立体顶点的一个带符号的排列，使得$000 \mapsto 001, 001 \mapsto 011, \cdots, 111 \mapsto 110$。试设计一个简单算法，它生成$\{1, 2, \cdots, n\}$的所有带符号的排列，其中每个步骤或者是交换两个相邻元素，或者是把头一个元素取负。

21. [M21]（埃·帕·麦克拉维（E. P. McCravy），1971。）在b进制中字母算术(6)有多少个解？

22. [M15] 真或假：如果一个字母算术在b进制下有一个解，则它在$b+1$进制下也有一

个解。

23. [*M20*] 真或假：当$j \neq k$时，一个纯粹的字母算术不能有两个相等的签名$s_j = s_k \neq 0$。

24. [*25*] 通过笔算或通过计算机算，求解下列字母算术：

a) SEND + A + TAD + MORE = MONEY.

b) ZEROS + ONES = BINARY. (Peter MacDonald, 1977)

c) DCLIX + DLXVI = MCCXXV. (Willy Enggren, 1972)

d) COUPLE + COUPLE = QUARTET. (Michael R. W. Buckley, 1977)

e) FISH + N + CHIPS = SUPPER. (Bob Vinnicombe, 1978)

f) SATURN + URANUS + NEPTUNE + PLUTO = PLANETS. (Willy Enggren, 1968)

g) EARTH + AIR + FIRE + WATER = NATURE. (Herman Nijon, 1977)

h) AN + ACCELERATING + INFERENTIAL + ENGINEERING + TALE + ELITE + GRANT + FEE + ET + CETERA = ARTIFICIAL + INTELLIGENCE.

i) HARDY + NESTS = NASTY + HERDS.

▶25. [*M21*] 给定签名向量$s = (s_1, \cdots, s_{10})$和一个字母算术问题的头字母集合$F$，试设计对于$\{0, \cdots, 9\}$所有有效的排列$a_1 \cdots a_{10}$的$\min(a \cdot s)$和$\max(a \cdot s)$的一个快速方法。（当考虑一个大的字母算术体系时，这样一个过程使得有可能快速地排除许多情况，如同在下列若干个习题的情况那样，因为仅当$\min(a \cdot s) \leq 0 \leq \max(a \cdot s)$时一个解才能存在。）

26. [*25*] 对于

NIIHAU ± KAUAI ± OAHU ± MOLOKAI ± LANAI ± MAUI ± HAWAII = 0

惟一字母算术的解是什么？

66 27. [*30*] 试构造所有词都有5个字母的一个纯加法的字母算术。

28. [*M25*] 整数n的一个分划是形如$n = n_1 + \cdots + n_t$这样一个表达式且$n_1 \geq \cdots \geq n_t > 0$。这样一个分划称为双倍地为真，如果$\alpha(n) = \alpha(n_1) + \cdots + \alpha(n_t)$也是一个纯粹的字母算术，其中$\alpha(n)$是在某种语言之下$n$的“名字”。双倍地为真的分划是由阿兰·怀尼（Alan Wayne）在[*AMM* 54 (1947), 38, 412-414]中引入的，其中他提出解TWENTY = SEVEN + SEVEN + SIX以及其他一些。

a) 当$1 \leq n \leq 20$时，寻找出英文中双倍地为真的所有分划。

b) 怀尼还给出EIGHTY = FIFTY + TWENTY + NINE + ONE的例子。试求出对于$1 \leq n \leq 100$中所有双倍地为真的分划，其中各部分是不同的，并使用名称ONE，TWO，…，NINETYNINE，ONEHUNDRED。

▶29. [*M25*] 继续上一道题，当$\{n_1, \cdots, n_t, n'_1, \cdots, n'_t\}$是小于20的不同的正整数时，试找出在数学上又在英语中字母算术两方面同为真的形如$n_1 + \cdots + n_t = n'_1 + \cdots + n'_t$的所有等式。例如

TWELVE + NINE + TWO = ELEVEN + SEVEN + FIVE

这些字母算术应当全是纯粹的。

30. [*25*] 通过笔算或计算机求解乘法的字母算术：

a) TWO × TWO = SQUARE (H. E. Dudeney, 1929)

b) HIP × HIP = HURRAY (Willy Enggren, 1970

c) PI × R × R = AREA (Brain Barwell, 1981)

d) NORTH/SOUTH = EAST/WEST (Nob Yoshigahara, 1995)

e) NAUGHT × NAUGHT = ZERO × ZERO × ZERO (Alan Wayne, 2003)

31. [*M22*]（芦ヶ原伸之。）当(A, …, I) = {1, …, 9}时，什么是A/BC + D/EF + G/HI = 1的惟一的解?

32. [*M25*]（亨·欧·达德尼，1901。）通过在数字{1, …, 9}的一个排列中插入一个加号和一个斜杠来找出表示100的所有方法。例如，100 = 91 + 5742/638，加号应居于斜杠之前。

33. [*25*] 继续上一道题，找出(a)不能以这样一种形式表示；(b)有惟一表示的少于150的所有正整数。

34. [*M26*] 当(a)x是一个完全的第5个幂；(b)x是一个完全的第7个幂时，使等式EVEN + ODD + PRIME = x双倍地为真。

▶ 35. [*M20*] 一个4立体的自同构有许多不同的西姆斯表，但在(14)中仅示出其中之一，当顶点如像在(12)中那样编号时，对于该群有多少个可能的不同的西姆斯表?

36. [*M23*]对于4 × 4的三子棋盘

0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

所有自同构的群，求一个西姆斯表，即把行变成行的排列，其中一“行”是属于一个行、列或对角线的四个元素的集合。

▶ 37. [*HM22*] 算法G或H可以使用多少个西姆斯表？当$n \to \infty$时试估计这个数的对数。

38. [*HM21*] 当使用奥德-史密斯算法(26)时，试证明每个排列的平均转置数近似地为 $\sinh 1 \approx 1.175$。

67

39. [16] 试写出通过(a)奥德-史密斯方法(26)，(b)希普方法(27)，对于$n = 4$生成的所有24个排列。

40. [*M23*] 试证明，希普方法(27)对应于一个正确的西姆斯表。

▶ 41. [*M33*] 试设计一个算法，它在从一个变差转到另一个变差时，只交换两个元素而生成{0, 1…, $n-1$}的所有r变差。（参见习题9。）提示：推广希普方法(27)，并得到一个数组$a_0 \cdots a_{n-1}$的在位置$a_{n-r} \cdots a_{n-1}$中的结果。例如，当$n = 5$和$r = 2$时，一个解使用分别的排列 01234，31204，30214，30124，40123，20143，24103，24013，34012，14032，13042，13402，23401，03421，02431，02341，12340，42310，41320，41230的最后两个元素。

42. [*M20*] 试对所有排列构造一个西姆斯表，在每一排列中对于$1 < j < k$，每一个$\sigma(k, j)$和每一个$\tau(k, j)$是长度< 3的一个循环。

43. [*M24*] 试对所有排列构造一个西姆斯表，在每一排列中，对于$1 < j < k$，每一个$\sigma(k,$

$k)$，$\omega(k)$ 以及$\tau(k, j)$ $\omega(k-1)^-$是长度$\leqslant 3$的一个循环。

44. [20] 当不要的排列的块通过扩充的算法G加以跳过时，奥德-史密斯方法(23)的西姆斯表是否比颠倒的协词典顺序方法(18)的西姆斯表优越?

45. [20] (a)当算法X访问排列314592687时下标$u_1\cdots u_9$是什么? (b) 当$u_1\cdots u_9 = 314157700$时被访问的排列是什么?

46. [20] 真或假: 当算法X访问$a_1\cdots a_n$时，我们有$u_k > u_{k+1}$当且仅当对于$1\leqslant k<n$，$a_k > a_{k+1}$。

▶47. [M21] 借助于数$N_0, N_1, \cdots, Nn$，这里N_k是对于$1\leqslant j\leqslant k$满足$t_j(a_1, \cdots, a_j)$ 的前缀$a_1\cdots a_k$的个数，来表达算法X的每一步骤被执行的次数。

▶48. [M25] 在测试$t_1(a_1), t_2(a_1, a_2), \cdots, t_n(a_1, a_2, \cdots, a_n)$ 总为真的情况下，试比较算法X和算法L的运行时间。

▶49. [28] 正文所建议的用算法X求解加法的字母算术的方法实质上是从右至左地选择数字; 换言之，在考虑对应于10的更高次幂的数字之前，它对最低有效位赋予尝试性的值。

试利用从左至右地选择数字的另一个方法。例如，当SEND + MORE = MONEY时，这样一个方法立即推导出M = 1。提示: 参见习题25。

50. [M15] 说明为什么对偶公式(32)从(13)得出。

51. [M16] 真或假: 如果对于所有排列的群，集合$S_k = \{\sigma(k, 0), \cdots, \sigma(k, k)\}$形成一个西姆斯表，则集合$S_k^- = \{\sigma(k, 0)^-, \cdots, \sigma(k, k)^-\}$也是这样。

▶52. [M22] 当对于西姆斯表(36)使用算法H时，什么排列$\tau(k, j)$ 和$\omega(k)$ 出现? 把得到的生成程序同算法P作比较。

▶53. [M26] (弗·马·艾弗斯 (F. M. Ives)。) 试构造这样一个西姆斯表，对于它，只须做$n! + O((n-2)!)$ 个转置，算法H就可生成所有排列。

54. [20]如果算法C的步骤C3作一个右循环的移动，并且置$a_1\cdots a_{k-1}a_k \leftarrow a_k a_1\cdots a_{k-1}$，而不是作左循环的移动，问算法C是否将正确地工作?

55. [M27] 考虑阶乘直尺函数

68

$$\rho!(m) = \max\{k | m \bmod k! = 0\}$$

令σ_k和τ_k是每当$j<k$时使得$\sigma_j\tau_k = \tau_k\sigma_j$的非负整数的排列。令$\alpha_0$和$\beta_0$是恒等排列，而且对于$m>0$，定义

$$\alpha_m = \beta_{m-1}^-\tau_{\rho!(m)}\beta_{m-1}\alpha_{m-1},\quad \beta m = \sigma_{\rho!(m)}\beta_{m-1}$$

例如，如果σ_k是触发操作 $(1\ k-1)(2\ k-2)\cdots = (0\ k)\phi(k)$ 和如果$\tau_k = (0\ k)$，而且如果算法E以对于$0\leqslant j<n$的$a_j = j$开始，则α_m和β_m是在步骤E5已被执行m次后$a_0\cdots a_{n-1}$和$b_0\cdots b_{n-1}$的内容。

a) 证明$\beta_{(n+1)!}\alpha_{(n+1)!} = \sigma_{n+1}\sigma_n^-\tau_{n+1}\tau_n^-(\beta_{n!}\ \alpha_{n!})^{n+1}$。

b) 使用(a)的结果来确定算法E的正确性。

56. [M22] 试证明，如果以

E5' [转置对偶。]如果$k>2$，则对于$j = k-2, k-4, \cdots,$(2或1)，

交换$b_{j+1}\leftrightarrow b_j$，返回E2。 ∎

来代替步骤E5，则算法E仍保持成立。

57. [HM22] 在步骤E5中所作的交换的平均次数是多少?

58. [*M21*] 真或假：如果算法E以$a_0\cdots a_{n-1} = x_1\cdots x_n$开始，则最后被访问的排列以$a_0 = x_n$开始。

59. [*M20*] 某些作者把凯利图的有向边定义为从π到$\pi\alpha_j$而不是从π下到$\alpha j\pi$的运行，这两个定义是否实质性地不同？

▶60. [*21*] 排列的格雷循环是包括$\{1, 2, \cdots, n\}$的每一个排列的一个循环$(\pi_0, \pi_1, \cdots, \pi_{n!-1})$，而且它有这样一个性质，即$\pi_k$和$\pi_{(k+1) \bmod n!}$在一个相邻的转置上不同。它也可被描述为在$\{1, 2, \cdots, n\}$上所有排列的群的凯利图上的一个哈密顿循环，并有（$n-1$）个生成元 ((12), (23), $\cdots$, $(n-1\ n)$)。这样一个格雷循环的δ序列是使得

$$\pi_{(k+1) \bmod n!} = (\delta_k \delta_{k+1})\pi_k$$

的整数$\delta_0\delta_1\cdots\delta_{n!-1}$的序列。（参见7.2.1.1-(24)，它描述对于二进制n元组的类似情况。）例如，图23示出当$n = 4$时由平易改动所定义的格雷循环。它的δ序列为 $(32131231)^3$。

a) 试求$\{1, 2, 3, 4\}$的排列的所有格雷循环。

b) 如果两个格雷循环的δ序列可以通过循环移位（$\delta_k\cdots\delta_{n!-1}\delta_0\cdots\delta_{k-1}$）和/或颠倒（$\delta_{n!-1}\cdots\delta_1\delta_0$）和/或取补 $((n-\delta_0)(n-\delta_1)\cdots(n-\delta_{n!-1}))$ 由彼此得到，则认为两者是等价的。(a) 中哪些格雷循环等价？

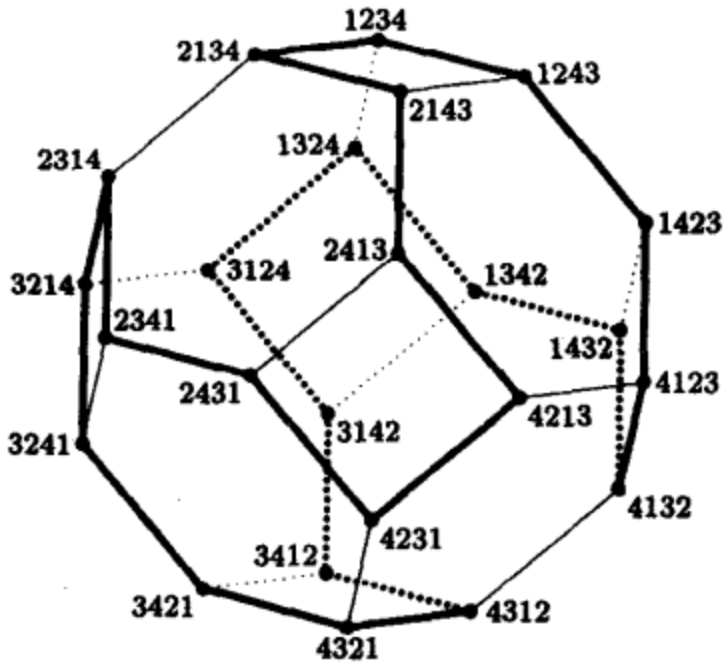


图23 算法P跟踪出在图5-1截取的八面体上的这个哈密顿循环 69

61. [*21*] 继续上一道题，排列的一个格雷码就好像一个格雷循环一样，只是不要求最后的排列$\pi_{n!-1}$和开头的排列π_0相邻。试研究对于$n = 4$且以1234开始的所有格雷码的集合。

▶62. [*M23*] 作为从$12\cdots n$开始的格雷码的最后元素，可以到达的排列是哪些？

63. [*M25*] 试估计$\{1, 2, 3, 4, 5\}$的排列的格雷循环的总数。

64. [*23*] 排列的一个“双重格雷”码是具有如下额外性质的一个格雷循环，即对所有的k，$\delta_{k+1} = \delta_k \pm 1$。康普顿和威廉森已经证明，对于所有$n \geq 3$这样的码存在。对于$n = 5$，有多少个双重格雷码存在？

65. [*M25*] 对于哪些整数N存在有通过$\{1, \cdots, n\}$的N个按词典顺序最小的排列的一个格雷通路？（习题7.2.1.1-26对于二进制n元组解决了类似的问题。）

66. [*22*] 厄尔里兹的交换方法提议另一类型的排列的格雷循环，其中$n-1$个生成程序是星形转置$(1\ 2), (1\ 3), \cdots, (1\ n)$。例如，图24示出当$n = 4$时的相关图，试分析这个图的哈密顿循环。

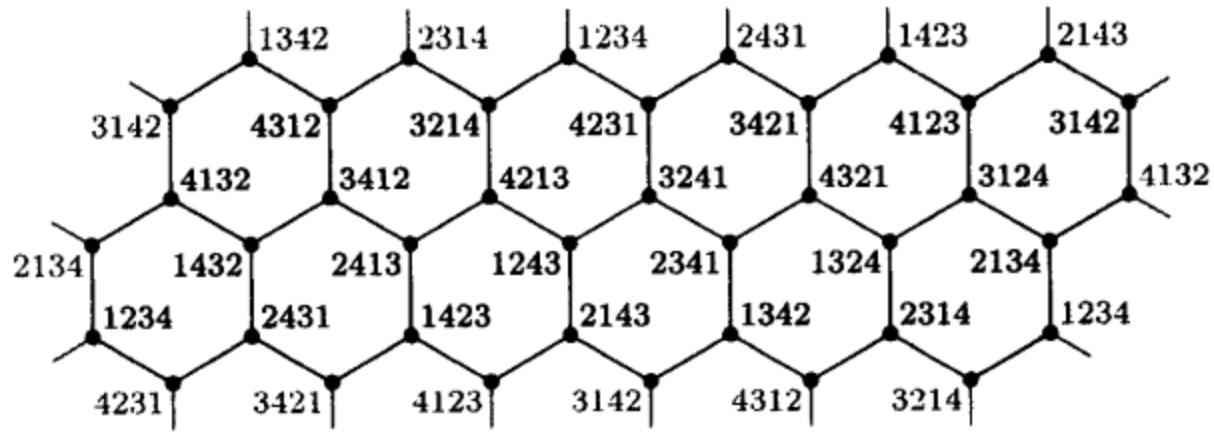


图24 由画成一个双扭的圆环体的星形转置(12)、(13)和(14)生成的{1, 2, 3, 4}的排列的凯利图

67. [*26*] 继续上一道题，试求对于$n = 5$的头元素交换的格雷循环，其中对于$2 < j < 5$，每个星形转置$(1\ j)$出现30次。

68. [*M30*]（康佩尔梅克（Kompel' makher）和里斯科维特斯（Liskovets），1975。）令G是$\{1, \cdots, n\}$的所有排列的凯利图，且有生成元$(\alpha_1, \cdots, \alpha_k)$，其中每个$\alpha_j$是转置$(u_j, v_j)$；还令$A$是对于$1 < j < k$，具有顶点$\{1, \cdots, n\}$和边$u_j$——$v_j$的图。试证明$G$有一个哈密顿循环当且仅当$A$是连通的。（图23是当$A$为一条通路时的特殊情况，图24是当$A$为一个“星”形时的特殊情况。）

▶69. [*28*] 如果$n \geq 4$，则下列算法只使用三个变换

$$\rho = (12)(34)(56)\cdots, \qquad \sigma = (23)(45)(67)\cdots, \qquad \tau = (34)(56)(78)\cdots$$

生成$\{1, 2, 3, \cdots, n\}$的所有排列$A_1A_2A_3\cdots A_n$，而且绝不彼此相邻地使用ρ和τ，试说明这为什么有效。

70

Z1. [初始化。] 对于$1 < j < n$，置$A_j \leftarrow j$。对于$j < n/2$置$a_j \leftarrow 2j$，以及对于$j < n/2$，置$a_{n-j} \leftarrow 2j + 1$。然后调用算法P，但使用参数$n - 1$而不是$n$。我们将把这个算法当做一个共行程序，每当在步骤P2处它“访问”$a_1\cdots a_{n-1}$时它就把控制转回给我们，我们也还将共享它的变量（n除外）。

Z2. [置x和y。] 再次调用算法P，并且得到一个新的排列$a_1\cdots a_{n-1}$和j的新值。如果$j = 2$，则交换$a_{1+s} \leftrightarrow a_{2+s}$（由此就对步骤P5的效果返工），并重复这一步骤；在这样的情况下，我们处于算法P的半途上。如果$j = 1$（使得算法P已经终止），则置$x \leftarrow y \leftarrow 0$并转到Z3，否则置$x \leftarrow a_{j-c_j+s+[o_j=-1]}$，$y \leftarrow a_{j-c_j+s-[o_j=+1]}$；这些是在步骤P5中最近交换的元素。

Z3. [访问。] 访问排列$A_1\cdots A_n$，然后如果$A_1 = x$和$A_2 = y$，则转到Z5。

Z4. [应用ρ，然后σ。] 交换$A_1 \leftrightarrow A_2$，$A_3 \leftrightarrow A_4$，$A_5 \leftrightarrow A_6$，$\cdots$；访问$A_1\cdots A_n$。然后交换$A_2 \leftrightarrow A_3$，$A_4 \leftrightarrow A_5$，$A_6 \leftrightarrow A_7$，$\cdots$。如果$A_1\cdots A_n = 1\cdots n$则结束，否则返回Z3。

Z5. [应用τ，然后σ。] 交换$A_3 \leftrightarrow A_4$，$A_5 \leftrightarrow A_6$，$A_7 \leftrightarrow A_8$，$\cdots$；访问$A_1\cdots A_n$。然后交换$A_2 \leftrightarrow A_3$，$A_4 \leftrightarrow A_5$，$A_6 \leftrightarrow A_7$，$\cdots$，并返回Z2。 ∎

提示：首先证明，如果在步骤Z1中修改成使得$A_j \leftarrow n + 1 - j$和$a_j \leftarrow j$，而且如果在步骤Z4和Z5中使用“触发”排列

$$\rho' = (1\ n)(2\ n-1)\cdots, \quad \sigma' = (2\ n)(3\ n-1)\cdots, \quad \tau' = (2\ n-1)(3\ n-2)\cdots$$

而不是使用ρ，σ，τ，则这个算法有效。在这个修改中，如果$A_1 = x$和$A_n = y$，则步骤Z3应转到Z5去。

▶70. [*M33*] 两个12-循环(41)可看做{1, 1, 3, 4}的12个排列

$$1134\to1341\to3411\to4311\to3114\to1143\to1431\to$$
$$4131\to1314\to3141\to1413\to4113\to1134$$

的σ-τ循环。以{1, 2}代替{1, 1}产生不同的循环，而且通过从一个跳到另一个去，我们得到一条哈密顿通路。基于{1, 1, 3, 4, 5, 6}的排列的360个循环，能否以类似的方法形成6个元素的所有排列的一个σ-τ通路?

71. [*48*] 每当$n \geq 3$为奇数时，是否具有生成元$\sigma = (12\cdots n)$和$\tau = (12)$的凯利图都有一个哈密顿循环?

72. [*M21*] 给定具有生成元（$\alpha_1, \cdots, \alpha_k$）的一个凯利图，假定每个$\alpha_j$使$x \mapsto y$。（例如，在习题71中$\sigma$和$\tau$两者都使$1 \mapsto 2$。）试证明在$G$中从$12\cdots n$开始的任何哈密顿通路必定在使$y \mapsto x$的一个排列处结束。

73. [*M30*] 令α，β和σ是一个集合X的排列；其中$X = A \cup B$，假设当$x \in A$时有$x\sigma = x\alpha$，当$x \in B$时有$x\sigma = x\beta$，而且$\alpha\beta^{-}$的阶为奇数。

a) 试证明所有三个排列α，β，σ有相同的符号；即，它们全为偶或全为奇。提示：一个排列有奇的阶当且仅当它的循环全有奇长度。

b) 从部分(a)推导定理R。

74. [*M30*]（罗·阿·兰金。）假设在定理R中$\alpha\beta = \beta\alpha$，试证明哈密顿循环存在当且仅当存在一个数$k$使得$0 < k < g/c$，以及$t + k \perp c$，其中$\beta^{g/c} = \gamma^t$，$\gamma = \alpha\beta^{-}$。提示：把群中的元素表示成$\beta^j\gamma^i$的形式。 71

75. [*M25*] 对于$0 \leq x < m$，$0 \leq y < n$，有向圆环体$O_m \times O_n$有mn个顶点(x, y)和有向弧$(x, y) \to (x, y)\alpha = ((x+1) \bmod m, y)$，$(x, y) \to (x, y)\beta = (x, (y+1) \bmod n)$。试证明，如果$m > 1$和$n > 1$，则这个有向图的哈密顿循环的个数是

$$\sum_{k=1}^{d-1} \binom{d}{k} [\gcd((d-k)m, kn) = d], \quad d = \gcd(m, n)$$

76. [*M31*] 图25中编号为0, 1, $\cdots$, 63的方格说明向东北方向的骑士在8×8圆环体上的旅游路线：如果k出现在方格(x_k, y_k)上，则$(x_{k+1}, y_{k+1}) \equiv (x_k + 2, y_k + 1)$，或者$(x_k + 1, y_k + 2)$，modulo 8，而且$(x_{64}, y_{64}) = (x_0, y_0)$。当$m, n \geq 3$时，在一个$m \times n$的圆环体上，有多少可能的这样的旅游路线?

29	24	19	14	49	44	39	34
58	53	48	43	38	9	4	63
23	18	13	8	3	62	33	28
52	47	42	37	32	27	22	57
17	12	7	2	61	56	51	46
6	41	36	31	26	21	16	11
35	30	1	60	55	50	45	40
0	59	54	25	20	15	10	5

图25 一个向东北方向的骑士的旅游路线

▶77. [*22*] 利用希普方法(27)，完成其内循环出现于(42)中的MMIX程序。

78. [*M23*] 试分析习题77中的程序的运行

时间，并推广它使得内循环做$r!$个访问（且$a_0 \cdots a_{r-1}$在全局寄存器中）。

79. [20] 哪7条MMIX指令将如(45)所要求的那样，＜转动四连续二进字……＞。例如，如果寄存器t包含值4和寄存器a包含四连续二进字#12345678，则寄存器a应该变成#12345687。

80. [21] 试只使用5个MMIX指令来求解上一习题。提示：使用MXOR。

▶81. [22] 通过确定如何来＜使用朗顿方法继续＞来完成MMIX程序(46)。

82. [M21] 试分析习题81中的程序运行时间。

83. [22] 使用习题70的σ-τ通路来设计类似于(42)的一个MMIX程序，它在寄存器a中生成#123456的所有排列。

84. [20] 试提出在并行地运行的p个处理器上生成$\{1, \cdots, n\}$的所有$n!$个排列的一个好方法。

▶85. [25] 假设n足够小，因而$n!$可装入一个计算机字中。什么是把$\{1, \cdots, n\}$的一个给定的排列$\alpha = a_1 \cdots a_n$转换成在$0 \leqslant k < n!$范围内的整数$k = r(\alpha)$的一个好方法？两个函数$k = r(\alpha)$和$\alpha = r^{[-1]}(k)$都应在仅仅$O(n)$步内就可计算。

86. [20] 一个偏序关系应当是传递的，即$x \prec y$和$y \prec z$应隐含$x \prec z$。但算法V不要求它的输入关系满足这个条件。

试证明，如果$x \prec y$和$y \prec z$，则算法V无论是否有$x \prec z$都将产生相等的结果。

87. [20]（弗·拉斯基。）考虑由算法V所访问的排列的反演表$c_1 \cdots c_n$，它们有什么值得注意的性质？（试把它同算法P中的反演表(4)作比较。）

72

88. [21] 试说明算法V可以用来生成把数字$\{0, 1, \cdots, 9\}$分划成两个3元素集合和两个2元素集合的所有方法。

▶89. [M30] 考虑(51)中的数$t_0, t_1, \cdots, t_n$，显然$t_0 = t_1 = 1$。

a) 如果$t_j = t_{j-1}$，则说下标j是“平凡的”。例如，相对于杨氏表景关系(48)，9是平凡的。试说明如何修改算法V使得变量k只取非平凡的值。

b) 试分析被修改了的算法的运行时间，什么公式代替(51)？

c) 如果对于$j \leqslant l < k$，我们没有$l \prec l+1$，则说区间$[j..k]$不是一个链。试证明，在这样的情况下$t_k \geqslant 2t_{j-1}$。

d) 每一个逆拓扑排序$a'_1 \cdots a'_n$定义对应于关系$a'_{j_1} \prec a'_{k_1}, \cdots, a'_{j_m} \prec a'_{k_m}$的一个标号，这些关系是等价于原来的关系$j_1 \prec k_1, \cdots, j_m \prec k_m$的；当$j$和$k$是不连续的非平凡的下标时，试说明如何来找出这样一个标号，使得$[j..k]$不是一个链。

e) 试证明对于这样一个标号，在部分(b)的公式中$M < 4N$。

90. [M21] 对于在一个给定集合中的所有h，算法V可用来产生h阶的所有排列，即对于$1 \leqslant j \leqslant n-h$使得$a'_j < a'_{j+h}$的所有$a'_1 \cdots a'_n$（见5.2.1节）。试分析当算法V生成2阶和3阶的所有两种排列时，它的运行时间。

91. [HM21] 当把算法V同关系(49)一起用来寻找匹配时，试分析它的运行时间。

92. [M18] 在“随机”情况下，算法V可能访问多少个排列？令P_n是在$\{1, \cdots, n\}$上的偏序的个数，即反身、反对称和传递的关系的个数。令Q_n是在这样的关系中，还有附加的性质，即每当$j \prec k$时则$j < k$的关系的个数。试表达对n个元素拓扑地进行排序的预期的方法个

数，并借助于P_n和Q_n对所有偏序求平均。

93. [*35*] 试证明，所有拓扑排序可以以这样一种方法来生成，即在每步中只有进行一个或两个相邻的转置（例子$1 \prec 2, 3 \prec 4$表明，即使我们允许非相邻的交换，但每步单个转置也并非总是可实现的，因为6个相关的排列中仅有2个是奇的。）

▶94. [*25*] 试说明，在匹配的情况下，使用(49)中的关系，所有拓扑排序能够以每步中仅用一个转置来生成。

95. [*21*] 试讨论如何来生成$\{1, \cdots, n\}$的所有由顶向下的排列，即使得$a_1 < a_2 > a_3 < a_4 > \cdots$的那些$a_1 \cdots a_n$。

96. [*21*] 试讨论如何来生成$\{1, \cdots, n\}$的所有循环排列，即其循环表示仅由单个n循环组成的那些$a_1 \cdots a_n$。

97. [*21*] 试讨论如何来生成$\{1, \cdots, n\}$的所有去排列，即使得$a_1 \neq 1$，$a_2 \neq 2$，$a_3 \neq 3$，$\cdots$的那些$a_1 \cdots a_n$。

98. [*HM23*] 试分析在上题中的方法的渐近运行时间。

99. [*M30*] 给定$n \geqslant 3$，试证明通过在访问之间至多作两个转置可以生成$\{1, \cdots, n\}$的所有去排列。

100. [*21*] 试讨论如何来生成$\{1, \cdots, n\}$的所有非可分解的排列，即使得对于$1 \leqslant j < n$，$\{a_1, \cdots, a_j\} \neq \{1, \cdots, j\}$的那些$a_1 \cdots a_n$。

101. [*21*] 试讨论如何来生成$\{1, \cdots, n\}$的所有卷积，即满足$a_{a_1} \cdots a_{a_n} = 1 \cdots n$的那些排列$a_1 \cdots a_n$。

102. [*M30*] 试证明在访问之间通过至多作两个转置就能生成$\{1, \cdots, n\}$的所有卷积。

103. [*M32*] 试证明，通过对三个连续元素的逐次转动，就能生成$\{1, \cdots, n\}$的所有偶排列。

▶104. [*M22*] 如果对于$\{1, \cdots, n\}$的一个排列$a_1 \cdots a_n$有

$$\sum_{k=1}^{n} k a_k = \sum_{k=1}^{n} (n+1-k)\, a_k$$

则说它是良平衡的。例如，当$n = 4$时，3142是良平衡的。

a) 证明当$n \bmod 4 = 2$时，没有良平衡的排列。

b) 证明如果$a_1 \cdots a_n$是良平衡的，则它的颠倒$a_n \cdots a_1$、它的取补$(n+1-a_1) \cdots (n+1-a_n)$以及它的逆$a_1' \cdots a_n'$也都是良平衡的。

c) 对于小的n值，确定良平衡的排列的个数。

▶105. [*26*] 一个弱序是这样一个关系$\preceq$，它是传递的（$x \preceq y$和$y \preceq z$，隐含$x \preceq z$）和完备的（$x \preceq y$或$y \preceq x$总成立）。如果$x \preceq y$和$y \preceq x$，我们可以写$x \equiv y$且如果$x \preceq y$和$y \npreceq x$则说$x \prec y$。在3个元素$\{1, 2, 3\}$上，有13个弱关系，即

$$1 \equiv 2 \equiv 3,\ 1 \equiv 2 \prec 3,\ 1 \prec 2 \equiv 3,\ 1 \prec 2 \prec 3,\ 1 \equiv 3 \prec 2,\ 1 \prec 3 \prec 2,$$
$$2 \prec 1 \equiv 3,\ 2 \prec 1 \prec 3,\ 2 \equiv 3 \prec 1,\ 2 \prec 3 \prec 1,\ 3 \prec 1 \equiv 2,\ 3 \prec 1 \prec 2,\ 3 \prec 2 \prec 1$$

a) 试说明如何系统地生成$\{1, \cdots, n\}$的所有弱序，作为以符号$\equiv$或$\prec$分开的数字的序列。

b) 一个弱序也可被表示为一个序列$a_1 \cdots a_n$，其中如果j为k个$\prec$符号所居前，则$a_j = k$。

例如，在{1, 2, 3}上的13个弱序分别为在这个形式下的000, 001, 011, 012, 010, 021, 101, 102, 100, 201, 110, 120, 210。求生成长度为n的所有这样序列的一个简单方法。

106. [*M40*] 用类似格雷码能否解习题105(b)?

▶107. [*30*]（约翰·霍·康维（John H. Conway），1973。）为玩“搅顶”的单人游戏，我们由洗一列标号为{1, …, n}的n张牌开始并把它们面朝上地放在一堆，然后如果顶上的牌是$k>1$，就把顶上的k张牌拿出然后又面朝下放在这堆的顶上。由此把排列从$a_1\cdots a_n$改变成$a_k\cdots a_1\ a_{k+1}\cdots a_n$。继续这样做直到顶上的牌是1为止。例如，当$n=5$时，7步骤序列

$$31452 \to 41352 \to 53142 \to 24135 \to 42135 \to 31245 \to 21345 \to 12345$$

可能出现。当$n=13$时，可能的最长序列是什么?

108. [*M27*] 如果“搅顶”游戏最长的n张牌有长度$f(n)$，试证明$f(n)\leqslant F_{n+1}-1$。

109. [*M47*] 试找出搅顶函数$f(n)$的好上界和下界。

110. [*25*]找出{0, …, 9}的使得

$$\{a_0, a_2, a_3, a_7\} = \{2, 5, 7, 8\}$$
$$\{a_1, a_4, a_5\} = \{0, 3, 6\}$$
$$\{a_1, a_3, a_7, a_8\} = \{3, 4, 5, 7\}$$
$$\{a_0, a_3, a_4\} = \{0, 7, 8\}$$

74 的所有排列，并且提议求解这种类型的大问题的一个算法。

▶111. [*M25*]人们已经提出了德·布鲁因循环的许多种面向排列的类似物。这其中最简单和最漂亮的是由布·沃·杰克逊（B. W. Jackson）在[*Discrete Math*（离散数学）**117** (1993), 141-150]上介绍的排列万有循环的思想，即$n!$个数字的这样一个循环，使得{1, …, n}的每一个排列作为$n-1$个连续数字恰好出现一次（并且抑制它多余的最后元素）。例如，(121323)是对于3的一个万有循环，而且实际上它是惟一这样的循环。

对于$n=4$，求排列的一个万有循环，并且证明对于所有$n\geqslant 2$，这样的循环存在。

75 ▶112. [*HM43*] 对于<9个对象的排列，精确地说有多少万有循环存在?

习题答案

7.2.1.1 节

1. 令$m_j = u_j - l_j + 1$，并访问$(a_1 + l_1, \cdots, a_n + l_n)$代替访问算法M中的$(a_1, \cdots, a_n)$。或者在此算法中把“$a_j \leftarrow 0$”改成“$a_j \leftarrow l_j$”和“$a_j = m_{j-1}$”改成“$a_j = u_j$”，并且在步骤M1中置$l_0 \leftarrow 0$，$u_0 \leftarrow 1$。

2. (0, 0, 1, 2, 3, 0, 2, 7, 0, 9)。

3. 当$j = k$时步骤M4被执行$m_1 m_2 \cdots m_k$次；因此总共是$\sum_{k=0}^{n} \prod_{j=1}^{k} m_j = m_1 \cdots m_n (1 + 1/m_n + 1/m_n m_{n-1} + \cdots + 1/m_n \cdots m_1)$。如果所有$m_j$是2或以上，这小于$2m_1 \cdots m_n$。（因此，我们应该记住有趣的格雷码方法，它每次访问只改动一个数字，实际上使数字改变的总数减少至多是2的一个因子。）

4. N1. [初始化。] 对于$0 < j < n$，置$a_j \leftarrow mj - 1$，其中$m_0 = 2$。

N2. [访问。] 访问n元组$(a_1, \cdots, a_n)$。

N3. [准备来减1。] 置$j \leftarrow n$。

N4. [必要时借。] 如果$a_j = 0$，置$a_j \leftarrow m_{j-1}$，$j \leftarrow j - 1$，并重复这一步骤。

N5. [减少，除非已完成。] 如果$j = 0$，则终止本算法，否则置$a_j \leftarrow a_{j-1}$，并返回到步骤N2。 ∎

5. 在一台像MMIX这样的机器上二进位的反射是容易的，但在其他计算机上，我们可进行如下：

R1. [初始化。] 置$j \leftarrow k \leftarrow 0$。

R2. [交换。] 交换$A[j + 1] \leftrightarrow A[k + 2^{n-1}]$。而且，如果$j < k$，则交换$A[j] \leftrightarrow A[k]$及$A[j + 2^{n-1} + 1] \leftrightarrow A[k + 2^{n-1} + 1]$。

R3. [推进k。] 置$k \leftarrow k + 2$，并且如果$k \geqslant 2^{n-1}$，则结束。

R4. [推进j。] 置$h \leftarrow 2^{n-2}$。如果$j \geqslant h$，则重复地置$j \leftarrow j - h$和$h \leftarrow h/2$直到$j < h$为止。然后置$j \leftarrow j + h$。（现在如果$k = (b_{n-1} \cdots b_0)_2$，则$j = (b_0 \cdots b_{n-1})_2$，返回到R2。 ∎

6. 如果$g((0b_{n-1} \cdots b_1 b_0)_2) = (0(b_{n-1}) \cdots (b_2 \oplus b_1)\ (b_1 \oplus b_0))_2$，则$g((1b_{n-1} \cdots b_1 b_0)_2) = 2^n + g((0\,\bar{b}_{n-1} \cdots \bar{b}_1 \bar{b}_0)_2) = (1(\bar{b}_{n-1} \cdots (\bar{b}_2 \oplus \bar{b}_1)\ (\bar{b}_1 \oplus \bar{b}_0))_2$，其中$\bar{b} = b \oplus 1$。 76

7. 为容纳$2r$个扇区，人们对于$2^n - r < k < 2^n + r$可以使用$g(k)$，其中$n = \lceil \lg r \rceil$，因为由(5)，$g(2^n - r) \oplus g(2^n + r - 1) = 2^n$。[乔·科·图梯尔（G. C. Tootill），*Proc. IEE* 103, Part B Supplement (1956), 434]。也参见习题26。

8. 对于$n \leftarrow n - 1$使用算法G，并且在右边包括校验位a_∞。（这产生$g(0)$, $g(2)$, $g(4)$, $\cdots$。）

9. 由于$v(1011000)$是奇数，因此代替最右环。

10. $A_n + B_n = g^{[-1]}(2^n - 1) = \lfloor 2^{n+1}/3 \rfloor$和$A_n = B_n + n$，因此$A_n = \lfloor 2^n/3 + n/2 \rfloor$，而且$B_n = \lfloor 2^n/3 - n/2 \rfloor$。

历史注记：早期日本数学家有马赖僮（1714—1783）在他的书*Shūki Sanpō* (1769) 的问题44中讨论了这个问题，并且发现在某个步骤数之后，n个环的难题可归结为（$n-1$）环的难题。在这个简化期间令$C_n = A_n - A_{n-1} = B_n - B_{n-1} + 1$是被删去的环数。有马赖僮注意到$C_n = 2C_{n-1} - [n\text{为偶}]$；因此对于$n = 9$，他可以计算$A_n = C_1 + C_2 + \cdots + C_n$，而无须真正知道公式 $C_n = \lceil 2^{n-1}/3 \rceil$。

两个多世纪以前，加尔达诺（Cardano）在他的*De Subtilitate Libri XXI* (Nuremberg: 1550) 第15本中已经提到“复杂的小环”。他写道，“它们是无用的但令人钦佩的微妙”，并错误地指出，为了删除7个环需要移动95次，而要把它们放回需要移动95次以上。约翰·沃里斯（Wallis）在他的*Algebra* 2 (Oxford: 1693) 的拉丁文版本第111章中，以7页篇幅来论述这个难题，给出了对于9环情况的详尽的但非最优的方法。他包括了使一个环通过杆滑动以及把它放进和放出的操作，而且他提示，有捷径可用，但他不试图来找出一个最短的解来。

11. 当$S_1 = S_2 = 1$时$S_n = S_{n-2} + 1 + S_{n-2} + S_{n-1}$的解是$S_n = 2^{n-1} - [n\text{为偶}]$。[*Math. Quest. Educational Times* 3 (1865), 66-67。]

12. (a) $n-1$个中国环的理论证明，格雷二进制码以方便的顺序（4, 31, 211, 22, 112, 1111, 121, 13）产生合成：

A1. [初始化。] 置$t \leftarrow 0$，$j \leftarrow 1$，$s_1 \leftarrow n$。（我们假定$n > 1$。）

A2. [访问。] 访问$s_1 \cdots s_j$，然后置$t \leftarrow 1 - t$，并且如果$t = 0$则转到A4。

A3. [奇步。] 如果$s_j > 1$，则置$s_j \leftarrow s_j - 1$，$s_{j+1} \leftarrow 1$，$j \leftarrow j + 1$；否则置$j \leftarrow j - 1$，以及$s_j \leftarrow s_j + 1$，返回A2。

A4. [偶步。] 如果$s_{j-1} > 1$，则置$s_{j-1} \leftarrow s_{j-1} - 1$，$s_{j+1} \leftarrow s_j$，$s_j \leftarrow 1$，$j \leftarrow j + 1$；否则置$j \leftarrow j - 1$，$s_j \leftarrow s_{j+1}$，$s_{j-1} \leftarrow s_{j-1} + 1$（但若$j - 1 = 0$则结束）。返回A2。 ∎

(b) 现在$q_1, \cdots, q_{t-1}$表示在杆上的环。

B1. [初始化。] 置$t \leftarrow 1$，$q_0 \leftarrow n$（我们假定$n > 1$）。

B2. [访问。] 置$q_t \leftarrow 0$，并访问$(q_0 - q_1) \cdots (q_{t-1} - q_t)$，如果$t$为偶则转到B4。

B3. [奇步。] 如果$q_{t-1} = 1$，则置$t \leftarrow t - 1$，否则置$q_t \leftarrow 1$和$t \leftarrow t + 1$。返回B2。

B4. [偶步。] 如果$q_{t-2} = q_{t-1} + 1$，则置$q_{t-2} \leftarrow q_{t-1}$，且$t \leftarrow t - 1$（但如果$t = 2$则结束）；否则置$q_t \leftarrow q_{t-1}$，$q_{t-1} \leftarrow q_t + 1$，$t \leftarrow t + 1$。返回B2。 ∎

这些算法[见加·密斯拉（J. Misra），*ACM Trans. Math. Software* 1 (1975), 285]即使在它们的初始化步骤中都无循环。

13. 在步骤A1中，也置$C \leftarrow 1$，在步骤A3中，如果$s_j > 1$则置$C \leftarrow s_j C$，否则
77 $C \leftarrow C/(s_{j-1} + 1)$。在步骤A4中，如果$s_{j-1} > 1$，则置$C \leftarrow s_{j-1} C$，否则$C \leftarrow C/(s_{j-2} + 1)$。类似的修改适用于步骤B1，B3，B4。对于合成$1 \cdots 1$，为容纳对于组成$1 \cdots 1$的值$C = n!$需要充分的精度；通过假定算术操作花费单位时间，我们引申无循环性的定义。

14. S1. [初始化。] 置$j \leftarrow 0$。

S2. [访问。] 访问串$a_1 \cdots a_j$。

S3. [加长。] 如果$j < n$，则置$j \leftarrow j+1$，并返回S2。

S4. [增加。] 如果$a_j < m_j - 1$，则置$a_j \leftarrow a_j + 1$，并返回S2。

S5. [缩短。] 置$j \leftarrow j-1$，而且如果$j > 0$则返回S4。 ∎

15. T1. [初始化。] 置$j \leftarrow 0$。

T2. [偶访问。] 如果j为偶，则访问串$a_1 \cdots a_j$。

T3. [加长。] 如果$j < n$，则置$j \leftarrow j+1$，$a_j \leftarrow 0$，并返回T2。

T4. [奇访问。] 如果j为奇，则访问串$a_1 \cdots a_j$。

T5. [增加。] 如果$a_j < m_j - 1$，则置$a_j \leftarrow a_j + 1$，并返回T2。

T6. [缩短。] 置$j \leftarrow j-1$，而且如果$j > 0$则返回T4。 ∎

尽管乍一看去它包含循环，但这个算法是无循环的；至多四个步骤分开连续的访问。基本思想同习题2.3.1-5有关，也和“预后根顺序”遍历有关（算法7.2.1.6 Q）。

16. 对于$1 \leqslant j \leqslant n$，假设$\texttt{LINK}(j-1) = j + nb_j$，对于$1 < j \leqslant n$，$\texttt{LINK}(j-1+n) = j + n(1-b_j)$。这些链接表示$(a_1, \cdots, a_n)$当且仅当$g(b_1 \cdots b_n) = a_1 \cdots a_n$，所以我们可以使用无循环的格雷二进生成程序实现期望的结果。

17. 对于$0 \leqslant j, k < 8$，把3个二进位码的连接$(g(j), g(k))$放在j行k列处。[不难证明这实际上是惟一的解，但排列和/或坐标取补和/或转动行除外，因为，当向北或向南移动时改变的坐标仅依赖于行，而且一个类似的命题也适用于列。4立体和4×4圆环体之间的卡诺自同构可以追溯到由威·凯斯特（W. Keister）、阿·恩·里特奇（A. E. Ritchie）和塞·哈·瓦斯布恩（S. H. Washburn）所著的*The Design of Switching Circuits*（开关线路设计）(1951) 的174页。顺便说一下，凯斯特继续研究来设计称为“转出”（SpinOut）的中国环的巧妙版本，以及叫做16进难题的一个推广。见美国专利3637215-3637216 (1972)。]

18. 使用2个二进位的格雷码来把数字$u_j = (0, 1, 2, 3)$表示为二进位对$u'_{2j-1}\, u'_{2j} = (00, 01, 11, 10)$。[李始元在（*IEEE Trans.* IT-4 (1958), 77-82）中介绍了他的度量，一个类似的$m/2$个二进位的编码对于m的偶数值有效；例如，当$m = 8$时，我们可以通过（0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000）来表示（0, 1, 2, 3, 4, 5, 6, 7）。但当$m > 4$时，这样一个方案忽略某些二进制的模式。]

19. (a) 一个取模的格雷四进制算法需要比算法M稍微少一些的计算，但关系不大，因为256是如此之小。结果是$z_0^8 + z_1^8 + z_2^8 + z_3^8 + 14(z_0^4 z_2^4 + z_1^4 z_3^4) + 56 z_0 z_1 z_2 z_3 (z_0^2 + z_2^2)(z_1^2 + z_3^2)$。

(b) 以$(1, z, z^2, z)$代替(z_0, z_1, z_2, z_3)给出$1 + 112z^6 + 30z^8 + 112z^{10} + z^{16}$；因此所有非零的李权是$\geqslant 6$。现在使用上道题中的构造来把$(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$的每一个转换成一个16个二进位的数。

20. 从u'恢复四进制向量$(u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_\infty)$，并且使用算法4.6.1 D来

求$u_0 + u_1x + \cdots + u_6x^6$除以$g(x)$mod 4的余数；该算法可被使用（尽管有系数不属于一个域的事实存在），因为$g(x)$是首一的。把余数表达为$x^j + 2x^k$(modulo $g(x)$和4),
78 并令$d = (k - j) \bmod 7$，$s = (u_0 + \cdots + u_6 + u_\infty) \bmod 4$。

情况1。$s = 1$：如果$k = \infty$，则误差为x^j（换言之，正确的向量有$u_j \leftarrow (u_j - 1) \bmod 4$)；否则会有三个或以上的误差。

情况2。$s = 3$：如果$j = k$则误差为$-x^j$；否则会出现≥ 3个误差。

情况3。$s = 0$：如果$j = k = \infty$，则不会造成误差；如果$j = \infty$且$k < \infty$，至少会造成4个误差。否则误差是$x^a - x^b$，按照$d = (0, 1, 2, 3, 4, 5, 6, \infty)$，其中$a = (j + (\infty, 6, 5, 2, 3, 1, 4, 0)) \bmod 7$，$b = (j + 2d) \bmod 7$。

情况4。$s = 2$：如果$j = \infty$，则误差是$2x^k$。否则误差是

$$
\begin{aligned}
&x_j + x^\infty，\text{如果} k = \infty;\\
&-x^j - x\infty，\text{如果} d = 0;\\
&x^a + x^b，\text{如果} d \in \{1, 2, 4\}, a = (j - 3d) \bmod 7, b = (j - 2d) \bmod 7;\\
&-x^a - x^b，\text{如果} d \in \{3, 5, 6\}, a = (j - 3d) \bmod 7, b = (j - d) \bmod 7。
\end{aligned}
$$

给定$u' = (1100100100001111)_2$，我们有$u = (2, 0, 3, 1, 0, 0, 2, 2)$和$2 + 3x^2 + x^3 + 2x^6 \equiv 1 + 3x + 3x^2 \equiv x^5 + 2x^6$，也有$s = 2$。因此误差是$x^2 + x^3$，而且最接近的无误差码字为 (2, 0, 2, 0, 0, 0, 2, 2)。算法4.6.1 D告诉我们，$2 + 2x^2 + 2x^6 \equiv (2 + 2x + 2x^3)g(x)$ (modulo 4)；所以8个信息位对应于 $(v_0, v_1, v_2, v_3) = (2, 2, 0, 2)$。[一个更智能化的算法将说："啊哈，这是π的头16个二进位"。]

关于推广到基于四进制向量的其他有效编码方案，参见哈姆蒙斯（Hammons）、库玛尔（Kumar）、卡尔德邦克（Calderbank）、斯罗安（Sloane）以及索里（Solé）的经典论文[*IEEE Trans.* IT-40 (1994), 301-319]。

21. (a) $C(\varepsilon) = 1$，$C(0\alpha) = C(1\alpha) = C(\alpha)$，以及$C(*\alpha) = 2C(\alpha) - [10\cdots0 \in \alpha]$。迭代这个递归式给出$C(\alpha) = 2^t - 2^{t-1}e_t - 2^{t-2}e_{t-1} - \cdots - 2^0e_1$，其中$e_j = [10\cdots0 \in \alpha_j]$且$\alpha_j$是后接第$j$个星号的$\alpha$的后缀。在这个例子里我们有$\alpha_1 = *10**0*$，$\alpha_2 = 10**0*$，$\cdots$，$\alpha_5 = \varepsilon$；因此$e_1 = 0$，$e_2 = 1$，$e_3 = 1$，$e_4 = 0$以及$e_5 = 1$（根据约定），因此$C(**10**0*) = 2^5 - 2^4 - 2^2 - 2^1 = 10$。

(b) 我们可以删去尾部星号使得$t = t'$，则$e_t = 1$意味着$e_{t-1} = \cdots = e_1 = 0$。[情况$C(\alpha) = 2^{t'-1}$出现当且仅当$\alpha$以$10^j*^k$结束。]

(c) 为计算在所有t子立体上$C(\alpha)$的和，注意$\binom{n}{t}$从集在n元组$0\cdots0$开始，而$\binom{n-1}{t}$在每个随继的n元组开始（即对于含该n元组的每个t子立体的一个丛集，并确定改变的二进位）。因此平均值是$\left(\binom{n}{t} + (2^n - 1)\binom{n-1}{t}\right)/2^{n-t}\binom{n}{t} = 2^t(1 - t/n) + 2^{t-n}(t/n)$。[(c) 中公式对于任何$n$个二进位的格雷通路成立，但(a)和(b)是特定于反射的格雷二进码的，这些结果是由克·法罗特索斯（C. Faloutsos）给出的，*IEEE Trans.* SE-14 (1988), 1381-1393。]

22. 令$\alpha*^j$和$\beta*^k$是一个格雷二进制检索结构的连续的叶，其中α和β是二进制的串且$j < k$。则α的最后$k - j$个二进位是使得α和$\beta\alpha'$为格雷二进制码的连续元素的串

α'，因此是相邻的。[在威廉·詹·达里（William J. Dally）所著的*A VLSI Architecture for Concurrent Data Structures*（一个并发数据结构的超大规模集成电路的结构）(Kluwer，1987)第3章中，讨论了这一性质在立体连通信息传送并行计算机上的有趣应用。]

23. $2^j = g(k) \oplus g(l) = g(k \oplus l)$ 意味着$l = k \oplus g^{[-1]}(2^j) = k \oplus (2^{j+1} - 1)$。换言之，如果$k = (b_{n-1}\cdots b_0)_2$，我们有$l = (b_{n-1}\cdots b_{j+1}\bar{b}_j\cdots\bar{b}_0)_2$。

24. 像通常一样，定义$g(k) = k \oplus \lfloor k/2 \rfloor$，我们发现$g(k) = g(-1-k)$；因此有2个二进制的整数$k$，使得$g(k)$ 有一个2进制的值l。其中之一为偶数，另一个为奇数。我们可以方便地把$g^{[-1]}$定义为偶数的解。则对于$j \geqslant 0$，(8) 由$b_j = a_{j-1} \oplus \cdots \oplus a_0$所代替。例如，由这个定义$g^{[-1]}(1) = -2$，当$l$是一个通常的整数时，$g^{[-1]}(l)$ 的“符号”是l的奇偶位。 79

25. 令$p = k \oplus l$；习题7.1-00告诉我们$2^{\lfloor \lg p \rfloor + 1} - p \leqslant |k-l| \leqslant p$，我们有$v(g(p) = v(g(k) \oplus g(l) = t$，当且仅当有正整数$j_1, \cdots, j_t$使得$p = (1^{j_1}0^{j_2}1^{j_3}\cdots(0\text{或}1)^{j_t})_2$。当$j_1 = n + 1 - t$和$j_2 = \cdots = j_t = 1$时，最大可能的$p < 2^n$出现，并且产生 $p = 2^n - \lceil 2^t/3 \rceil$。当$j_2 = \cdots = j_t = 1$时，最小可能的$2^{\lfloor \lg p \rfloor + 1} - p = (1^{j_2}0^{j_3}\cdots(1\text{或}0)^{j_t})_2 + 1$出现，并且产生 $p = \lceil 2^t/3 \rceil$。[阮宗光，*IEEE Trans.* IT-20 (1974), 668；也见斯·罗·卡威欧（S. R. Cavior），*IEEE Trans.* IT-21 (1975)，596。]

26. 令$N = 2^{n_t} + \cdots + 2^{n_1}$，其中$n_t > \cdots > n_1 \geqslant 0$；而且，令$\Gamma_n$是对于$\{0, 1, \cdots, 2^n - 1\}$的以0开始和以1结束的任何格雷码，而$\Gamma_0$为简单的0而已，使用

$$\Gamma_{n_t}^R,\ 2^{n_t} + \Gamma_{n_{t-1}},\ \cdots,\ 2^{n_t} + \cdots + 2^{n_3} + \Gamma_{n_2}^R,\ 2^{n_t} + \cdots + 2^{n_2} + \Gamma_{n_1},\text{如果}t\text{为偶；}$$

$$\Gamma_{n_t},\ 2^{n_t} + \Gamma_{n_{t-1}}^R,\ \cdots,\ 2^{n_t} + \cdots + 2^{n_3} + \Gamma_{n_2}^R,\ 2^{n_t} + \cdots + 2^{n_2} + \Gamma_{n_1},\text{如果}t\text{为奇。}$$

27. 一般地说，如果$k = (b_{n-1}\cdots b_0)_2$，则$S_n$的第（$k+1$）个最大的数等于对应于符号模式$g(k) = (a_{n-1}\cdots a_0)_2$的

$$1/(2-(-1)^{a_{n-1}}/(2-\cdots/(2-(-1)^{a_1}/(2-(-1)^{a_0}))\cdots))$$

因此，我们在给定其秩的前提下，可以在$O(n)$ 步内计算S_n的任何元素。置$k = 2^{100} - 10^{10}$和$n = 100$产生答案373065177/1113604409。[每当$f(x)$ 为正的和单调的函数时，正如由希·埃·萨尔泽（H. E. Salzer）所发现的（*CACM* 16 (1973)，180），2^n个元素$f(\pm f(\cdots \pm f(\pm x)\cdots))$ 按照格雷二进制编码是有序的。然而，在这个特殊情况下，存在另一个获得答案的方法，因为使用4.5.3节的记号，我们也有$S_n = //2, \pm 2, \cdots, \pm 2, \pm 1//$，通过对于$k$的交替二进位取补，在这个形式下的连分数是有序的。]

28. (a) 当$t = 1, 2, \cdots$时，均值（G_t）的二进位a_j跑遍周期序列

$$0, \cdots, 0, *, 1, \cdots, 1, *, 0, \cdots, 0, *, \cdots$$

连同在每2^{1+j}步时出现星号。因此对应于$\lfloor (t-1)/2 \rfloor$和$\lfloor t/2 \rfloor$的二进制表示的串是均值。而且，在所有均值同$\lfloor (t-1)/2 \rfloor$和$\lfloor t/2 \rfloor$的共同二进位相一致这个意义下，这些串实

际上是“极端”情况，因此星号出现在它们不一致的地方。例如当$t = 100 = (01100100)_2$和$n = 8$时，我们有均值$(G_{100}) = 001100{**}$。

(b) 由于$G_{2t} = 2G_t \cup (2G_t + 1)$，我们可以假定$t = (a_{n-2}\cdots a_1 a_0 1)_2$为奇数。如果在格雷二进制编码中$\alpha$是$g(p)$且$\beta$是$g(q)$，我们有$p = (p_{n-1}\cdots p_0)_2$和$q = (p_{n-1}\cdots p_{j+1}\bar{p}_j\cdots\bar{p}_0)_2$，而且$a_{n-1}a_{n-2} = 01 = p_{n-1}p_{n-2}$。我们不能有$p < t \leqslant q$，因为这样意味着$j = n-1$和$p_{n-3} = p_{n-4} = \cdots = p_0 = 1$。[参见阿·杰·伯恩斯坦（A. J. Bernstein）、肯·斯泰格里兹（K. Steiglitz）以及约·爱·霍普格罗夫特（J. E. Hopcroft），*IEEE Trans.* IT-12 (1966)，425-430。]

29. 假设$p \neq 0$，令$l = \lfloor \lg p \rfloor$和对于$0 \leqslant a < 2^{n-l}$，$S_a = \{s \mid 2^l a \leqslant s < 2^l(a+1)\}$。于是对于所有$k \in S_a$，$(k \oplus p) - k$有一个不变符号，而且

$$\sum_{k\in S_a} |(k \oplus p) - k| = 2^l |S_a| = 2^{2l}$$

再有$g^{[-1]}(g(k) \oplus p) = k \oplus g^{[-1]}(p)$，而且$\lfloor \lg g^{[-1]}(p) \rfloor = \lfloor \lg p \rfloor$。因此

$$\frac{1}{2^n}\sum_{k=0}^{2^n-1} \left|g^{[-1]}(g(k)\oplus p) - k\right| = \frac{1}{2^n}\sum_{a=0}^{2^{n-l}-1}\ \sum_{k\in S_a} \left|(k \oplus g^{[-1]}(p)) - k\right| = \frac{1}{2^n}\sum_{a=0}^{2^{n-1}-1} 2^{2l} = 2^l$$

[参见小莫尔干·马·巴赫纳尔（Morgan M. Buchner, Jr.），*Bell System Tech. J.* 48 (1969)，3113-3130。]

80

30. 包含$k > 1$的循环有长度$2^{\lfloor \lg \lg k \rfloor + 1}$，因为由等式(7)容易证明，如果$k = (b_{n-1}\cdots b_0)_2$，我们有

$$g^{[2^l]}(k) = (c_{n-1}\cdots c_0)_2 \quad \text{其中} c_j = b_j \oplus b_{j+l+1}$$

为排列使得$\lfloor \lg k \rfloor = t$的所有元素$k$，有两种情况：如果$t$是一个2的幂，则含$2\lfloor k/2 \rfloor$的循环也含$2\lfloor k/2 \rfloor + 1$。所以我们必须对于$t-1$的循环前导实行加倍。否则含$2\lfloor k/2 \rfloor$的循环同含$2\lfloor k/2 \rfloor + 1$的循环不相交。所以，$L_t = (2L_{t-1}) \cup (2L_{t-1}+1) = (L_{t-1}{*})_2$。由约格·安德特（Jörg Arndt）在2001年所发现的这一论断，确立了提示并产生了以下的算法：

P1. [初始化。] 置$t \leftarrow 1$，$m \leftarrow 0$。（我们可以假设$n \geqslant 2$。）

P2. [通过前导的循环。] 置$r \leftarrow m$。以$k = 2^t + r$执行算法Q。然后如果$r > 0$，则置$r \leftarrow (r-1)\ \&\ m$并重复直到$r = 0$为止。[见习题7.1-00。]

P3. [增加$\lg k$。] 置$t \leftarrow t + 1$。如果t现在等于n则终止；否则置$m \leftarrow 2m + [t\ \&\ (t-1) \neq 0]$且返回P2。 ▮

Q1. [开始一个循环。] 置$s \leftarrow X_k$，$l \leftarrow k$，$j \leftarrow l \oplus \lfloor l/2 \rfloor$。

Q2. [遵循此循环。] 如果$j \neq k$，则置$X_l \leftarrow X_j$，$l \leftarrow j$，$j \leftarrow l \oplus \lfloor l/2 \rfloor$，并重复直到$j = k$为止，然后置$X_l \leftarrow s$。 ▮

31. 我们从f_n获得一个域当且仅当我们从$f_n^{[2]}$获得一个域，它把$(a_{n-1}\cdots a_0)_2$变为$((a_{n-1}\oplus a_{n-2})(a_{n-1}\oplus a_{n-3})(a_{n-2}\oplus a_{n-4})\cdots(a_2 \oplus a_0)(a_1))_2$。令$c_n(x)$为定义这个变换的

矩阵A的特征多项式mod 2；则$c_1(x)=x+1$，$c_2(x)=x^2+x+1$，且$c_{j+1}(x)=xc_j(x)+c_{j-1}(x)$。由凯利-哈密顿定理，$c_n(A)$是一个零矩阵，一个域被得到当且仅当$c_n(x)$是一个本原多项式，而且这一条件可如同在3.2.2节中那样进行测试。N最初的这样的值是1, 2, 3, 5, 6, 9, 11, 14, 23, 26, 29, 30, 33, 35, 39, 41, 51, 53, 65, 69, 74, 81, 83, 86, 89, 90, 95。

[向后运行这个递归式表明$c_{-j-1}(x)=c_j(x)$，因此$c_j(x)$整除$c_{(2j+1)k+j}(x)$；例如，$c_{3k+1}(x)$总是$x+1$的倍数。因此当$j>0$且$k>0$时所有形如$2jk+j+k$的数n都被排除。多项式$c_{18}(x)$，$c_{50}(x)$，$c_{98}(x)$和$c_{99}(x)$是不可约的，但不是本原的。]

32. 大部分为真，但在$w_k(x)$改变符号的一些点处为假。(沃尔斯原来提议$w_k(x)$在这样一些点应为零，但在这里所采取的约定是更好的，因为它使像(15)-(19)这样的简单公式对于所有x都成立。)

33. 由对k的归纳法我们有，对于$0\leqslant x<1/2$

$$w_k(x)=w_{\lfloor k/2\rfloor}(2x)=r_1(2x)^{b_1+b_2}r_2(2x)^{b_2+b_3}+\cdots=r_1(x)^{b_0+b_1}r_2(x)^{b_1+b_2}r_3(x)^{b_2+b_3}\cdots$$

因为在这个范围内$r_j(2x)=r_{j+1}(x)$且$r_1(x)=1$。而且当$1/2\leqslant x<1$时，

$$\begin{aligned}w_k(x)=(-1)^{\lceil k/2\rceil}w_{\lfloor k/2\rfloor}(2x-1)&=r_1(x)^{b_0+b_1}r_1(2x-1)^{b_1+b_2}r_2(2x-1)^{b_2+b_3}\cdots\\&=r_1(x)^{b_0+b_1}r_2(x)^{b_1+b_2}r_3(x)^{b_2+b_3}\cdots\end{aligned}$$

因为$\lceil k/2\rceil\equiv b_0+b_1(\text{modulo }2)$，且对于$j\geqslant 1$，$r_j(2x-1)=r_{j+1}(x-1/2)=r_{j+1}(x)$。

34. $p_k(x)=\prod_{j>0}r_{j+1}^{bj}$，因此$w_k(x)=p_k(x)p_{\lfloor k/2\rfloor}(x)=p_{g(k)}(x)$。[雷·爱·阿·克·佩里(R. E. A. C. Paley)，*Proc. London Math. Soc.* (2) **34** (1932)，241-279。] 81

35. 如果$j=(a_{n-1}\cdots a_0)_2$且$k=(b_{n-1}\cdots b_0)_2$，则在j行k列的元素为$(-1)^{f(j,k)}$，其中$f(j,k)$是使得：$r=s$(哈达玛)；$r+s=n-1$(佩里)；$r+s=n$或$n-1$(沃尔斯)的所有a_rb_s之和。

令R_n，F_n和G_n是把$j=(a_{n-1}\cdots a_0)_2$分别变为$k=(a_0\cdots a_{n-1})_2$，$k=2^n-1-j=(\bar{a}_{n-1}\cdots\bar{a}_0)_2$和$k=g^{[-1]}(j)=((a_{n-1})\cdots(a_{n-1}\oplus\cdots\oplus a_0))_2$的排列的排列矩阵。于是，使用克洛尼克矩阵乘积，我们有递归公式

$$R_{n+1}=\begin{pmatrix}R_n\otimes(1\ \ 0)\\R_n\otimes(0\ \ 1)\end{pmatrix},\quad F_{n+1}=F_n\otimes\begin{pmatrix}0&1\\1&0\end{pmatrix},\quad G_{n+1}=\begin{pmatrix}G_n&0\\0&G_nF_n\end{pmatrix}$$

$$H_{n+1}=H_n\otimes\begin{pmatrix}1&1\\1&\bar{1}\end{pmatrix},\quad P_{n+1}=\begin{pmatrix}P_n\otimes(1\ \ 1)\\P_n\otimes(1\ \ \bar{1})\end{pmatrix},\quad W_{n+1}=\begin{pmatrix}W_n\otimes(1\ \ 1)\\F_nW_n\otimes(1\ \ \bar{1})\end{pmatrix}$$

于是$W_n=G_n^TP_n=P_nG_n$；$H_n=P_nR_n=R_nP_n$；而且$P_n=W_nG_n^T=G_nW_n=H_nR_n=R_nH_n$。

36. W1. [哈达玛变换。] 对于$k=0,1,\cdots,n-1$，对于所有使$\lfloor j/2^k\rfloor$为偶数的j($0\leqslant j<2^n$)，以$(X_j+X_{j+2^k},X_j-X_{j+2^k})$来代替$(X_j,X_{j+2^k})$。(这些操作有效地置$X^T\leftarrow H_nX^T$。)

W2. [位颠倒。] 应用习题5的算法到向量X上。(在习题35的记号下，这些操作有效地置$X^T\leftarrow R_nX^T$。)

W3. [格雷二进制排列。] 应用习题30的算法到向量X上。(这些操作有效地置$X^T \leftarrow G_n^T X_n^T$。) ▮

如果n有习题31中的特殊值之一，则把步骤W2同步骤W3组合在一起成为单个排列步骤可能更快。

37. 如果$k = 2^{e_1} + \cdots + 2^{e_t}$且$e_1 > \cdots > e_t \geqslant 0$，则符号改变出现在$S_{e_1} \cup \cdots \cup S_{e_t}$处，其中

$$S_0 = \left\{\frac{1}{2}\right\},\ S_1 = \left\{\frac{1}{4}, \frac{3}{4}\right\},\ \cdots,\ S_e = \left\{\frac{2j+1}{2^e} \middle| 0 \leqslant j < 2^e\right\}$$

因此在$(0..x)$中符号改变的个数为$\sum_{j=1}^{t} \left\lfloor 2^{e_j} x + \frac{1}{2} \right\rfloor$。置$x = l/(k+1)$给出$l + O(t)$个改变；所以第$l$个距$l/(k+1)$的距离至多为$O(v(k)/2^{\lfloor \lg k \rfloor})$。

[这个论断使得满足$|z_{kl} - l/(k+1)| = \Omega((\log k)/k)$的无限多个对$(k, l)$似乎为真。但是却没有这样的“坏”对的显式构造立即显现出来。]

38. 令$t_0(x) = 1$且$t_k(x) = \omega^{\lfloor 3x \rfloor \lceil 2k/3 \rceil} t_{\lfloor k/3 \rfloor}(3x)$，其中$\omega = e^{2\pi i/3}$，则$t_k(x)$在$x$从0增为1时围绕原点摇晃$2/3k$次。如果$s_k(x) = \omega_{\lfloor 3^k x \rfloor}$是拉德曼彻函数$r_k(x)$的三进制类似，如同在取模的三进制格雷码那样，当$k = (b_{n-1} \cdots b_0)_3$时，我们有$t_k(k) = \prod_{j>0} s_{j+1}(x)^{b_j - b_{j+1}}$。

39. 让我们把符号叫做$\{x_0, x_1, \cdots, x_7\}$而不是$\{a, b, c, d, e, f, g, h\}$。我们要来求$\{0, 1, \cdots, 7\}$的一个排列$p$，使得在$j$行$k$列处有对于$(-1)^{j \cdot k} x_{p(j) \oplus k}$的矩阵正交行的行；这个条件等价于要求：对于$0 \leqslant j < j' < 8$

$$(j + j') \cdot (p(j) + p(j')) \equiv 1 \text{ (modulo 2)}$$

一个解是$p(0) \cdots p(7) = 01725634$，产生恒等式$(a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2)$
82 $(A^2 + B^2 + C^2 + D^2 + E^2 + F^2 + G^2 + H^2) = \mathcal{A}^2 + \mathcal{B}^2 + \mathcal{C}^2 + \mathcal{D}^2 + \mathcal{E}^2 + \mathcal{F}^2 + \mathcal{G}^2 + \mathcal{H}^2$，其中

$$\begin{pmatrix} \mathcal{A} \\ \mathcal{B} \\ \mathcal{C} \\ \mathcal{D} \\ \mathcal{E} \\ \mathcal{F} \\ \mathcal{G} \\ \mathcal{H} \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f & g & h \\ b & -a & d & -c & f & -e & h & -g \\ h & g & -f & -e & d & c & -b & -a \\ c & -d & -a & b & g & -h & -e & f \\ f & e & h & g & -b & -a & -d & -c \\ g & -h & e & -f & -c & d & -a & b \\ d & c & -b & -a & -h & -g & f & e \\ e & -f & -g & h & -a & b & c & -d \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{pmatrix}$$

[这个恒等式是由卡·费·迪根（C. F. Degen）发现的，见*Mémoires de l' Acad. Sci. St. Petersbourg* (5) 8 (1818)，207-219。有关的八进的钟在约·卡·巴耶兹（J. C. Baez）所写的一篇有趣综述中作了讨论，见*Bull. Amer. Math. Soc.* 39 (2002)，145-205。]

(b) 没有16 × 16的解。人们可得到最接近的一个是

$$p(0) \cdots p(15) = 0\ 1\ 11\ 2\ 14\ 15\ 13\ 4\ 9\ 10\ 7\ 12\ 5\ 6\ 3\ 8$$

当且仅当$j \oplus j' = 5$时它不成立。(参见*Philos. Mag.* **34** (1867), 461-475。在此文的§9、§10、§11和§13中，西尔威斯特指出并证明了被当作哈达玛变换而为人所知的一些基本结果——尽管哈达玛本人把这归功于西尔威斯特[*Bull. des Sciences Mathématiques* (2) 17 (1893), 240-246]。而且，西尔威斯特使用1的第m个根，在§14中引进m^n个元素的变换。)

40. 是的：这个改变事实上将跑遍在词典顺序的二进制顺序（而不是在格雷二进制顺序）下的被交换的子集。(当我们跑遍矩阵的行的所有线性组合时，在mod 2下的0和1的任何非奇异5×5矩阵将生成所有32个可能性。）最重要的事情是直尺函数的出现，或者是某个其他格雷码δ序列的出现，而不是在每步中只有一个a_j改变的事实。在像这样的情况下，任何个数的a_j都可在相同的耗费下同时改变。

41. 至多16个；例如，`fired`, `fires`, `finds`, `fines`, `fined`, `fares`, `fared`, `wares`, `wards`, `wands`, `wanes`, `waned`, `wines`, `winds`, `wires`, `wired`。从`paced/links`和`paled/mints`我们也得到16个，也许从一个词和一个反面的非词的混合。

42. 假设$n \leqslant 2^{2^r} + r + 1$且令$s = 2^r$。对于$0 \leqslant j < 2^s$和$0 \leqslant k < s$，我们使用$2^{r+s}$个二进位$f_{jk}$的一个辅助表，表示如同在算法L中的焦点指针，连同一个辅助的s个二进位“寄存器”$j = (j_{s-1}\cdots j_0)_2$，以及一个$(r+2)$个二进位“程序计数器”$p = (p_{r+1}\cdots p_0)_2$。在每步中我们考察程序计数器，可能还有$j$寄存器以及$f$二进位之一；然后基于看到的二进位，我们对格雷码的一个位取补，对程序计数器的一个位取补。可能还改变一个j或f的二进位，因此相对于最高有效的$n-r-2$个二进位仿真步骤L3。

例如，以下是当$r = 1$时的构造：

$p_2p_1p_0$	改变	设置		$p_2p_1p_0$	改变	设置	
0 0 0	a_0, p_0	$j_0 \leftarrow f_{00}$	$j \leftarrow f_0$	1 1 0	a_0, p_0	$f_{j0} \leftarrow f_{(j+1)0}$	$f_j \leftarrow f_{j+1}$
0 0 1	a_1, p_1	$j_1 \leftarrow f_{01}$		1 1 1	a_1, p_1	$f_{j1} \leftarrow f_{(j+1)1}$	
0 1 1	a_0, p_0	$f_{00} \leftarrow 0$	$f_0 \leftarrow 0$	1 0 1	a_0, p_0	$f_{(j+1)0} \leftarrow (j+1)_0$	$f_{j+1} \leftarrow j+1$
0 1 0	a_2, p_2	$f_{01} \leftarrow 0$		1 0 0	a_{j+3}, p_2	$f_{(j+1)1} \leftarrow (j+1)_1$	

当这个过程趋向于改变二进位a_n时停止。 83

事实上，如果我们允许自己来考察某些格雷二进制的位以及辅助位，则每一步我们只需改变一个辅助二进位，因为$p_r\cdots p_0 = a_r\cdots a_0$，而且当$j$没有它的最后值$2^s - 1$时，我们可以以更灵巧的方法来设置$f_0 \leftarrow 0$。这个于2001年由弗雷德曼所提出的构造，对于他在 [*SICOMP* 7 (1978), 134-146] 上发表的另一个构造作了改进。通过一个更精心设计的构造，有可能把辅助二进位的个数减少到$O(n)$。

43. 这个数是由希尔弗曼、维克斯和萨姆普森估计的[*IEEE Trans.* IT-29 (1983), 894-901]，大约为7×10^{22}。精确的计算可能是可行的，因为每个6个二进位的格雷循环仅有5个或更少的段，它们位于对应于至少6个坐标之一的5立体中。(在尚未发表的工作中，史蒂夫·温格尔（Steve Winker）已经使用类似的思想于

1972年在一个“普通的”计算机上花费少于15分钟来计算$d(5)$。)

44. 所有恰带有坐标j的两次出现的$(n+1)$个二进位的δ序列是通过下列构造产生的；令$\delta_1\cdots\delta_{2^n-1}$和$\varepsilon_1\cdots\varepsilon_{2^n-1}$为对于格雷通路的$n$个二进位的$\delta$序列，且$2^{\delta_1}\oplus\cdots\oplus 2^{\delta_{2^n-1}}=2^{\varepsilon_1}\oplus\cdots\oplus 2^{\varepsilon_{2^n-1}}$。对于某个满足$0\leqslant k<2^n$的$k$，形成循环

$$\delta_{k+1}\cdots\delta_{2^n-1}n\varepsilon_1\cdots\varepsilon_{2^n-1}n\delta_1\cdots\delta_k$$

然后交换$n\leftrightarrow j$。

类似地，恰有坐标h和j的两次出现（且h在j之前）的所有$(n+2)$个二进位的δ序列是由四个n个二进位的序列$\delta_1\cdots\delta_{2^n-1}$，…，$\eta_1\cdots\eta_{2^n-1}$，其中$2^{\delta_1}\oplus\cdots\oplus 2^{\eta_{2^n-1}}=0$，通过在

$$\delta_{k+1}\cdots\delta_{2^n-1}n\varepsilon_1\cdots\varepsilon_{2^n-1}(n+1)\varsigma_1\cdots\varsigma_{2^n-1}n\eta_1\cdots\eta_{2^n-1}(n+1)\delta_1\cdots\delta_k$$

中交换$n\leftrightarrow h$和$n+1\leftrightarrow j$产生的。

设$a(n)$和$b(n)$是在部分(a)和部分(b)中定义的n个二进位循环的个数；则$(a(1),\cdots,a(5))=(1,0,0,1920,318996480)$，$(b(1),\cdots,b(5))=(0,2,12,384,4200960)$。上述构造证明，如果有$A(n)$和$B(n)$种方法来构造分别的序列$\delta$、$\varepsilon$、$\varsigma$和$\eta$，则$a(n+1)+2b(n+1)=2^n(n+1)A(n)$，$b(n+2)=2^n(n+2)(n+1)B(n)$。如果我们限定格雷通路可扩充到格雷循环的那些情况且$\delta_0=\varepsilon_0=\varsigma_0=\eta_0$，我们得到$a'(n+1)$和$b'(n+2)$序列，其中$a'(n+1)+2b'(n+1)=2^n(n+1)d(n)^2/n$且$b'(n+2)=2^n(n+2)(n+1)\,d(n)^4/n^3$。

45. 我们有$d(n+1)\geqslant 2^n d(n)^2/n$，因为$2^n d(n)^2/n$是恰好带有两次0的出现的$(n+1)$个二进位的$\delta$序列个数的下界。因此$d(n+1)^{1/2^{n+1}}>d(n)^{1/2^n}$；而且对于$n\geqslant 5$，$d(n)\geqslant\frac{5}{32}a^{2^n}$，其中$\alpha=\left(\frac{32}{5}d(5)\right)^{1/32}\approx 2.06$。

其实，通过使用上一习题，我们能确立甚至更快的增长，因为对于$n\geqslant 5$，$d(n+1)\geqslant a'(n+1)+b'(n+1)$而且$b'(n+1)\leqslant(25/64)(n+1)d(n)^2/n$。因此对于$n\geqslant 5$，$d(n+1)\geqslant(2^n-(25/64))(n+1)d(n)^2/n$，而且对于这个关系式的迭代证明

$$\lim_{n\to\infty} d(n)^{1/2^n}\geqslant d(5)^{1/32}\prod_{n=5}^{\infty}\left(2^n-\frac{25}{64}\right)^{1/2^{n+1}}\left(\frac{n+1}{n}\right)^{1/2^{n+1}}\approx 2.3606$$

[参见罗·詹·道格拉斯（R. J. Douglas），*Disc. Math.* 17 (1977), 143-146；迈·莫拉德（M. Mollard），*European J. Comb.* 9 (1988), 49-52]。然而，这个极限真正的值大概是∞。

46. 利奥·莫泽（Leo Moser）（未发表）已经猜测，它是~ n/e。迄今为止仅有大约$n/\sqrt{2}$的一个上界已被确立，请见上一题答案中的参考文献。

48. 如果这些循环的$d(n,k,v)$以$g(0)\cdots g(k-1)v$开始，则此猜测意味着$d(n,k,v)$
84 $\leqslant d(n,k,g(k))$，因为格雷循环的颠倒还是一个格雷循环。因此这个提示从$d(n)=d(n,1)$得出，而且

$$d(n,k)=\sum_{v}\{d(n,k,v)\mid(v\text{——}g(k-1),v\notin S_k\}\leqslant c_{nk}d(n,k,g(k))=d(n,k+1)$$

最后，$d(n,2^n)=1$，因此$d(n)\leqslant\prod_{k=1}^{2^n-1}c_{nk}=\prod_{k=1}^{n}k^{\binom{n}{k}}=n\prod_{k=1}^{n-1}(k(n-k))^{\binom{n}{k}/2}\leqslant n\prod_{k=1}^{n-1}(n/2)^{\binom{n}{k}}=n(n/2)^{2^n-2}$。[*IEEE Trans.* IT-29 (1983), 894-901]。

49. 在$(2n-1)$立体中取从0…0到1…1的任何哈密顿通路P，例如，萨维吉-温格尔码，并使用$0P$，$1\bar{P}$。（当$n=1$或$n=2$时，通过这一构造可得到所有这样的循环，但当$n>2$时，还有更多的可能性存在。）

50. $\alpha_1(n+1)\ \alpha_1^Rn\alpha_1j_1\alpha_2n\alpha_2^R(n+1)\ \alpha_2\cdots j_{l-1}\alpha_ln\alpha_l^R(n+1)\alpha_ln\alpha_l^Rj_{l-1}\cdots j_1\alpha_1^Rn$。

51. 我们能假设$n>3$且我们有其转换计数为$c_j=2\lfloor(2^{n-1}+j)/n\rfloor$的一个$n$个二进位的格雷循环；我们要来构造其转换计数为$c_j'=2\lfloor(2^{n+1}+j)/(n+2)\rfloor$的一个$(n+2)$个二进位的循环。如果$2^{n+1}\bmod(n+2)\geqslant 2$，对于$l=2\lfloor(2^{n+1})/(n+2)\rfloor+1$，我们可以使用定理D，并且下划线$j$的$b_j$个副本，其中$b_j=4\lfloor(2^{n-1}+j)/n\rfloor-\lfloor(2^{n+1}+j)/(n+2)\rfloor-[j=0]$，并且把一个下划线的0放在最后。这总是容易做的，因为$|b_j-2^{n+2}/n(n+2)|<5$。如果$2^{n+1}\bmod(n+2)\leqslant n$，而且$l=2\lfloor 2^{n+1}/(n+2)\rfloor-1$且$b_j=4\lfloor(2^{n-1}+j)/n\rfloor-\lfloor(2^{n+1}+j+2)/(n+2)\rfloor-[j=0]$，一个类似的构造也有效。事实上，$2^{n+1}\bmod(n+2)$总是$\leqslant n$[参见基·克德拉亚（K. Kedlaya），*Electronic J. Combinatorics* 3 (1996), Comment on # R25 (1997年4月9日)]。这个证明的基本思想是由约·鲍·罗宾森和马·科恩给出的[*IEEE Trans.* C-30 (1981)，17-23]。

52. 在最小j坐标位置中不同代码模式的个数至多为$c_0+\cdots+c_{j-1}$。

53. 注意定理D对于某个j，仅产生具有$c_j=c_{j+1}$的那些循环，因此它不能产生计数（2, 4, 6, 8, 12）。习题50的扩充也给出$c_j=c_{j+1}-2$，但它不能产生（6, 10, 14, 18, 22, 26, 32）。满足习题52的条件的数的集合精确地是由以$\{2,2,4,\cdots,2^{n-1}\}$开始，并且重复地以对$\{c_j+2,c_k-2\}$ $(c_j<c_k)$代替某些对$\{c_j,c_k\}$可以得到的那些集合。

54. 假设这些值是$\{p_1,\cdots,p_n\}$，并令x_{jk}是在$(a_1\cdots a_k)$中p_j出现的次数。对于某个$k<l$，我们必定有$(x_{1k},\cdots,x_{nk})\equiv(x_{1l},\cdots,x_{nl})$(modulo 2)。但是如果诸$p$是素数，随着一个$n$个二进位格雷循环的$\delta$序列而变化，则惟一的解是$k=0$和$l=2^n$。[*AMM* 60 (1953), 418; 83 (1976), 54。]

56. [*Bell System Tech. J.* 37 (1958), 815-826]。112个典范的δ序列产生

类	例 子	t	类	例 子	t	类	例 子	t
A	0102101302012023	2	D	0102013201020132	4	G	0102030201020302	8
B	0102303132101232	2	E	0102032021202302	4	H	0102101301021013	8
C	0102030130321013	2	F	0102013102010232	4	I	0102013121012132	1

这里B是平衡码（图13b），G是标准格雷二进码（图10b），且H是取补码（图13a）。在习题18的对应下，类H也等价于取模的（4, 4）格雷码。带有t个自同构的A类对应于2688个不同的δ序列$\delta_0\delta_1\cdots\delta_{15}$的$32\times 24/t$个。

类似地（参见习题7.2.3-00），5个二进位的格雷循环落入237 675个不同的等价类中。

57. 仅仅对于类型1，有480个顶点被孤立，就是在上面答案中类D、F、G的那
85 些。仅仅对于类型2，图有384个分量，其中的288个是类F和类G的孤立顶点。有大小为9的64个分量，每个含来自E的3个顶点和来自A的6个顶点；大小为30的16个分量，每个有来自H的6个顶点和来自C的24个顶点；大小为84的16个分量，每个有来自D的12个顶点，来自B的24个顶点，来自I的48个顶点。仅仅对于类型3(或类型4)，整个图是连通的。[类似地，如果认为通路$\alpha\beta$同通路$\alpha^R\beta$是相邻的，则4个二进位的所有91 392个格雷通路是连通的。维克斯（Vickers）和希尔弗曼(Silverman)（见*IEEE Trans.* C-29 (1980)，329-331）已经猜测，对于所有$n \geqslant 3$，类型3的变化将足以连接n个二进位的格雷循环的图。]

58. 如果$\beta\beta$的某个非空子串涉及每个坐标偶数次，则该子串不能有$|\beta|$的长度，所以β的某个循环移位有具有同样偶性性质的前缀γ。但那么一来α不定义一个格雷循环，因为我们可以把γ的每个n改回成0。

59. 假定$q>1$且0在α中出现多于$q+1$次，如果在习题58中α不是局部的，$\beta\beta$也一样。因此以(30)中的α开始，但0和1交换，对于$n \geqslant 5$，我们得到非局部的循环，其中坐标0恰巧改变6次。[马克·拉姆拉斯（Mark Ramras），*Discrete Math.* 85 (1990)，329-331。]另一方面，一个4个二进位的格雷循环不可能是非局部的，因为它总有长度为2的运程；如果$\delta_k = \delta_{k+2}$，则元素$\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}$形成一个2子立体。

60. 对$q=1$使用习题58的构造。

61. 其想法是，通过形成连接

$$W = (u_{i_0}v_{j_0}, u_{i_1}v_{j_1}, u_{i_2}v_{j_2}, \cdots),\ i_k = \bar{a}_0 + \cdots + \bar{a}_{k-1},\ j_k = a_0 + \cdots + a_{k-1}$$

来把一个m个二进位的循环$U = (u_0, u_1, u_2, \cdots)$同一个$n$个二进位的循环$V = (v_0, v_1, v_2, \cdots)$穿插，其中$a_0a_1a_2\cdots$是控制位$\alpha\alpha\alpha\cdots$的一个周期的串；当$a_k = 0$时我们前进到$U$的下一个元素，否则就到$V$的下一个元素。

如果α是长度$2^m < 2^n$的任何串，并包含为0的s个二进位和包含为1的$t = 2^m - s$位，如果s和t为奇数，则W将是一个$(m+n)$个二进位的格雷循环。因为我们有$i_{k+l} \equiv i_k (\text{modulo } 2^m)$和$j_{k+l} \equiv j_k (\text{modulo } 2^n)$仅当$l$是$2^m$的一个倍数时，这是$i_k + j_k = k$所致。假设$l = 2^m c$；则$j_{k+l} = j_k + t_c$，所以$c$是$2^n$的一个倍数。

(a) 令$\alpha = 0111$；则长度为8的运程出现在左2位而长度$\geqslant \lfloor 4/3\ r(n) \rfloor$的运程出现在右边$n$个二进位中。

(b) 设s为$< 2^m\ r(m)/(r(m)+r(n))$的最大奇数。也置$t = 2^m - s$和$a_k = \lfloor (k+1)\ t/2^m \rfloor - \lfloor kt/2^m \rfloor$，使得$i_k = \lceil ks/2^m \rceil$和$j_k = \lfloor kt/2^m \rfloor$。如果一个长度为$l$的运程出现在左边$m$个二进位中，我们有$i_{k+l+1} \geqslant i_k + r(m) + 1$，因此$l + 1 > 2^m r(m)/s \geqslant r(m) + r(n)$。而且如果它出现在右边$n$个二进位中，我们有$j_{k+l+1} \geqslant j_k + r(n) + 1$。因此

$$l+1 > 2^m r(n)/t > 2^m r(n)/(2^m r(n)/(r(m)+r(n)) + 2)$$

$$= r(m) + r(n) - \frac{2(r(m)+r(n))^2}{2^m r(n) + 2(r(m)+r(n))} > r(m) + r(n) - 1$$

因为$r(m) \leqslant r(n)$。

在限制较少的情况下这个构造也经常有效。参见介绍格雷码运程的研究的文章 [路 · 戈德丁、乔 · 默 · 劳伦斯和埃 · 尼梅特 (E. Nemeth), *Utilitas Math.* 34 (1988), 179-192]。

63. 对于$0 \leqslant k < 2^{10}$置$a_k \leftarrow k \bmod 4$，但当$k \bmod 16 = 15$或$k \bmod 64 = 42$或$k \bmod 256 = 133$时$a_k = 4$除外。也置$(j_0, j_1, j_2, j_3, j_4) \leftarrow (0, 2, 4, 6, 8)$。然后对于$k = 0, 1, \cdots, 1023$，置$\delta_k \leftarrow j_{a_k}$，以及$j_{a_k} \leftarrow 1 + 4a_k - j_{a_k}$。(这个构造推广了习题61的方法。) 86

64. (a) 每个元素u_k同$\{v_k, v_k + 2^m, \cdots, v_{k+2^m(2^{n-1}-1)}\}$以及$\{v_{k+1}, v_{k+1+2^m}, \cdots, v_{k+1+2^m(2^{n-1}-1)}\}$出现在一起。因此排列$\sigma_0 \cdots \sigma_{2^m-1}$必定是一个$2^{n-1}$循环，包含偶校验的$n$个二进位的顶点，乘以其他顶点的任何排列。这个条件也是充分的。

(b) 设τ_j是使$v \mapsto v \oplus 2^j$的排列，并设$\pi_j(u, w)$是排列$(uw)\tau_j$。如果$u \oplus w = 2^i + 2^j$，则$\pi_j(u, w)$使$u \mapsto u \oplus 2^i$和$w \mapsto w \oplus 2^i$，同时对于所有其他顶点v，$v \mapsto v \oplus 2^j$，所以它把每个顶点变成为一个邻居。

如果S是任何集合$\subseteq \{0, \cdots, n-1\}$，令$\sigma(S)$是对于所有$j \in \{0, \cdots, n-1\} \backslash S$，所有排列$\tau_j$的流，它在$j$的递增顺序下，并重复两次；例如，如果$n = 5$，我们有$\sigma(\{1, 2\}) = \tau_0\tau_3\tau_4\tau_0\tau_3\tau_4$，则格雷流

$$\Sigma(i, j, u) = \sigma(\{i, j\})\pi_j(u, u \oplus 2^i \oplus 2^j)\sigma(\{i, j\})\tau_j\sigma(\{j\})$$

由其乘积是转置$(u\ u \oplus 2^i \oplus 2^j)$的$6n - 8$个排列组成。而且，当把这个流应用于任何$n$个二进位的顶点$v$时，它的所有运程都有$n - 2$或以上的长度。

我们可以假定$n \geqslant 5$。令$\delta_0 \cdots \delta_{2^n-1}$为对于其所有运程的长度为3或以上的一个$n$个二进位的格雷循环$(v_0, v_1, \cdots, v_{2^n-1})$的$\delta$序列，则在

$$\Sigma = \prod_{k=1}^{2^{n-1}-1} (\Sigma(\delta_{2k-1}, \delta_{2k}, v_{2k-1})\Sigma(\delta_{2k}, \delta_{2k+1}, v_{2k}))$$

中所有排列的乘积是$(v_1v_3)(v_2v_4)\cdots(v_{2^n-3}\ v_{2^n-1})\ (v_{2^n-2}\ v_0) = (v_{2^n-1}\cdots v_1)\ (v_{2^n-2}\cdots v_0)$，所以它满足(a)的循环条件。

而且，所有幂$(\sigma(\emptyset)\Sigma)^t$在应用到任何顶点$v$时产生长度$\geqslant n - 2$的运程。通过任意地重复$\Sigma$中个别的因式$\sigma(\{i, j\})$或$\sigma(\{j\})$，我们可以调整$\sigma(\emptyset)\Sigma$的长度，对于任何整数$a, b \geqslant 0$得到$2n + (2^{n-1} - 1)(12n - 16) + 2(n - 2)a + 2(n - 1)b$；因此由习题5.2.1-21，假定$2^m \geqslant 2n + (2^{n-1} - 1)(12n - 16) + 2(n^2 - 5n + 6)$，我们可以把它的长度精确地增加到$2^m$。

(c) 对于$n \geqslant 5$可以证明界$r(n) \geqslant n - 4 \lg n + 8$如下。首先我们由习题60 ~ 63的方法发现，对于$5 \leqslant n < 33$，它成立。然后，对于某个$m \geqslant 20$，对于每个整数$N \geqslant 33$可以写成$N = m + n$或$N = m + n + 1$，其中

$$n = m - \lfloor 4 \lg m \rfloor + 10$$

如果$m \geqslant 20$，2^m对于使部分(b)中的构造正确而言是充分大了，因此

$$\begin{aligned} r(N) &\geqslant r(m+n) \geqslant 2\min(r(m), n-2) \geqslant 2(m - \lfloor 4\lg m \rfloor + 8) \\ &= m + n + 1 - \lfloor 4\lg(m+n) - 1 + \varepsilon \rfloor + 8 \\ &\geqslant N - 4\lg N + 8 \end{aligned}$$

其中$\varepsilon = 4\lg(2m/(m+n)) < 1$。[*Electronic Journal of Combinatorics* **10** (2003), #R27, 1-10。]事实上(b)的递归使用给出$r(1024) \geqslant 1000$。

65. 一个计算机查找发现，8个实质上不同的模式（及它们的颠倒）是可能的，它们之一有δ序列01020314203024041234214103234103，而且它接近于其他模式中的两个。

66.（由马克·库克（Mark Cooke）给出的解。）一个适合的δ序列是
012345607012132435657607102135346267015374123625670173142620657013421
465605731024645375710204353761407363046427370356402713275054121027564
87 150240365425013602541615604312576032572043157624321760452041751635476
703564757062543724213262416152341751436714316431 4。[对于$n > 8$的解仍然未知。]

67. 令$v_{2k+1} = \tilde{v}_{2k}$且$v_{2k} = 0u_k$，其中$(u_0, u_1, \cdots u_{2^n-1})$是任何$(n-1)$个二进位的格雷循环。[见罗宾森和科恩，*IEEE Trans.* C-30 (1981), 17-23。]

68. 是的。最简单的方法大概是取$(n-1)$个三进位的取模的格雷三进制码，并对每个串加0…0, 1…1, 2…2(modulo 3)。例如，当$n = 3$时，这个码为000, 111, 222, 001, 112, 220, 002, 110, 221, 012, 120, 201, …, 020, 101, 212。

69. (a) 对于$j = 1, 2\cdots$，当对二进位$b_{j-1}, \cdots, b_0$同时取补时，我们只需验证在h中的变化；这些变化分别为$(1110)_2$，$(1101)_2$，$(0111)_2$，$(1011)_2$，$(10011)_2$，$(100011)_2$，…。为证明每个n元组出现，注意当$0 \leqslant k < 2^n$和$n > 3$时，$0 \leqslant h(k) < 2^n$；而且$h^{[-1]} = ((a_{n-1}\cdots a_0)_2) = (b_{n-1}\cdots b_0)_2$，其中$b_0 = a_0 \oplus a_1 \oplus a_2 \oplus \cdots$, $b_1 = a_0$, $b_2 = a_2 \oplus a_3 \oplus a_4 \oplus \cdots$, $b_3 = a_0 \oplus a_1 \oplus a_3 \oplus \cdots$，且对于$j \geqslant 4$，$b_j = a_j \oplus a_{j+1} \oplus \cdots$。

(b) 令$h(k) = (\cdots a_2 a_1 a_0)_2$，其中$a_j = b_j \oplus b_{j+1} \oplus b_0[j \leqslant t] \oplus b_{t-1}[t-1 \leqslant j \leqslant t]$。

70. 如同在(32)和(33)中一样，我们通过假定权1的串顺序出现，可以删去$n!$的一个因式。然后对于$n = 5$，有14个解且以00000开始，以及21个解以00001开始。当$n = 6$时，每个类型（通过颠倒和取补而相关联）有46 935个。当$n = 7$时，这个数很大，但同7个二进位格雷码的总数相比仍然是非常小的。

71. 假设对于$0 \leqslant j < n-1$，$\alpha_{n(j+1)}$和α_{nj}在坐标t_j处不同。则由(44)和(38)，$t_j = j_{\pi n}$。现在等式(34)告诉我们$t_0 = n-1$；而且如果$0 < j < n-1$，我们通过(40)有$t_j = ((j-1)\pi_{n-1})\pi_{n-1}$。因此对于$0 \leqslant j < n-1$，$t_j = j\sigma_n\, \pi_{n^2-1}$，而且$(n-1)\pi_n$的值是剩下的无论什么样的值。（排列的记号众所周知易让人混淆，因此仔细地校验一些小的情况总是明智的。）

72. δ序列是0102132430201234012313041021323。

73. 令$Q_{nj} = P_{nj}^R$并通过S_n和T_n来记序列(41)、(42)，因此如果我们省略逗号的话，则$S_n = P_{n0}Q_{n1}P_{n2}\cdots$且$T_n = Q_{n0}P_{n1}Q_{n2}\cdots$；而且我们有

$$S_{n+1} = 0P_{n0}\,0Q_{n1}\,1Q^{\pi}_{n0}\,1P^{\pi}_{n1}\,0P_{n2}\,0Q_{n3}\,1Q^{\pi}_{n2}\,1P^{\pi}_{n3}\,0P_{n4}\cdots$$
$$T_{n+1} = 0Q_{n0}\,1P^{\pi}_{n0}\,0P_{n1}\,0Q_{n2}\,1Q^{\pi}_{n1}\,1P^{\pi}_{n2}\,0P_{n3}\,0Q_{n4}\,1Q^{\pi}_{n3}\cdots$$

其中$\pi = \pi_n$，并揭示S_n和T_n的δ序列Δ_n和E_n之间相当简单的联合递归。就是说，如果我们写

$$\Delta_n = \phi_1\,a_1\,\phi_2\,a_2\cdots\phi_{n-1}\,a_{n-1}\,\phi_n,\quad E_n = \psi_1\,b_1\,\psi_2\,b_2\cdots\psi_{n-1}\,b_{n-1}\,\psi_n$$

其中每个ϕ_j和ψ_j是长度为$2\binom{n-1}{j-1}-1$的串，则下一个序列为

$$\Delta_{n+1} = \phi_1\,a_1\,\phi_2\,n\,\psi_1\pi\,b_1\pi\,\psi_2\pi\,n\,\phi_3\,a_3\,\phi_4\,n\,\psi_3\pi\,b_3\pi\,\psi_4\pi\,n\cdots$$
$$E_{n+1} = \psi_1\,n\,\phi_1\pi\,n\,\psi_2\,b_2\,\psi_3\,n\,\phi_2\pi\,a_2\pi\,\phi_3\pi\,n\,\psi_4\,b_4\,\psi_5\,n\,\phi_4\pi\,a_4\pi\,\phi_5\pi\,n\cdots$$

例如，如果我们下划线诸a和诸b来把它们同诸ϕ和诸ψ相区别，我们有$\Delta_3 = 0\,\underline{1}\,0\,2\,1\,\underline{0}\,1$和$E_3 = 0\,\underline{2}\,1\,2\,0\,\underline{2}\,1$，而且

$$\Delta_4 = 0\,1\,0\,2\,1\,3\,0\pi\,2\pi\,1\pi\,2\pi\,0\pi\,3\,1\,3\,1\pi = 0\,\underline{1}\,0\,2\,1\,3\,2\,\underline{1}\,0\,1\,2\,3\,1\,\underline{3}\,0$$
$$E_4 = 0\,3\,0\pi\,3\,1\,2\,0\,2\,1\,3\,0\pi\,2\pi\,1\pi\,0\pi\,1\pi = 0\,\underline{3}\,2\,3\,1\,2\,0\,\underline{2}\,1\,3\,2\,1\,0\,\underline{2}\,0$$

这里$a_3\phi_4$和$b_3\psi_4$为空。对于下一步，元素已被下划线。

因此我们可以计算在存储器中的δ序列如下。此处对于$1 \leqslant j < n$，$p[j] = j\pi_n$；而且对于$0 \leqslant k < 2^n - 1$，$s_k = \delta_k$，$t_k = \varepsilon_k$，且$u_k = [\delta_k$和ε_k为下划线的]。 88

R1. [初始化。] 置$n \leftarrow 1$，$p[0] \leftarrow 0$，$s_0 \leftarrow t_0 \leftarrow u_0 \leftarrow 0$。

R2. [推进n。] 执行以下的算法S，它对于n的下一个值计算数组s'，t'和u'，然后置$n \leftarrow n+1$。

R3. [就绪了吗？] 如果n充分大，则所希的δ序列Δ_n是在数组s'中；结束。否则继续进行。

R4. [计算π_n。] 置$p'[0] = n-1$，而且对于$1 \leqslant j < n$，$p'[j] = p[p[j-1]]$。

R5. [准备来推进。] 对于$0 \leqslant j < n$，置$p[j] \leftarrow p'[j]$；对于$0 \leqslant k < 2^n - 1$，置$s_k \leftarrow s'_k$，$t_k \leftarrow t'_k$，以及$u_k \leftarrow u'_k$。返回R2。 ∎

在下列步骤中，“当$u_j = 0$时传送Stuff(l, j)”是对于“如果$u_j = 0$，则重复地进行Stuff(l, j)，$l \leftarrow l+1$，$j \leftarrow j+1$，直到$u_j \neq 0$为止”的省略。

S1. [准备计算Δ_{n+1}。] 置$j \leftarrow k \leftarrow l \leftarrow 0$和$u_{2^n-1} \leftarrow -1$。

S2. [推进j。] 当$u_j = 0$时传送$s'_l \leftarrow s_j$和$u'_l \leftarrow 0$，然后如果$u_j < 0$则转到S5。

S3. [推进j和k。] 置$s'_l \leftarrow s_j$，$u'_l \leftarrow 1$，$l \leftarrow l+1$，$j \leftarrow j+1$。然后当$u_j = 0$时传送$s'_l \leftarrow s_j$和$u'_l \leftarrow 0$。然后置$s'_l \leftarrow n$，$u'_l \leftarrow 0$，$l \leftarrow l+1$。然后当$u_k = 0$时传送$s'_l \leftarrow p[t_k]$和$u'_l \leftarrow 0$。然后置$s'_l \leftarrow p[t_k]$，$u'_l \leftarrow 1$，$l \leftarrow l+1$，$k \leftarrow k+1$。当$u_k = 0$时再次传送$s'_l \leftarrow p[t_k]$和$u'_l \leftarrow 0$。

S4. [对Δ_{n+1}完成了吗？] 如果$u_k < 0$转到S6。否则置$s'_l \leftarrow n$，$u'_l \leftarrow 0$，$l \leftarrow l+1$，$j \leftarrow j+1$，$k \leftarrow k+1$，并返回S2。

S5. [完成Δ_{n+1}。] 置$s_l'\leftarrow n$，$u_l'\leftarrow 1$，$l\leftarrow l+1$。然后当$u_k=0$时传送$s_l'\leftarrow p[t[k]]$和$u_l'\leftarrow 0$。

S6. [准备计算E_{n+1}。] 置$j\leftarrow k\leftarrow l\leftarrow 0$。当$u_k=0$时传送$t_l'\leftarrow t_k$，然后置$t_l'\leftarrow n$，$l\leftarrow l+1$。

S7. [推进j。] 当$u_j=0$时传送$t_l'\leftarrow p[s_j]$。如果$u_j<0$则结束；否则置$t_l'\leftarrow n$，$l\leftarrow l+1$，$j\leftarrow j+1$，$k\leftarrow k+1$。

S8. [推进k。] 当$u_k=0$时传送$t_l'\leftarrow t_k$。然后，如果$u_k<0$则转到S10。

S9. [推进k和j。] 置$t_l'\leftarrow t_k$，$l\leftarrow l+1$，$k\leftarrow k+1$。然后，当$u_k=0$时传送$t_l'\leftarrow t_k$。然后置$t_l'\leftarrow n$，$l\leftarrow l+1$。当$u_j=0$时则传送$t_l'\leftarrow p[s_j]$。然后置$t_l'\leftarrow p[s_j]$，$l\leftarrow l+1$，$j\leftarrow j+1$，返回S7。

S10. [完成E_{n+1}。] 置$t_l'\leftarrow n$，$l\leftarrow l+1$。当$u_j=0$时则传送$t_l'\leftarrow p[s_j]$。 ∎

为生成对于相当大的n的单调萨维吉-温格尔码，人们可以先生成Δ_{10}和E_{10}，比如说，或者甚至生成Δ_{20}和E_{20}。使用这些表，则一个适当的递归过程将能够以平均每步都很小的计算开销，而达到n的较高的值。

74. 如果单调的通路是$v_0,\cdots,v_{2^n-1}$，而且如果v_k有权j，则我们有

$$2\sum_{t\ge 0}\binom{n}{j-2t}+((j+\nu(v_0))\bmod 2)\le k\le 2\sum_{t\ge 0}\binom{n}{j-2t}+((j+\nu(v_0))\bmod 2)-2$$

因此分别有权j和$j+1$的顶点之间的极大距离是$2(\binom{n-1}{j-1}+\binom{n-1}{j}+\binom{n-1}{j+1})-1$。当$j$近似地为$n/2$时，近似地为$3\cdot 2^n\sqrt{2\pi n}$的极大值出现[这大约仅是在顶点的任何序之下可达到的最小值的3倍，由习题7.10-00它是$\sum_{j=0}^{n-1}\binom{j}{\lfloor j/2\rfloor}$]。

75. 只有5 个实际上不同的解，事实上它们结果都是格雷循环，δ序列为

0 1 2 3 0 1 2 4 2 1 0 3 2 1 0 1 2 1 0 3 2 1 0 4 0 1 2 3 0 1 2 (1)
0 1 2 3 0 1 2 4 2 1 0 3 2 1 0 1 3 0 1 2 3 0 1 4 1 0 3 2 1 0 3 (1)
0 1 2 3 0 1 2 4 2 1 0 3 2 1 0 2 0 3 2 1 0 3 2 4 2 3 0 1 2 3 0 (2)
0 1 2 3 0 1 2 4 2 3 0 1 2 3 0 2 0 1 2 3 0 1 2 4 2 3 0 1 2 3 0 (2)
0 1 2 3 4 1 0 1 2 1 0 3 0 1 4 3 2 1 0 3 0 1 4 1 0 1 2 3 4 1 0 (3)

76. 如果$v_0,\cdots,v_{2^n-1}$是无趋势的，因此$(n+1)$个二进位的循环$0v_0$，$1v_0$，$1v_1$，$0v_1$，$0v_2$，$1v_2$，$\cdots$，$1v_{2^n-1}$，$0v_{2^n-1}$也是。图14g示出一个稍微更有趣的构造，它把习题75的头一个解推广到一个$(n+2)$个二进位的循环

$$00\Gamma''^R,\ 01\Gamma'^R,\ 11\Gamma',\ 10\Gamma'',\ 10\Gamma,\ 11\Gamma''',\ 01\Gamma'''^R,\ 00\Gamma^R$$

其中Γ是n个二进位序列$g(1),\cdots,g(2^{n-1})$，且$\Gamma'=\Gamma\oplus g(1)$，$\Gamma''=\Gamma\oplus g(2^{n-1})$，$\Gamma'''=\Gamma\oplus g(2^{n-1}+1)$。[一个几乎是一个格雷码的$n$（$n\ge 3$）个二进位的无趋势设计，它只有使$\nu(v_k\oplus v_{k+1})=2$的四个步骤，是由郑清水发现的，*Proc. Berkeley Conf. Neyman and Kiefer* 2 (Hayward, Calif.: Inst. of Math. Statistics, 1985), 619-633。]

77. 用哨兵值的一个数组$(s_{n-1},\cdots,s_0)$代替数组$(o_{n-1},\cdots,o_0)$且在H1步中$s_j\leftarrow m_j-1$。在H4步中置$a_j\leftarrow(a_j+1)\bmod m_j$。如果在H5步中$a_j=s_j$，则置$s_j\leftarrow(s_j-1)$

mod m_j，$f_j \leftarrow f_{j+1}$，$f_{j+1} \leftarrow j+1$。

78. 对于(50)，注意B_{j+1}是在坐标j中反射已经出现的次数，因为我们已经绕过作为$m_j \cdots m_0$的倍数的那些步中的坐标j。因此，如果$b_j < m_j$，b_j加1引起a_j也适当地加1或减1。而且，如果对于$0 \leqslant i < j$，$b_i = m_i - 1$，则当增加b_j使所有这些b_i变为0将使$B_0, \cdots, B_j$的每一个都加1，由此使(50)中的$a_0, \cdots, a_{j-1}$的值不变。

对于(51)，注意$B_j = m_j B_{j+1} + b_j \equiv m_j B_{j+1} + a_j + (m_j - 1)B_{j+1} \equiv a_j + B_{j+1}$(modulo 2)；因此，$B_j \equiv a_j + a_{j+1} + \cdots$，因此(51)明显地等价于(50)。

在对于一般进制$(m_{n-1}, \cdots, m_0)$的取模的格雷码中，当k是由(46)给出时，令

$$\overline{g}(k) = \begin{bmatrix} a_{n-1}, \cdots, a_2, a_1, a_0 \\ m_{n-1}, \cdots, m_2, m_1, m_0 \end{bmatrix}$$

则$a_j = (b_j - B_{j+1}) \bmod m_j$，因为如果我们由$(0, \cdots, 0)$开始，则坐标$j$已在modulo m_j之下增加恰好$B_j - B_{j+1}$次。从取模的格雷码的诸a确定诸b的逆函数，在每个m_j是m_{j+1}的一个因子的特殊情况下（例如，如果所有的m_j都相等）是$b_j = (a_j + a_{j+1} + a_{j+2} + \cdots) \bmod m_j$。但一般说来，逆没有简单形式。对于$j = n-1, \cdots, 0$，并以$B_n = 0$开始，它可通过使用递归式$b_j = (a_j + B_{j+1}) \bmod m_j$，$B_j = m_j B_{j+1} + b_j$来加以计算。

（对于进制$m > 2$的反射格雷码，是由伊凡·弗洛里斯（Ivan Flores）在 [*IRE Trans.* EC-5 (1956), 79-82] 中引进的。他在所有m_j都相等的情况下推导了(50)和(51)。取模的并带有一般进制的格雷码由约瑟夫·罗森鲍姆（Joseph Rosenbaum）在 [*AMM* 45 (1938), 694-696] 中含蓄地讨论了，但没有转换公式；当所有m_j有公共值m时的转换公式是由马丁·科恩发表的 [*Info. and Control* 6 (1963), 70-78]）

79. (a) 最后的n元组总有$a_{n-1} = m_{n-1} - 1$，因此如果$m_{n-1} = 2$，它离$(0, \cdots, 0)$只是一步而已。而且这个条件足以来使最后的n元组成为$(1, 0, \cdots, 0)$。（类似地，由算法K得到的最后的子森林输出相邻于初始的，当且仅当最左的树是一个孤立的顶点。） 90

(b) 最后的n元组是$(m_{n-1} - 1, 0, \cdots, 0)$，当且仅当对于$0 \leqslant j < n-1$，$m_{n-1} \cdots m_{j+1} \bmod m_j = 0$，因为$b_j = m_j - 1$且$B_j = m_{n-1} \cdots m_j - 1$。

80. 使用带有进制$m_j = e_j + 1$的反射格雷码跑遍$p_1^{a_1} \cdots p_t^{a_t}$。

81. 第一个循环包含从(x, y)到$(x, (y+1) \bmod m)$的边，当且仅当$(x+y) \bmod m \neq m-1$当且仅当第二个循环包含从(x, y)到$((x+1) \bmod m, y)$的边。

82. 有两个4个二进制位的覆盖4立体所有边的格雷循环$(u_0, \cdots, u_{15})$和$(v_0, \cdots, v_{15})$。（其实，在和它们的补相同的类中，习题56中的类A，B，D，H和I没有边形成格雷循环。）因此，对于16进制的取模格雷码我们可以形成四个所希的循环$(u_0u_0, u_0u_1, \cdots, u_0u_{15}, u_1u_{15}, \cdots, u_{15}u_0)$，$(u_0u_0, u_1u_0, \cdots, u_{15}u_0, u_{15}u_1, \cdots, u_0u_{15})$，$(v_0v_0, \cdots, v_{15}v_0)$，$(v_0v_0, \cdots, v_0v_{15})$。

类似地，当n是16、32、64等时，存在$n/2$个不相交边的n个二进位的格雷循环。[*Abhandlungen Math. Sem. Hamburg*（汉堡数学研究所论文集）20 (1956), 13-16。]

雅·奥伯特（J. Aubert）和伯·施奈德（B. Schneider）[*Discrete Math.* **38** (1982), 7-16]已经证明，对于所有$n \geqslant 4$的偶数值，相同性质成立，但还未发现简单的构造。

83. 马克·库克在2002年12月找到了以下完全不对称的解:

(1) 27374650573202656123165467436105251060520424163143721451014217372506246064173213107351607103156205713172463452102434643207054702414735614673762504735074513062065641507312373142737643256124026430167354675324025246374752176402707360651052151060735754632531053;

(2) 06167134172321751716715404602471647424732025316216735316327360526710141503047313570615453627623241426465272021632075363710750740315767476154565275651045102402310735342465123040654530621371053726205017524534067034373435315026024630456276741527524060216104343;

(3) 37010637515071312362437657351030120423537472074104736216172473246505132565057121565024570473247421427640231034362703262764130574056062034174561315175631470272172520561321260405350626046017364267176417435134012453602417306365450615630274145356764326257450513;

(4) 67065464356721472362104054320545107374051705321454316364305046734560621206416201320742373627204506473140171020514126107452343672132045275235341051542637060136356730710542016315121053506173123642725371656172175425107602154623754526742570373464036473762716571;

（这些δ序列都应从这个立体的同一个顶点开始。）是否有一个对称的方法来做这件工作?

84. 调用初始位置（2, 2），图A-1的8步的解表明这个序列如何向下进行到（0, 0）。例如，在头一个移动中，绳子的前一半绕过和放在右桅的后边，然后通过最大的右循环，中间的行应从右往左读。类似地推广到n个对的循环将花费$3^n - 1$步。

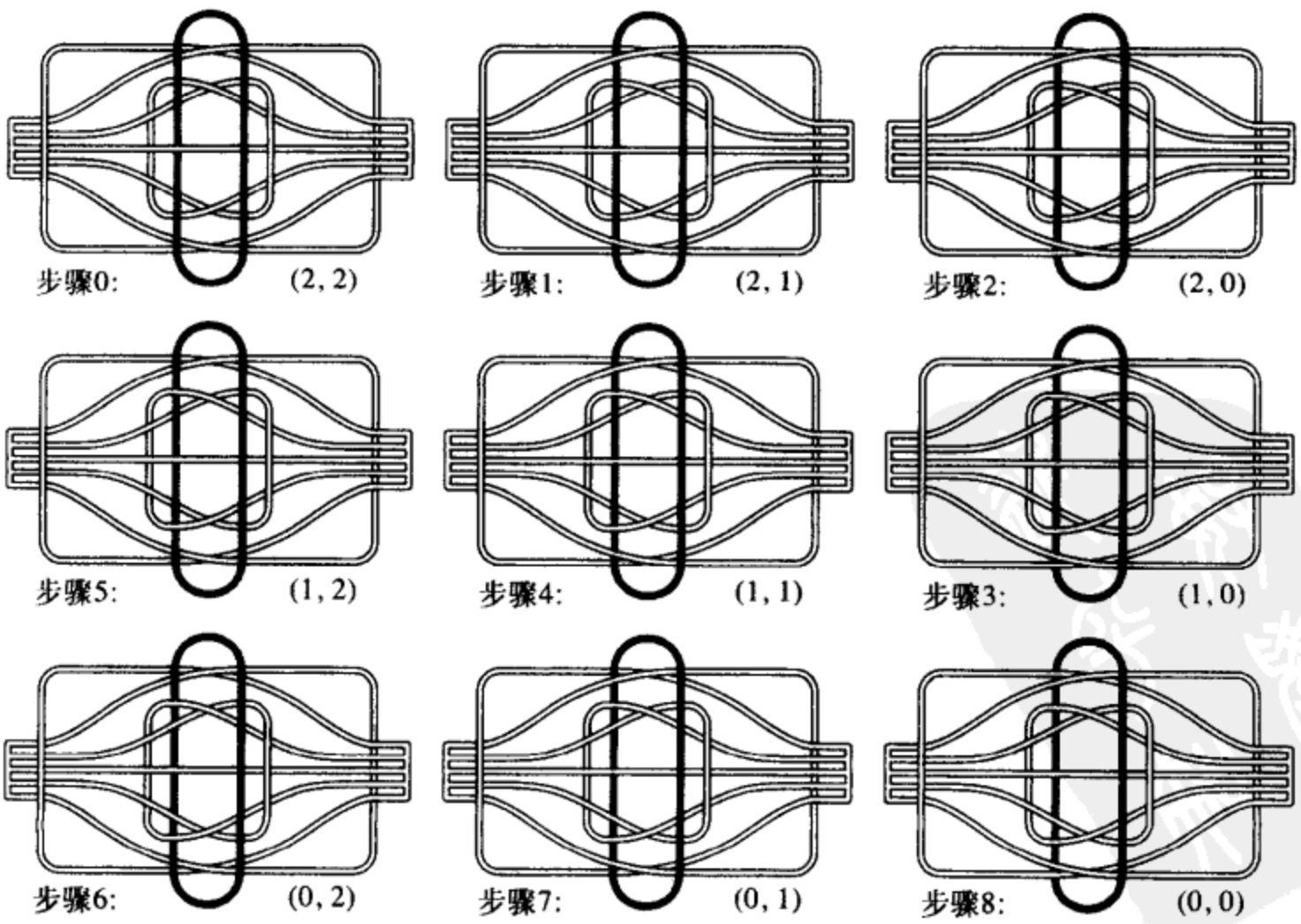


图 A-1

[这个令人感兴趣的难题的起源还不确定。杰里·斯洛克姆（Jerry Slocum）和杰克·博特曼（Jack Botermans）的书*The Book of Ingenious & Diabolical Puzzles*（关于巧妙的和恶魔的难题的书）（1994）示出，大概于1850年左右在中国制造的用牛角雕刻出来的2个循环的版本[101页]和大约于1988年在马来西亚作成的现代的6个循环的版本[93页]。斯洛克姆自己拥有大约于1884年在英国用竹子作成的4循环的版本。他在如下几处发现了它：亨利·诺弗拉（Henry Novra）的书*Catalogue of Conjuring Tricks and Puzzles*（魔术技巧和难题目录）（1858或1859）中，威·亨·克里梅尔（W. H. Cremer）的*Games, Amusements, Pastimes and Magic*（游戏、娱乐、消遣和魔术）（1867）中，以及在哈姆利（Hamley）的1895年的目录*Marvellous Canoe Puzzle*（奇妙的独木舟难题）的标题之下。也可见美国专利（U. S. Patents）2091191 (1937)，*D*172310(1954)，3758114(1973)，*D*406866(1999)。迪克曼在1972年8月2日给马丁·加德纳的信中说明了它同反射的格雷三进制码的联系。]

85. 由(50)，在对于进制（t, t'）的反射格雷码中如果 $\hat{g}\left(\begin{bmatrix} b,b' \\ t,t' \end{bmatrix}\right) = \begin{bmatrix} a,a' \\ t,t' \end{bmatrix}$，则 $\Gamma \boxtimes \Gamma'$的元素$\begin{bmatrix} b,b' \\ t,t' \end{bmatrix}$是$\alpha_a \alpha'_{a'}$。我们现在可以证明，如果在对于进制（$t, t', t''$）的反射格雷码中 $\hat{g}\left(\begin{bmatrix} b,b',b'' \\ t,t',t'' \end{bmatrix}\right) = \begin{bmatrix} a,a',a'' \\ t,t',t'' \end{bmatrix}$，则 $(\Gamma \boxtimes \Gamma') \boxtimes \Gamma''$和$\Gamma \boxtimes (\Gamma' \boxtimes \Gamma'')$ 的元素$\begin{bmatrix} b,b',b'' \\ t,t',t'' \end{bmatrix}$都是$\alpha_a \alpha'_{a'} \alpha''_{a''}$。参见习题4.1-10，还注意混合进制的法则

$$m_1 \cdots m_n - 1 - \begin{bmatrix} x_1, \cdots, x_n \\ m_1, \cdots, m_n \end{bmatrix} = \begin{bmatrix} m_1 - 1 - x_1, \cdots, m_n - 1 - x_n \\ m_1, \cdots, m_n \end{bmatrix}$$

一般地说，对于进制（$m_1, \cdots, m_n$）的反射格雷码是 $(0, \cdots, m_1 - 1) \boxtimes \cdots \boxtimes (0, \cdots, m_n - 1)$。[*Information Processing Letters* 22 (1986), 201-205。]

86. 令Γ_{mn}是反射的m进制格雷码，它可由$\Gamma_{m0} = \varepsilon$和

$$\Gamma_{m(n+1)} = (0, 1, \cdots, m-1) \boxtimes \Gamma_{mn}, n \geqslant 0$$

来定义。当m为偶数时，这条通路从（$0, 0, \cdots, 0$）通到（$m-1, 0, \cdots 0$）。考虑由$\Pi_{m0} = \Lambda$和

$$\Pi_{m(n+1)} = \begin{cases} (0,1,\cdots,m-1) \boxtimes \Pi_{mn}, m\Gamma^R_{(m+1)n}, & \text{如果}m\text{为奇} \\ (0,1,\cdots,m) \boxtimes \Pi_{mn}, m\Gamma^R_{mn}, & \text{如果}m\text{为偶} \end{cases}$$

91 ~ 92

定义的格雷通路Π_{mn}。这个通路由（$0, \cdots, 0, m$）开始，并以（$m, 0, \cdots, 0$）结束，遍历所有 $(m+1)^n - m^n$个非负整数的n元组，对于它们来说，$\max(a_1, \cdots, a_n) = m$。所希的无穷格雷通路是$\Pi_{0n}$，$\Pi^R_{1n}$，$\Pi_{2n}$，$\Pi^R_{3n}$，$\cdots$。

87. 当n为奇数时，这是不可能的，因为满足$\max(|a_1|, \cdots, |a_n|) = 1$的$n$元组包括奇校验的 $(1/2)(3^n + 1)$ 和偶校验的 $(1/2)(3^n - 3)$。当$n = 2$时，我们可以使用一个回旋$\Sigma_0, \Sigma_1, \Sigma_2, \cdots$，其中$\Sigma_m$在$m > 0$时以反时钟方向从$(m, 1-m)$转向（$m, -m$）。对于$n \geqslant 2$的偶数值，如果$T_m$是从（$m, 1-m, m-1, 1-m, \cdots, m-1, 1-m$）到（$m, -m,$

$m, -m, \cdots, m, -m$）的n元组的一条通路，则我们对于具有相同性质的（$n+2$）元组，可以使用$\Sigma_m \boxtimes (T_0, \cdots, T_{m-1}), (\Sigma_0, \cdots, \Sigma_m)^R \boxtimes T_m$，其中$\boxtimes$是对偶操作

$$\Gamma \boxtimes \Gamma' = (\alpha_0\alpha'_0, \cdots, \alpha_{t-1}\alpha'_0, \alpha_{t-1}\alpha'_1, \cdots, \alpha_0\alpha'_1, \alpha_0\alpha'_2, \cdots, \alpha_{t-1}\alpha'_2, \alpha_{t-1}\alpha'_3, \cdots)$$

[不带有数量限制的无穷n维格雷码首先是由恩·瓦泽索尼（E. Vázsonyi）构造的，*Acta Litterarum ac Scientiarum*, sectio Scientiarum Mathematicarum 9 (Szeged: 1938), 163-173。]

88. 它将再次访问所有的子森林，不过是以颠倒的顺序，并且以（0, …, 0）结束，且返回到在初始化步骤K1之后它拥有的状态。（这个反射原理，事实上是理解算法K如何工作的关键。）

89. (a) 令$M_0 = \varepsilon$，$M_1 = \bullet$，且$M_{n+2} = \bullet M^R_{n+1}, -M^R_n$。这个构造有效，因为$M^R_{n+1}$的最后元素是$M_{n+1}$的头一个元素，就是一个点后边接着是$M^R_n$的头一个元素。

(b) 给定一个串$d_1 \cdots d_l$，其中每一个d_j是 • 或 —，我们可以通过令$k = l - [d_l = \bullet]$来定义它的后继并且进行如下：如果k为奇数且$d_k = \bullet$，则把$d_k d_{k+1}$变成 —；如果k为偶数且$d_k = -$，则把d_k变成 ••；否则k减1并重复，直到或者作一个改变或者达到$k = 0$为止。给定的字的后继是 •— —•••• —• —• 。

90. 仅当码字的个数为偶时，一个循环才能存在，因为在每一步中斜杠的个数改变 ± 1。于是我们必定有$n \bmod 3 = 2$。习题89的格雷通路M_n不适合；它们由$(\bullet -)^{\lfloor n/3 \rfloor} \bullet^{\,n \bmod 3}$开始，并以$(-\bullet)^{\lfloor n/3 \rfloor} \bullet^{[n \bmod 3 = 1]} -^{[n \bmod 3 = 2]}$结束。但当$n = 3k+2$时$M_{3k+1}\bullet$，$M^R_{3k}-$是莫斯码图中的一个哈密顿循环。

91. 等价地，n元组$a_1\bar{a}_2a_3\bar{a}_4\cdots$没有两个紧挨着的1。如果我们填加0并且分别以0和10来表示 • 和 —，则这样的n元组对应于长度为$n+1$的莫斯码序列。在这个对应之下，通过包含有点或折线（但最后的点除外）开始的下标的边缘，我们可以把习题89的通路M_{n+1}转换成像算法K那样的一个过程。

Q1. [初始化。] 对于$1 \leqslant j \leqslant n$，置$a_j \leftarrow \lfloor ((j-1) \bmod 6)/3 \rfloor$和$f_j \leftarrow j$。还置$f_0 \leftarrow 0$，$r_0 \leftarrow 1$，$l_1 \leftarrow 0$，以及对于$1 \leqslant j \leqslant n$，$r_j \leftarrow j + (j \bmod 3)$和$l_{j+(j \bmod 3)} \leftarrow j$，除非如果$j + (j \bmod 3) > n$，则置$r_j \leftarrow 0$和$l_0 \leftarrow j$（“边缘”现在含1, 2, 4, 5, 7, 8, …）。

Q2. [访问。] 访问n元组$(a_1, \cdots, a_n)$。

Q3. [选择p。] 置$q \leftarrow l_0$，$p \leftarrow f_q$，$f_q \leftarrow q$。

Q4. [校验a_p。] 如果$p = 0$，则结束此算法。否则如果$a_p + p$现在为偶数，置$a_p \leftarrow 1 - a_p$并转到Q6。

Q5. [插入$p+1$。] 如果$p < n$，则置$q \leftarrow r_p$，$l_q \leftarrow p+1$，$r_{p+1} \leftarrow q$，$r_p \leftarrow p+1$，$l_{p+1} \leftarrow p$。转到Q7。

93

Q6. [删去$p+1$。] 如果$p < n$，则置$q \leftarrow r_{p+1}$，$r_p \leftarrow q$，$l_q \leftarrow p$。

Q7. [使p成为被动的。] 置$f_p \leftarrow f_{l_p}$和$f_{l_p} \leftarrow l_p$，返回Q2。 ∎

这个算法也可作为一个由李钢、弗朗克·拉斯基和唐·欧·克努特给出的更一般方法的特殊情况导出，该方法通过允许用户对于每个（父，子）对（p，q）

确定或者$a_p \geqslant a_q$或者$a_p \leqslant a_q$，来推广算法K。[见克努特和拉斯基，*Lecture Notes in Computer Science* **2635** (2004)，183-204]。在另一个方向上的推广，它产生不包含某些子串的长度为n的所有串，已经由马·布·斯奎尔（M. B. Squire）所发现[*Electronic J. Combinatorics* **3** (1996)，#R17，1-29]。

顺便指出，下面这一点是有趣的，即映射$k \mapsto g(k)/2$是在没有奇数长度的1的运程的所有二进制n元组和没有两个紧挨着的1的所有二进制n元组之间的一一对应。

92. 是的，因为所有满足$x_1, \cdots, x_{n-1} \leqslant m$和弧$(x_1, \cdots, x_{n-1}) \rightarrow (x_2, \cdots, x_n)$的$(n-1)$元组$(x_1, \cdots, x_{n-1})$的有向图每当$\max(x_1, \cdots, x_n) = m$时是连通和平衡的；见定理2.3.4. 2G。其实，如果我们注意到长度除以n的素串的最后k^n个元素，当从$m-1$减去时，对于所有$m \geqslant k$，是相同的，我们就可从算法F得到这样一个序列。例如，当$n = 4$时，序列Φ_4的头81个数字是$2 - \alpha^R = 00001010011\cdots$，其中$\alpha$是串(62)。[给定任何固定的$m \geqslant 3$，存在有无穷的$m$进制序列，对于所有$n$，其头$m^n$个元素是德·布鲁因循环。参见拉·吉·坎明斯（L. J. Cummings）和道·怀德曼（D. Wiedemann），*Cong. Numerantium* **53** (1986)，155-160。]

93. 由$f(\,)$生成的循环是$\alpha 1$的一个循环排列，其中α有$m^n - 1$的长度而且以1^{n-1}结束。由算法R生成的循环是$\gamma = c_0 \cdots c_{m^{n+1}-1}$的一个循环排列，其中$c_k = (c_0 + b_0 + \cdots + b_{k-1}) \bmod m$且$b_0 \cdots b_{m^{n+1}-1} = \beta = \alpha^m 1^m$。

如果$x_0 \cdots x_n$在γ中出现，比如说对于$0 \leqslant j \leqslant n$，$x_j = c_{k+j}$，则对于$0 \leqslant j \leqslant n$，$y_j = b_{k+j}$，其中$y_j = (x_{j+1} - x_j) \bmod m$。[这是同取模的$m$进制格雷码的联系，参见习题78。]现在如果$y_0 \cdots y_{n-1} = 1^n$，我们有$m^{n+1} - m - n < k \leqslant m^{n+1} - n$；否则有一个使得$-n < k' < m^n - n$的下标$k'$，而且对于$0 < r < m$，$y_0 \cdots y_{n-1}$在$\beta$的位置$k = (k' + r(m^n - 1)) \bmod m^{n+1}$中出现。在两种情况下，$k$的$m$种选择有不同的$x_0$值，因为当$n \geqslant 2$时$\alpha$中所有元素之和是$m - 1 (\text{modulo } m)$。[如果$m \bmod 4 \neq 2$，算法R对于$n = 1$也是正确的，因为在该情况下$m \perp \sum \alpha$。]

94. 0 0 1 0 2 0 3 0 4 1 1 2 1 3 1 4 2 2 3 2 4 3 3 4 4（划底线的数字被有效地插入0 0 1 1 2 2 3 4和34的穿插中。算法D当$n = 1$和$r = m - 2 \geqslant 0$时可一般地使用；但鉴于(54)，这样做毫无意义）。

95. (a) 令$c_0 c_1 c_2 \cdots$有周期r。如果r为奇数，我们有$p = q = r$，所以仅当$p = q = 1$和$a_0 = b_0$的平凡情况时才有$r = pq$。否则由4.5.2-(10)，$r/2 = \operatorname{lcm}(p, q) = pq/\gcd(p, q)$，因此$\gcd(p, q) = 2$。在后一情况下，对于$0 \leqslant j < p$，$0 \leqslant k < q$，$j \equiv k (\text{modulo } 2)$，出现的$2n$元组$c_l c_{l+1} \cdots c_{l+2n-1}$是$a_j b_k \cdots a_{j+n-1} b_{k+n-1}$，而且对于$0 \leqslant j < p$，$0 \leqslant k < q$，$j \equiv k (\text{modulo } 2)$，是$b_k a_j \cdots b_{k+n-1} a_{j+n-1}$。

(b) 输出将是穿插在周期分别为$m^n + r$和$m^n - r$的两个序列$a_0 a_1 \cdots$和$b_0 b_1 \cdots$；对于$0 \leqslant x < r$诸a是$f(\,)$的循环且x^n被改变成x^{n-1}，而诸b是$f'(\,)$的循环且x^n被改变成x^{n-1}。由(58)和部分(a)，周期长度为$m^{2n} - r^2$，而且对于$0 \leqslant x, y < r$，每个$2n$元组都出现但$(xy)^n$除外。

(c) 当$t \geqslant n$，$t' = n$且$0 \leqslant x' = x < r$时，通过转到D3，实际的步骤D6改变(b)的特性。当x^{2n-1}刚被输出而b正要被发出时，这个改变发射额外的x，其中b是在这个循

环中跟随x^n的数字。D6也允许在$t \geqslant n$和$x < x' < r$的情况下以$t' = n$把控制传到D7而后D3；当$(xx')^{n-1}x$刚被输出且下一个将输出b时，这一行为发出一个额外的$x'x$。这r^2个额外的数位提供了(b)的r^2个不见的$2n$元组。

96. (a) 递归式$S_2 = 1, S_{2n+1} = S_{2n} = 2S_n, R_2 = 0, R_{2n+1} = 1 + R_{2n}, R_{2n} = 2R_n, D_2 = 0, D_{2n+1} = D_{2n} = 1 + 2D_n$有解$S_n = 2^{\lfloor \lg n \rfloor - 1}, R_n = n - 2S_n, D_n = S_n - 1$。因此$S_n + R_n + D_n = n - 1$。

(b) 每个顶部一级的输出通常涉及$\lfloor \lg n \rfloor - 1$个D激活和$\nu(n) - 1$个R激活，加上在底部一级的基本激活。但有例外：如果头一个激活完成一个序列1^n，则算法R可以调用两次它的$f(\,)$；而有时算法R全然不需要调用$f(\,)$。如果头一次激活完成一个序列$(x')^n$，则算法D可能调用两次它的$f'(\,)$。但有时算法D既不需要调用$f(\,)$，也不需要调用$f'(\,)$。

算法R完成一个序列x^{n+1}当且仅当它的儿子$f()$刚好完成一个序列0^n。对于$x < r$算法D完成一个序列x^{2n}，当且仅当它刚从D6跳转到D3去而不调用任何儿子。

从这些发现我们可以得出结论，如果$r > 1$，对于每个顶部级至多有$\lfloor \lg n \rfloor + \nu(n) + 1$个激活是可能的；当对于$n = 6$算法D从D6转到D4时，这种情况发生。但当$r = 1$时，我们可以有$2\lfloor \lg n \rfloor + 3$这样多的激活，例如，当对于$n = 25$算法R从R4转到R2时。

97. (a) (0011)，(00011101)，(0000101001111011) 以及(0000011000101101111100111010101001)。因此$j_2 = 2, j_3 = 3, j_4 = 9, j_5 = 15$。

(b) 对于$0 \leqslant k < j_n + n$，我们显然有$f_{n+1}(k) = \sum f_n(k) \bmod 2$。下一个值$f_{n+1}(j_n + n)$依赖于在计算$y = f_n(j_n + n - 1)$之后步骤R4是否跳转到R2去。如果它跳转（即如果$f_{n+1}(j_n + n - 1) \neq 0$），对于$j_n + n \leqslant k < 2^n + j_n + n$，有$f_{n+1}(k) \equiv 1 + \sum(k-1)$。否则对于那些$k$的值，我们有$f_{n+1}(k) \equiv 1 + \sum(k-1)$。特别是当$2^n \leqslant k + \delta_n \leqslant 2^n + n$时，$f_{n+1}(k) = 1$。所述公式对于下标$k$有较简单的值域，当$j_n < k < j_n + n$或$2^n + j_n < k < 2^n + j_n + n$时成立，因为$1 + \sum(k \pm 1) \equiv \sum(k)$。

(c) 穿插的循环有$c_n(2k) = f_n^+(k)$和$c_n(2k+1) = f_n^-(k)$，其中

$$f_n^+(k) = \begin{cases} f_n(k-1), & \text{如果} 0 < k \leqslant j_n + 1 \\ f_n(k-2), & \text{如果} j_n + 1 < k \leqslant 2^n + 2 \end{cases}$$

$$f_n^-(k) = \begin{cases} f_n(k+1), & \text{如果} 0 \leqslant k < j_n \\ f_n(k+2), & \text{如果} j_n \leqslant k < 2^n - 2 \end{cases}$$

$f_n^+(k) = f_n^+(k \bmod (2^n + 2))$，$f_n^-(k) = f_n^-(k \bmod (2^n - 2))$。因此在$c_n$循环中，子序列$1^{2n-1}$在位置$k_n = (2^{n-1} - 2)(2^n + 2) + 2j_n + 2$处开始。这将使$j_{2n}$为奇数。如果$j_n \bmod 4 = 1$，子序列$(0\ 1)^{n-1}0$在位置$l_n = (2^{n-1} + 1)(j_n - 1)$处开始。而如果$j_n \bmod 4 = 3$，则在位置$l_n = (2^{n-1} + 1)(2^n + j_n - 3)$处开始。还有$k_2 = 6$，$l_2 = 2$。

(d) 算法D把4个元素插入c_n循环；因此，当$j_n \bmod 4 < 3$ $(l_n < k_n)$时：

$$f_{2n}(k) = \begin{cases} c_n(k-1), & \text{如果} 0 < k \leqslant l_n + 2 \\ c_n(k-3), & \text{如果} l_n + 2 < k \leqslant k_n + 3 \\ c_n(k-4), & \text{如果} k_n + 3 < k \leqslant 2^{2n} \end{cases}$$

当$j_n \bmod 4 = 3$, $(k_n < l_n)$ 时:

$$f_{2n}(k) = \begin{cases} c_n(k-1), & \text{如果} 0 < k \leqslant k_n + 1 \\ c_n(k-2), & \text{如果} k_n + 1 < k \leqslant l_n + 3 \\ c_n(k-4), & \text{如果} l_n + 3 < k \leqslant 2^{2n} \end{cases}$$

(e) 结果$j_{2n} = k_n + 1 + 2[j_n \bmod 4 < 3]$。其实，居于$1^{2n}$之前的元素由$f_n^+(\,)$的$2^{n-2}-1$个完整周期同$f_n^-(\,)$的$2^{n-2}$个完整周期穿插组成，而且如果$l_n < k_n$，则同一个0以及10也穿插，后边接有$f_n(1)\,f_n(1)\,f_n(2)\,f_n(2)\cdots f_n(j_n-1)\,f_n(j_n-1)$。所有这些元素之和为奇数，除非$l_n < k_n$；因此$\delta_{2n} = 1 - 2[j_n \bmod 4 = 3]$。 95

令$n = 2^t q$，其中q为奇数且$n > 2$。这些递归式意味着，如果$q = 1$，我们有$j_n = 2^{n-1} + b_t$，其中$b_t = 2^t/3 - (-1)^t/3$。而且如果$q > 1$，我们有$j_n = 2n - 1 \pm b_{t+2}$，其中 + 号被选定当且仅当$\lfloor \lg q \rfloor + [\lfloor 4q/2^{\lfloor \lg q \rfloor} \rfloor = 5]$为偶数。

98. 当k处于某一个范围时，如果$f(k) = g(k)$，则存在一个常数C，使得对于在这个范围内的k，$\Sigma f(k) = C + \Sigma g(k)$。我们因此可以轻松导出另外的递归式：如果$n > 1$，我们有

当$j_n \bmod 4 < 3$ $(l_n < k_n)$ 时:

$$\Sigma f_{2n}(k) = \begin{cases} \Sigma c_n(k-1), & \text{如果} 0 < k \leqslant l_n + 2 \\ 1 + \Sigma c_n(k-3), & \text{如果} l_n + 2 < k \leqslant k_n + 3 \\ \Sigma c_n(k-4), & \text{如果} k_n + 3 < k \leqslant 2^{2n} \end{cases}$$

当$j_n \bmod 4 = 3$ $(k_n < l_n)$ 时:

$$\Sigma f_{2n}(k) = \begin{cases} \Sigma c_n(k-1), & \text{如果} 0 < k \leqslant k_n + 1 \\ 1 + \Sigma c_n(k-2), & \text{如果} k_n + 1 < k \leqslant l_n + 3 \\ \Sigma c_n(k-4), & \text{如果} l_n + 3 < k \leqslant 2^{2n} \end{cases}$$

$$\Sigma c_n(k) = \Sigma f_n^+\left(\lceil k/2 \rceil\right) + \Sigma f_n^-\left(\lfloor k/2 \rfloor\right)$$

$$\Sigma f_n^+(k) = \begin{cases} \Sigma f_n(k-1), & \text{如果} 0 < k \leqslant j_n + 1 \\ 1 + \Sigma f_n(k-2), & \text{如果} j_n + 1 < k \leqslant 2^n + 2 \end{cases}$$

$$\Sigma f_n^-(k) = \begin{cases} \Sigma f_n(k+1), & \text{如果} 0 \leqslant k < j_n \\ 1 + \Sigma f_n(k+2), & \text{如果} j_n \leqslant k < 2^n - 2 \end{cases}$$

$$\Sigma f_n^{\pm}(k) = \left\lfloor k/(2^n \pm 2) \right\rfloor + \Sigma f_n^{\pm}(k \bmod (2^n \pm 2)); \quad \Sigma f_n(k) = \Sigma f_n(k \bmod 2^n)$$

$$\Sigma f_{2n+1}(k) = \begin{cases} \Sigma\Sigma f_{2n}(k), & \text{如果} 0 < k \leqslant j_{2n} \text{或} 2^{2n} + j_{2n} < k \leqslant 2^{2n+1} \\ 1 + k + \Sigma\Sigma f_{2n}(k + \delta_{2n}), & \text{如果} j_{2n} < k \leqslant 2^{2n} + j_{2n} \end{cases}$$

当$j_n \bmod 4 < 3$ $(l_n < k_n)$ 时:

$$\Sigma\Sigma f_{2n}(k) = \begin{cases} \Sigma\Sigma c_n(k-1), & \text{如果} 0 < k \leqslant l_n + 2 \\ 1 + k + \Sigma\Sigma c_n(k-3), & \text{如果} l_n + 2 < k \leqslant k_n + 3 \\ \Sigma\Sigma c_n(k-4), & \text{如果} k_n + 3 < k \leqslant 2^{2n} \end{cases}$$

当$j_n \bmod 4 = 3$ $(k_n < l_n)$ 时:

$$\Sigma\Sigma f_{2n}(k) \equiv \begin{cases} \Sigma\Sigma c_n(k-1), & \text{如果} 0 < k \leqslant k_n + 1 \\ 1 + k + \Sigma c_n(k-2), & \text{如果} k_n + 1 < k \leqslant l_n + 3 \\ 1 + \Sigma\Sigma c_n(k-4), & \text{如果} l_n + 3 < k \leqslant 2^{2n} \end{cases}$$

$$\Sigma\Sigma f_{2n}(k) \equiv [j_n \bmod 4 < 3]\left\lfloor k/2^{2n} \right\rfloor + \Sigma\Sigma f_{2n}(k \bmod 2^{2n})$$

那么，啊哈，有了结尾了:

$$\Sigma\Sigma c_n(2k) = \Sigma f_n^+(k), \quad \Sigma\Sigma c_n(2k+1) = \Sigma f_n^-(k)$$

如果$n = 2^t q$，其中q为奇数，通过这递归公式组来计算$f_n(k)$ 的运行时间是$O(t + S(q))$，其中$S(1) = 1, S(2k) = 1 + 2S(k)$ 且$S(2k+1) = 1 + S(k)$。显然$S(k) < 2k$，因此，这个计算对于n个二进位数至多涉及$O(n)$ 个简单运算。事实上，这个算法经常快得多，如果我们对于使$\lfloor \lg k \rfloor = s$的所有$k$取$S(k)$ 的平均，可得到 $(3^{s+1} - 2^{s+1})/2^s$，它比$3k^{\lg(3/2)} < 3k^{0.59}$小。(顺便指出，如果$k = 2^{s+1} - 1 - (2^{s-e_1} + 2^{s-e_2} + \cdots + 2^{s-e_t})$，我们有$S(k) = s + 1 + e_t + 2e_{t-1} + 4e_{t-2} + \cdots + 2^t e_1$。)

99. 在$f_n(\,)$ 的位置k处开始的一个串在$f_n^+(\,)$中的位置$k^+ = k + 1 + [k > j_n]$处且在$f_n^-(\,)$ 中的位置$k^- = k - 1 - [k > j_n]$处开始，除开0^n和1^n在$f_n^+(\,)$ 中两次出现而在$f_n^-(\,)$ 中全然不出现之外。

为了在循环$f_{2n}(\,)$ 中求$\gamma = a_0 b_0 \cdots a_{n-1} b_{n-1}$，令$\alpha = a_0 \cdots a_{n-1}$且$\beta = b_0 \cdots b_{n-1}$。假设在$f(\,)$ 中α在位置j处，而β在位置k处开始，并且假定无论是α还是β都不是0^n和1^n。如果$j^+ \equiv k^+$ (modulo 2)，令$l/2$是方程$j^+ + (2^n + 2)x = k^- + (2^n - 2)y$的解。如果$j > k$，我们可以取$l/2 = k + (2^n - 2)(2^{n-3}(j-k) \bmod (2^{n-1} + 1))$，否则$l/2 = j + (2^n + 2)(2^{n-3}(k-j) \bmod (2^{n-1} - 1))$。否则令 $(l-1)/2 = k^+ + (2^n + 2)x = j^- + (2^n - 2)y$。于是$\gamma$在循环
96 $c_n(\,)$中在位置l处开始；在循环$f_{2n}(\,)$ 中它在位置$l + 1 + [l > k_n] + 2[l > l_n]$处开始。当$\alpha \in \{0^n, 1^n\}$或$\beta \in \{0^n, 1^n\}$时（但并非两者都成立），类似的公式成立。最后，$0^{2n}$、$1^{2n}$、$(01)^n$以及 $(10)^n$分别在位置$0, j_{2n}, l_n + 1 + [k_n < l_n]$以及$l_n + 2 + [k_n < l_n]$处开始。

当n为偶数时，为求在$f_{n+1}(\,)$ 中的$\beta = b_0 b_1 \cdots b_n$，假设在$f_n(\,)$ 中n个二进位串 $(b_0 \oplus b_1) \cdots (b_{n-1} \oplus b_n)$ 在位置j处开始。如果$f_{n+1}(k) = b_0$，则β在位置$k = j - \delta_n[j > j_n] + 2^n[j = j_n][\delta_n = 1]$处开始，否则按照（$j < j_n, j = j_n, j > j_n$）在位置$k + (2^n - \delta_n, \delta_n, 2^n + \delta_n)$ 处开始。

这个递归式的运行时间满足$T(n) = O(n) + 2T(\lfloor n/2 \rfloor)$，所以它是$O(n \log n)$。[习题97 ~ 99是基于约 · 图里阿尼（J. Tuliani）的工作给出的，他还建立了对于m的某些更大值的一些方法；参见*Discrete Math.* **226** (2001)，313-336。]

100. 没有明显的缺点出现，但在能推荐任何序列之前应进行广泛的测试。相反，由算法F含蓄地产生的德 · 布鲁因循环是设想为随机二进位的一个糟糕来源，尽管在定义3.5D的意义下它是n分布的，因为在开始时诸0预支配了该序列。其实，当n为素数时，对于$0 \leqslant t < (2^n - 2)/n$，该序列的$t_n + 1$个二进位为0。

101. (a) 令β为$\lambda\lambda'$的一个真后缀且$\beta \leqslant \lambda\lambda'$。或者$\beta$是$\lambda'$的一个后缀，因此$\lambda < \lambda' \leqslant \beta$，或者$\beta = \alpha\lambda'$且我们有$\lambda < \alpha < \beta$。

现在$\lambda < \beta \leqslant \lambda\lambda'$意味着对于某个$\gamma \leqslant \lambda'$，$\beta = \lambda\gamma$。但$\gamma$是$\beta$的一个后缀且$1 \leqslant |\gamma| = |\beta| - |\lambda| < |\lambda'|$；因此$\gamma$是$\lambda'$的一个真后缀，而且$\lambda' < \gamma$。矛盾。

(b) 长度为l的任何串为素的。通过(a)以任何顺序把相邻的素串结合起来，直到不再可能进行结合为止。[参见马·鲍·舒特曾伯格（M. P. Schützenberger），*Proc. Amer. Math. Soc.* **16** (1965)，21-24。]

(c) 如果$t \neq 0$，令λ为$\lambda_1 \cdots \lambda_t$最小的后缀。则由定义$\lambda$为素串，而且它有$\beta\gamma$的形式，其中$\beta$是某个$\lambda_j$的一个非空后缀。因此$\lambda_t \leqslant \lambda_j \leqslant \beta \leqslant \beta_\gamma = \lambda \leqslant \lambda_t$，所以我们必定有$\lambda = \lambda_r$。删除$\lambda_t$并且重复直到$t = 0$为止。

(d) 真的。因为如果对于某个素数λ且$|\lambda| > |\lambda_1|$，我们有$\alpha = \lambda\beta$，则可以附加β因式来得到α的另一个因式分解。

(e) 3 · 14159265358979323846264338327 95 · 02884197。(在习题106中有一个有效的算法。知道π的更多数位并不改变头两个因式，在博雷尔（参见3.5节）的意义下，作为“正常的”任何数的无穷十进展开可分解成有限长度的一些素数。

102. 我们必定有 $1/1(1-mz) = 1/\prod_{n=1}^{\infty}(1-z^n)^{L_m(n)}$。如同在习题4.6.2-4中一样，这意味着(60)。

103. 当$n = p$是素数时，(59)告诉我们，$L_m(1) + p\,L_m(p) = m^p$，而且我们还有$L_m(1) = m$。[此组合证明提供了同定理1.2.4F的传统代数证明的有趣对照。]

104. 4483个非素串是abaca, agora, ahead, …；1274个素串是…, rusts, rusty, rutty。(由于prime不是素的，也许我们应该把素串叫做普通的(lowly)。)

105. (a) 令α'是α且它有增加的最后字母，并假设$\alpha' = \beta\gamma'$，其中$\alpha = \beta\gamma$且$\beta \neq \varepsilon$，$\gamma \neq \varepsilon$。令θ是α的前缀且$|\theta| = |\gamma|$。由假设，有一个串ω使得$\alpha\omega$是素的；因此$\theta \leqslant \alpha\omega < \gamma\omega$，所以我们必定有$\theta \leqslant \gamma$。结果$\theta < \gamma'$，而且我们有$\alpha' < \gamma'$。

(b) 设$\alpha = \lambda_1\beta = a_1 \cdots a_n$，其中$\lambda_1\beta\omega$是素的。条件$\lambda_1\beta\omega < \beta\omega$意味着对于$1 \leqslant j \leqslant n-r$，$a_j \leqslant a_{j+r}$，其中$r = |\lambda_1|$。但我们不能有$a_j < a_{j+r}$；否则$\alpha$就将以比$\lambda_1$更长的一个素串开始，这同习题101(d)矛盾。

(c) 如果α是λ和λ'两者的n扩充，其中$|\lambda| > |\lambda'|$，我们必定有$\lambda = (\lambda')^q\theta$，其中$\theta$是$\lambda'$的一个非空前缀。但那样，$\theta \leqslant \lambda' < \lambda < \theta$。 97

106. B1. [初始化。] 置$a_1 \leftarrow \cdots \leftarrow a_n \leftarrow m-1$，$a_{n+1} \leftarrow -1$且$j \leftarrow 1$。

B2. [访问。] 以下标j来访问$(a_1, \cdots, a_n)$。

B3. [减1。] 如果$a_j = 0$则结束。否则置$a_j \leftarrow a_j - 1$，并且对于$j < k \leqslant n$，$a_k \leftarrow m-1$。

B4. [准备因式。] (按照习题105(b)，我们现在要找$a_1, \cdots, a_n$的头一个因式λ_1，置$j \leftarrow 1$和$k \leftarrow 2$。

B5. [找新的j。] (现在$a_1 \cdots a_{k-1}$是素的$a_1 \cdots a_j$的$(k-1)$扩充。)如果$a_{k-j} > a_k$，则返回B2。否则，如果$a_{k-j} < a_k$，则置$j \leftarrow k$。然后k加1并重复这一步。 ∎

步骤B4和B5的有效分解算法是由吉·波·杜瓦尔（J. P. Duval）给出的，见*J. Algorithms* 4 (1983), 363-381。关于进一步的信息，请参见卡特尔（Cattell）、拉斯基、萨瓦达（Sawada）、塞尔拉（Serra）以及米尔斯（Miers），见*J. Algorithms* 37 (2000)，267-282。

107. 所访问的n元组的个数是$P_m(n)=\sum_{j-1}^{n}L_m(j)$。由于$L_m(n)=\frac{1}{n}m^n+O(m^{n/2}/n)$，我们有$P_m(n)=Q(m,n)+O(Q(\sqrt{m},n))$，其中

$$Q(m,n)=\sum_{k=1}^{n}\frac{m^k}{k}=\frac{m^n}{n}R(m,n)$$

$$R(m,n)=\sum_{k=0}^{n-1}\frac{m^{-k}}{1-k/n}=\sum_{k=0}^{n/2}\frac{m^{-k}}{1-k/n}+O(nm^{-n/2})$$

$$=\frac{m}{m-1}\sum_{j=0}^{t-1}\frac{1}{n^j}\sum_{l}\left\{{j\atop l}\right\}\frac{l!}{(m-1)^l}+O(n^{-t})$$

因此$P_m(n)\sim m^{n+1}/((m-1)n)$。对运行时间的主要贡献来自于步骤F3和F5中的循环，它们对于长度j的每个素因式花费$n-j$，因此总共为$nP_m(n)-\sum_{j=1}^{n}jL_m(j)=m^{n+1}(1/((m-1)^2n)+O(1/(mn^2)))$。这比输出德·布鲁因循环的$m^n$个数字的时间要少。

108. (a) 如果$\alpha\neq 9\cdots 9$，我们有$\lambda_{k+1}\leqslant\beta 9^{|\alpha|}$。因为后者为素的。

(b) 我们可以假设β不是全0，因为9^j0^{n-j}是$\lambda_{r-1}\lambda_r\lambda_1\lambda_2=89^n0^n1$的子串。令$k$是满足$\beta\leqslant\lambda_k$的极小，则$\lambda_k<\beta\alpha$，所以$\beta$是$\lambda_k$的一个前缀。由于$\beta$是一个预素的，因此它是某个素$\beta'<\beta$的$|\beta|$扩充。由习题106，刚好在$\beta'$之前由算法F所访问的预素串是$(\beta'-1)9^{n-|\beta'|}$，其中$\beta'-1$表示比$\beta'$小1的十进数。因此，如果$\beta'$不是$\lambda_{k-1}$，则这个提示（它也从习题106得出）意味着$\lambda_{k+1}$以至少$n-|\beta'|\geqslant n-|\beta|$个9结束，而且$\alpha$是$\lambda_{k-1}$的一个后缀。另一方面，如果$\beta'=\lambda_{k-1}$，则$\alpha$是$\lambda_{k-2}$的一个后缀，而且$\beta$是$\lambda_{k-1}\lambda_k$的一个前缀。

(c) 如果$\alpha\neq 9\cdots 9$，我们有$\lambda_{k+1}<(\beta\alpha)^{d-1}\beta 9^{|\alpha|}$，因为后者为素的。否则$\lambda_{k-1}$以至少$(d-1)|\beta\alpha|$个9结束，而且$\lambda_{k+1}\leqslant(\beta\alpha)^{d-1}9^{|\beta\alpha|}$，所以$(\alpha\beta)^d$是$\lambda_{k-1}\lambda_k\lambda_{k+1}$的一个子串。

(d) 在素135899 135914, 787899 787979, 129999 13 131314, 09 090911, 089999 09 090911, 118999 119 119122之内。

(e) 是的：在所有情况下，如果$0\leqslant a_n<9$（而且，如果我们假定像9^j0^{n-j}这样的串出现在开始处），则$a_1\cdots a_n$的位置居前于子串$a_1\cdots a_{n-1}(a_n+1)$的位置。其次，9^j0^{n-j-1}仅出现于对于$1\leqslant a<9$的$9^{j-1}0^{n-j}a$已经出现之后，所以我们不能把0放在9^j0^{n-j-1}之后。

98

109. 假设我们要来放置子矩阵

$$\begin{pmatrix}(w_{n-1}\cdots w_1w_0)_2 & (x_{n-1}\cdots x_1x_0)_2\\ (y_{n-1}\cdots y_1y_0)_2 & (z_{n-1}\cdots z_1z_0)_2\end{pmatrix}$$

二进制情况$n=1$是给定的例子，而如果$n>1$，由归纳法可以假设我们只须确定前导二进位数字a_{2n-1}、a_{2n-2}、b_{2n-1}和b_{2n-2}。情况$n=3$是典型的，我们必须求解

$$\begin{aligned}
b_5 &= w_2, & b_4 &= x_2, & a_5 \oplus b_5 &= y_2, & a_4 \oplus b_4 &= z_2, & &\text{如果} a_0 = 0,\ b_0 = 0;\\
b_4 &= w_2, & b_5' &= x_2, & a_4 \oplus b_4 &= y_2, & a_5 \oplus b_5' &= z_2, & &\text{如果} a_0 = 0,\ b_0 = 1;\\
a_5 \oplus b_5 &= w_2, & a_4 \oplus b_4 &= x_2, & b_5 &= y_2, & b_4 &= z_2, & &\text{如果} a_0 = 1,\ b_0 = 0;\\
a_4 \oplus b_4 &= w_2, & a_5 \oplus b_5' &= x_2, & b_4 &= y_2, & b_5' &= z_2, & &\text{如果} a_0 = 1,\ b_0 = 1;
\end{aligned}$$

这里$b_5' = b_5 \oplus b_4 b_3 b_2 b_1$，考虑当$j$变成$j+1$时的进位。

110. 令$a_0 a_1 \cdots a_{m^2-1}$是一个m进制的德·布鲁因循环，例如(54)的头m^2个元素。如果m为奇数，则当i为偶数时，令$d_{ij} = a_j$；而当i为奇数时，令$d_{ij} = a_{(j+(i-1)/2) \bmod m^2}$。[发现这个构造的许多人中的头一位似乎是约翰·克·科克（John C. Cock），他在（*Discrete Math.* 70 (1988), 209-210）中也构造了其他形状和大小的德·布鲁因圆环体。]

如果$m = m'm''$，其中$m' \perp m''$。借助于求解对于m'和m''的问题的矩阵，我们使用中国剩余定理来定义

$$d_{ij} = d_{ij}' \text{ (modulo } m') \text{ 和} d_{ij} = d_{ij}'' \text{ (modulo } m'')$$

因此上道习题导致对于任何m的一个解。

安托·伊凡伊（Autal Iványi）和佐尔坦·托特（Zoltán Tóth）发现了偶数值m的另一个有趣的解。[2*nd Conf. Automata, Lauguages and Programming Systems* (1988), 165-172；也参见哈尔贝特（Hurlbert）和伊萨克（Isaak），*Contemp. Math.* 178 (1994), 153-160。]无穷序列

00110213312032230415243553425140544506172637465775 64⋯0766708⋯

的头m^2个元素a_j定义一个具有如下性质的德·布鲁因循环，即ab和ba的出现之间的距离总是偶数。由此如果$i+j$为偶数，我们可以令$d_{ij} = a_j$，如果$i+j$为奇数，可以令$d_{ij} = a_i$。例如，当$m = 4$时，我们有

$$\begin{pmatrix}
0010021220302232\\
0001020320212223\\
0111031321312333\\
1011121330313233\\
0010021220302232\\
0203000122232021\\
0111031321312333\\
1213101132333031\\
0010021220302232\\
2021222300010203\\
0111031321312333\\
3031323310111213\\
0010021220302232\\
2223202102030001\\
0111031321312333\\
3233303112131011
\end{pmatrix}$$

习题（109）

$$\begin{pmatrix}
0010001030203020\\
0001020301000203\\
0111011131213121\\
1011121311101213\\
0010001030203020\\
2021222321202223\\
0111011131213121\\
3031323331303233\\
0313031333233323\\
1011121311101213\\
0212021232223222\\
0001020301000203\\
0313031333233323\\
2021222321202223\\
0212021232223222\\
3031323331303233
\end{pmatrix}$$

（托特）

111. (a) 对于$1 \leqslant j \leqslant 9$，$a_9 \neq 0$，令$d_j = j$和$0 \leqslant a_j < 3$。通过规则$s_1 = 0$，$t_1 = d_1$；对于$1 \leqslant j < 9$，$t_{j+1} = d_{j+1} + 10t_j[a_j = 0]$形成序列$s_j$, t_j。对于$a_j = (0, 1, 2)$和$1 \leqslant j \leqslant 9$，$s_{j+1} = s_j + (0, t_j, -t_j)$。则$s_{10}$是一个可能的结果；我们只须记住出现的略小的一些值即可。通过在$s_k = 0$时不允许$a_k = 2$可节省一半以上的工作，然后使用$|s_{10}|$来代替s_{10}。由于少于$3^8 = 6561$种可能性需要加以试验，通过算法M的三进制版本的硬算很有效；为推导出小于211（不含211）的所有整数是可表示的，需要少于24 000次存储访问和1600个乘法。

另一种方法（使用格雷码来把数字以2^8种可能的方式分成一些块之后来改变符号）把乘法次数减少到255，但其代价是再使用额外约500次存储访问。因此在这个应用中格雷码并非是有利的。

(b) 现在（通过73 000次存储访问和4900个乘法），我们可以达到小于241的所有数，但241不在内。有46种方式来表示100，包括著名的$9 - 87 + 6 + 5 - 43 + 210$。

[亨·欧·达德尼在*The Weekly Dispatch* (1899年6月4日和18日) 中介绍他的"世纪"问题，也参见由马丁·加德纳所著的*Numerology of Dr. Matrix*，第6章。史蒂文·卡汉 (Steven Kahan)，*J. Recreational Math.* **23** (1991)，19-25。]

112. 习题111的方法现在需要1亿6千7百万次以上存储访问和1千万次乘法，因为3^{16}要比3^8大得多。通过首先对于$1 \leqslant k < 9$，从头k位和后k位中可得到的可能性造成表，我们可以做得好得多（1040万次存储访问，1100次乘法），然后考虑使用9
100 的所有数字块，共有60 318种表示100的方法，而且最初不可达的个数是16 040。

7.2.1.2 节

1. [约·帕·诺·菲利普斯 (J. P. N. Phillips)，*Comp. J.* **10** (1967)，311。] 假设$n \geqslant 3$，我们可用下列步骤来代替步骤L2 ~ L4:

L2'. [最容易的情况？] 置$y \leftarrow a_{n-1}$和$z \leftarrow a_n$。如果$y < z$则置$a_{n-1} \leftarrow z$，$a_n \leftarrow y$，并返回L1。

L2.1'. [下一个最容易的情况？] 置$x \leftarrow a_{n-2}$，如果$x \geqslant y$则在L2.2'步上继续。否则如果$x < z$，则置$(a_{n-2}, a_{n-1}, a_n) \leftarrow (z, x, y)$，如果$x \geqslant z$，则为$(y, z, x)$，返回L1。

L2.2'. [求j。] 置$j \leftarrow n - 3$和$y \leftarrow a_j$。如果$y \geqslant x$，置$j \leftarrow j - 1$，$x \leftarrow y$，$y \leftarrow a_j$，并重复直到$y < x$为止。如果$j = 0$则结束。

L3'. [容易增加？] 如果$y < z$，则置$a_j \leftarrow z$，$a_{j+1} \leftarrow y$，$a_n \leftarrow x$并转到L4.1'。

L3.1'. [增加a_j。] 置$l \leftarrow n - 1$；如果$y \geqslant a_l$。重复地做l减1直到$y < a_l$为止，然后置$a_j \leftarrow a_l$及$a_l \leftarrow y$。

L4'. [开始颠倒。] 置$a_n \leftarrow a_{j+1}$和$a_{j+1} \leftarrow z$。

L4.1'. [颠倒$a_{j+2} \cdots a_{n-1}$。] 置$k \leftarrow j + 2$，$l \leftarrow n - 1$。然后如果$k < l$，交换$a_k \leftrightarrow a_l$，置$k \leftarrow k + 1$，$l \leftarrow l - 1$，并重复直到$k \geqslant l$。返回L1。 ∎

对于$0 \leqslant t \leqslant n$，如果把$a_t$存储到存储单元$A[n - t]$中，这个程序可能运行得还要

更快，或者如果像在下道题那样，使用颠倒的协词典顺序也将如此。

2. 我们再次假设，开始时$a_1 \leqslant a_2 \leqslant \cdots \leqslant a_n$；然而，从{1, 2, 2, 3}生成的排列将是1223, 2123, 2213, …, 2321, 3221。令a_{n+1}是大于a_n的一个辅助元素。

L1. [访问。] 访问排列$a_1 a_2 \cdots a_n$。

L2. [找j。] 置$j \leftarrow 2$。如果$a_{j-1} \geqslant a_j$，则j加1直到$a_{j-1} < a_j$为止。如果$j > n$则结束。

L3. [减少a_j。] 置$l \leftarrow 1$。如果$a_l \geqslant a_j$，则增加l直到$a_l < a_j$为止。然后交换$a_l \leftrightarrow a_j$。

L4. [颠倒$a_1 \cdots a_{j-1}$。] 置$k \leftarrow 1$和$l \leftarrow j-1$。然后如果$k < l$，交换$a_k \leftrightarrow a_l$，置$k \leftarrow k+1$，$l \leftarrow l-1$，并且重复直到$k \geqslant l$为止。返回L1。 ∎

3. 如同在习题5.1.1-7中那样，令$C_1 \cdots C_n = c_{a_1} \cdots c_{a_n}$为反序表，则$\operatorname{rank}(a_1 \cdots a_n)$是混合进制数$\begin{bmatrix} C_1, \cdots, C_{n-1}, C_n \\ n, \cdots, 2, 1 \end{bmatrix}$。[参见海·奥·罗塞（H. A. Rothe），*Sammlung combinatorisch -analytischer Abhandlungen* 2 (1800)，263-264；也可参见在7.2.1.7节所引用的纳拉燕纳的先驱性工作。] 例如，314592687有秩$\begin{bmatrix} 2,0,1,1,4,0,0,1,0 \\ 9,8,7,6,5,4,3,2,1 \end{bmatrix} = 2 \cdot 8! + 6! + 5! + 4 \cdot 4! + 1! = 81577$。这是在等式4.1-(10)中所表征的阶乘数字。

4. 使用递归式$\operatorname{rank}(a_1 \cdots a_n) = \frac{1}{n} \sum_{j=1}^{t} n_j [x_j < a_1] \binom{n}{n_1, \cdots, n_t} + \operatorname{rank}(a_2 \cdots a_n)$。例如rank (314159265)是

$$\frac{3}{9}\binom{9}{2,1,1,1,2,1,1} + 0 + \frac{2}{7}\binom{7}{1,1,1,2,1,1} + 0 + \frac{1}{5}\binom{5}{1,2,1,1} + \frac{3}{4}\binom{4}{1,1,1,1} + 0 + \frac{1}{2}\binom{2}{1,1} = 30991$$

5. (a) 步骤L2被执行$n!$次。恰好作k次比较的概率是$q_k - q_{k+1}$，其中q_t是$a_{n-t+1} > \cdots > a_n$的概率，就是$[t \leqslant n]/t!$；因此均值是$\sum k(q_k - q_{k+1}) = q_1 + \cdots + q_n = \lfloor n!e \rfloor / n! - 1 \approx e - 1 \approx 1.718$，而且方差为

$$\sum k^2(q_k - q_{k+1}) - \text{mean}^2 = q_1 + 3q_2 + \cdots + (2n-1)q_n - (q_1 + \cdots + q_n)^2 \approx e(3-e) \approx 0.766$$

[关于更高的动量，请参见莱·肯普（R. Kemp），*Acta Informatica* 35 (1998)，17-89，定理4。] 101

顺便指出，在步骤L4中交换操作的平均次数是$\sum \lfloor k/2 \rfloor (q_k - q_{k+1}) = q_2 + q_4 + \cdots \approx \cosh 1 - 1 = (e + e^{-1} - 2)/2 \approx 0.543$。这是由理·詹·奥德-史密斯[*Comp. J.* 13 (1970)，152-155] 给出的一个结果。

(b) 步骤L3仅被执行$n! - 1$次，但为方便起见，我们将假设它出现多一次（有0个比较），则对于$1 \leqslant k < n$，恰好作k次比较的概率是$\sum_{j=k+1}^{n} 1/j!$。而对于$k = 0$，为$1/n!$。因此均值为$\frac{1}{2}\sum_{j=0}^{n-2} 1/j! \approx e/2 \approx 1.359$。习题1把这个数减少2/3，方差是$\frac{1}{3}\sum_{j=0}^{n-3} 1/j! + \frac{1}{2}\sum_{j=0}^{n-2} 1/j! - \text{mean}^2 \approx \frac{5}{6}e - \frac{1}{4}e^2 \approx 0.418$。

6. (a) 令 $e_n(z)=\sum_{k=0}^{n} z^k/k!$，则不同前缀$a_1\cdots a_j$的个数为$j![z^j]e_{n_1}(z)\cdots e_{n_t}(z)$。这是$N=\binom{n}{n_1,\cdots,n_t}$乘以在步骤L2中至少作$n-j$次比较的概率$q_{n-j}$。因此均值为$\frac{1}{N}w(e_{n_1}(z)\cdots e_{n_t}(z))-1$，其中$w(\sum x_k z^k/k!)=\sum x_k$。在二进制情况下这个均值是$M/\binom{n}{s}-1$，其中$M=\sum_{l=0}^{s}\sum_{k=l}^{n-s+l}\binom{k}{l}=\sum_{l=0}^{s}\binom{n-s+l+1}{l+1}=\binom{n+2}{s+1}-1=\binom{n}{s}\left(2+\frac{s}{n-s+1}+\frac{n-s}{s+1}\right)-1$。

(b) 如果$\{a_1,\cdots,a_j\}=\{n_1'\cdot x_1,\cdots,n_t'\cdot x_t\}$，则前缀$a_1\cdots a_j$一起对在步骤L3中的总比较次数贡献$\sum_{1<k<l<t}(n_k-n_k')[n_l>n_l']$。因此均值是$\frac{1}{N}\sum_{1<k<l<t}w(f_{kl}(z))$，其中

$$f_{kl}(z)=\Bigg(\prod_{\substack{1<m<t\\ m\neq k,\,m\neq l}} e_{n_m}(z)\Bigg)\Bigg(\sum_{r=0}^{n_k}(n_k-r)\frac{z^r}{r!}\Bigg)e_{n_{l-1}}(z)$$

$$=e_{n_1}(z)\cdots e_{n_t}(z)(n_k-zr_k(z))r_l(z),\qquad \text{其中 } r_k(z)=\frac{e_{n_k-1}(z)}{e_{n_k}(z)}$$

在二值情况下，这个公式简化为$\frac{1}{N}w((se_s(z)-ze_{s-1}(z))e_{n-s-1}(z))=\frac{s}{N}\left(\binom{n+1}{s+1}-1\right)-\frac{1}{N}\left(\binom{n+1}{s+1}\left(s-\frac{s+1}{n-s+1}\right)+1\right)=\frac{1}{N}\left(-s-1+\binom{n+1}{s}\right)=\frac{n+1}{n-s+1}-\frac{s+1}{N}$。

7. 在上道题的记号下，量$\frac{1}{N}w(s_{n_s}(z)\cdots e_{n_t}(z))-1$为

$$\frac{n_1+\cdots+n_t}{n}+\frac{(n_1n_2+n_1n_3+\cdots+n_{t-1}n_t)+n_1(n_1-1)+\cdots+n_t(n_t-1)}{n(n-1)}+\cdots$$

利用等式1.2.9-(38)人们能证明这个极限是$-1+\exp\sum_{k>1}r_k/k$，其中$r_k=\lim_{t\to\infty}(n_1^k+\cdots+n_t^k)/(n_1+\cdots+n_t)^k$。在情况(a)和(b)中我们有$r_k=[k=1]$。所以这个极限为$e-1\approx 1.71828$。在情况(c)，我们有$r_k=1/(2^k-1)$。所以极限为$-1+\exp\sum_{k>1}1/(k(2^k-1))\approx 2.46725$。

8. 假设j开始时为0，并把步骤L1改成为

L1'. [访问。] 访问变形$a_1\cdots a_j$。如果$j<n$，则置$j\leftarrow j+1$，并重复此步骤。 ∎

这个算法是由路·约·菲希尔（L. J. Fischer）和卡·克·克劳斯（K. C. Krause）给出的[*Lehrbuch der Combinationslehre und der Arithmetik* (Dresden:1812), 55-57]。

顺便指出，在答案6的记号下变形的总数为$w(e_{n_1}(z)\cdots e_{n_t}(z))$。这个计数问题首先是由詹姆斯·贝努利（James Bernoulli）在*Ars Conjectandi* (1713) 第2部分第9章中讨论的。

9. R1. [访问。] 访问变形$a_1\cdots a_r$。（这时$a_{r+1}\leqslant\cdots\leqslant a_n$。）

R2. [容易情况吗？] 如果$a_r<a_n$，则交换$a_r\leftrightarrow a_j$，其中j是使得$j>r$和$a_j>a_r$的最小下标，并返回R1。

102 R3. [颠倒。] 像在步骤L4中那样，置$(a_{r+1},\cdots,a_n)\leftarrow(a_n,\cdots,a_{r+1})$。

R4. [找j。] 置$j\leftarrow r-1$。如果$a_j\geqslant a_{j+1}$，则j减1，并重复直到$a_j<a_{j+1}$为止。

如果$j=0$则结束。

R5. [增加a_j。] 置$l\leftarrow n$，如果$a_j\geqslant a_l$，则l减1，重复到$a_j<a_l$为止。然后交换$a_j\leftrightarrow a_l$。

R6. [再次颠倒。] 像在步骤L4中那样，置$(a_{j+1},\cdots,a_n)\leftarrow(a_n,\cdots,a_{j+1})$，并返回R1。 ∎

输出的个数为$r![z^r]e_{n_1}(z)\cdots e_{n_t}(z)$，当诸元素都不同时，这当然是$n^t$。

10. 如果$n\geqslant 2$，$a_1a_2\cdots a_n=213\cdots n$，$c_1c_2\cdots c_n=010\cdots 0$，$o_1o_2\cdots o_n=1(-1)1\cdots 1$。

11. 步骤（P1…P7）被执行$(1, n!, n!, n!+x_n, n!, (x_n+3)/2, x_n)$次，其中$x_n=\sum_{k=1}^{n-1}k!$，因为当$2<j<n$时P7被执行$(j-1)!$次。

12. 我们要秩为999999的排列。答案是：(a) 2783915460（由习题3）；(b) 8750426319，因为由7.2.1.1-(50)，对应于$\begin{bmatrix}0,0,1,2,3,0,2,7,0,9\\1,2,3,4,5,6,7,8,9,10\end{bmatrix}$的混合进制数是$\begin{bmatrix}0,0,\ 1,3-2,3,5-0,2,7,8-0,9-9\\1,2,3,\ 4,\ \ 5,\ 6,\ \ 7,8,\ 9,\ \ 10\end{bmatrix}$ (c) 这个乘积是$(01\cdots 9)^9(01\cdots 8)^0(01\cdots 7)^7(01\cdots 6)^2\cdots(012)^1$，即9703156248。

13. 对于所有$n\geqslant 2$这个命题为真。但当2穿过1时，即当c_2从0变成1时，我们有$c_3=2, c_4=3, c_5=\cdots=c_n=0$。而当$n\geqslant 5$时下一个排列为$432156\cdots n$[见*Time Travel* (1988), 74]。

14. 在步骤P4、P5和P6开始时为真，因为恰有$j-1-c_j+s$个元素位于x_j的左边，即从$\{x_1,\cdots,x_{j-1}\}$有$j-1-c_j$个和从$\{x_{j+1},\cdots,x_n\}$有s个。（在某种意义下，这个公式是算法P的要点。）

15. 如果$\begin{bmatrix}b_{n-1},\cdots,b_0\\1,\ \ \cdots,n\end{bmatrix}$对应于反射格雷码$\begin{bmatrix}c_1,\cdots,c_n\\1,\cdots,n\end{bmatrix}$，由7.2.1.1-(50)我们到达步骤P6当且仅当对于$j\leqslant k\leqslant n$，$b_k=k-1$和B_{n-j+1}为偶数。但对于$j\leqslant k\leqslant n$，$b_{n-k}=k-1$意味着对于$j<k\leqslant m$，B_{n-k}为奇数。因此在步骤P5中$s=[c_{j+1}=j]+[c_{j+2}=j+1]=[o_{j+1}<0]+[o_{j+2}<0]$。（参见*Math. Comp.* 17 (1963), 282-285。）

16. P1'. [初始化。] 对于$1\leqslant j<n$，置$c_j\leftarrow j$和$o_j\leftarrow -1$；也置$z\leftarrow a_n$。

P2'. [访问。] 访问$a_1\cdots a_n$。如果$a_1=z$，则转到P3.5'。

P3'. [向下搜寻。] 对于$j\leftarrow n-1, n-2, \cdots, 1$（以此顺序），置$a_{j+1}\leftarrow a_j$，$a_j\leftarrow z$并访问$a_1\cdots a_n$。然后置$j\leftarrow n-1$，$s\leftarrow 1$，并转到P4'。

P3.5' [向上搜寻。] 对于$j\leftarrow 1, 2, \cdots, n-1$，（以这个顺序），置$a_j\leftarrow a_{j+1}$，$a_{j+1}\leftarrow z$，并访问$a_1\cdots a_n$，然后置$j\leftarrow n-1$，$s\leftarrow 0$。

P4'. [准备好来改动吗？] 置$q\leftarrow c_j+o_j$。如果$q=0$，则转到P6'；如果$q>j$，则转到P7'。

P5'. [改动。] 交换$a_{c_j+s}\leftrightarrow a_{q+s}$。然后置$c_j\leftarrow q$并返回P2'。

P6'. [增加s。] 如果$j=1$则结束；否则置$s\leftarrow s+1$。

P7'. [转向。] 置$o_j\leftarrow -o_j$，$j\leftarrow j-1$，并转回P4'。 ∎

17. 开始时，对于$1 \leqslant j \leqslant n$，$a_j \leftarrow a'_j \leftarrow j$。步骤P5现在应置$t \leftarrow j - c_j + s$, $u \leftarrow j - q + s$, $v \leftarrow a_u$, $a_t \leftarrow v$, $a'_v \leftarrow t$, $a_u \leftarrow j$, $a'_j \leftarrow u$, $c_j \leftarrow q$。（参见习题14。）

但对于要求的和可利用的逆，我们实际上可大大简化算法，避免偏离变量s和令控制表$c_1 \cdots c_n$向下计数，如同吉·厄尔里兹所说明的那样 [*JACM* 20 (1973), 505-506]:

103 Q1. [初始化。] 对于$1 \leqslant j \leqslant n$，置$a_j \leftarrow a'_j \leftarrow j$，$c_j \leftarrow j - 1$和$o_j \leftarrow -1$，还置$c_0 = -1$。

Q2. [访问。] 访问排列$a_1 \cdots a_n$和它的逆$a'_1 \cdots a'_n$。

Q3. [找k。] 置$k \leftarrow n$。如果$c_k = 0$则置$c_k \leftarrow k - 1$，$o_k \leftarrow -o_k$和$k \leftarrow k - 1$，并重复直到$c_k \neq 0$为止。如果$k = 0$则结束。

Q4. [改动。] 置$c_k \leftarrow c_k - 1$，$j \leftarrow a'_k$和$i = j + o_k$，然后置$t \leftarrow a_i$，$a_i \leftarrow k$，$a_j \leftarrow t$，$a'_i \leftarrow j$，$a'_k \leftarrow i$，并返回Q2。 ∎

18. 置$a_n \leftarrow n$并使用算法P的$(n-1)!/2$个迭代来生成$\{1, \cdots, n-1\}$的所有排列，使得1居于2之前[莫·库·罗伊（M. K. Roy），*CACM* 16 (1973), 312-313；也参见习题13]。

19. 例如，我们可以使用算法P的思想，同时相对于进制（1, 2, …, n）像在算法7.2.1.1H中那样改动n元组$c_1 \cdots c_n$。尽管该算法对下标进行不同的编号，但它正确地维持方向。算法P所需要的偏离s可如同在习题15中那样来计算，或者逆排列可如同在习题17中那样来维持。[参见吉·厄尔里兹，*CACM* 16 (1973), 690-691。]另一个算法像希普算法那样，也可无循环地加以实现。

（注意：在排列生成的大多数应用中，我们感兴趣于极小化总的运行时间，而不是逐次访问之间的极大时间。从这个观点出发，无循环性是不合适的，除非是在一台并行计算机上。但是也确有在智能上满足无循环算法这样一个事实存在，无论它实用与否。）

20. 例如，当$n = 3$时我们可以开始123, 132, 312, $\bar{3}12$, $1\bar{3}2$, $12\bar{3}$, $21\bar{3}$, …, 213, $\bar{2}13$, …。如果对于n的δ序列是$(\delta_1 \delta_2 \cdots \delta_{2^n n!})$，对于$n+1$对应的序列是$(\Delta_n \delta_1 \Delta_n \delta_2 \cdots \Delta_n \delta_{2^n n!})$，其中$\Delta_n$是$2n-1$个操作$n\ n-1 \cdots 1\ -1 \cdots n-1\ n$的序列；这里$\delta_k = j$意味着$a_j \leftrightarrow a_{j+1}$，$\delta_k = -$意味着$a_1 \leftarrow -a_1$。

（带符号的排列以另一个状态出现在习题5.1.4-43和44中，所有带符号的排列的集合叫做八面体群。）

21. 显然M = 1，因此O必定为0而S必定为$b-1$。然后N = E + 1，R = $b-2$，且D + E = b + Y。当Y = $k \geqslant 2$时这恰好对E留下max(0, b-7-k) 个选择，因此当$b > 8$时总数为$\sum_{k=2}^{b-7}(6-7-k) = \binom{b-8}{2}$个解。[*Math. Mag.* 45 (1972), 48-49。顺便指出，戴·爱泼斯坦（D. Eppstein）已经指出，对于一个给定的进制求解字母算术的任务是NP完全的，参见*SIGACT News* 18, 3 (1987)，38-40。]

22. 仅当$b = 2$时$(\mathrm{XY})_b + (\mathrm{XX})_b = (\mathrm{XYX})_b$才是可解的。

23. 几乎为真，因为解的个数将为偶数，除非$[j \in F] \neq [k \in F]$。（考虑三进制字母算术X + $(\mathrm{XX})_3$ + $(\mathrm{YY})_3$ + $(\mathrm{XZ})_3$ = $(\mathrm{XYX})_3$。）

24. (a). 9283 + 7 + 473 + 1062 = 10825; (b). 698392 + 3192 = 701584; (c). 63952 + 69275 = 133227; (d). 653924 + 653924 = 1307848; (e). 5718 + 3 + 98741 = 104462; (f). 127503 + 502351 + 3947539 + 46578 = 4623971; (g). 67432 + 704 +

8046 + 97364 = 173546; (h). 59 + 577404251698 + 69342491650 + 49869442698 + 1504 + 40614 + 82591 + 344 + 41 + 741425 = 5216367650 + 691400684974。[所有解都是惟一的，对于(b) ~ (g)的参考文献: *J. Recreational Math.* 10 (1977)，115; 5(1972)，296; 10(1977)，41; 10(1978)，274; 12(1979)，133-134; 9(1977)，207。]

(i) 在此情况下，有 $\frac{8}{10}10!=2903040$个解，因为除了把H或N赋值为0之外的{0, 1, …, 9}的每个排列都有效。(在这样的情况下，一个编写良好的一般性加法字母算术求解程序将仔细地减少输出的数量。)

25. 我们可以假设$s_1 \leqslant \cdots \leqslant s_{10}$。令$i$是$\notin F$的最小下标，并置$a_i \leftarrow 0$; 然后以$j$的递增顺序来置剩下的元素$a_j$。和定理6.1S类似的一个证明表明，这个过程极大化$a \cdot s$。 104
一个类似的过程产生极小值，因为$\min(a \cdot s) = -\max(a \cdot (-s))$。

26. 400739 + 63930 − 2379 − 1252630 + 53430 − 1390 + 738300。

27. 读者大概可对下列例子进行改进。BLOOD + SWEAT + TEARS = LATER; EARTH + WATER + WRATH = HELLO + WORLD, AWAIT + ROBOT + ERROR = SOBER + WORDS; CHILD + THEME + PEACE + EHTIC = IDEAL + ALPHA + METIC。(本题是受到WHERE + SEDGE + GRASS + GROWS = MARSH [小阿 · 威 · 约翰森 (A. W. Johnson, Jr.), *J. Recr. Math.* 15 (1982), 51]的激励而设计出的，除了D和O有相同签名外，这将是奇妙地纯粹的。)

28. (a) 11 = 3 + 3 + 2 + 2 + 1, 20 = 11 + 3 + 3 + 3, 20 = 11 + 3 + 3 + 2 + 1, 20 = 11 + 3 + 3 + 1 + 1 + 1, 20 = 8 + 8 + 2 + 1 + 1, 20 = 7 + 7 + 6, 20 = 7 + 7 + 2 + 2 + 2, 20 = 7 + 7 + 2 + 1 + 1 + 1 + 1, 20 = 7 + 5 + 5 + 2 + 1, 20 = 7 + 5 + 2 + 2 + 2 + 1 + 1, 20 = 7 + 5 + 2 + 2 + 1 + 1 + 1 + 1, 20 = 7 + 3 + 3 + 2 + 2 + 1 + 1 + 1, 20 = 7 + 3 + 3 + 1 + 1 + 1 + 1 + 1 + 1 + 1, 20 = 5 + 3 + 3 + 3 + 3 + 3。[这14个解首先是由罗伊 · 蔡德斯 (Roy Childs) 于1999年计算出来的，n的下一个可双重分划的值是30 (以20种方式)，然后40 (以94种方式)，41 (以67种方式)，42 (以57种方式)，50 (以190种方式，包括50 = 2 + 2 + … + 2)，等等。]

(b) 51 = 20 + 15 + 14 + 2, 51 = 15 + 14 + 10 + 9 + 3, 61 = 19 + 16 + 11 + 9 + 6, 65 = 17 + 16 + 15 + 9 + 7 + 1, 66 = 20 + 19 + 16 + 6 + 5, 69 = 18 + 17 + 16 + 10 + 8, 70 = 30 + 20 + 10 + 7 + 3, 70 = 20 + 16 + 12 + 9 + 7 + 6, 70 = 20 + 15 + 12 + 11 + 7 + 5, 80 = 50 + 20 + 9 + 1, 90 = 50 + 12 + 11 + 9 + 5 + 2 + 1, 91 = 45 + 19 + 11 + 10 + 5 + 1。[头两个51是由史蒂文 · 卡汉给出的，见他的书*Have Some Sums to Solve* (Farmingdale, New York: Baywood, 1978), 36-37, 84, 112。带有在意大利文中17个不同的项和58个在罗马数字下不同的项的有趣例子，已被久里欧 · 西沙利 (Giulio Cesare) 发现，见J. Recr. Math. 30 (1999), 63。]

注意: 漂亮的例子，THREE = TWO + ONE + ZERO[理查德 · 刘 · 布赖斯 (Richard L. Breisch), *Recreational Math. Magazine* 12 (1962年12月), 24]不幸地为我们的约定排除掉。在英文中，双重地为真分划成不同部分的总数大概是有限的，尽管对于任意大的整数而言不是标准的，有没有比NINETYNINENONILLIONNINETYNINESEXTILLIONSIXTYONE =

NINETYNINENONILLIONNINETYNINESEXTILLIONNINETEEN + SIXTEEN + ELEVEN + NINE + SIX更大的例子?（由热·宫扎里兹-莫利斯（G. González-Morris）提出。）

29. 10 + 7 + 1 = 9 + 6 + 3，11 + 10 = 8 + 7 + 6，12 + 7 + 6 + 5 = 11 + 10 + 9，…，19 + 10 + 3 = 14 + 13 + 4 + 1（全部共有31个例子）。

30. (a) $567^2 = 321489$，$807^2 = 651249$，或$854^2 = 729316$。(b) $958^2 = 917764$。(c) $96 \times 7^2 = 4704$。(d) 51304/61904 = 7260/8760。(e) $328509^2 = 4761^3$。[*Strand* 78 (1929), 91, 208; *J. Recr. Math.* 3(1970), 43; 13(1981), 212; 27(1995), 137; 31 (2003), 133。(b)、(c)、(d)和(e)的解是惟一的，通过基于算法X的一个自右到左的方法，这些答案分别在（14, 13, 11, 3423, 42）千内存访问中解出。芦ケ原伸之还注意到，NORTH/SOUTH = WEST/EAST有惟一解67104/27504 = 9320/3820。]

31. 5/34 + 7/68 + 9/12(!)。在大约265Kμ中，使用附带条件A < D < G，通过算法X人们可以验证惟一性。[*Quark Visual Science Magazine*, No.136(*Tokyo*: *Kodansha*, 1993年10月)。] 奇怪的是，一个非常类似的难题也有一个惟一解：1/(3 × 6) + 5/(8 × 9) + 7/(2 × 4) = 1；参见斯戈特·莫里斯（Scot Morris），*Omni* 17, 4(1995年1月)，97。

32. 共有11种方法，其中最令人惊讶的是3 + 69258/714。[参见*Weekly*
105 *Dispatch* (1901年6月9日和23日)；*Amusements in Mathematics* (1917)，158-159。]

33. (a) 1, 2, 3, 4, 15, 18, 118, 146; (b)6, 9, 16, 20, 27, 126, 129, 136, 145。[*The Weekly Dispatch* (1902年11月11日和30日)；*Amusements in Math.* (1917)，159。]

在这种情况下，一个适当的策略是找出所有变形，其中$a_k \cdots a_{l-1}/a_l \cdots a_9$为一个整数，然后对于$a_1 \cdots a_{k-1}$的所有排列来记录解。恰有带有惟一解的164959个整数，最大的是9876533。除开2091年外，对于21世纪的所有年都有解。当$n = 6443$时出现最多的解(125)，可表示的n的最长的延伸是$5109 < n < 7060$。通过“去9”对于小的n用手算，达德尼能够给出正确的答案。

34. (a)$x = 10^5$，7378 + 155 + 92467 = 7178 + 355 + 92467 = 1016 + 733 + 98251 = 100000，(b)$x = 4^7$。3036 + 455 + 12893 = 16384是惟一的。求解这个问题最快的方法大概是用2529个素数的一个表开始，这些素数都由5个不同数字组成（就是10243, 10247, …, 98731），然后来排列剩下的数字。

顺便指出，不加限制的字母算术EVEN + ODD = PRIME有10个解：而在这些解当中，ODD和PRIME都是素数的就只有一个。[参见有泽诚（M. Arisawa），*J. Recr.* Math. 8(1975)，153。]

35. 一般说来，如果对于$1 < k < n$，$s_k = |S_k|$，则有$s_1 \cdots s_{k-1}$种方法来选择S_k中每个不相等的元素，因此答案为 $\prod_{k=1}^{n-1}\left(\prod_{j=1}^{k-1} s_j^{s_k-1}\right)$。在此情况下，它是$2^2 \cdot 6^3 \cdot 24^{15} =$ 436196692474023836123136。

（但如果对顶点重新编号，s_k个值可能改变。例如，如果(12)的顶点(0, 3, 5)同(e, d, c)交换，我们有$s_{14} = 1$，$s_{13} = 6$，$s_{12} = 4$，$s_{11} = 1$和$4^5 \cdot 24^{15}$个西姆斯表。）

36. 由于{0, 3, 5, 6, 9, a, c, f}的每一个位于3条线上，但每个其他元素仅位于两条线上，显然我们可以令$S_f = \{(\,), \sigma, \sigma^2, \sigma^3, \alpha, \alpha\sigma, \alpha\sigma^2, \alpha\sigma^3\}$，其中$\sigma =$

(03fc)(17e4)(2bd4)(56a9) 是一个90°转动而α = (05)(14)(27)(36)(8d)(9c)(af)(be)是一个由内向外的扭绞。还有$S_e = \{(\ ), \beta, \gamma, \beta\gamma\}$，其中$\beta$ = (14)(28)(3c)(69)(be)是一个转置，而γ = (12)(48)(5a)(69)(7b)(de)是另一个扭绞。$S_d = \cdots = S_1 = \{(\ )\}$。(共有$4^7 - 1$个可能的答案。)

37. 集合S_k可以$k!^{k-1}$种方式来选出（参见习题35），而且它的不相等元素可以用进一步的$k!$种方式来赋值给$\sigma(k, 1), \cdots, \sigma(k, k)$。因此答案为 $A_n = \prod_{k=1}^{n-1} k!^k = n!^{\binom{n}{2}} / \prod_{k=1}^{n} k^{\binom{k}{2}}$。例如，$A_{10} \approx 6.256 \times 10^{148}$，通过欧拉求和公式我们有

$$\sum_{k=1}^{n-1} \binom{k}{2} \ln k = \frac{1}{2}\int_1^n x(x-1)\ln x \ dx + O(n^2 \log n) = \frac{1}{6} n^3 \ln n + O(n^3)$$

因此 $\ln A_n = \frac{1}{3} n^3 \ln n + O(n^3)$

38. 对于$1 \leqslant k < n$，在G4步中需要$\phi(k)$的概率是$1/k! - 1/(k+1)!$，我们全然达不到步骤G4的概率是$1/n!$。由于$\phi(k)$做$\lceil k/2 \rceil$个转置，平均值是 $\sum_{k=1}^{n-1}(1/k! - 1/(k+1)!)\lceil k/2 \rceil = \sum_{k=1}^{n-1}\big(\lceil k/2 \rceil - \lceil (k-1)/2 \rceil\big)/k! - \lceil (n-1)/2 \rceil / n! = \Sigma_{k\text{奇数}} 1/k! + O(1/(n-1)!)$。

39. (a) 0123, 1023, 2013, 0213, 1203, 2103, 3012, 0312, 1302, 3102, 0132, 1032, 2301, 3201, 0231, 2031, 3021, 0321, 1230, 2130, 3120, 1320, 2310, 3210, (b) 0123, 1023, 2013, 0213, 1203, 2103, 3102, 1302, 0312, 3012, 1032, 0132, 0231, 2031, 3021, 0321, 2301, 3201, 3210, 2310, 1320, 3120, 2130, 1230。

40. 由归纳法，我们发现$\sigma(1, 1)$ = (0 1)，$\sigma(2, 2)$ = (0 12)。

$$\sigma(k,k) = \begin{cases} (0\ \ k)(k-1\ \ k-2\cdots 1), & \text{如果} k \geqslant 3 \text{为奇数} \\ (0\ \ k-1\ \ k-2\ \ 1\cdots k-3\ \ k), & \text{如果} k \geqslant 4 \text{为偶数} \end{cases}$$

106

当k为偶数时还有$\omega(k) = (0\ k)$，当$k \geqslant 3$为奇数时$\omega(k) = (0\ k-2\cdots 1\ k-1\ k)$。因此当$k \geqslant 3$为奇数时，$\sigma(k, 1) = (k\ k-1\ 0)$ 而且对于$1 < j < k$，$\sigma(k, j)$ 使$k \mapsto j-1$。当$k \geqslant 4$为偶数时，对于$1 \leqslant j \leqslant k$，$\sigma(k, j) = (0\ k\ k-3\cdots 1\ k-2\ k-1)^j$。

注意：引起算法G通过单个转置来生成所有排列的头一个方案是由马克·维尔斯（Mark Wells）[*Math. Comp.* 15 (1961), 192-195]设计的；但它颇为复杂。小维·里普斯基（W. Lipski. Jr.）研究了一般的这样的方案，并且找到了各种另外的方法[*Computing* 23 (1979)，357-365]。

41. 我们可以假设$r < n$。假定我们重新定义$\omega(k)$ 为$\sigma(n-r, n-r)\cdots\sigma(k, k)$ 而不是使用(16)，如果在步骤G3中我们简单地把“$k \leftarrow 1$”改成为“$k \leftarrow n-r$”，则算法G将生成对于任何西姆斯表的r变形。

如果$n-r$为奇数，则(27)的方法仍然有效，尽管当$k < n-r+2$时在答案40中的公式要加以修改。当$k = n-r$时，新的公式是$\sigma(k, j) = (k\ j-1\ \cdots 1\ 0)$ 和$\omega(k) = (k\cdots 1\ 0)$；而当$k = n-r+1$时，$\sigma(k, j) = (k\cdots 1\ 0)^j$。

如果$n-r$为偶数，且如果$r < 3$，我们可以使用(27)，但把偶和奇两者调换。但当$r \geqslant 4$时，需要一个更复杂的方案，因为像（$k\ 0$）这样一个固定的转置仅当$\omega(k-1)$

是一个k循环时才可对奇的k使用。这意味着$\omega(k-1)$必须是一个偶排列，但对于$k \geqslant n-r+2$，$\omega(k)$是奇的。

当$n-r$为偶的时，以下方案有效：对于$1 \leqslant j \leqslant k = n-r$令$\tau(k,j)\ \omega(k-1)^- = (k\ k-j)$，而且当$k > n-r$时使用(27)。于是当$k = n-r+1$时，我们有$\omega(k-1) = (01\cdots k-1)$，因此对于$1 \leqslant j \leqslant k$，$\sigma(k,j)$使$k \mapsto (2j-1) \bmod k$，而且$\sigma(k,k) = (k, k-1, k-3, \cdots, 0, k-2, \cdots, 1)$，$\omega(k) = (k\cdots 1\ 0)$，$\sigma(k+1,j) = (k+1\cdots 0)^j$。

42. 如果$\sigma(k,j) = (k\ j-1)$，我们有$\tau(k,1) = (k\ 0)$，而且对于$2 \leqslant j \leqslant k$，$\tau(k,j) = (k\ j-1)(k\ j-2) = (k\ j-1\ j-2)$。

43. 当然，$\omega(1) = \sigma(1,1) = \tau(1,1) = (0\ 1)$。对于所有$k \geqslant 2$下列构造使$\omega(k) = (k-2, k-1, k)$：令$\alpha(k,j) = \tau(k,j)\ \omega(k-1)^-$，其中$\alpha(2,1) = (2\ 0)$，$\alpha(2,2) = (2\ 0\ 1)$，$\alpha(3,1) = \alpha(3,3) = (3\ 1)$，$\alpha(3,2) = (3\ 1\ 0)$；这使$\sigma(2,2) = (0\ 2)$，$\sigma(3,3) = (0\ 3\ 1)$。然后对于$k \geqslant 4$，令$\alpha(k,1) = (k\ k-4)$，对$1 < j < k-2$，令$\alpha(k,j) = (k\ k-3-j\ k-2-j)$，且

$k \bmod 3 = 0$		$k \bmod 3 = 1$		$k \bmod 3 = 2$
$\alpha(k,k-2) = (k\ k-2\ 0)$	或	$(k\ k-3\ 0)$	或	$(k\ k-1\ 0)$,
$\alpha(k,k-1) = (k\ k-2\ k-3)$	或	$(k\ k-3)$	或	$(k\ k-1\ k-3)$,
$\alpha(k,k) = (k\ k-2)$	或	$(k\ k-3\ k-2)$	或	$(k\ k-2)$;

这使得如同所要求的那样，$\sigma(k,k) = (k-3\ k\ k-2)$。

44. 否。因为$\tau(k\ j)$是一个$(k+1)$循环，不是一个转置。（参见(19)和(24)。）

45. (a) 202280070，因为$u_k = \max(\{0, 1, \cdots, \alpha_{k-1}\} \backslash \{a_1, \cdots, a_{k-1}\})$。（实际上，$u_n$绝不由这个算法设置，但我们可以假设它为零。）(b) 425368917。

46. 真的（假设$u_n = 0$）。如果$u_k > u_{k+1}$或者$a_k > a_{k+1}$，我们必定有$a_k > u_k \geqslant a_{k+1} > u_{k+1}$。

47. 步骤（X1, X2, …, X6）被分别执行（1, A, B, $A-1$, $B-N_n$, A）次，其中$A = N_0 + \cdots + N_{n-1}$且$B = nN_0 + (n-1)N_1 + \cdots + 1N_{n-1}$。

48. 步骤（X2, X3, X4, X5, X6）分别被执行$A_n + (1, n!, 0, 0, 1)$次，其中$A_n = \sum_{k=1}^{n-1} n^{\underline{k}} = n! \sum_{k=1}^{n-1} 1/k! \approx n!(e-1)$。假设对于涉及$a_j$、$l_j$或$u_j$的操作，它们分别耗费（1, 1, 3, 1, 3）个存储访问，总的耗费是每个排列大约$9e-8 \approx 16.46$个存储访问。

在步骤（L2，L3，L4）中，算法L对于每个排列近似地使用$(e, 2+e/2, 2e+2e^{-1}-4)$个存储单元，总的费用为$3.5e + 2e^{-1} - 2 \approx 8.25$（参见习题5）。

107

当k接近于n时，通过使这个代码流水线化，对于这一情况算法X可加以调节。但如同习题1中所示，算法L也能这样做。

49. 对签名进行排序使得$|s_0| \geqslant \cdots \geqslant |s_9|$；也准备表$w_0\cdots w_9$，$x_0\cdots x_9$，$y_0\cdots y_9$使得签名$\{s_k\cdots s_9\}$是$w_{x_k} \leqslant \cdots \leqslant w_{y_k}$。例如，当SEND + MORE = MONEY时，我们对于分别的字母（M, S, O, E, N, R, D, Y, A, B）有$(S_0, \cdots, S_9) = (-9000, 1000, -900, 91, -90, 10, 1, -1, 0, 0)$；还有$(w_0, \cdots, w_9) = (-9000, -900, -90, -1, 0, 0, 1, 10, 91, 1000)$，以及$x_0\cdots x_9 = 0112233344$，$y_0\cdots y_9 = 9988776554$。如果对应于$w_j$的数字不能为零，则另一个表$f_0\cdots f_9$还有$f_j = 1$；在此情况下$f_0\cdots f_9 = 1000000001$。使用习题25

的方法，这些表使得对于剩下数字的所有选择$a_k \cdots a_9$来计算$s_k a_k + \cdots + s_9 a_9$的最大值和最小值就变得容易了，因为链接$l_j$告诉我们在递增顺序下的那些数字。

这个方法要求在查找树的每个节点处有一个稍微昂贵的计算，但它通常在保持该树是小的方面是成功的。例如，它在求解习题24的8个字母算术时，费用仅为7, 13, 7, 9, 5, 343, 44和89千存储访问。在情况(a)、(b)、(e)和(h)中这是相当大的改进，尽管情况(f)是相当差的。另一个坏情况是答案(27)的“CHILD”例子，其中从左到右需要2947千个存储访问，与从右到左的方法需588个存储访问形成强烈对照。然而从左到右的方法，在BLOOD + SWEAT + TEARS（73对360）和HELLO + WORLD（340对400）上的费用都较好。

50. 如果α是在一个排列群中，则它的幂$\alpha^2, \alpha^3 \cdots$（包括$\alpha^{m-1} = \alpha^-$在内）也是，其中$m$是$\alpha$的阶（它的循环长度的最小公倍数），而且(32)等价于$\alpha^- = \sigma_1 \sigma_2 \cdots \sigma_{n-1}$。

51. 假的。例如$\sigma(k, i)^-$和$\sigma(k, j)^-$两者都可能使$k \mapsto 0$。

52. $\tau(k, j) = (k - j\ k - j + 1)$ 是一个相邻的交换，而且

$$\omega(k) = (n-1 \cdots 0)(n-2 \cdots 0) \cdots (k \cdots 0) = \phi(n-1)\phi(k-1)$$

是一个k触发后边接着一个n触发。对于$0 \leqslant j < n$，对应于算法H中的控制表$c_0 \cdots c_{n-1}$的排列有小于j的c_j个元素在j的右边。所以，除了诸下标被移动了1之外，它和算法P中对应于$c_1 \cdots c_{n-1}$的排列相同。

算法P和算法H的这个版本之间仅有的实质性差别在于，算法P使用一个反射的格雷码来跑遍它的控制表的所有可能性，而算法H则以递增（词典）顺序来跑遍这些混合进制的数。

其实，通过修改算法G或修改算法H，可以以任何西姆斯表来使用格雷码。于是所有转换是通过$\tau(k, j)$或通过$\tau(k, j)^-$，而排列$\omega(k)$是无关紧要的。

53. 在课文中，关于对$n = 4$不能实现$n! - 1$个转置的证明也表明，在单个转置$(n-1\ n-2)$的费用之下我们可以把这个问题从n简化为$n-2$，在该证明的记号下，这个转置被叫做“$(3c)$”。

因此我们通过在步骤H4中做下列转换可以生成所有的排列：如果$k = n-1$或$k = n-2$，转置$a_{j \bmod n} \leftrightarrow a_{(j-1) \bmod n}$，其中$j = c_{n-1} - 1$。如果$k = n-3$或$k = n-4$，转置$a_{n-1} \leftrightarrow a_{n-2}$，并且也转置$a_{j \bmod (n-2)} \leftrightarrow a_{(j-1) \bmod (n-2)}$，其中$j = c_{n-3} - 1$。而且一般地说，对于$1 \leqslant i \leqslant t$，如果$k = n - 2t - 1$或$k = n - 2t - 2$，转置$a_{n-2i+1} \leftrightarrow a_{n-2i}$，也转置 108
$a_{j \bmod (n-2t)} \leftrightarrow a_{(j-1) \bmod (n-2t)}$，其中$j = c_{n-2t-1} - 1$[参见*CACM* 19 (1976), 68-72]。

对应的西姆斯表排列可写出如下，尽管它们在算法中并不明显地出现：

$$\sigma(k, j)^- = \begin{cases} (0\ 1 \cdots j-1\ k), & \text{如果} n-k \text{为奇} \\ (0\ 1 \cdots k)^j, & \text{如果} n-k \text{为偶} \end{cases}$$

在交换之后$a_{j \bmod (n-2t)}$的值将是$n - 2t - 1$。为有效起见，我们也可以使用k通常等于$n-1$的事实，转置的总数为$\sum_{t=0}^{\lfloor n/2 \rfloor} (n-2t)! - \lfloor n/2 \rfloor - 1$。

54. 是的；转换可以是在位置$\{1, \cdots, k\}$上的任何k循环。

55. (a) 当$n>\rho!(m)$时，由于$\rho!(m)=\rho!(m \bmod n!)$，对于$0<m<n\cdot n!=(n+1)!-n!$，我们有$\rho!(n!+m)=\rho!(m)$。因此对于$0\leqslant m<n\cdot n!$，$\beta_{n!+m}=\sigma_{\rho!(n!+m)}\cdots\sigma_{\rho!(n!+1)}\beta_{\rho m!}=\sigma_{\rho!(m)}\cdots\sigma_{\rho!(1)}\beta_{n!}=\beta_m\beta_{n!}$，而且特别地我们有

$$\beta_{(n+1)!}=\sigma_{n+1}\beta_{(n+1)!-1}=\sigma_{n+1}\beta_{n!-1}\beta_{n!}^{n}=\sigma_{n+1}\sigma_n^{-}\beta_{n!}^{n+1}$$

类似地，对于$0\leqslant m<n\cdot n!$，$\sigma_{n!+m}=\beta_{n!}^{-}\alpha_m\beta_{n!}\alpha_{n!}$。

由于$\beta_{n!}$和τ_n及τ_{n+1}可交换，我们发现$\alpha_{n!}=\tau_n\alpha_{n!-1}$，而且

$$\begin{aligned}\alpha_{(n+1)!}=\tau_{n+1}\alpha_{(n+1)!-1}&=\tau_{n+1}\beta_{n!}^{-}\alpha_{(n+1)!-1-n}\beta_{n!}\alpha_{n!}=\cdots\\&=\tau_{n+1}\beta_{n!}^{-n}\alpha_{n!-1}(\beta_{n!}\alpha_{n!})^n\\&=\beta_{n!}^{-n-1}\tau_{n+1}\tau_n^{-}(\beta_{n!}\alpha_{n!})^{n+1}\\&=\beta_{(n+1)!}^{-}\sigma_{n+1}\sigma_n^{-}\tau_{n+1}\tau_n^{-}(\beta_{n!}\alpha_{n!})^{n+1}\end{aligned}$$

(b) 在此情况下$\sigma_{n+1}\sigma_n^{-}=(n\ n-1\cdots1)$和$\tau_{n+1}\tau_n^{-}=(n+1\ n\ 0)$，而且由归纳法我们有$\beta_{(n+1)!}\ \alpha_{(n+1)!}=(n+1\ n\cdots0)$。因此对于$0\leqslant j\leqslant n$和$0\leqslant m<n!$，$\alpha_{jn!+m}=\beta_{n!}^{-j}\alpha_m(n\cdots0)^j$。$\{0,\cdots,n\}$的所有排列都被实现，因为$\beta_{n!}^{-j}\ \alpha_m$固定$n$且$(n\cdots0)^j$使$n\mapsto n-j$。

56. 如果在上道题中那样，我们置$\sigma_k=(k-1\ k-2)\ (k-3\ k-4)\cdots$，由归纳法我们发现$\beta_{n!}\ \alpha_{n!}$是$(n+1)$循环$(0\ n\ n-1\ n-3\cdots$ (2或1) (1或2)$\cdots n-4\ n-2)$。

57. 如同在答案5中那样论证，我们得到$\sum_{k=2}^{n-1}[k\text{奇数}]/k!-(\lfloor n/2\rfloor-1)/n!=\sinh 1-1-O(1/(n-1)!)$。

58. 真的。由习题55的公式我们有$\alpha_{n!-1}=(0\ n)\ \beta_{n!}^{-}(n\cdots0)$，而且这使$0\mapsto n-1$，因为$\beta_{n!}$固定$n$。（因此，算法E将在习题66的图上定义一个哈密顿循环，当且仅当$\beta_{n!}=(n-1\cdots2\ 1)$，而且这成立当且仅当$\beta_{(n-1)!}$的每个循环的长度是$n$的一个因子。对于$n=2,3,4,6,12,20$和40，后者为真，但对于其他$n\leqslant250\,000$则不成立。）

59. 在正文的定义中带有生成元（$\alpha_1,\cdots,\alpha_k$）的凯利图同构于在另外的定义中带有生成元（$\alpha_1^{-},\cdots,\alpha_k^{-}$）的凯利图。因为在前者中$\pi\rightarrow\alpha_j\pi$，当且仅当在后者中$\pi^{-}\rightarrow\pi^{-}\alpha_j^{-}$。

60. 有88个δ序列，它们归结为4类：$P=(32131231)^3$（由8个不同的δ序列表示的平易改动）；$Q=(32121232)^3$（具有8个表示的平易改动的一个双重格雷变形）；$R=(121232321232)^2$（具有24个表示的一个双重格雷码）；$S=2\alpha3\alpha^R$，$\alpha=12321312121$（48个表示）。类P和Q是它们的补的循环移动；类P、Q和S是它们的颠倒的移动；类R是它的补的一个移动的颠倒。[参见阿·林·利·西尔弗（A. L. Leigh Silver），*Math. Gazette* **48** (1964)，1-16。]

109

61. 分别有（26, 36, 20, 26, 28, 40, 40, 20, 26, 28, 28, 26）这样的通路，并在（1243, 1324, 1432, 2134, 2341, 2413, 3142, 3214, 3421, 4123, 4231, 4312）处结束。

62. 当$n=3$时仅有两条通路，分别在132和213处结束。但当$n\geqslant4$时有从$12\cdots n$引到任何奇排列$(a_1a_2\cdots a_n)$的格雷码。当$n=4$时习题61确立这一点，而且当$n>4$时，我们可以用归纳法如下证明它。

令$A(j)$是以j开始的所有排列的集合，并令$A(j,k)$是以j_k开始的那些。如果

$(\alpha_0, \alpha_1, \cdots, \alpha_n)$ 是使得$\alpha_j \in A(x_j, x_{j+1})$ 的任何奇排列，则$(12)\alpha_j$是在$A(x_{j+1}, x_j)$ 中的一个偶排列。结果，如果$x_1x_2\cdots x_n$是$\{1, 2, \cdots, n\}$的一个排列，则至少有一条形如

$$(12)\ \alpha_0 — \cdots — \alpha_1 — (12)\alpha_1 — \cdots — \alpha_2 — \cdots — (12)\alpha_{n-1} — \cdots — \alpha_n$$

的哈密顿通路，从$(12)\alpha_{j-1}$到α_j的子通路包括$A(x_j)$ 的所有元素。

当$\alpha_1 \neq 1$时，这个构造至少以 $(n-2)!^n/2^{n-1}$种不同的方法解这个问题，因为我们可以取$\alpha_0 = 2\ 1\cdots n$和$\alpha_n = a_1a_2\cdots a_n$；有 $(n-2)!$种方法来选择$x_2\cdots x_{n-1}$和 $(n-2)!/2$种方法来选择每一个$\alpha_1, \cdots, \alpha_{n-1}$。

最后，如果$a_1 = 1$，取任何通路$12\cdots n — \cdots — a_1a_2\cdots a_n$，它跑遍所有$A(1)$，而且对于某些$j \neq j'$，及$\alpha \in A(1, j)$ 和$\alpha' \in A(1, j')$，选择任何步骤$\alpha — \alpha'$。使用类似于上面的哈密顿通路，但现在有$\alpha_1 = \alpha$，$\alpha_n = (12)\alpha'$，$x_1 = 1$，$x_2 = j$，$x_n = j'$和$x_{n+1} = 1$的一个构造，以

$$\alpha — (12)\alpha_1 — \cdots — \alpha_2 — \cdots — (12)\ \alpha_{n-1} — \cdots — \alpha_n — \alpha'$$

来代替该步骤。（在此情况下，排列$\alpha_1, \cdots, \alpha_n$可能全为偶的。）

63. 使用7.2.3节的技术的蒙特卡罗估计提示，等价类的总数将粗略地为1.2×10^{21}；这些类大多数将包含480个格雷循环。

64. 精确地说，有2 005 200个δ序列有双重格雷性质。在循环移动、颠倒、和/或取补之下，它们属于4206个等价类。9个类，例如码$2\alpha2\alpha^R$，其中

$$\alpha = 123432343212321212323212321212343432121234321234321212321232321$$

是它们颠倒的移动；48个类由重复的60个循环组成。后一种类型中最有趣之一是$\alpha\alpha$，其中

$$\alpha = \beta2\beta4\beta4\beta4\beta4,\quad \beta = 32121232123$$

65. 对于任何给定的$N < n!$，这样一条通路存在：令第N个排列是$\alpha = a_1\cdots a_n$，并令$j = a_1$，还令Π_k是所有排列$\beta = b_1\cdots b_n$的集合，对于它$b_1 = k$且$\beta < \alpha$。通过对N的归纳法，对于Π_j来说存在一条格雷通路P_1。对于$2 < k < j$，我们可以构造对于$\Pi_j \cup \Pi_1 \cup \cdots \cup \Pi_{k-1}$的一条格雷通路$P_k$，并逐次地把$P_{k-1}$同对于$\Pi_{k-1}$的一个格雷循环组合在一起。（参见答案62的一个“吸收”构造，事实上，当N是6的一个倍数时，P_j将是一个格雷循环。）

66. 通过规则$\pi_{(k+1) \bmod n!} = (1\delta_k)\ \pi_k$定义$\delta$序列，我们发现恰有36个这样的序列，所有这些序列都是像 $(xyzyzyxzyzyz)^2$这样模式的循环移动。（下一情况$n = 5$，大概有10^{18}个解，相对于循环移动、颠倒以及坐标的排列来说，它们是不等价的。因此大约有6×10^{21}个不同的δ序列。）顺便指出，雨果·帕克（Igor Pak）已经证明，通过星形转置生成的凯利图一般是一个（$n-2$）维圆环体。 110

67. 如果令π等价于$\pi(12345)$，我们得到有40768个哈密顿循环的24个顶点上的一个既约图。这些哈密顿循环中的240个导致形如α^5的δ序列，其中α使用每个转置6次（例如$\alpha = 3542325342345324543524 52$）。这个问题的解的总数大概是$10^{16}$。

68. 如果A不连通，则G也不连通。如果A连通，则可以假设它是一个自由树。而且，在此情况下我们可以证明习题62中结果的一个推广。对于$n \geq 4$，有从恒等排列到任何奇排列的G中的一条哈密顿通路。因为不失一般性，我们可以假设A包含边1—2，其中1是树的叶，因而就可应用类似习题62的一个证明。

[这个优雅的构造是由莫·特淳特（M. Tchuente）给出的，见*Ars Combinatoria* **14** (1982), 115-122。拉斯基和萨维吉在（*SIAM J. Discrete Math.* **6** (1993), 152-166）中对广泛的推广做了讨论。也可参见在（*Kibernetika* 11, 3 (1975), 17-25）中俄文的原始论文。英译稿见（*Cybernetics* 11 (1975), 362-366）。]

69. 遵循题目中的提示，当$n = 5$时修改的算法就像以下那样动作:

1234	1243	1423	4123	4132	1432	1342	1324	3124	3142	3412	4312
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑
54321	24351	24153	54123	14523	14325	24315	24513	54213	14253	14352	54312
12345	15342	35142	32145	32541	52341	51342	31542	31245	35241	25341	21345
15342	12435	32415	35412←31452		51432	52431	32451←35421		31425	21435	25431
23451	53421	51423	21453→25413		23415	13425	15423→12453		52413	53412	13452
21543	51243	53241	23541	23145	25143	15243	13245	13542	53142	52143	12543
34512	34215	14235	14532	54132	34152	34251	54231	24531	24135	34125	34521
32154→35124		15324→12354		52314	32514←31524		51324	21354→25314		35214→31254	
45123←42153		42351←45321		41325	41523→42513		42315	45312←41352		41253←45213	
43215	43512←41532		41235	45231→43251		43152→45132		42135	42531←43521		43125
51234	21534→23514		53214	13254←15234		25134←23154		53124	13524→12534		52134
↓	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓	↑

这里列表示以所有$2n$种方式循环地转动和/或反射的排列集合；因此每列恰含一个“玫瑰花园排列”（见习题18）。我们可以使用算法P来系统地跑遍玫瑰花园排列，并且知道对偶xy将出现在它的列中的对偶yx之前，在那时τ'而不是ρ'将把我们向右或向左移动。步骤Z2省略交换$a_1 \leftrightarrow a_2$，由此引起排列$a_1 \cdots a_{n-1}$向后进行重复它们自己。（我们含蓄地使用在算法T的输出中$t[k] = t[n! - k]$这一事实。）

现在如果以24…31代替1…n，而且把$A_1 \cdots A_n$改成$A_1 A_n A_2 A_{n-1} \cdots$，我们得到其结果示于图22b中的未修改的算法。

这个方法是受由埃·斯·拉坡博特（E. S. Rapoport）提出的一个（非构造性）定理（*Scripta Math.* **24** (1959)，51-58）的启示给出的。它说明了由卡尔拉·萨维吉在1989年发现的更为一般的事实，就是通过3个卷积ρ, σ, τ生成的任何群的凯利图，当$\rho\tau = \tau\rho$时有一个哈密顿循环。[参见雨·帕克和拉·拉多依希克（R. Radoičić），“*Hamiltonin path in a Cayley graph*”，待发表。]

70. 否；在这个有向图中最长的循环的长度为358。但确实存在不相交的180个
111 循环的对偶，从中可以推导出长度为720的一条哈密顿通路。例如考虑循环$\alpha\sigma\beta\sigma$和$\gamma\sigma\sigma$，其中

$$\alpha = \tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3$$
$$\tau\sigma^2\tau\sigma^1\tau\sigma^1$$
$$\beta = \sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3$$
$$\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^4$$

$$\gamma = \sigma\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^3\tau\sigma^2\tau\sigma^1\tau\sigma^2\tau\sigma^3$$
$$\tau\sigma^1\tau\sigma^1\tau\sigma^3\tau\sigma^2\tau\sigma^5\tau\sigma^5\tau\sigma^5\tau\sigma^3\tau\sigma^5\tau\sigma^2\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^1\tau\sigma^5\tau\sigma^1\tau\sigma^3$$
$$\tau\sigma^3\tau\sigma^5\tau\sigma^5\tau\sigma^1\tau\sigma^5\tau\sigma^2\tau\sigma^3\tau\sigma^1\tau\sigma^2$$

如果我们以134526开始并且沿着$\alpha\sigma\beta\tau$到达163452，然后沿着$\gamma\sigma\tau$并到达126345，然后沿着$\sigma\gamma\tau$并到达152634，然后沿着$\beta\sigma\alpha$并在415263处结束。

71. 当$n = 7$、9和11时，通过计算机，布伦丹·麦凯（Brendan Mckay）和弗朗克·拉斯基已经发现这样的一些循环，但没有明显的优美结构。

72. 任何哈密顿通路包括使$y \mapsto x$的$(n-1)!$个顶点，这些顶点的每一个（如果不是最后一个）后边跟着使$x \mapsto x$的顶点。所以1必定是最后的；否则$(n-1)! + 1$个顶点将使$x \mapsto x$。

73. (a) 首先假设β是恒等排列( )。则包含A的一个元素的α的每个循环完全处于A之内。因此σ的循环是由省略不含A的元素的α的所有循环得到的。σ所有剩下的循环都有奇长度，所以σ是一个偶排列。

如果β不是恒等排列，我们可应用这个论证到$\alpha' = \alpha\beta^-$、$\beta' = (\,)$以及$\sigma' = \sigma\beta^-$，并且得出结论σ'是一个偶排列；因此σ和β有相同的符号。

类似地，σ和α有相同的符号，因为$\beta\alpha^- = (\alpha\beta^-)^-$有和$\alpha\beta^-$相同的阶。

(b) 令X是在定理R中的凯利图的顶点，并令α是使一个顶点π变成$\alpha\pi$的排列；这个排列有长度a的g/a个循环。类似地定义排列β，然后$\alpha\beta^-$有长度c的g/c个循环。如果c为奇数，则在这个图中的任何哈密顿循环定义一个循环σ，它已含所有顶点并且满足(a)的假设。因此α和β有奇数个循环，因为有r个循环的n个元素上的一个排列的符号是$(-1)^{n-r}$（参见习题5.2.2-2）。

[这个证明表明X不能是任何奇数个循环的并，它是由兰金给出的（*Proc. Cambridge Phil. Soc.* **62** (1966), 15-16）。]

74. 如果我们要求$0 < j < g/c$和$0 < k < c$，则表示$\beta^j\gamma^k$是惟一的。因为如果对于满足$0 < j < g/c$的某个j我们有$\beta^j = \gamma^k$，则这个图将至多有jc个元素。由此得出，对于某个t，$\beta^{g/c} = \gamma^t$。

如同在上一题的答案中那样，令σ为一个哈密顿循环，如果$\pi\sigma = \pi\alpha$，则$\pi\gamma\sigma$必定是$\pi\gamma\alpha$。因为$\pi\gamma\beta = \pi\alpha$。而且如果$\pi\sigma = \pi\beta$，则$\pi\gamma\sigma$不可能是$\pi\gamma\alpha$，因为这将意味着$\pi\gamma^c\sigma = \pi\gamma^c\alpha$，因此元素$\pi\gamma^k$相对于它们在$\sigma$中的后继会有等价的行为。

注意如果$j > 0$，则有一个$k < j$使得$\pi\sigma^j = \pi\alpha^k\beta^{j-k} = \pi\beta^j\gamma^k$，因此$\pi\sigma^{g/c} = \pi\gamma^{t+k}$等价于一个$\pi$，而且相同的行为将重复。我们在$g$步中头一次返回到$\pi$，当且仅当$t + k$同$c$互素。

75. 对于$g = mn$, $a = m$, $b = n$, $c = mn/d$应用上一道题。数t满足$t \equiv 0 (\text{modulo } m)$, $t + d \equiv 0 (\text{modulo } n)$，而且由此得出$k + t \perp c$，当且仅当$(d-k)m/d \perp kn/d$。

注意：习题7.2.1.1-78的取模的格雷码是从$(0, 0)$到$(m-1, (-m) \bmod n)$的一条哈密顿通路，所以它是一个哈密顿循环，当且仅当m是n的一个倍数。自然猜测（但是错的）每当$d > 1$时，至少存在一个哈密顿循环。但是鲍·厄尔多斯（P.

112 Erdös）和威·托·特洛特尔（W. T. Trotter）已经发现[*J. Graph Theory* 2 (1978), 137-142]，如果p和$2p+1$是奇素数，则当$m = p(2p+1)(3p+1)$且$n = (3p+1)\prod_{q=1}^{3p} q^{[q\text{为素数}][q\neq p][q\neq 2p+1]}$时，没有适当的$k$存在。

关于在$O_m \times O_n$中其他类型的循环的有趣事实，参见约·安·加里安（J. A. Gallian），*Mathematical Intelligencer* 13, 3 (1991年夏)，40-43。

76. 我们可以假设旅游线路在左下角开始。当m和n都为3所整除时无解，因为在此情况下有2/3个单元是不可到达的。否则，令$d = \gcd(m, n)$并且如同在上题中那样进行论证，但通过$(x, y)\alpha = ((x+2) \bmod m, (y+1) \bmod n)$以及$(x, y)\beta = ((x+1) \bmod m, (y+2) \bmod n)$进行，我们发现答案

$$\sum_{k=0}^{d}\binom{d}{k}[\gcd(2d-k)m,(k+d)n) = d \text{ 或 } (mn\perp 3 \text{ 和 } \gcd((2d-k)m,(k+d)n) = 3d)]$$

77.
```
01  * Permutation generator \'a la Heap
02  N      IS     10              n的值（3或更多，但不大）
03  t      IS     $255
04  j      IS     $0              8j
05  k      IS     $1              8k
06  ak     IS     $2
07  aj     IS     $3
08         LOC    Data_Segment
09  a      GREG   @               对于a_0...a_{n-1}的基地址
10  A0     IS     @
11  A1     IS     @+8
12  A2     IS     @+16
13         LOC    @+8*N           为a_0...a_{n-1}提供的空间
14  c      GREG   @-8*3           8c_0的单元
15         LOC    @-8*3+8*N       8c_3...8c_{n-1}，开始时为零
16         OCTA   -1              8c_n = -1，一个方便的哨兵
17  u      GREG   0               a_0的内容，在内循环中除外
18  v      GREG   0               a_1的内容，在内循环中除外
19  w      GREG   0               a_2的内容，在内循环中除外
20         LOC    #100
21  1H     STCO   0,c,k           B-A     c_k←0
22         INCL   k,8             B-A     k←k+1
23  0H     LDO    j,c,k           B       j←c_k
24         CMP    t,j,k           B
25         BZ     t,1B            B       如果c_k = k则循环
26         BN     j,Done          A       如果c_k<0（k = n）则结束
27         LDO    ak,a,k          A-1     取a_k
28         ADD    t,j,8           A-1
29         STO    t,c,k           A-1     c_k←j+1
30         AND    t,k,#8          A-1
31         CSZ    j,t,0           A-1     如果k为偶数则置j←0
32         LDO    aj,a,j          A-1     取a_j
33         STO    ak,a,j          A-1     用a_k代替它
34         CSZ    u,j,ak          A-1     如果j = 0则置u←a_k
35         SUB    j,j,8           A-1     j←j-1
36         CSZ    v,j,ak          A-1     如果j = 0则置v←a_k
```

113

```
37         SUB    j,j,8        A-1    j←j-1
38         CSZ    w,j,ak       A-1    如果j=0则置w←a_k
39         STO    aj,a,k       A-1    用a_j以前的值代替a_k
40  Inner  PUSHJ  0,Visit      A
           ...                        (参见(42))
55         PUSHJ  0,Visit      A
56         SET    t,u          A      交换u↔w
57         SET    u,w          A
58         SET    w,t          A
59         SET    k,8*3        A      k←3
60         JMP    OB           A
61  Main   LDO    u,A0         1
62         LDO    v,A1         1
63         LDO    w,A2         1
64         JMP    Inner        1      ∎
```

78. 行31～38变成$2r-1$个指令，行61～63变成r个指令，而行56～58变成$3+(r-2)$[r为偶数]个指令（参见答案40中的$\omega(r-1)$）。因此总的运行时间是$((2r!+2)A+2B+r-5)\mu+((2r!+2r+7+(r-2)[r\text{为偶数}])A+7B-r-4)\nu$，其中$A=n!/r!$且$B=n!(1/r!+\cdots+1/n!)$。

79. `SLU u,[#f],t; SLU t,a,4; XOR t,t,a; AND t,t,u; SRU u,t,4; OR t,t,u; XOR a,a,t;` 这里，如同在习题1.3.1'-34的答案中那样，记号‘`[#f]`’表示包含常数值`#f`的一个寄存器。

80. `SLU u,a,t; MXOR u,[#8844221188442211],u; AND u,u,[#ff000000]; SRU u,u,t; XOR a,a,u`。这是欺骗，因为当$t=4$时它把`#12345678`变换成`#13245678`，但(45)仍然有效。

甚至更快和更有技巧的将是类似于(42)的一个例程。考虑

```
PUSHJ 0,Visit; MXOR a,a,c1; PUSHJ 0,
Visit; ··· MXOR a,a,c5; PUSHJ 0,Visit
```

其中$c_1,\cdots,c_5$是将引起`#12345678`逐步地变成`#12783456`，`#12567834`，`#12563478`，`#12785634`，`#12347856`的常数。其他仅执行通常的1/6或1/24时间的指令，可以负责在字节内和字节之间移动四个连续二进字，但就PUSHJ/POP的开销而言，它赢不过(46)。

```
81. t  IS   $255 ;k IS $0 ;kk IS $1 ;c IS $2 ;d IS $3
       SET  k,1      k←1
    3H SRU  d,a,60   d←最左的四个连续二进字
       SLU  a,a,4    a←16a mod 16^16
       CMP  c,d,k
       SLU  kk,k,2
       SLU  d,d,kk
       OR   t,t,d    t←t+16^k d
       PBNZ c,1B     如果d≠k，则返回主循环
       INCL k,1      k←k+1
       PBNZ a,3B     如果k<n，则返回第二个循环  ∎
```

82. $\mu+(5n!+11A-(n-1)!+6)\nu=((5+10/n)\nu+O(n^{-2}))n!$，加上访问时间，

其中$A = \sum_{k=1}^{n-1} k!$是在3H处使用循环的次数。

83 通过适当的初始化和一个13个四字（奥克塔）字节的表，大约只需要十余条MMIX指令：

114

```
magic  GREG #8844221188442211
0H     ⟨Visit register a⟩
       PBN  c,Sigma
Tau    MXOR t,magic,a; ANDNL t,#ffff; JMP 1F
Sigma  SRU  t,a,20; SLU a,a,4; ANDNML a,#f00
1H     XOR  a,a,t; SLU c,c,1
2H     PBNZ c,0B; INCL p,8
3H     LDOU c,p,0; PBNZ c,0B                    ∎
```

84. 假设所有处理器实际上速度相同，使用基于控制表的任何方法，对于$(k-1)n!/p \leq r < k\ n!/p$，我们可以令第$k$个处理器生成秩为$r$的所有排列。通过把它们的秩转换成混合进制记号（见习题12），容易计算起始和结束的控制表。

85. 我们可以使用类似于算法3.4.2P那样的技术：为计算$k = r(\alpha)$，首先对于$1 \leq j \leq n$置$a'_{a_j} \leftarrow j$（逆排列）。然后置$k \leftarrow 0$，并且对于$j = n, n-1, \cdots, 2$（以此顺序），置$t \leftarrow a'_j$, $k \leftarrow kj + t - 1$, $a_t \leftarrow a_j$, $a'_{a_j} \leftarrow t$。为计算$r^{[-1]}(k)$，由$a_1 \leftarrow 1$开始。然后对于$j = 2, \cdots, n-1, n$（以此顺序）置$t \leftarrow (k \bmod j) + 1$, $a_j \leftarrow a_t$, $a_t \leftarrow j$, $k \leftarrow \lfloor k/j \rfloor$。[参见斯·普利斯泽仁斯基（S. Pleszczyński），*Inf. Proc. Letters* 3 (1975)，180-183；温·米尔沃德（W. Myrvold）和弗·拉斯基，*Inf. Proc. Letters* 79 (2001)，281-284。]

如果我们仅仅要对$\{1, \cdots, n\}$的n^m个变形$a_1, \cdots, a_m$计秩和不计秩，另一个方法是更可取的。为计算$k = r(a_1, \cdots, a_m)$，以$b_1 \cdots b_n \leftarrow b'_1 \cdots b'_n \leftarrow 1 \cdots n$开始；然后对于$j = 1, \cdots, m$（以此顺序），置$t \leftarrow b'_{a_j}$，$b_t \leftarrow b_{n+1-j}$和$b'_{b_t} \leftarrow t$。最后置$k \leftarrow 0$，以及对于$j = m, \cdots, 1$（以此顺序）置$k \leftarrow k \times (n+1-j) + b'_{a_j} - 1$。为计算$r^{[-1]}(k)$，由$b_1 \cdots b_n \leftarrow 1 \cdots n$开始，而且对于$j = 1, \cdots, m$（以此顺序）置$t \leftarrow (k \bmod (n+1-j)) + 1$, $a_j \leftarrow b_t$, $b_t \leftarrow b_{n+1-j}$, $k \leftarrow \lfloor k/(n+1-j) \rfloor$。[对于很大的$n$和小的$m$的情况，参见习题3.4.2-15。]

86. 如果$x \prec y$和$y \prec z$，则此算法将绝不把y移动到x的左边，也绝不把z移动到y的左边，所以它绝不对x和z进行测试。

87. 它们以词典顺序出现；算法P使用一个反射的格雷顺序。

88. 对于$a'_0 < a'_1 < a'_2$, $a'_3 < a'_4 < a'_5$, $a'_6 < a'_7$, $a'_8 < a'_9$, $a'_0 < a'_3$, $a'_6 < a'_8$生成反演排列。

89. (a) 令$d_k = \max\{j | 0 \leq j \leq k$且$j$是非平凡的$\}$，其中把0当作是非平凡的。这个表容易预先计算，因为j是非平凡的当且仅当它必定跟随$\{1, \cdots, j-1\}$。在步骤V2中置$k \leftarrow d_n$并在步骤V5中置$k \leftarrow d_{k-1}$。（假设$d_n > 0$。）

(b) 现在$M = \sum_{j=1}^{n} t_j$ [j是非平凡的]。

(c) 至少有两个集合$\{j, \cdots, k\}$的拓扑排序$a_j \cdots a_k$，而且它们中的任何一个都可以被放在$\{1, \cdots, j-1\}$的任何拓扑排序$a_1 \cdots a_{j-1}$之后。

(d) 算法2.2.3T重复地输出极小元素（没有前驱的元素），并从关系图中删去它们。我们颠倒过来使用它们，并且重复地删去，且把最高标号给予极大元素（没有后继的元素）。如果只有一个极大元素存在，则它是平凡的。如果k和l都是极大

的，则它们在满足$x \prec k$或$x \prec l$的任何元素x之前被输出，因为步骤T5和T7在一个队列（非一个栈）中保持极大元素。因此如果k是非平凡的而且首先输出，则元素l可能变成平凡的，但下一个非平凡元素j将不在l之前输出；因此k与l无关。

(e) 令非平凡的t个数是$s_1 < s_2 < \cdots < s_r = N$。于是由(c)我们有$s_j \geqslant 2s_{j-2}$。结果

$$M = s_2 + \cdots + s_r \leqslant s_r\left(1+\frac{1}{2}+\frac{1}{4}+\cdots\right) + s_{r-1}\left(1+\frac{1}{2}+\frac{1}{4}+\cdots\right) < 4s_r。$$

115

（如同马·皮克扎斯基（M. Peczarski）所发现的那样，一个更精确的估计事实上是真的。令$s_0 = 1$，令非平凡的下标是$0 = k_1 < k_2 < \cdots < k_r$，并且对于$j \geqslant 1$，令$k_j' = \max\{k \mid 1 \leqslant k < k_j, k \not\prec k_j\}$。于是$k_j' \geqslant k_{j-1}$，以$k_{j+1}$结束的$\{1, \cdots, k_{j+1}\}$的拓扑排序有$s_j$个；而且至少有$s_{j-1}$个以$k_{j+1}'$结束，因为$s_{j-1}$个$\{1, \cdots, k_j - 1\}$的拓扑排序的每一个都可加以扩充。因此对于$1 \leqslant j < r$

$$s_{j+1} \geqslant s_j + s_{j-1}$$

现在令$y_0 = 0$，$y_1 = F_2 + \cdots + F_r$，以及对于$1 < j < r$，$y_j = y_{j-2} + y_{j-1} - F_{r+1}$。于是

$$F_{r+1}(s_1 + \cdots + s_r) + \sum_{j=1}^{r-1} y_j(s_{r+1-j} - s_{r-j} - s_{r-1-j}) = (F_2 + \cdots + F_{r+1})s_r$$

而且每一个$y_j = F_{r+1} - 2F_j + (-1)^j F_{r+1-j}$是非负的。因此$s_1 + \cdots + s_r \leqslant ((F_2 + \cdots + F_{r+1})/F_{r+1})s_r \approx 2.6s_r$。下面的习题表明这个界是最好可能的。）

90. 由习题5.2.1-25，这样的排列的个数N为F_{n+1}，因此$M = F_{n+1} + \cdots + F_2 = F_{n+3} - 2 \approx \phi^2 N$。顺便说明，所有这样的排列满足$a_1 \cdots a_n = a_1' \cdots a_n'$。可把它们安排成一条格雷通路（习题7.2.1.1-89）。

91. 由于$t_j = (j-1)(j-3)\cdots(2或1)$，我们发现$M = (1 + 2/\sqrt{\pi n} + O(1/n))N$。

注意：对于满足(49)的排列的反演表$c_1 \cdots c_{2n}$，可通过条件$c_1 = 0$，$0 \leqslant c_{2k} \leqslant c_{2k-1}$，$0 \leqslant c_{2k+1} \leqslant c_{2k-1} + 1$来加以表征。

92. 对偶（R，S）的总数等于P_n乘以拓扑排序预期的个数，这里R是一个偏序，而S是包括R在内的一个线性序；它也是Q_n乘以$n!$，因此答案为$n!Q_n/P_n$。

在7.2.3节我们将讨论P_n和Q_n的计算，对于$1 \leqslant n \leqslant 12$预期值结果近似地是

$$(1, 1.33, 2.21, 4.38, 10.1, 26.7, 79.3, 262, 950, 3760, 16200, 74800)$$

当$n \to \infty$时的近似值已由布赖特威尔（Brightwell）、普罗梅尔（Prömel）和史蒂吉尔（Steger）推导出来[*J. Combinatorial Theory* **A 73** (1996), 193-206]。但极限特性十分不同于当n是在一个实用范围中时的情况。当$n \leqslant 5$时，Q_n的值首先由舍·帕·艾万（S. P. Avann）所确定[*Æquationes Math.* **8** (1972), 95-102]。

93. 对于$1 \leqslant j \leqslant n$，基本思想是引进满足$j \prec n+1$和$j \prec n+2$的虚拟元素$n+1$和$n+2$，并且通过相邻交换找出这样一个扩充关系的所有拓扑排序；然后取每第二个排列；抑制虚拟元素。可以使用类似于算法V的一个算法。但是以所有可能的方法来把$n-1$和n插入$a_1 \cdots a_{n-2}$中，而把n减小成$n-2$这样一个递归来进行，并假定$n-1 \not\prec n$，偶尔地把$n+1$和$n+2$加以交换。[参见加·普鲁伊斯（G. Pruesse）和

弗·拉斯基，*SICOMP* **23** (1994)，373-386；坎菲尔德（Canfield）和威廉森已经描述了一个无循环的实现，*Order* **12** (1995)，57-75。]

94. $n = 3$的情况说明了以$1\cdots(2n)$ 开始和以$1(2n)\ 2(2n-1)\cdots n(n+1)$ 结束的一个模式的一般思想：123456，123546，123645，132645，132546，132456，142356，142536，142635，152634，152436，152346，162345，162435，162534。

匹配也可被认为是有n个循环的$\{1, \cdots, 2n\}$的卷积，通过这个表示这个模式在每步中涉及两个转置。

注意：刚才列出的排列的C反演表分别为000000，000100，000200，010200，010100，010000，020000，020100，020200，030200，030100，030000，040000，
116 040100，040200。一般地说，$C_1 = C_3 = \cdots = C_{2n-1} = 0$，而且在进制 $(2n-1, 2n-3, \cdots, 1)$ 上n元组 $(C_2, C_4, \cdots, C_{2n})$ 跑遍一个反射的格雷码。因此如果愿意，可以容易地把这个生成过程作为无循环的。[参见蒂莫西·沃尔斯（Timothy Walsh），*J. Combinatorial Math. and Combinatorial Computing* **36** (2001)，95-118，第一节。]

注意：生成所有匹配的算法可追溯到约·费·普发芙（J. F. Pfaff）[*Abhandlungen Akad. Wissenschaften* (Berlin:1814-1815)，124-125]。他描述了两个这样的过程：头一个方法是词典顺序的，它也对应于C反演表的词典顺序。他的第二个方法对应于那些表的协词典顺序。在两种情况下偶和奇排列交替出现。

95. 用算法V对于$a_1' < a_n' > a_2' < a_{n-1}' > \cdots$生成反演排列（关于解的个数参见习题5.1.4-23）。

96. 例如，我们可以以$a_1\cdots a_{n-1}a_n = 2\cdots n\ 1$和$b_1b_2\cdots b_nb_{n+1} = 1\ 2\cdots n\ 1$开始，并且使用算法P来生成$\{2, \cdots, n\}$的 $(n-1)!$个排列$b_2\cdots b_n$。在该算法刚刚交换$b_i \leftrightarrow b_{i+1}$之后，我们置$a_{b_{i-1}} \leftarrow b_i$，$a_{b_i} \leftarrow b_{i+1}$，$a_{b_{i+1}} \leftarrow b_{i+2}$并且访问$a_1\cdots a_n$。

97. 对于$t_k(a_1, \cdots, a_k) = \text{`}a_k \neq k\text{'}$ 使用算法X。

98. 使用习题47的记号，通过容斥的方法（习题1.3.3-26）我们有 $N_k = \Sigma\binom{k}{j}(-1)^j (n-j)^{\underline{k-j}}$。如果$k = O(\log n)$，则$N_{n-k} = (n!e^{-1}/k!)(1 + O(\log n)^2/n)$；因此$A/n! \approx (e-1)/e$且$B/n! \approx 1$。在答案48的假设之下，存储访问的数目，因此是 $\approx A + B + 3A + B - N_n + 3A \approx n!(9 - 8/e) \approx 6.06n!$，大约是每个去排列用16.5个。[一个类似的方法参见塞·乔·阿克尔（S. G. Akl），*BIT* **20** (1980)，2-7。]

99. 假设L_n生成以 $(1\ 2\cdots n)$ 开始而后是 $(2\ 1\cdots n)$ 并以 $(1\cdots n-1)$ 结尾的$D_n \cup D_{n-1}$；例如，$L_3 = (123)$，(213)，(12)。于是我们可以生成D_{n+1}作为$K_{nn}, \cdots K_{n2}, K_{n1}$，其中$K_{nk} = (1\ 2\cdots n)^{-k}(n\ n+1)L_n(1\ 2\cdots n)^k$；例如$D_4$为

$$(1234), (2134), (12)(34), (3124), (1324), (31)(24), (2314), (3214), (23)(14)$$

注意：K_{nk}以循环 $(k+1\cdots n\ 1\cdots k\ n+1)$ 开始并以 $(k+1\cdots n\ 1\cdots k-1)(k\ n+1)$ 结尾，所以通过 $(k-1\ k)$ 的预乘法把我们从K_{nk}带到$K_{n(k-1)}$。还有，通过 $(1\ n)$ 的预乘法将从D_{n+1}的最后元素返回到头一个元素。通过 $(1\ 2\ n+1)$ 的预乘法把我们从

D_{n+1}的最后元素带到 (2 1 3…n)，通过向后沿着D_n的循环，从该元素我们可以返回到 (1 2…n)，由此如所希望的那样完成列表L_{n+1}。

100. 对于$t_k(a_1, \cdots, a_k)$ = '$p>0$或$l[q] \neq k+1$'，使用算法X。

注意：不可分解的排列的个数是$[z^n]\left(1-1/\sum_{k=0}^{\infty} k!z^k\right)$；参见路·康姆特（L. Comtet），*Comptes Rendus Acad. Sci.* **A275** (Paris 1972)，569-572。安·道·金（A. D. King）已经证明，通过在每一步进行单个转置可有效地生成不可分解的排列。事实上，由相邻的转置也可以生成不可分解的排列。例如，当n = 4时，它们是3142，3412，3421，3241，2341，2431，4231，4321，4312，4132，4123，4213，2413。

101. 这里是类似于算法X的一个词典顺序卷积生成元：

Y1. [初始化。] 对于$1 \leqslant k \leqslant n$置$a_k \leftarrow k$和$l_{k-1} \leftarrow k$。然后置$l_n \leftarrow 0$，$k \leftarrow 1$。

Y2. [进入级k。] 如果$k>n$，则访问$a_1 \cdots a_n$并转到Y3。否则置$p \leftarrow l_0$，$u_k \leftarrow p$，$l_0 \leftarrow l_p$，$k \leftarrow k+1$并重复这个步骤。（我们已决定令$a_p = p$。） 117

Y3. [减少k。] 置$k \leftarrow k-1$，而且如果$k=0$则结束。否则置$q \leftarrow u_k$和$p \leftarrow a_q$。如果$p=q$，则置$l_0 \leftarrow q$，$q \leftarrow 0$，$r \leftarrow l_p$和$k \leftarrow k+1$（准备使$a_p > p$）。否则置$l_{u_{k-1}} \leftarrow q$，$r \leftarrow l_q$（准备使$a_p > q$）。

Y4. [增加a_p。] 如果$r=0$则转到Y5。否则置$l_q \leftarrow l_r$，$u_{k-1} \leftarrow q$，$u_k \leftarrow r$，$a_p \leftarrow r$，$a_q \leftarrow q$，$a_r \leftarrow p$，$k \leftarrow k+1$，并转到Y2。

Y5. [恢复a_p。] 置$l_0 \leftarrow p$，$a_p \leftarrow p$，$a_q \leftarrow q$，$k \leftarrow k-1$，并返回Y3。 ∎

令$t_{n+1} = t_n + n\,t_{n-1}$，$a_{n+1} = 1 + a_n + n\,a_{n-1}$，$t_0 = t_1 = 1$，$a_0 = 0$，$a_1 = 1$（参见等式5.1.4-(40)）。对于$k>n$步骤Y2被执行$t_n$次，对于$k \leqslant n$执行$a_n$次。对于$p=q$步骤Y3被执行$a_n$次，一共执行$a_n + t_n$次。步骤Y4被执行$t_n - 1$次；步骤Y5被执行$a_n$次。因此对于所有$t_n$个输出，访问存储单元的总数近似地是$11a_n + 12t_n$，其中$a_n < 1.25331414t_n$（如果速度是必须的则优化显然可能）。

102. 我们构造一个表L_n，它以()开始并以$(n-1\ n)$结束，从L_3 = ()，(1 2)，(1 3)，(2 3)开始。如果n是奇数，则L_{n+1}是$L_n, K_{n1}^R, K_{n2}, \cdots, K_{nn}^R$，其中$K_{nk} = (k \cdots n)^{-}\ L_{n-1}(k \cdots n)(k\ n+1)$。例如

$$L_4 = (), (1\ 2), (1\ 3), (2\ 3), (2\ 3)(1\ 4), (1\ 4), (2\ 4), (1\ 3)(2\ 4), (1\ 2)(3\ 4), (3\ 4)$$

如果n为偶数，则L_{n+1}是$L_n, K_{n(n-1)}, K_{n(n-2)}^R, \cdots, K_{n1}, (1\ n-2)L_{n-1}^R(1\ n-2)(n\ n+1)$。

关于进一步的发展，参见在答案94中所引用的沃尔斯的文章。

103. 对于$1<j<n$，下列由卡尔拉·萨维吉给出的优美的解只须$n-2$个不同的操作ρ_j，其中当j为偶数时，ρ_j以$a_{j+1}a_{j-1}a_j$来代替$a_{j-1}a_ja_{j+1}$；而当j为奇数时，以$a_ja_{j+1}a_{j-1}$来代替$a_{j-1}a_ja_{j+1}$。我们可以假设$n \geqslant 4$；令$A_4 = (\rho_3\rho_2\rho_2\rho_3)^3$。一般地说，$A_n$将以$\rho_{n-1}$开始和结束，而且它将包含$\rho_{n-1}$的$2n-2$个出现。为得到$A_{n+1}$，用$\rho_nA_n'\rho_n$来代替$A_n$的第$k$个$\rho_{n-1}$，其中如果$n$为偶数，$k = 1, 2, 4, \cdots, 2n-2$，如果$n$为奇数，则$k = 1, 3, \cdots, 2n-3, 2n-2$，而且$A_n'$是由$A_n$通过删去它的头一个或最后一个元素得到

的。然后，如果我们以$a_1\cdots a_n = 1\cdots n$开始，则$A_n$的操作$\rho_{n-1}$将使位置$a_n$跑遍逐次的值$n\to p_1\to n\to p_2\to\cdots\to p_{n-1}\to n$。其中$p_1\cdots p_{n-1} = (n-1-[n\text{为偶数}])\cdots 4213\cdots(n-1-[n\text{为奇数}])$；最后的排列将再次是$1\cdots n$。

104. (a) 一个很好地平衡的排列有 $\sum_{k=1}^{n} ka_k = n(n+1)^2/4$个，一个整数。

(b) 当对k求和时以a_k来代替k。

(c) 当n不是很大时，有一个相当快的计数方法，可以以习题16的流水线平易改动算法为基础。因为数量Σka_k对于每个相邻的交换以一个简单的方式改变，而且因为每n步中的$n-1$步是可以快速地完成的“搜寻”。通过只考虑1在2之前的排列，我们可节省一半的工作。对于$1\leqslant n\leqslant 15$的值是0, 0, 0, 2, 6, 0, 184, 936, 6688, 0, 420480, 4298664, 44405142, 0, 6732621476。

105. (a) 对于每个排列$a_1\cdots a_n$，如果$a_j > a_{j+1}$，则在a_j和a_{j+1}间插入$\prec$，如果$a_j < a_{j+1}$，则在它们之间插入$\equiv$或$\prec$。(因此带有k个“递增”的一个排列产生2^k个弱序。弱序有时叫做“优先安排”。习题5.3.1-4表明它们中近似地有$n!/(2(\ln 2)^{n+1})$。弱序的格雷码可以通过在上升处把算法P同格雷二进制码组合一起得到，其中每步改变$\prec\leftrightarrow\equiv$和/或$a_j\leftrightarrow a_{j+1}$。)

(b) 以$a_1\cdots a_n a_{n+1} = 0\cdots 00$和$a_0 = -1$开始。执行算法L直到它以$j = 0$停止为止。找出使$a_1 > \cdots > a_k = a_{k+1}$的$k$，而且如果$k = n$则结束。否则对于$1\leqslant l\leqslant k$置$a_l\leftarrow a_{k+1}+1$，
118 并转到步骤L4。[参见莫·莫尔（M. Mor）和阿·森·弗伦克尔（A. S. Fraenkel），*Discrete Math.* **48** (1984), 101-112。弱顺序序列由下列性质来表征，即如果k出现且$k>0$，则$k-1$也出现。]

106. 通过一序列初等操作$a_i\leftrightarrow a_j$或$a_i\leftarrow a_j$，可以得到所有弱序序列。(也许人们能进一步限制变换，即对于$1\leqslant j<n$，只允许$a_j\leftrightarrow a_{j+1}$或$a_j\leftarrow a_{j+1}$。)

107. 如同希·索·威尔弗所说明的，每一步增加数量 $\sum_{k=1}^{n} 2^k[a_k = k]$，因此这个游戏必然结束。至少这个解的三个方法是似真的。一个坏，一个好，还有一个更好。

坏的方法以所有13!个洗牌和记录最长者来玩这个游戏。这个方法确实产生出正确的答案，但是13! = 6 227 020 800，平均来说游戏持续≈ 8.728步。

好的方法 [安·佩珀迪尼（A. Pepperdine），*Math. Gazette* **73** (1989), 131-133]是倒退来玩，即由最后的位置1*…*开始，其中*表示一张朝下的牌；仅当它的值变成有关系时我们才将这张牌翻开。为了从一个给定位置$a_1\cdots a_n$倒退，考虑使得或者$a_k = k$或者$a_k = *$以及k还未翻开的所有$k>1$。于是倒数第二的位置是21*…*，3*1*…*，…，n*…*1。某些位置（像对于$n = 6$的6**213）没有前驱，尽管我们未曾把所有的牌都翻开。容易系统地考察可能的倒退的游戏的树，因而事实上人们可以说明，带有t个*的节点的数目恰好是$(n-1)!/t!$。因此所考虑节点的总数恰好是$\lfloor(n-1)!e\rfloor$。当$n = 13$时，这是1 302 061 345。

更好的一个方法是向前玩，以初始位置*…*开始而且当牌朝下时翻开顶上一张，随着牌被翻开，跑遍$\{2,\cdots,n\}$的所有$(n-1)!$个排列。如果已知底部的$n-m$张

牌等于 $(m+1)(m+2)\cdots n$（以此顺序），则至多还有$f(m)$个可能的移动；因此如果它不能持续足够长时间成为有趣的游戏，我们就无须继续玩下去。像算法X这样的一个排列生成程序允许我们对于具有相同前缀的所有排列共享计算，并且拒绝不重要的前缀。当被翻转时，在位置j处的牌不必取j的值。当$n=13$时，这个方法只须在 (1, 2, …, 12) 的级别上分别考虑 (1, 11, 940, 6960, 44745, 245083, 1118216, 4112676, 11798207, 26541611, 44380227, 37417359) 个转移，而且只做总数为482 663 902的向前移动。尽管它重复游戏的某些线路，但当$n=13$时对某些无得益转移的早期切除使它比倒退方法快上11倍。

实现长度80的惟一方法是由2 9 4 5 11 12 10 1 8 13 3 6 7开始。

108. 对于其中

$$a_1\cdots a_n\rightarrow a_k a_p(k,2)\cdots a_p(k,k-1)a_1a_{k+1}\cdots a_n$$

的任何游戏，当$a_1=k$时，此结果成立，其中$p(k,2)\cdots p(k,k-1)$是$\{2,\cdots,k-1\}$的任意排列。假设在玩游戏期间，a_1恰好取m个不同的值$d(1)<\cdots<d(m)$；我们将证明至多出现F_{m+1}个排列，包括最初的洗牌在内；当$m=1$时这个断言是明显的。

令$d(j)$是$a_{d(m)}$最初的值，其中$j<m$，并且假设在步骤r时$a_{d(m)}$改变。如果$d(j)=1$，则排列的个数是$r+1<F_m+1<F_{m+1}$。否则$r<F_{m-1}$，而且至多F_m个进一步的排列跟在步骤r之后[*SIAM Review* 19 (1977), 239-241]。

对于$1<n<16$，$f(n)$的值为 (0, 1, 2, 4, 7, 10, 16, 22, 30, 38, 51, 65, 80, 101, 113, 139)，而且它们分别在 (1, 1, 2, 2, 1, 5, 2, 1, 1, 1, 1, 1, 1, 4, 6, 1) 种方式下可实 119
现。对于$n=16$，惟一最长的摇动的排列为

9 12 6 7 2 14 8 1 11 13 5 4 15 16 10 3

109. 答案107的向前的方法提示，$f(n)$大概像$n\log n$这样快地增长（同证券收集相对照）。

110. 对于$0<j<9$，构造位向量$A_j=[a_j\in S_1]\cdots[a_j\in S_m]$和$B_j=[j\in S_1]\cdots[j\in S_m]$。那么使得$A_j=v$的$j$的个数，对于所有位向量$v$来说，必定等于使得$B_k=v$的$k$的个数。而且如果这样，$\{a_j|A_j=v\}$的值应该以所有可能的方法赋给$\{k|B_k=v\}$的排列。

例如，在给定问题中位向量在十六进制记号下为

$$(A_0,\cdots,A_9)=\{9,6,8,\mathrm{b},5,4,0,\mathrm{a},2,0\}$$
$$(B_0,\cdots,B_9)=(5,0,8,6,2,\mathrm{a},4,\mathrm{b},9,0)$$

因此$a_0\cdots a_9=8327061549$或8327069541。

在一个更大的问题中，我们将在一个散列表中保持位向量表。借助于等价类，而不是排列来给出答案是更好的。其实，这个问题同排列的关系并不大。

111. 在具有$n!/2$个顶点$a_1\cdots a_{n-2}$和$n!$条弧$a_1\cdots a_{n-2}\rightarrow a_2\cdots a_{n-1}$的有向图中（对每个排列有一条弧）。每个顶点有入度2和出度2。其次，从像$a_1\cdots a_{n-2}\rightarrow a_2\cdots a_{n-1}\rightarrow a_3\cdots a_n\rightarrow a_4\cdots a_n a_2\rightarrow a_5\cdots a_n a_2 a_1\rightarrow\cdots\rightarrow a_2a_1a_3\cdots a_{n-2}$这样的通路，我们可以看到，从任何其他顶点开始，任何顶点都是可抵达的。因此由定理2.3.4.2D，一个欧拉路径存

在；而且这样的路径显然等价于排列的一个万有循环。当$n = 4$时，词典顺序下最小的例子是（123124132134214324314234）。

[格·哈尔伯特和加·伊萨克在（*Discrete Math.* **149** (1996), 123-129）中已经提议另一个有趣的方法：称排列的一个取模的万有循环是具有如下性质的$n!$个数字$\{0, \cdots, n\}$的循环，即$\{1, \cdots, n\}$的每个排列$a_1 \cdots a_n$从令$a_j = (u_j - c) \bmod (n + 1)$ 的连续数字$u_1 \cdots u_n$产生，其中c是在$\{u_1, \cdots, u_n\}$中“不出现”的数字。例如，取模万有循环（012032）对于$n = 3$来说实质上是惟一的。而对于$n = 4$词典顺序下最小的是（012301420132014321430243）。当$a_1 - a'_1 \equiv \cdots \equiv a_{n-2} - a'_{n-2}$(modulo n) 时，如果我们认为在上一节中的有向图的顶点$a_1 \cdots a_{n-2}$和$a'_1 \cdots a'_{n-2}$是等价的，我们就得到$(n-1)!/2$个顶点的欧拉路径对应于$\{1, \cdots, n-1\}$的取模万有循环的一个有向图。]

112. 由习题2.3.4.2-22，在前边答案的有向图中，只须计算在$12\cdots(n-2)$ 处为根的有向树就够了。而这些树可通过习题2.3.4.2-19来计算。对于$n \leqslant 6$，数U_n的结果相当简单：$U_2 = 1, U_3 = 3, U_4 = 2^7 \cdot 3, U_5 = 2^{33} \cdot 3^8 \cdot 5^3, U_6 = 2^{190} \cdot 3^{49} \cdot 5^{33}$。（这里我们认为（121323）和（213231）是相同的循环，但不同于（131232）。）

马克·库克已经发现下列启发性方法来有效地计算这些值：首先注意，排列的一个万有循环也等价于在具有生成元$\sigma = (1\ 2\cdots n)$ 和$\rho = (1\ 2\cdots n-1)$ 的凯利图上的一个哈密顿循环。例如，对于$n = 4$，前边的答案对应于循环$\sigma^3\rho^2\sigma\rho\sigma^2\rho^2\sigma^3\rho\sigma^2\rho^2\rho\sigma\rho\sigma^2\rho$。

现在考虑$n! \times n!$的矩阵$M = 2I - R - S$，其中$R_{\pi\pi'} = [\pi' = \pi\rho]$且$S_{\pi\pi'} = [\pi' = \pi\sigma]$。有一个矩阵$H$，使得$H^{-}RH$和$H^{-}SH$每一个都有块对角形式，对于$n$的每个分划$\lambda$，
120 它由$k_\lambda \times k_\lambda$的矩阵$R_\lambda$和$S_\lambda$的$k_\lambda$个副本组成，其中$k_\lambda$是$n!$除以形如$\lambda$的勾的长度的乘积（定理5.1.4H），而且$R_\lambda$和$S_\lambda$是基于杨氏表景的$\rho$和$\sigma$的矩阵表示。[在布鲁斯·萨甘（Bruce Sagan），*The Symmetric Group* (Pacific Grove, Calif.: Wadsworth & Brooks/Cole, 1991) 中可找到一个证明。]例如，当$n = 3$时，我们有（当把行和列通过分别的排列$1, \sigma, \sigma^2, \rho, \rho\sigma, \rho\sigma^2$来作下标时）：

$$R = \begin{pmatrix} 0&0&0&1&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0\\ 1&0&0&0&0&0\\ 0&0&1&0&0&0\\ 0&1&0&0&0&0 \end{pmatrix},\quad S = \begin{pmatrix} 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 0&0&0&1&0&0 \end{pmatrix},\quad H = \begin{pmatrix} 1&1&1&-1&1&0\\ 1&1&-1&0&0&-1\\ 1&1&0&1&-1&1\\ 1&-1&-1&1&0&1\\ 1&-1&1&0&1&-1\\ 1&-1&0&-1&-1&0 \end{pmatrix},$$

$$H^{-}RH = \begin{pmatrix} 1&0&0&0&0&0\\ 0&-1&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&1&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&1&0 \end{pmatrix},\qquad H^{-}SH = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&-1&0&0\\ 0&0&1&-1&0&0\\ 0&0&0&0&0&-1\\ 0&0&0&0&1&-1 \end{pmatrix}$$

这里$k_3 = k_{111} = 1$且$k_{21} = 2$，因此M的特征值是：在$k_\lambda \times k_\lambda$矩阵$2I - R_\lambda - S_\lambda$的$k_\lambda$-重重复的特征值在$\lambda$上的并。在这个例子中，(0)、(2)和两个$\left(\begin{smallmatrix} 2&0\\ -2&3 \end{smallmatrix}\right)$的特征值是$\{0\}$、$\{2\}$和两个$\{2, 3\}$。

在习题2.3.4.2-19中，M的特征值直接同矩阵A的那些特征值相关。其实，如果

我们把对于排列π和$\pi\rho\sigma^{-}$的分量等置，则A的每个特征向量产生M的一个特征向量，因为$R+S$的行π和$\pi\rho\sigma^{-}$相等。例如，对于特征值0, 3, 3

$$A=\begin{pmatrix}2&-1&-1\\-1&2&-1\\-1&-1&2\end{pmatrix}\text{ 有特征向量 }\begin{pmatrix}1\\1\\1\end{pmatrix},\begin{pmatrix}1\\-1\\0\end{pmatrix},\begin{pmatrix}1\\0\\-1\end{pmatrix}$$

并且产生对于相同特征值的M的特征向量 $(1,1,1,1,1,1)^T$, $(1,-1,0,0,-1,1)^T$, $(1,0,-1,-1,0,1)^T$。而且M有$n!/2$个额外的特征向量，除开对某个π由π和$\pi\sigma^{-}\rho$作为下标的分量以外，所有分量皆为零，因为$R+S$只有行$\pi\rho^{-}$和$\pi\sigma^{-}$在列π和$\pi\sigma^{-}\rho$中有非零的条目。这样的向量产生出$n!/2$个额外的特征值，全都等于2。

因此U_n（它是$2/n!$乘以A的非零特征值的乘积）是$2^{1-n!/2}/n!$乘以M的非零特征值的乘积。

不幸的是，小素数因式的现象不再继续；U_7等于$2^{1217}3^{123}5^{119}7^{5}11^{28}43^{35}73^{20}79^{21}109^{35}$，而且$U_9$可被$59229013196333^{168}$所整除。

如同在7.2.1.1小节中我们对德·布鲁因循环所介绍的那样，这些循环中至少有一个必定易于描述和计算。但是却还未找到简单的构造。 121

索引和词汇表

索引中的页码为英文部分的页码，与书中中文部分边栏的页码一致。当一个索引条款所指页码包含相关习题时，也请参见该习题的答案以获取更多信息。这里对答案页未加索引，除非其中有习题陈述中未包含的话题。

0-origin indexing（以0开始的下标），46

2-adic numbers（2-adic数。即向左无限伸展而右边有限的二进制数），31

4-cube（4立体），47-48, 78, 91

8-cube（8立体），17, 35

Kμ:（千次存储访问。参见Kilomems条目）

Mμ:（百万次存储访问。参见Megamems条目）

$\nu(k)$,（参见Lee weight条目，也是一数各位数之和，参见Sideways sum条目）

π(circle ratio), as "random" example（（圆周率3.14159…）作为"随机"的例子），30, 37, 65, 68, 74, 79

$\rho(k)$, 参见Ruler function（直尺函数）条目

$\sigma-\tau$ path（$\sigma-\tau$通路），58-59, 71

$\phi(k)$ permutation（$\phi(k)$ 排列），50-51, 69

A

Additive alphametics（加法性字母算术），44-45, 52-53, 68

Adjacent interchanges（相邻交换），40-45, 69, 73, 117, 118

Akl, Selim George (سليم جورج عقل)（阿克尔，塞林·乔治），117

Almost-linear recurrence（几乎线性递归），23

Alphametics（字母算术），44

- additive（加法性字母算术），44-45, 52-53, 68
- doubly true（双重为真的加法性字母算术（指字母和数值均为真）），67
- multiplicative（乘法性字母算术），67
- pure（纯字母算术），45, 66-67

Alternating group（交替群），43, 74

Analog-to-digital conversion（模拟-数字转换），3-4, 15

Analysis of algorithms（算法分析），28, 37, 38, 64-69, 72-73

Anti-Gray code（反格雷码），35

Antipodal words（对跖的，反面的词），11

Applying a permutation（应用一个排列），46-48

Arima, Yoriyuki（有马赖僮），77

Arisawa, Makoto（有泽诚），106

Arndt, Jörg（安德特，约格），81

Artificial intelligence（人工智能），66, 79

Ascents（递增，上升），118

Assignment problem（赋值问题），64

Aubert, Jacques（奥伯特，雅格奎斯），91

Automorphisms（自同构），47-48, 66, 67, 85

Avann, Sherwin Parker（艾万，舍尔文·帕克），116

B

Baez, John Carlos（巴耶兹，约翰·卡洛斯），83

Balanced Gray codes（平衡格雷码），14-17, 35, 85

Balanced permutations（平衡排列），74

Bandwidth of n-cube（n立体的带宽），35

Barwell, Brian Robert(巴威尔，布里安·罗伯特），67

baud: One transmission unit(e.g., one bit)per second（波特：每秒一个传输单位（例如，一个二进位）），4

Baudot, Jean Maurice Émile（鲍多特，吉恩·莫里斯·埃米里），4-5

Beckett, Samuel Barclay（贝克特，萨姆尔·巴克莱），34-35

Beidler, John Anthony（贝德勒，约翰·安托尼），44
Bell ringing（钟鸣响），39, 42-43, 59
Bennett, William Ralph（贝内特，威廉·拉尔芙），4
Bernoulli, Jacques(= Jakob = James)（贝努里，雅格奎斯（=雅各布=詹姆斯）），102
Bernstein, Arthur Jay（贝恩斯坦，阿瑟·杰伊），80
Binary Gray codes（二进制格雷码），12-17, 33-35
Binary number system（二进制数系），1, 4
Binary recurrences（二进制递归式），79, 96
Binary trie（二进制检索结构），30
Bit reversal（位颠倒（即首尾颠倒）），28, 31
Bitner, James Richard(毕特纳，詹姆斯·理查德），9
Bitwise operations（按位操作），4, 11-12, 32, 81
Borel, Émile Félix Édouard Justin（波里尔，埃米里·费里克·爱多华德·雅士登），97
Borrow（借位），76
Botermans, Jacobus(= Jack) Petrus Hermana（博特曼，雅各布斯（=杰克）彼特鲁斯·赫尔曼纳），91
Boustrophedon product（每行改换方向的乘积），36, 93
Breisch, Richard Lewis（布赖斯，理查德·刘易斯），105
Brightwell, Graham Richard（布赖特威尔，格拉姆·理查德），116
Bruijn, Nicolaas Govert de（布鲁因，尼古拉斯·戈威特·德），22
  cycles（布鲁因循环），22-27, 36-38, 75, 99, 121
  toruses（布鲁因圆环体），38
Bubble sort（气泡排序），41
Buchner, Morgan Mallory, Jr., 80（小巴赫纳尔，莫尔干·马罗里
Buckley, Michael R. W.（巴克利，迈克尔R. W.），66
Bypassing blocks of permutations（绕开排列块），51-54, 68, 117

C

Calderbank, Arthur Robert（卡尔德邦克，阿瑟·罗伯特），79
Cambridge Forty-Eight（剑桥48（个钟）），42, 43
Canfield, Earl Rodney(坎菲尔德，厄尔·罗德尼），116
Canoe puzzle（独木舟难题），92
Canonical delta sequence（规范的δ序列），13, 85
Cardano, Girolamo (= Hieronymus Cardanus)（加尔达诺，吉奥拉芙（=希尔罗尼姆斯·加尔达姆斯）），77
Carry（进位），2, 99
Casting out nines（弃9），106
Castown, Rudolph W.（卡斯汤，鲁道夫），11
Cattell, Kevin Michael（卡特尔，基文·迈克尔），98
Cavior, Stephan Robert（卡威欧，斯蒂芬·罗伯特），80
Cayley, Arthur（凯利，阿瑟），58
  graphs（凯利图），58, 69-72, 111, 120
  Hamilton theorem（哈密顿定理），81
Center of gravity（重心），17
Cesare, Giulio (pen name of Dani Ferrari, Luigi Rafaiani, Luigi Morelli, and Dario Uri)（西沙里，久里欧（丹尼费拉里，路易基拉发伊尼，鲁伊基莫勒里及达里奥·奥里的笔名）），105
Chain（链），73
Change ringing（改变钟响），39, 42-43, 59
Characteristic polynomial（特征多项式），81
Chen, Kuo-Tsai（陈国才），26
Cheng, Ching-Shui（郑清水），90
Childs, Roy Sydney（蔡德斯，罗伊·悉尼），105
Chinese remainder theorem（中国剩余定理），99
Chinese ring puzzle（中国（九龙）环难题），5-6, 28, 77-78
Cock, John Crowle（科克，约翰·克罗里），99
Cohn, Martin（科恩，马丁），85, 88, 90
Colex order（协词典顺序），也请参见Reverse colex order（颠倒的协词典顺序）条目，117
Complementary Gray codes（取补的格雷码），13,

16-17, 33, 85
Complete relation（完全关系），74
Compositions（合成），28-29
Compton, Robert Christopher（坎普顿，罗伯特·克里斯多弗），59, 70
Comtet, Louis（康姆特，路易斯），117
Concatenation（连接），25, 35, 85
Concurrent computing（并发计算），79
Conjugate permutation（共轭排列），50
Connected components（连通分量），34
Conway, John Horton（康威，约翰·霍尔顿），74, 83
Cooke, Raymond Mark（库克，莱蒙德·马克），87, 91, 120
Coordinates（坐标），13
Coroutines（共行程序），70-71
  recursive（递归共行程序），24-25, 36-37
Coupon collecting（证券收集），120
Cremer, William Henry, Jr.（小克里梅，威廉·亨利），92
Cryptarithms（密码数学），44
Cube, see *n*-cube（立体，见*n*立体）
Cube-connected computers（立体连通计算机），79
Cummings, Larry Jean（坎明斯，拉里·吉恩），94
Cycle, undirected（无向循环），66
Cycle leaders（循环导元），31
Cycle structure of a permutation（一个排列的循环结构），46, 50
Cyclic permutation（循环排列），73
Cyclic shifts（多个循环移位），26
Cyclic shift（一个循环移位），56, 58, 61, 68

D

Dally, William James（达里，威廉·詹姆斯），79
de Bruijn, Nicolaas Govert（德·布鲁因·尼科拉斯·戈威特），22
  cycles（布鲁因循环），22-27, 36-38, 75, 99, 121
  toruses（布鲁因圆环体），38
Decimal number system（十进制数系），2, 18-19, 39
Degen, Carl Ferdinand（迪根，卡尔·费尔迪南德），83
Delta sequences（δ序列），13, 69
Derangements（去排列），73
Dijkstra, Edsger Wybe（狄伊克斯特拉，埃德斯盖尔·怀比），42
Dilation of an embedded graph（一个嵌入图的扩大），35
Directed torus graphs（有向圆环体图），72
Discrete Fourier transform（离散傅里叶变换），9, 27, 83
Divisors of a number（一个数的因子），35
Doubly Gray code（双重格雷码），70
Doubly linked lists（双重链接表），21, 93-94
Doubly true alphametics（双重为真的字母算术），67
Douglas, Robert James（道格拉斯，罗伯特·詹姆斯），84
Dual boustrophedon product（对偶每行改变方向的乘积），93
Dual permutation generation（对偶排列生成），55-57, 68
Duckworth, Richard（达克沃思，理查德），v, 42
Dudeney, Henry Ernest（杜迪尼，亨利·欧内斯特），5, 44, 67, 100, 106
Duval, Jean Pierre（杜瓦尔，吉恩·皮尔里），98
Dyckman, Howard Lloyd（迪克曼，霍华德·李洛德），36, 92

E

Edge covering（边覆盖），35
Ehrlich, Gideon (גדעון אהרליך)（吉迪安·厄尔里兹），9, 57, 70, 103, 104
  swap method（厄尔里兹变换方法），57-58, 69-70
Eigenvalues and eigenvectors（特征值和特征向量），121
Enggren, Willy（恩格林，威里），66, 67
Enumeration（枚举），1
Eppstein, David Arthur（爱泼斯坦，戴维·阿瑟），104
Equivalent Gray codes（等价格雷码），33-34
Er, Meng Chiau（余明昭），54
Erdös, Pál(= Paul)（埃尔多斯，帕尔（= 保罗）），

112
Error-correcting codes（误差校正码），30
Etzion, Tuvi (טובי עציון, born טובי הולצר)（埃特津，图维），25
Euler, Leonhard (Ейлеръ, Леонардъ = Эйлер, Леонард) summation formula（欧拉，伦纳德，求和公式），106
Eulerian trails in a directed graph（在一个有向图中的欧拉路径），120
Even permutation（偶排列），43, 74
Exclusive or（异或），114
Exponential series, partial sums of（幂级数的部分和），102
Extending a partial order（扩充一个偏序），62
Extension of a string（一个串的扩充），26

F

Factorial number system（分数数系），101
Factorial ruler function（分数直尺函数），68
Factorization of strings（串的分解），37
  algorithm for（串分解算法），98
Faloutsos, Christos (Φαλούτσος, Χρήστος)（法罗特索斯·克里斯托斯），79
Fast Fourier transform（快速傅里叶变换），28
Fast Walsh transform（快速沃尔斯变换），32
Fermat, Pierre de, theorem（费尔玛，皮尔里·德，定理），38
Ferrari, Dani（费拉里，达尼），122
Fibonacci, Leonardo, of Pisa, numbers（斐波那契·比萨的伦纳多数），36, 74, 116
Field, finite（有限域），32
First-element swaps（首元素交换），57-58, 70
Fischer, Ludwig Joseph（菲希尔，路德维吉·约瑟夫），102
Five-letter words（五个字母的词），11, 32-33, 38, 66
Flip operation（触发操作），50-51, 69, 71, 74, 108
Flores, Ivan（弗罗里斯，伊凡），90
Focus pointers（焦点指针），10-11, 20-21, 93
Forest（森林），20-21
Fourier, Jean Baptiste Joseph（傅里叶，吉恩·巴帕蒂斯特·约瑟夫）
  series, 7（傅里叶级数）
  transform, discrete（离散傅里叶变换），9, 28, 83
Fox, Ralph Hartzler（福克，拉尔夫·哈特泽），26
Fraenkel, Aviezri S (אביעזרי פרנקל)（弗朗克尔，阿维泽热），119
Fredman, Michael Lawrence（费雷德曼，迈克尔·劳伦斯），33, 84
Fredricksen, Harold Marvin（费雷德里克森·哈罗德·马文），26, 27
Fringe（边缘），21, 93

G

Gallian, Joseph Anthony（加里安，约瑟夫·安托尼），113
Galois, Évariste（伽罗华，埃瓦里斯特），47
Gardner, Martin（马丁·加德纳），57, 65, 92, 100
General permutation generator（一般排列生成程序），48-51, 60-61, 67-68
Generating functions, techniques for using（使用生成函数的技术），65, 97, 102-103, 117
Generation（生成），1
  constant amortized time（常数平摊时间的生成），76
  loopless（无循环的生成），9-12, 20, 28, 29, 36, 66, 78, 104, 117
Gilbert, Edgar Nelson（吉尔伯特，埃德加·尼尔森），33
Gilbert, William Schwenck（吉尔伯特，威廉·斯文克），1
Goddyn de la Vega, Luis Armando（戈德丁·德·拉·维加，路易斯·阿尔曼多），34, 86
Goldstein, Alan Jay（戈德斯坦，阿兰·杰伊），61
Gomes, Peter John（戈姆斯，彼特·约翰），iii
González-Morris, Germán Antonio（宫扎里兹-莫利斯，热尔曼·安托尼奥），105
Gordian Knot puzzle（戈迪安结难题），35
Grandsire Doubles（格兰德希尔加倍），43
Gray, Elisha（格雷，埃里莎），5
Gray, Frank（格雷，弗朗克），4

Gray binary code（格雷二进制编码），2-12, 16, 28-33, 36, 41, 94
  permutation（排列的格雷二进制排列方法），3, 31
Gray binary trie（格雷二进制检索结构），30
Gray code: A sequence of adjacent objects（格雷码：相邻对象的一个序列）
Gray code for matchings（匹配的格雷码），116-117
Gray code for mixed radices(混合进制的格雷码），41, 103, 108, 112, 117
Gray code for *n*-tuples（*n*元组的格雷码），12, 15, 18
  advantages of（*n*元组格雷码的优点），6, 11-12
  binary, see Binary Gray codes, Gray binary code（格雷二进制编码。参见二进制格雷码）
  limitations of（格雷码的限制），76, 100
  nonbinary（非二进制的格雷码），18-20, 35-36, 80, 82, 88, 90-92
Gray code for permutations（排列的格雷码），69-70, 116-118
Gray code for weak orders（弱序的格雷码），118-119
Gray cycle: A cyclic Gray code（格雷循环：一个循环的格雷码），12, 15
Gray fields（格雷域），31
Gray path（格雷通路），15. see Gray code（参见格雷码）
Gray stream（格雷流），34
Gray ternary code（格雷三进制码），19, 36
Gros, Luc Agathon Louis（格罗斯，拉克·阿格顿·路易斯），5
Group of permutations（排列群），47-48, 58, 108
Gvozdjak, Pavol（格沃兹德扎克，帕沃尔），34

H

h-ordered permutation（阶数为*h*的排列），73
Hadamard, Jacques, Salomon（哈达玛，杰克奎斯·索罗门），83
  transform（哈达玛变换），9, 32, 82, 83
Hamilton, William Rowan, see Cayley（哈密顿，威廉·罗万，参见凯利）
  cycle（哈密顿循环），13, 34, 41, 58-59, 69-72, 111, 120
  path（哈密顿通路），15, 33, 41, 58-59, 70-71, 85, 110-111
Hamley, William, and sons（哈姆利，威廉及其儿子们），92
Hammons, Arthur Roger, Jr.（小哈姆蒙斯，阿瑟·罗杰尔），79
Hariguchi, Yoichi（播口阳一），iv
Harmuth, Henning Friedolf（哈姆特，亨宁·费里多夫），7
Hawaii（夏威夷），66
Heap, Brian Richard（希普，布里安·理查德），51, 53, 59, 68, 72, 104
Hexadecimal digits（十六进制数字），47, 120
Hexadecimal puzzle（十六进制难题），78
Hindenburg, Carl Friedrich（辛登伯格，卡尔·弗雷德里奇），40
Hook lengths（勾的长度），120
Hopcroft, John Edward（霍普克罗夫特，约翰·爱德华），80
Hunter, James Alston Hope（汉特，詹姆斯，阿尔斯顿·霍普），44
Hurlbert, Glenn Howland（哈尔伯特，格林·豪兰德），99, 120

I

Identity permutation（恒等排列），47
Image of an element（一个元素的映像），46
in situ permutation（在就地的排列），28, 31
in situ transformation（在就地的变换），9
Inclusion and exclusion principle（容斥原理），6, 117
Indecomposable permutations（不可分解的排列），73
Inline expansion（行中可扩展），11-12
Interleaving（穿插），37, 86, 99
Internet（因特网），ii, iii
Inverse function（逆函数），4, 31
Inverse permutation（逆排列），62-63, 66, 115
Inversion tables（反演表），41, 101, 116
Inversions of a permutation（一个排列的反演），

41, 43
Involutions（卷积），73-74, 111, 116
Isaak, Garth Timothy（伊萨克，加尔特·提莫西），99, 120
Isomorphic Gray cycles（自同构格雷循环），33-34
Iteration of functions（函数的迭代），32, 81
Ivanyi, Antal（伊凡伊，安托），99
Ives, Frederick Malcolm（伊弗斯，费雷德里克·马林科姆），68

J

Jackson, Bradley Warren（杰克森，布赖德利·沃伦），75
Japanese mathematics（日本数学），77
Jiang, Ming（姜明），59
Johnson, Allan William, Jr.（小约翰森，阿兰·威廉），105
Johnson, Selmer Martin（约翰，塞尔默尔·马丁），66

K

Kahan, Steven Jay（卡汉，史蒂文·杰伊），100, 105
Karnaugh, Maurice（卡诺，莫里斯），29
Kedlaya, Kiran Sridhara（凯德拉雅，吉兰·塞里德哈拉），85
Keister, William（凯斯特，威廉），78
Kemp, Rainer（凯姆普，赖安纳），101
Kent Treble Bob Major（肯特·特里贝尔·波布梅约），39
Kiefer, Jack Carl（基佛尔，杰克·卡尔），90
Kilomem: One thousand memory accesses（千存储：对存储的一千次访问）
King, Andrew Douglas（金，安德鲁·道格拉斯），117
Klügel, Georg Simon（克鲁格尔，乔治·西蒙），51
Knight's tour, northeasterly（骑士旅行路线，沿东北方向），72
Knuth, Donald Ervin（克努特，唐纳德·欧文，本书著者。中文名高德纳），i, ii, iv, 94
Koda, Yasunori（黄田保宪），20-21
Kompel'makher, Vladimir Leont'evich (**Компельмахер, Владимир Леонтьевич**)（康培尔马克尔，伏拉基米尔·伦特那维奇），70
Krause, Karl Christian Friedrich（克洛斯，卡尔，克里斯蒂安·费雷德里奇），102
Kronecker, Leopold, product（克洛尼克，利奥波尔德乘积），82
Kumar, Panganamala Vijay (**პანგანამალა ვიჯეი კუმარ**)（库马尔，邦加纳马拉·维杰伊），79

L

Langdon, Glen George, Jr.（小朗顿，格林·乔治），57, 61, 72
Larrivee, Jules Alphonse（拉尔里威，朱利斯·阿尔方斯），6
Lawrence, George Melvin（劳伦斯，乔治·默尔文），15, 86
Lee, Chester Chi Yuan（李始元）= Chi Lee (李济), 78
distance（李距离），29
weight（李权），29
Lehmer, Derrick Henry（莱默，德里克·亨利），39
Lempel, Abraham (**אברהם למפל**)（廉培尔，阿布拉罕），25
Lexicographic order（词典顺序），2-3, 25, 29, 39, 46, 83
Lexicographic permutation generation（词典顺序排列的生成），50, 53, 64-65
for involutions（用于卷积的词典顺序的排列生成），117
Lexicographic successor（词典顺序的后继），40
Li, Gang（= Kenny）（李钢）（肯尼李），94
Lieves（叶子），30
Linear embedding（线性嵌入），62
Linked allocation（链接分配），28, 29, 53-54, 117
Lipski, Witold, Jr.（小利普斯基·维特托尔德），107
Liskovets, Valery Anisimovich (**Лисковец, Валерий Анисимович**)（利斯科维奇，瓦列里·阿尼西莫维奇），70

Listing（列表），1
Loony Loop（伦尼环），35-36
Loopless generation（无循环生成），9-12, 20, 28, 29, 36, 66, 78, 104, 117
Loyd, Samuel（洛伊德，萨姆尔），iv
Lyndon, Roger Conant（林顿，罗杰尔·科南特），26
  Words, 26, see Prime strings（林顿字，参见素串）

M

m-ary digit: An integer between 0 and *m*-1 inclusive（*m*进制数字：0与*m*－1（含）之间的一个整数），2, 22
MacDonald, Peter S.（麦克唐纳德·彼特），66
Macro-processor（宏处理器），11
Maiorana, James Anthony（梅奥兰纳，詹姆斯·安托尼），26, 27
Mantel, Willem（曼特尔，威林），23
Martin, Monroe Harnish（马丁，蒙罗·哈尼斯），27-28
Matchings（匹配），33, 63, 73
Matrix(Bush), Irving Joshua（马特里克（布什），伊文·约舒亚），100
Matrix tree theorem（矩阵树定理），120
Maximal element（极大元），115
McClintock, William Edward（麦克林托克，威廉·爱德华），15
McCravy, Edwin Parker, Jr.（小麦克拉维，埃德温·帕克），66
McKay, Brendan Damien（麦基凯，布伦丹·达米恩），112
Median（均值），31
Megamem: One million memory accesses（百万存储访问：一百万次存储访问）
Mems : Memory accesses（存储访问）
Miers, Charles Robert（米尔斯，查尔斯·罗伯特），98
Military sayings（军事上的说法），1
Minimal element（极小元），115
Misra, Jayadev (ଜୟଦେବ ମିଶ୍ର)（米斯拉，加牙迪夫），77
Mitchell, Christopher John（米彻尔，克里斯多弗·约翰），25
Mixed-radix number system（混合进制数系），2, 19-21, 35, 55, 65, 90, 92, 101
MMIX computer（MMIX计算机，本书所用虚拟计算机），ii, iv, 59-61, 72, 76
Modular Gray codes（模块化格雷码），19-20, 35, 90
  decimal（十进制模块化格雷码），19
  m-ary（*m*进制模块化格雷码），24, 91, 94
  mixed-radix（混合进制模块化格雷码），112
  quaternary（四进制模块化格雷码），78, 85
  ternary（三进制模块化格雷码），82, 88
Modular universal cycles（模块化万用循环），120
Mollard, Michel（莫拉德，米彻尔），84
Monic polynomial（首一系数多项式），78
Monotonic binary Gray codes（单调的二进制格雷码），15-18, 35
Morse, Samuel Finley Breese, code（莫斯·萨缪尔，芬利·布里斯码），36, 93
Monte Carlo estimates（蒙特卡罗估计），110
Mor, Moshe (משה מור)（莫尔，莫西），118
Morelli, Luigi（莫勒里，雷吉），122
Morris, Ernest（莫里斯，欧内斯特），42
Morris, Scot Anderson（莫里斯，斯戈特·安德森），105
Moser, Leo（莫泽，利奥），84
Multinomial coefficient（多项式系数），29, 65
Multiplication of permutations（排列的乘法），46
Multiplicative alphametics（乘法性字母算法），67
Multisets（多重集合），39, 41, 62, 65, 71
Mundy, Peter（芒迪·彼特），42
MXOR (multiple exclusive-or)（多个异或），72
Myrvold, Wendy Joanne（米尔沃德，温迪·乔安尼），115

N

n-cube: The graph of *n*-bit strings, adjacent when they differ in only one position（*n*立体：*n*个二进位串的图。当这些串只在一个位上不同时它们相邻），13, 15, 33-34, 47-48, 66

subcubes of（*n*立体的子立体），30-31
n-distributed sequence（*n*分布序列），97
n-extension of a string（一个串的*n*位扩充），26
n-tuple: A sequence or string of length *n*（*n*元组：长度为*n*的一个序列或串），1-2
Nārāyana Pandita (**नारायण पण्डित**)（纳拉燕纳·潘迪塔），40, 101
Nemeth, Evelyn (= Evi) Hollister Pratt（尼梅特，埃维林（=埃维），霍利斯特·普拉特），86
Neyman, Jerzy（尼曼，杰尔齐），90
Nijenhuis, Albert（尼珍休斯，阿尔伯特），58
Nijon, Herman（尼均，赫尔曼），66
Nonbinary Gray codes（非二进制格雷码），18-20, 35-36, 82, 88, 90-92
Nonlocal Gray codes（非局部格雷码），16-17, 34
Nordstrom, Alan Wayne（诺德斯特洛姆，阿兰·韦恩），30
Normal numbers（规格化数字），97
Northeasterly knight's tour（朝东北方向的骑士旅行路线），72
Novra, Henry（诺弗拉，亨利），91
NP-complete problem（NP完全问题），104
Nybble: A 4-bit quantity（尼贝尔：四个连续二进位的一个量），60-61

O

Octacode（八进制码），30
Octahedral group（八面体的群），104
Octonions（八个事物），83
Odd length runs（奇数长度的运程），94
Odd permutation（奇排列），43, 110-111
Ord-Smith, Richard Albert James (= Jimmy)（奥德-史密斯，理查德·阿尔伯特·詹姆斯），50, 51, 56, 67, 68, 102
Order of a group element（群元素的顺序），58, 108
Organ pipe order（风琴管顺序），111, 118
Orthogonal vectors（正交向量），8, 32
Ourotoruses（外圆环体），38-39

P

Paley, Raymond Edward Alan Christopher（佩里，雷蒙德，爱德华·阿兰·克里斯多弗），81
functions（佩里函数），32
Pak, Igor Markovich (**Пак, Игорь Маркович**)（帕克，雨果·马科维奇），111
Pan-digital puzzles（泛数字难题），39, 67
Parallel computation（并行计算），72, 104
Parity bit（奇偶位），6, 28, 29
Partial ordering（偏序），62, 72, 73
Partitions of a number（一个数的分划），67, 120
Peterson, Kenneth Graham（佩得森，肯尼思·格拉罕），25
Peczarski, Marcin Piotr（皮克扎斯基，马辛·皮奥特），116
Pepperdine, Andrew Howard（佩珀迪尼，安德鲁·霍华德），119
Permutation generation（排列生成），39-75
  cyclic shift method（排列生成的循环移位方法），56, 58, 61, 68
  dual（对偶的排列生成），55-57, 68
  Ehrlich swap method（排列生成的厄尔里兹交换方法），57-58, 69-70
  fastest（最快的排列生成），59-62
  general（通用的排列生成方法），48-51, 60-61, 67-68
  lexicographic（排列生成的词典顺序方法），50, 53, 64-65
  lexicographic with restricted prefixes（排列生成的带有限制前缀的词典顺序方法），54, 68, 116
  plain changes（排列生成的平易改动法），42-45, 55, 61, 63, 65-66, 71, 118
  when to use（何时使用排列生成），64
Permutations（排列），39-75
  applying（应用排列），46-48
  conjugate（共轭排列），50
  cycles of（排列的循环），46, 50
  cyclic（循环排列），73
  derangements（排列的去排列），73
  even（偶排列），43, 74
  groups of（排列群），47-48, 58, 108
  Gray codes for（排列的格雷码），69-70, 116-118
  h-ordered（*h*阶的排列），73

indecomposable（不可分解的排列），73
inverse（排列的逆），62-63, 66, 115
inversions of（排列的反演），41, 43
involutions（排列的卷积），73-74, 116
multiplication of（排列的乘法），46
notations for（排列的记号），46
odd（奇排列），43, 110-111
of a multiset（一个多重集合的排列），39-40, 62
r-element（*r*元素排列），65, 68
rank of（排列的秩），65, 72, 115
sign of（排列的符号），43, 71
signed（带符号的排列），66
universal cycles of（排列的万用循环），75
up-down（由顶向下的排列），73
well-balanced（一个平衡得好的排列），74
Perverse, Rufus Quentin（珀弗斯，鲁弗斯·昆廷），35
Pfaff, Johann Friedrich（普发夫，约翰·费珀里），117
Phillips, John Patrick Norman（菲利浦，约翰·帕特里克，诺尔曼），101
Pi(π), as "random" example（π圆周率。作为随机数例子），30, 37, 65, 68, 74, 79
Plain changes（平易改动），42-45, 55, 61, 63, 65-66, 71, 118
Pleszczyński, Stefan（普利斯泽仁斯基，斯蒂芬），115
Postmultiplication（后乘法），47
Preferential arrangements（合意的安排），118
Prefix of a string（一个串的前缀），25
Premultiplication（预乘法），47, 49-50, 52, 117
Preorder in a tree（在一棵树中的前根顺序），49, 52
Prepostorder（预后根顺序），78
Preprime strings（预素串），26-28, 37
Prime strings（素串），25-28, 37
factorization（因式分解为素串），37, 98
Primitive polynomials modulo *p*（本原多项式 modulo *p*），23, 81
Principal subforests（主要的子森林），20-21
Prömel, Hans Jürgen（普罗梅尔，汉斯·尤尔根），116
Proper prefixes or suffixes（真前缀或后缀），25
Pruesse, Gara（普鲁伊斯，加拉），116
Pseudorandom bits（虚拟随机二进位），37
Pulse code modulation（脉冲码调制），4
Pure alphametics（纯字母算术），45, 66-67
Purkiss, Henry John（帕基斯，亨利·约翰），28
Puzzles（难题），iv

Q

Quaternary *n*-tuples（四进制*n*元组），29, 85
Quaternions and octonions（四事物和八事物体），32
Queue（队列），115

R

R&D method（R（递归的德·布鲁因生成）和D（双重递归德·布鲁因生成）方法），25, 37
Rademacher, Hans（拉德曼彻，汉斯），8
functions（拉德曼彻函数），8, 32, 82
Radoičić, Radoš（拉多伊希克，拉多斯），111
Rafaiani, Luigi（拉斐雅尼，雷吉），122
Ramras, Mark Bernard（拉姆拉斯，马克·贝尔纳德），86
Random number generation（随机数生成），37
Rankin, Robert Alexander（兰金，罗伯特·亚历山大），58, 71, 112
Ranking a permutation（对一个排列排秩），65, 72, 115
Ranking an *n*-tuple（对一个*n*元组排秩），4, 19, 35
Rapoport, Elvira Strasser（拉坡博特，埃尔维拉·斯特拉泽），111
Recurrences（递归式），23, 79, 81, 90, 95-96, 101
Reflected Gray codes（反射格雷码），19-21, 35, 80, 90, 92
decimal（十进制反射格雷码），19
mixed-radix（混合进制反射格雷码），41, 103, 108, 117
ternary（三进制反射格雷码），36
Reingold, Edward Martin（ריינגולד, משה בן חיים）

יצחק)（赖因戈德，爱德华·马丁），9
Reversal of a string（一个串的颠倒），69, 74
Reverse colex order（颠倒的协词典顺序），46, 50, 53, 55, 64, 101
Reversing a string（颠倒一个串，也参见Flip operation条目），40, 101, 103
Reversing bits（颠倒二进位（首尾交换）），28, 31
Richards, Dana Scott（理查德，达纳·斯戈特），36
Right subcube（正确的子立体），30
Ringel, Gerhard（林吉尔，杰哈德），35
Ritchie, Alistair English（里彻，阿历斯佚尔·英格里斯），78
Robinson, John Paul（罗宾森，约翰·鲍罗），30, 85, 88
Roman numerals（罗马数字），105
Rosary permutations（玫瑰花园排列），66, 111
Rosenbaum, Joseph（罗森鲍姆，约瑟夫），90
Rotem, Doron(דורון רותם)（罗特姆，多伦），60, 63
Rothe, Heinrich August（罗思，海固里兹·奥古斯特），101
Roy, Mohit Kumar(মোহিত কুমার রায়)（罗伊，莫希特·库玛尔），104
Rüdiger, Christian Friedrich（鲁迪吉尔，克里斯蒂安，弗雷德里兹），40
Ruler function（直尺函数），6, 8, 12, 13, 83
  decimal（十进制直尺函数），19
  factorial（分数直尺函数），68
Run lengths（运程长度），15-17, 34, 86, 94
Ruskey, Frank（拉斯基，弗朗克），iv, 20, 21, 28, 31, 33, 59, 72, 94, 98, 111, 112, 115, 116

S

Sagan, Bruce Eli（萨甘，布鲁斯·埃利），121
Salzer, Herbert Ellis（萨尔泽，希尔伯特·埃利斯），80
Sampson, John Laurence（萨姆普森，约翰·劳伦斯），33, 84
Savage, Carla Diane（萨维吉，卡尔拉·戴安尼），17-18, 28, 33, 35, 85, 111, 118
Sawada, Joseph James（萨瓦达，约瑟夫·詹姆斯），98
Sayers, Dorothy Leigh（塞耶斯，多萝西·赖格），39
Schäffler, Otto（沙菲勒尔，奥托），5
Schneider, Bernadette（施奈德，贝尔纳德梯），91
Schützenberger, Marcel Paul（舒曾伯格，马希尔·鲍罗），97
Sedgewick, Robert（塞奇威克，罗伯特），59
Seitz, Richard（塞特兹，理查德），57
Sequency（序列），7
Serra, Micaela（塞尔拉，迈凯拉），98
Shapiro, Harold Seymour（萨皮罗，哈罗德·塞莫尔），33
Shift register sequences（移位寄存器序列），22-28, 36-38
Sideways sum（横向加（一个数各个数字之和）），15, 80
Sign of a permutation（一个排列的符号），43, 71
Signature of an alphametic（一个字母算术的签名），44
Signed permutation（带符号的排列），66
Silver, Alfred Lindsey Leigh（西尔弗，阿尔弗雷德·林希·赖格），109
Silverman, Jerry（希尔弗曼，杰里），33, 84-86
Sims, Charles Coffin（西姆斯，查尔斯·科芬），47
  tables（西姆斯表），47-53, 55-56, 67-68
Skipping blocks of permutations（跳过排列的块），51-54, 68, 117
Sloane, Neil James Alexander（施罗安，奈尔·詹姆斯·亚历山大），79
Slocum, Gerald Kenneth(= Jerry)（斯洛克姆，杰拉德·肯尼思（=杰里）），91
Solé, Patrick（索利，帕特里克），79
SpinOut puzzle（转出（中国环）难题），78
Squire, Matthew Blaze（斯奎尔，马修斯·布拉泽），94
Stahnke, Wayne Lee（斯坦克，韦恩·李），23
Standard sequences（标准序列），26
Stanford GraphBase（斯坦福图库），ii, iii, 11,

32-33, 38
Star graph（星图），70
Star transpositions（星形转置），57-58, 70
Stedman, Fabian（斯蒂德曼，法比安），42
  Doubles（斯蒂德曼加倍），43
Steger, Angelika（史蒂吉尔，安琪尔卡），116
Steiglitz, Kenneth（斯泰格里兹，肯尼思），80
Stevens, Brett（斯蒂文斯，布里特），34
Stewart, Ian Nicholas（斯图尔特，伊恩·尼古拉斯），38
Stibitz, George Robert（史蒂比兹，乔治·罗伯特），4, 6
Stringology（字符串学），25-28, 37-38
Subcubes（子立体），30-31
Subforests（子森林），20-21, 36
Subsets（子集合），1, 6
Suffix of a string（一个串的后缀），25
Sums of squares（平方和），32
Swapping with the first element（同头元素的交换），57-58, 70
Sylvester, James Joseph（西尔威斯特，詹姆斯·约瑟夫），32, 83
Symmetries（对称），47-48, 66, 67

T

Tableaux（表景），62-63, 120
Tangle puzzle, see Loony Loop（困惑难题，参见伦尼环）
Taylor, Lloyd William（泰勒，李洛德·威廉），5
Tchuente, Maurice（特纯特，莫里斯），110
Telephone（电话），5
Television（电视），4
Ternary *n*-tuples（三进制*n*元组），19, 26-27, 35, 36, 82, 88, 100
Tic-tac-toe board（三子棋盘），67
Tiring irons（疲劳的钢），5
Tompkins, Charles Brown（汤普金斯，查尔斯·布朗），57
Tootill, Geoffrey Colin（图特梯尔，杰弗里·科林），14, 77
Topological sorting（拓扑排序），62-64, 72-73
Topswops（搅顶（单人纸牌游戏）），74
Torture test（拷打测试），35
Toruses（圆环体），29, 38, 47, 78
  directed（有向圆环体），72
  twisted（扭曲圆环体），70
Total ordering（全序），62
Tóth, Zoltán（托特，佐尔坦），99
Transition counts（转换计数），14, 33
Transitive relation（传递关系），72, 74
Transposing adjacent elements（转置相邻元素），40-45, 69, 73, 117, 118
Traveling salesrep problem（旅行的销售商问题），64
Traversal（遍历），1
Trend-free Gray codes（无趋势格雷码），16-17, 35
Trie（检索结构），30
Trotter, Hale Freeman（特洛特，哈尔·弗里曼），43
Trotter, William Thomas（特洛特，威廉·托马斯），112
Tuliani, Jonathan R.（图里安尼，约纳丹·R），97
Tuple: A sequence containing a given number of elements（元组：含有给定个数的元素序列）
Twisted torus（扭曲的圆环体），70
Two-line form of permutation（排列的两行形式），46

U

Undirected cycle（无向循环），66
Undoing（返工），54, 117
Universal cycles of permutations（排列的万有循环），75
  modular（模块化的排列万有循环），120
Unranking a permutation（对一个排列去掉阶次），65, 72, 115
Unranking an *n*-tuple（对一个*n*元组去掉阶次），3-4, 19, 28, 35
Up-down permutations（由顶向下的排列），73
Up-down sequences（由顶向下的序列），36
Uri, Dario（尤里，达里奥），122

V

Variations（变异、变差），65, 68, 106, 115
Varol, Yaakov Leon(יעקב לאון ורול)（瓦洛尔，雅可夫·里昂），60, 63
Vatriquant, Simon（瓦特里宽特，西蒙），44
Vázsonyi, Endre（瓦泽索尼，恩德里），93
Vickers, Virgil Eugene（维克斯，维吉尔·尤金），33, 84-86
Vinnicombe, Robert Ian James（文尼康比，罗伯特·伊恩·詹姆斯），66
Visitation（访问），1

W

Wallis, John（沃里斯，约翰），6, 77
Walsh, Joseph Leonard（沃尔斯，约瑟夫·伦纳德），7, 8, 81
functions（沃尔斯函数），7-9, 32
transform（沃尔斯变换），8-9, 32
Walsh, Timothy Robert Stephen（沃尔斯，蒂莫西·罗伯特·斯蒂芬），117, 118
Wang, Terry Min Yih（王珉懿），28
Washburn, Seth Harwood（瓦斯布恩，塞特·哈尔伍德），78
Wayne, Alan（韦恩，阿兰），67
Weak orders（弱序），74
Weight enumeration（权枚举），78
Well-balanced permutation（平衡了的排列），74
Wells, Mark Brimhall(韦尔斯，马克·布林哈尔），107
Weston, Andrew（威斯通，安德鲁），59
White, Arthur Thomas（怀特，阿瑟·托马斯），II, 43
Wiedemann, Douglas Henry（怀德曼，道格拉斯·亨利），94
Wilf, Herbert Saul（威尔芙，希尔伯特·索尔），58, 119
Williamson, Stanley Gill（威廉森，斯坦利·吉尔），59, 70, 116
Wilson, Wilfrid George（威尔森，威尔弗雷德·乔治），43
Winker, Steven Karl（温格尔，斯蒂文·卡尔），84
Winkler, Peter Mann（温格尔，彼特·曼），17-18, 35, 85
Wrapping around（包绕起来），19, 29, 38

X

XOR(bitwise exclusive-or)（XOR（指令）（按位异或）），114

Y

Yates, Frank（雅特斯，弗朗克），9
Yoshigahara, Nobuyuki(= Nob)（芦ケ原伸之），67, 105
Young, Alfred, tableaux（杨，阿尔弗雷德表景），62-63, 120
Yuen, Chung Kwong（阮宗光），80