THE ART OF COMPUTER PROGRAMMING

PRE-FASCICLE 2A

A DRAFT OF SECTION 7.2.1.1: GENERATING ALL n-TUPLES

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

Internet page `http://www-cs-faculty.stanford.edu/~knuth/taocp.html` contains current information about this book and related books.

See also `http://www-cs-faculty.stanford.edu/~knuth/sgb.html` for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

Copyright © 2001 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 14), 10 December 2004

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 contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, and 3 were at the time of their first printings. And those carefully-checked volumes, alas, were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make it both interesting and authoritative, as far as it goes. But the field is so vast, I cannot hope to have surrounded it enough to corral it completely. Therefore I beg you to let me know about any deficiencies you discover.

To put the material in context, this is Section 7.2.1.1 of a long, long chapter on combinatorial algorithms. 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 the topic of bitwise manipulations. (I drafted about 60 pages about that subject in 1977, but those pages need extensive revision; meanwhile I've decided to work for awhile on the material that follows it, so that I can get a better feel for how much to cut.) 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 this booklet, Section 7.2.1.1, where I get the ball rolling at last by dealing with the generation of n-tuples. Then will come Section 7.2.1.2 (about permutations), Section 7.2.1.3 (about combinations), etc. 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.

Even the apparently lowly topic of n-tuple generation turns out to be surprisingly rich, with ties to Sections 1.2.4, 1.3.3, 2.3.1, 2.3.4.2, 3.2.2, 3.5, 4.1, 4.3.1, 4.5.2, 4.5.3, 4.6.1, 4.6.2, 4.6.4, 5.2.1, and 6.3 of the first three volumes. I strongly believe in building up a firm foundation, so I have discussed this topic much more thoroughly than I will be able to do with material that is newer or less basic. To my surprise, I came up with 112 exercises, a new record, even though — believe it or not — I had to eliminate quite a bit of the interesting material that appears in my files.

Some of the material is new, to the best of my knowledge, although I will not be at all surprised to learn that my own little "discoveries" have been discovered before. Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 43, 46, 47, 53, 55, and 62. Are these problems still open? The question in exercise 53 might not have been posed previously, but it seems to deserve attention. Other problems, like exercises 66 and 83, suggest additional research topics. Please let me know if you know of a solution to any of these intriguing problems. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to get credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who I should have credited, with respect to the ideas found in exercises 15, 16, 31, 37, 38, 69, 73, 76, 86, 87, 89, 90, and/or 109.

I shall happily pay a finder's fee of \$2.56 for each error in this draft 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 book was written. And I also want to thank Frank Ruskey for bravely foisting this material on college students, and for providing valuable feedback about his classroom experiences.

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

Happy reading!

Stanford, California D. E. K.
August 2001 (revised, September 2001)

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 $a_1 = 0, 1, \ldots, m_1 - 1$ (in this order) do the following:
 For $a_2 = 0, 1, \ldots, m_2 - 1$ (in this order) do the following:
  For $a_3 = 0, 1, \ldots, m_3 - 1$ (in this order) do the following: (3)
   For $a_4 = 0, 1, \ldots, m_4 - 1$ (in this order) do the following:
    Visit (a_1, a_2, a_3, a_4).

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

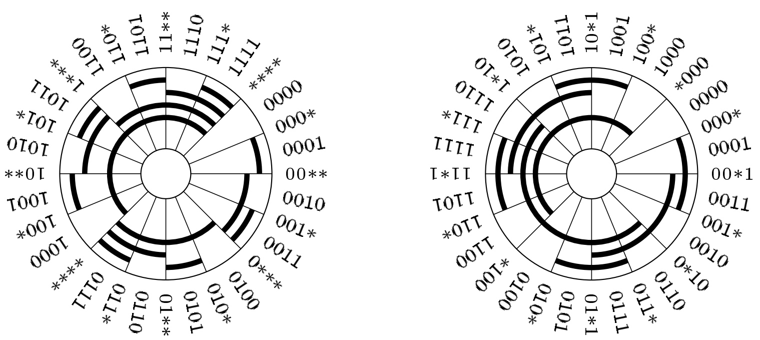


Fig. 10. (a) Lexicographic binary code. (b) Gray binary code.

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

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

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

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

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, $g\bigl((111001000011)_2\bigr) = (100101100010)_2$. 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.)

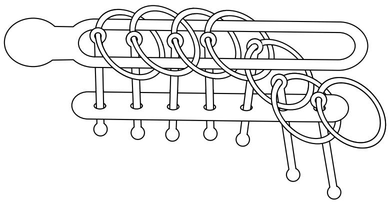


Fig. 11. The Chinese ring puzzle.

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 $k\oplus1$; 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–(00). 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)

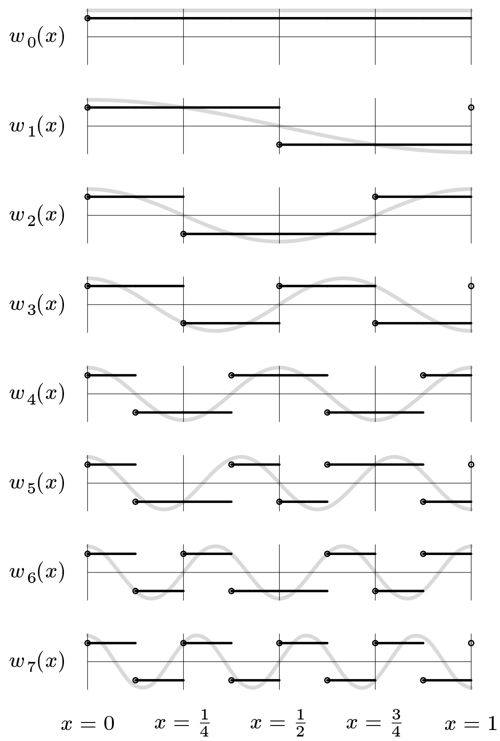


Fig. 12. Walsh functions $w_k(x)$ for $0 \le k < 8$, with the analogous trigonometric functions $\sqrt{2}\cos k\pi x$ shown in gray for comparison.

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	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$
a_2	0	0	0	0	$\underline{1}$	$\underline{1}$	$\underline{1}$	$\underline{1}$	1	1	1	1	$\underline{0}$	$\underline{0}$	$\underline{0}$	$\underline{0}$
a_1	0	0	$\underline{1}$	$\underline{1}$	1	1	$\underline{0}$	$\underline{0}$	0	0	$\underline{1}$	$\underline{1}$	1	1	$\underline{0}$	$\underline{0}$
a_0	0	$\underline{1}$	1	$\underline{0}$	0	$\underline{1}$	1	$\underline{0}$	0	$\underline{1}$	1	$\underline{0}$	0	$\underline{1}$	1	$\underline{0}$
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 $\bigl((z-m)\oplus z\oplus m\bigr)\wedge m' \neq 0$, 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 $m_0 \leftarrow z \wedge (2^5-1)$, $m_1 \leftarrow z \wedge (2^{10}-2^5)$, $m_2 \leftarrow z \wedge (2^{15}-2^{10})$, $m_3 \leftarrow z \wedge (2^{20}-2^{15})$, and $m_4 \leftarrow z \wedge (2^{25}-2^{20})$, 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`	(23)
`pines`	`piney`	`pinky`	`pinks`	`punks`	`punky`	`pucks`	

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

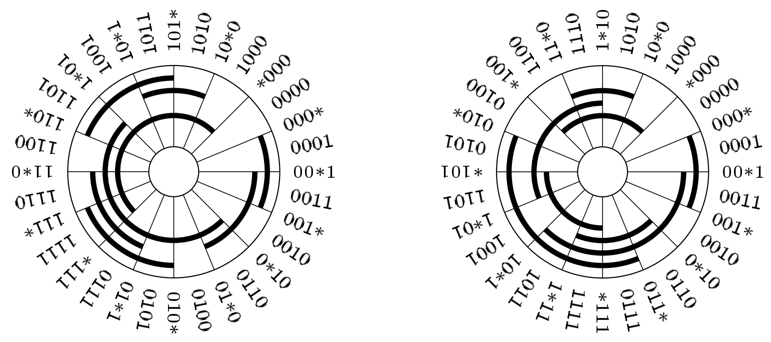


Fig. 13. (a) Complementary Gray code. (b) Balanced Gray code.

cycle on the n-cube. We can assume that subscripts have been chosen so that $v_0 = 0\ldots0$.

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

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

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 $\overline{a}_3\overline{a}_2\overline{a}_1\overline{a}_0$. 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*

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

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

$$01410301020131024201043401020103.\tag{28}$$

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 $01v_{lm_l}$, …, $01v_{10}$ 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

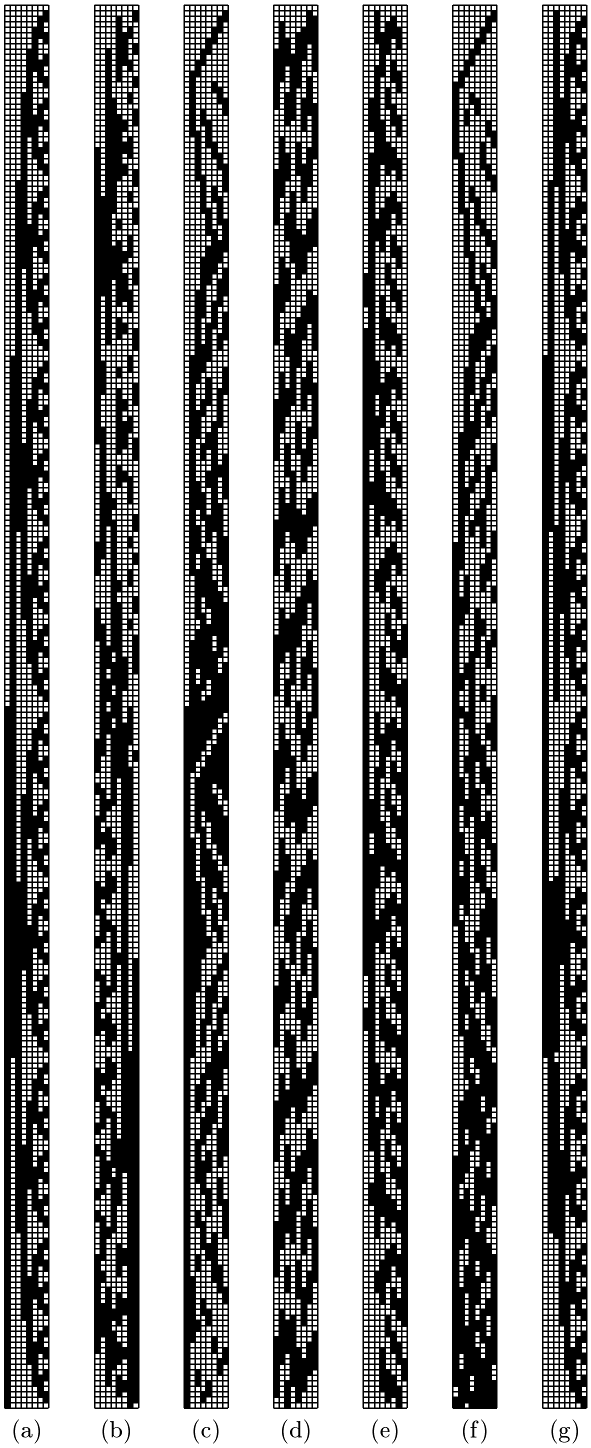


Fig. 14. Examples of 8-bit Gray codes:

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 $n > 0$,

$$P_{(n+1)j} = 1P^{\pi_n}_{n(j-1)},\ 0P_{nj}; \tag{34}$$
$$P_{nj} = \emptyset \quad \text{if } j < 0 \text{ or } j \ge n. \tag{35}$$

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

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

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

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

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

$$\begin{aligned} \pi_1 &= (0), & \pi_4 &= (0\,3),\\ \pi_2 &= (0\,1), & \pi_5 &= (0\,4\,3\,2\,1),\\ \pi_3 &= (0\,2\,1), & \pi_6 &= (0\,5\,2\,4\,1\,3); \end{aligned}$$

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

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

into its reflected-Gray equivalent

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

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

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

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:

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

(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 $(a_{n-1},\ldots,a_1,a_0)$.

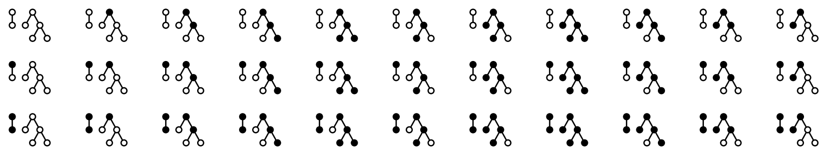
H3. [Choose j.] Set $j \leftarrow f_0$ and $f_0 \leftarrow 0$. (As in Algorithm L, j was the rightmost active coordinate; all elements to its right have now been reactivated.)

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).

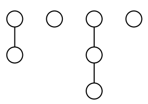
***Subforests.** An interesting and instructive generalization of Algorithm H, discovered by Y. Koda and F. Ruskey [*J. Algorithms* **15** (1993), 324–340], sheds further light on the subject of Gray codes and loopless generation. Suppose we have a forest of n nodes, and we want to visit all of its "principal 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.

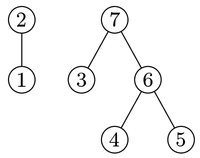
If the given forest consists of degenerate nonbranching trees, the principal subforests are equivalent to mixed-radix numbers. For example, a forest like



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 forest. 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.

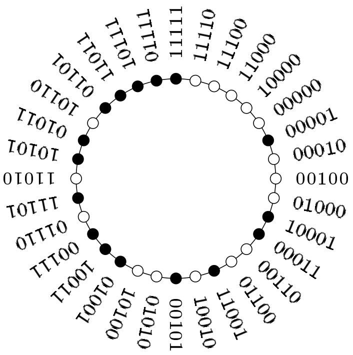


Fig. 15.
A de Bruijn cycle for 5-bit numbers.

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

$$0011021220313233041424344\ldots \tag{54}$$

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

$$f(x_1,\ldots,x_n)=\begin{cases}c_1, & \text{if } (x_1,x_2,\ldots,x_n)=(0,0,\ldots,0);\\ 0, & \text{if } (x_1,x_2,\ldots,x_n)=(1,0,\ldots,0);\\ (c_1x_1+c_2x_2+\cdots+c_nx_n)\bmod m, & \text{otherwise.}\end{cases} \tag{55}$$

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

$$x^n-c_nx^{n-1}-\cdots-c_2x-c_1 \tag{56}$$

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: 121211112121211112121211112121211112121211112121 ...
x' in step D7: 0 11021220 11021220 11021220 11021220 11021220 1 ...;

so the 81-cycle produced in steps D3 and **D7** is 0**0**00**1**0**110**1**2**...2**2**22 0**0**00**1**....

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 $a_1 \ldots a_n$ are $a_2 \ldots a_n a_1$, $a_3 \ldots a_n a_1 a_2$, $\ldots$, $a_n a_1 \ldots a_{n-1}$.)

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 $a_1 \ldots a_n$, 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 $\lambda\lambda\lambda\ldots$. 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 $n = 4$:

$0_\wedge000$	$0011_\wedge$	$0022_\wedge$	$0111_\wedge$	$0122_\wedge$	$0212_\wedge$	$1_\wedge111$	$12_\wedge12$	
$0001_\wedge$	$0012_\wedge$	$01_\wedge01$	$0112_\wedge$	$02_\wedge02$	$022_\wedge0$	$1112_\wedge$	$122_\wedge1$	(61)
$0002_\wedge$	$002_\wedge0$	$0102_\wedge$	$012_\wedge0$	$021_\wedge0$	$0221_\wedge$	$112_\wedge1$	$1222_\wedge$	
$001_\wedge0$	$0021_\wedge$	$011_\wedge0$	$0121_\wedge$	$0211_\wedge$	$0222_\wedge$	$1122_\wedge$	$2_\wedge222$	

(The digits preceding '$_\wedge$' 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:

$$0\;0001\;0002\;0011\;0012\;0021\;0022\;01\;0102\;0111\;0112$$
$$0121\;0122\;02\;0211\;0212\;0221\;0222\;1\;1112\;1122\;12\;1222\;2. \qquad (62)$$

(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 $2222122202211 \ldots 10000$, 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:* $\left[{0,\,0,\,1,\,2,\,3,\,0,\,2,\,7,\,1,\,0 \atop 1,\,2,\,3,\,4,\,5,\,6,\,7,\,8,\,9,\,10}\right] = 1000000$.

▸ **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 $s_1s_2\ldots s_j$.

b) Similarly, design a loopless algorithm that represents the compositions implicitly in an array of pointers $q_0q_1\ldots q_t$, 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

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

(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

$$(v_0 + v_1x + v_2x^2 + v_3x^3)g(x) \bmod 4 = u_0 + u_1x + u_2x^2 + u_3x^3 + u_4x^4 + u_5x^5 + u_6x^6$$

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 $10**$.)

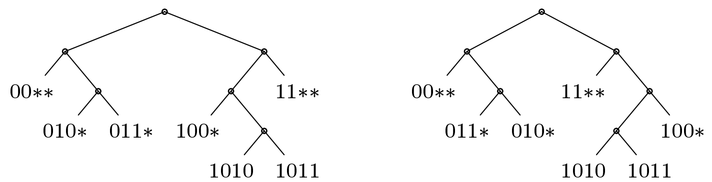


Fig. 16. (a) Normal binary trie. (b) Gray binary trie.

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,

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

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

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

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

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

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

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

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 $m_j = x \wedge (2^{5j} - 1)$ 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 $j = n$.

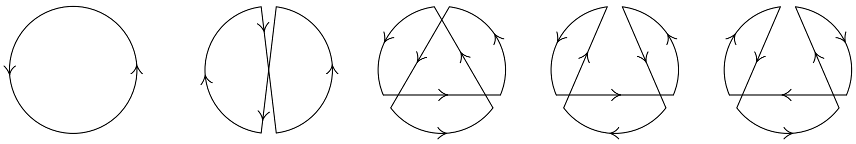
53. [*M46*] If the numbers $(c_0, \ldots, c_{n-1})$ are even and satisfy the condition of the previous exercise, is there always an n-bit Gray cycle with these transition counts?

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 $v_0 = 0\ldots0$ 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 $h(0)$, $h(1)$, …, $h(2^n-1)$ 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, …, 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), …, (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.

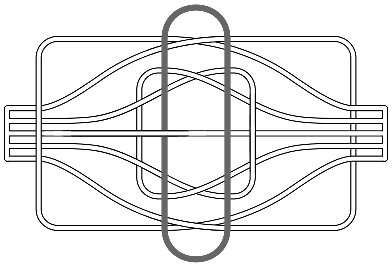


Fig. 17. The Loony Loop puzzle.

▸ **85.** [*M25*] (Dana Richards.) If $\Gamma = (\alpha_0, \ldots, \alpha_{t-1})$ is a sequence of t strings of length n and $\Gamma' = (\alpha'_0, \ldots, \alpha'_{t'-1})$ is a sequence of t' strings of length n', the *boustrophedon product* $\Gamma \boxtimes \Gamma'$ is the sequence of tt' strings of length $n + n'$ 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 •—, •••, —• or its reverse.

b) What string follows •——••——•—• 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.** [*M23*] 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 for the base case $n = 2$.

a) How many coroutines 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 $\bigl(f_n(0)\ldots f_n(2^n-1)\bigr)$ for $n = 2$, 3, 4, and 5.
b) Prove that, for all even values of n, there is a number $\delta_n = \pm 1$ 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 $\bigl(c_n(0)c_n(1)\ldots c_n(2^{2n}-5)\bigr)$ 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 $\bigl(f_n(0)f_n(1)\ldots f_n(2^n-1)\bigr)$.

100. [*40*] Do the de Bruijn cycles of exercise 97 provide a useful source of pseudorandom 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 3141592653589793238462643383279502884197?

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 a_{ij} such that each of the m^4 submatrices

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

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 A 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 $a_{ij} = (\ldots c_2 c_1 c_0)_2$, where

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

Show that the upper left $2^{2n} \times 2^{2n}$ submatrix of A 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, $100 = -1234 - 5 - 6 + 7898 - 7 - 6543 - 2 - 1$.

ANSWERS TO EXERCISES

All that heard him were astonished
at his understanding and answers.
— Luke 2:47

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 $g\bigl((0b_{n-1} \ldots b_1 b_0)_2\bigr) = \bigl(0(b_{n-1}) \ldots (b_2 \oplus b_1)(b_1 \oplus b_0)\bigr)_2$ then $g\bigl((1b_{n-1} \ldots b_1 b_0)_2\bigr) = 2^n + g\bigl((0\overline{b}_{n-1} \ldots \overline{b}_1 \overline{b}_0)_2\bigr) = \bigl(1(\overline{b}_{n-1}) \ldots (\overline{b}_2 \oplus \overline{b}_1)(\overline{b}_1 \oplus \overline{b}_0)\bigr)_2$, where $\overline{b} = b \oplus 1$.

40

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, ..., 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 $\mathtt{LINK}(j-1) = j + nb_j$ for $1 \le j \le n$ and $\mathtt{LINK}(j-1+n) = j + n(1-b_j)$ 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 $\bigl(g(j), g(k)\bigr)$ 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 $a=\bigl(j+(\infty,6,5,2,3,1,4,0)\bigr) \bmod 7$ 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 $C(\alpha)=2^t-2^{t-1}e_t-2^{t-2}e_{t-1}-\cdots-2^0e_1$, 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 $\alpha_1={*}10{*}{*}0{*}$, $\alpha_2=10{*}{*}0{*}$, …, $\alpha_5=\epsilon$; 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 $\bigl(\binom{n}{t}+(2^n-1)\binom{n-1}{t}\bigr)/2^{n-t}\binom{n}{t}=2^t(1-t/n)+2^{t-n}(t/n)$. [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

$$\sum_{k \in S_a} \left| (k \oplus p) - k \right| = 2^l |S_a| = 2^{2l}.$$

Also $g^{[-1]}(g(k) \oplus p) = k \oplus g^{[-1]}(p)$, and $\lfloor \lg g^{[-1]}(p) \rfloor = \lfloor \lg p \rfloor$. Therefore

$$\frac{1}{2^n} \sum_{k=0}^{2^n - 1} \left| g^{[-1]}(g(k) \oplus p) - k \right| = \frac{1}{2^n} \sum_{a=0}^{2^{n-l}-1} \sum_{k \in S_a} \left| (k \oplus g^{[-1]}(p)) - k \right| = \frac{1}{2^n} \sum_{a=0}^{2^{n-l}-1} 2^{2l} = 2^l.$$

[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 $r \leftarrow (r-1) \wedge m$ 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 $m \leftarrow 2m + [t \wedge (t-1) \neq 0]$ 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 $\big((a_{n-1} \oplus a_{n-2})(a_{n-1} \oplus a_{n-3})(a_{n-2} \oplus a_{n-4}) \dots (a_2 \oplus a_0)(a_1)\big)_2$. 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

$$w_k(x) = w_{\lfloor k/2 \rfloor}(2x) = r_1(2x)^{b_1+b_2} r_2(2x)^{b_2+b_3} \dots = r_1(x)^{b_0+b_1} r_2(x)^{b_1+b_2} r_3(x)^{b_2+b_3} \dots$$

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 $k = (a_0 \dots a_{n-1})_2$, $k = 2^n - 1 - j = (\overline{a}_{n-1} \dots \overline{a}_0)_2$, and $k = g^{[-1]}(j) = \bigl((a_{n-1}) \dots (a_{n-1} \oplus \cdots \oplus a_0)\bigr)_2$, respectively. Then, using the Kronecker product of matrices, we have the recursive formulas

$$R_{n+1} = \begin{pmatrix} R_n \otimes (1\;0) \\ R_n \otimes (0\;1) \end{pmatrix}, \qquad F_{n+1} = F_n \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad G_{n+1} = \begin{pmatrix} G_n & 0 \\ 0 & G_n F_n \end{pmatrix},$$

$$H_{n+1} = H_n \otimes \begin{pmatrix} 1 & 1 \\ 1 & \overline{1} \end{pmatrix}, \qquad P_{n+1} = \begin{pmatrix} P_n \otimes (1\;1) \\ P_n \otimes (1\;\overline{1}) \end{pmatrix}, \qquad W_{n+1} = \begin{pmatrix} W_n \otimes (1\;1) \\ F_n W_n \otimes (1\;\overline{1}) \end{pmatrix}.$$

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

$$S_0 = \Bigl\{\frac{1}{2}\Bigr\}, \quad S_1 = \Bigl\{\frac{1}{4}, \frac{3}{4}\Bigr\}, \quad \dots, \quad S_e = \Bigl\{\frac{2j+1}{2^e} \Bigm| 0 \le j < 2^e\Bigr\}.$$

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 $|z_{kl} - l/(k+1)| = \Omega\bigl((\log k)/k\bigr)$. 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 $p(0) \dots p(7) = 0\,1\,7\,2\,5\,6\,3\,4$, 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 $r = 1$:

$$\begin{array}{lll} p_2p_1p_0 & \text{Change} & \text{Set} \\ 0\ 0\ 0 & a_0, p_0 & \left.\begin{array}{l} j_0 \leftarrow f_{00} \\ j_1 \leftarrow f_{01} \end{array}\right\} j \leftarrow f_0 \\ 0\ 0\ 1 & a_1, p_1 & \\ 0\ 1\ 1 & a_0, p_0 & \left.\begin{array}{l} f_{00} \leftarrow 0 \\ f_{01} \leftarrow 0 \end{array}\right\} f_0 \leftarrow 0 \\ 0\ 1\ 0 & a_2, p_2 & \end{array} \qquad \begin{array}{lll} p_2p_1p_0 & \text{Change} & \text{Set} \\ 1\ 1\ 0 & a_0, p_0 & \left.\begin{array}{l} f_{j0} \leftarrow f_{(j+1)0} \\ f_{j1} \leftarrow f_{(j+1)1} \end{array}\right\} f_j \leftarrow f_{j+1} \\ 1\ 1\ 1 & a_1, p_1 & \\ 1\ 0\ 1 & a_0, p_0 & \left.\begin{array}{l} f_{(j+1)0} \leftarrow (j{+}1)_0 \\ f_{(j+1)1} \leftarrow (j{+}1)_1 \end{array}\right\} f_{j+1} \leftarrow j{+}1 \\ 1\ 0\ 0 & a_{j+3}, p_2 & \end{array}$$

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 $p_r \ldots p_0 = a_r \ldots a_0$, 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 $\delta_1 \ldots \delta_{2^n-1}$ and $\varepsilon_1 \ldots \varepsilon_{2^n-1}$ 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

$$\delta_{k+1} \ldots \delta_{2^n-1}\, n\, \varepsilon_1 \ldots \varepsilon_{2^n-1}\, n\, \delta_1 \ldots \delta_k$$

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 $\delta_1 \ldots \delta_{2^n-1}$, $\ldots$, $\eta_1 \ldots \eta_{2^n-1}$ where $2^{\delta_1} \oplus \cdots \oplus 2^{\eta_{2^n-1}} = 0$, by interchanging $n \leftrightarrow h$ and $n+1 \leftrightarrow j$ in

$$\delta_{k+1} \ldots \delta_{2^n-1}\, n\, \varepsilon_1 \ldots \varepsilon_{2^n-1}\, (n{+}1)\, \zeta_1 \ldots \zeta_{2^n-1}\, n\, \eta_1 \ldots \eta_{2^n-1}\, (n{+}1)\, \delta_1 \ldots \delta_k.$$

Let $a(n)$ and $b(n)$ be the number of n-bit cycles defined in parts (a) and (b); then $(a(1), \ldots, a(5)) = (1, 0, 0, 1920, 318996480)$ and $(b(1), \ldots, b(5)) = (0, 2, 12, 384, 4200960)$. 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

$$\lim_{n\to\infty} d(n)^{1/2^n} \ge d(5)^{1/32} \prod_{n=5}^{\infty} \Bigl(2^n - \frac{25}{64}\Bigr)^{1/2^{n+1}} \Bigl(\frac{n+1}{n}\Bigr)^{1/2^{n+1}} \approx 2.3606.$$

[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 $g(0) \ldots g(k-1)v$, 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 $U = (u_0, u_1, u_2, \ldots)$ with an n-bit cycle $V = (v_0, v_1, v_2, \ldots)$, by forming concatenations

$$W = (u_{i_0}v_{j_0},\ u_{i_1}v_{j_1},\ u_{i_2}v_{j_2},\ \ldots), \qquad i_k = \bar{a}_0 + \cdots + \bar{a}_{k-1}, \quad j_k = a_0 + \cdots + a_{k-1},$$

where $a_0a_1a_2\ldots$ is a periodic string of control bits $\alpha\alpha\ldots$; 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

$$\begin{aligned} l+1 &> 2^m r(n)/t > 2^m r(n)/(2^m r(n)/(r(m)+r(n)) + 2) \\ &= r(m) + r(n) - \frac{2(r(m)+r(n))^2}{2^m r(n) + 2(r(m)+r(n))} > r(m) + r(n) - 1 \end{aligned}$$

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 $k = 0, 1, \ldots, 1023$, 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 $(u\,w)\tau_j$. 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

$$\Sigma = \prod_{k=1}^{2^{n-1}-1} \bigl(\Sigma(\delta_{2k-1}, \delta_{2k}, v_{2k-1})\,\Sigma(\delta_{2k}, \delta_{2k+1}, v_{2k})\bigr)$$

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 01020314203024041234 214103234103, and it is close to two of the others.

66. (Solution by Mark Cooke.) One suitable delta sequence is 012345607012132435 65760710213534626701537412362567017314262065701342146560573102464537 57102043537614073630464273703564027132750541210275641502403654250136

0254161560431257603257204315762432176045204175163547670356475706254 3
7242132624161523417514367143164314. (Solutions for $n > 8$ are still unknown.)

67. Let $v_{2k+1} = \overline{v}_{2k}$ 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 $h(k) = (\ldots a_2a_1a_0)_2$ 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 $\Delta_3 = 0\,\underline{1}\,0\,2\,1\,\underline{0}\,1$ and $E_3 = 0\,\underline{2}\,1\,2\,0\,\underline{2}\,1$, 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 $2\bigl(\binom{n-1}{j-1} + \binom{n-1}{j} + \binom{n-1}{j+1}\bigr) - 1$. 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

$$
\begin{array}{l}
0\,1\,2\,3\,0\,1\,2\,4\,2\,1\,0\,3\,2\,1\,0\,1\,2\,1\,0\,3\,2\,1\,0\,4\,0\,1\,2\,3\,0\,1\,2\,(1)\\
0\,1\,2\,3\,0\,1\,2\,4\,2\,1\,0\,3\,2\,1\,0\,1\,3\,0\,1\,2\,3\,0\,1\,4\,1\,0\,3\,2\,1\,0\,3\,(1)\\
0\,1\,2\,3\,0\,1\,2\,4\,2\,1\,0\,3\,2\,1\,0\,2\,0\,3\,2\,1\,0\,3\,2\,4\,2\,3\,0\,1\,2\,3\,0\,(2)\\
0\,1\,2\,3\,0\,1\,2\,4\,2\,3\,0\,1\,2\,3\,0\,2\,0\,1\,2\,3\,0\,1\,2\,4\,2\,3\,0\,1\,2\,3\,0\,(2)\\
0\,1\,2\,3\,4\,1\,0\,1\,2\,1\,0\,3\,0\,1\,4\,3\,2\,1\,0\,3\,0\,1\,4\,1\,0\,1\,2\,3\,4\,1\,0\,(3)
\end{array}
$$

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 $n \ge 3$ 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 $0 \le i < j$, 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 $\bigl(x, (y+1) \bmod m\bigr)$ if and only if $(x+y) \bmod m \ne m-1$ if and only if the second cycle contains the edge from (x, y) to $\bigl((x+1) \bmod m, y\bigr)$.

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) 2737465057320265612316546743610525106052042416314372145101421737
2506246064173213107351607103156205713172463452102434643207054702
4147356146737625047350745130620656415073123731427376432561240264
3016735467532402524637475217640270736065105215106073575463253105;

(2) 0616713417232175171671540460247164742473202531621673531632736052
6710141503047313570615453627623241426465272021632075363710750740
3157674761545652756510451024023107353424651230406545306213710537
2620501752453406703437343531502602463045627674152752406021610434;

(3) 3701063751507131236243765735103012042353747207410473621617247324
6505132565057121565024570473247421427640231034362703262764130574
0560620341745613151756314702721725205613212604053506260460173642
6717641743513401245360241730636545061563027414535676432625745051;

(4) 6706546435672147236210405432054510737405170532145431636430504673
4560621206416201320742373627204506473140171020514126107452343672
1320452752353410515426370601363567307105420163151210535061731236
4272537165617217542510760215462375452674257037346403647376271657.

(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

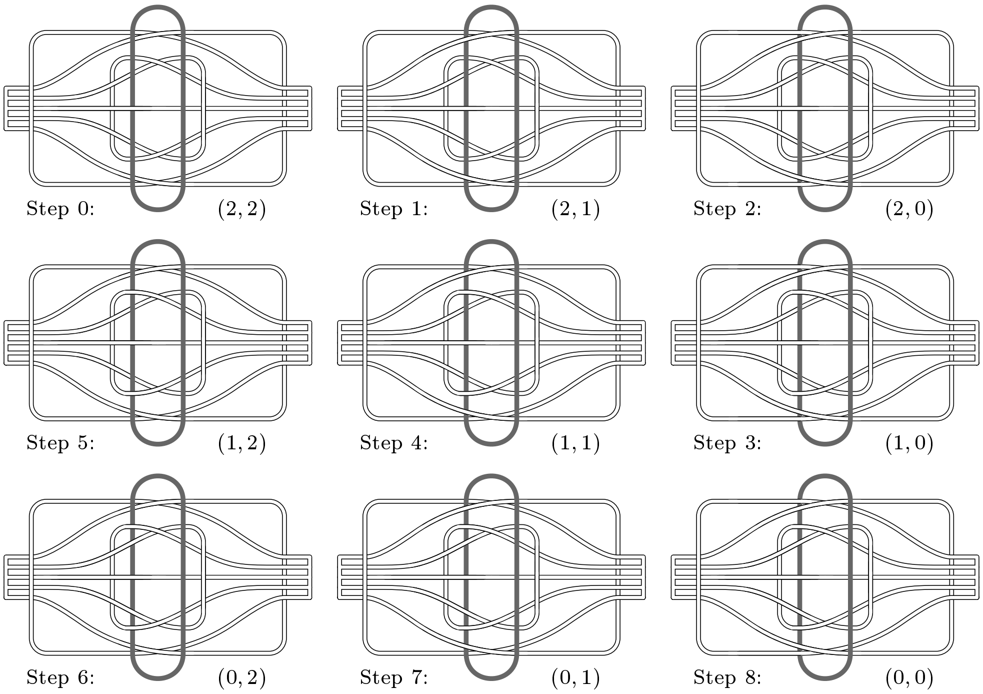


Fig. A-1.

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 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." 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, $\ldots$.

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, $\ldots$, 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 $M_{n+2} = \bullet\, M_{n+1}^R,\ \boldsymbol{-}\, M_n^R$. 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 $\boldsymbol{-}$, 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 $\boldsymbol{-}$; if k is even and $d_k = \boldsymbol{-}$, 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 $\bullet\boldsymbol{-}\boldsymbol{-}\bullet\bullet\bullet\bullet\boldsymbol{-}\bullet\boldsymbol{-}\bullet$.

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 $(\bullet\boldsymbol{-})^{\lfloor n/3 \rfloor}\bullet^{n \bmod 3}$ and end with $(\boldsymbol{-}\bullet)^{\lfloor n/3 \rfloor}\bullet^{[n \bmod 3 = 1]}\boldsymbol{-}^{[n \bmod 3 = 2]}$. But $M_{3k+1}\bullet$, $M_{3k}^R\boldsymbol{-}$ 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 $\bullet-$ 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, $\ldots$.)

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. $0\underline{01020}3041\underline{121}3142\underline{2}3243344$. (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 bits 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 (00000110001011011111 001110101001). 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

$$\begin{array}{cc} \text{when } j_n \bmod 4 < 3 \ (l_n < k_n): & \text{when } j_n \bmod 4 = 3 \ (k_n < l_n): \\ f_{2n}(k) = \begin{cases} c_n(k-1), & \text{if } 0<k\leq l_n+2; \\ c_n(k-3), & \text{if } l_n+2<k\leq k_n+3; \\ c_n(k-4), & \text{if } k_n+3<k\leq 2^{2n}; \end{cases} & = \begin{cases} c_n(k-1), & \text{if } 0<k\leq k_n+1; \\ c_n(k-2), & \text{if } k_n+1<k\leq l_n+3; \\ c_n(k-4), & \text{if } l_n+3<k\leq 2^{2n}. \end{cases} \end{array}$$

(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

$\Sigma f_{2n}(k)$, when $j_n \bmod 4 < 3$ $(l_n < k_n)$: $\qquad$ when $j_n \bmod 4 = 3$ $(k_n < l_n)$:

$$\equiv \begin{cases} \Sigma c_n(k-1), & \text{if } 0 < k \le l_n+2; \\ 1+\Sigma c_n(k-3), & \text{if } l_n+2 < k \le k_n+3; \\ \Sigma c_n(k-4), & \text{if } k_n+3 < k \le 2^{2n}; \end{cases} \qquad \equiv \begin{cases} \Sigma c_n(k-1), & \text{if } 0 < k \le k_n+1; \\ 1+\Sigma c_n(k-2), & \text{if } k_n+1 < k \le l_n+3; \\ \Sigma c_n(k-4), & \text{if } l_n+3 < k \le 2^{2n}. \end{cases}$$

$$\Sigma c_n(k) \equiv \Sigma f_n^+(\lceil k/2 \rceil) + \Sigma f_n^-(\lfloor k/2 \rfloor).$$

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

$$\Sigma f_n^\pm(k) \equiv \lfloor k/(2^n \pm 2) \rfloor + \Sigma f_n^\pm(k \bmod (2^n \pm 2)); \quad \Sigma f_n(k) = \Sigma f_n(k \bmod 2^n).$$

$$\Sigma f_{2n+1}(k) \equiv \begin{cases} \Sigma\Sigma f_{2n}(k), & \text{if } 0 < k \le j_{2n} \text{ or } 2^{2n}+j_{2n} < k \le 2^{2n+1}; \\ 1+k+\Sigma\Sigma f_{2n}(k+\delta_{2n}), & \text{if } j_{2n} < k \le 2^{2n}+j_{2n}. \end{cases}$$

$\Sigma\Sigma f_{2n}(k)$, when $j_n \bmod 4<3$ $(l_n<k_n)$: $\qquad$ when $j_n \bmod 4=3$ $(k_n<l_n)$:

$$\equiv \begin{cases} \Sigma\Sigma c_n(k-1), & \text{if } 0<k\le l_n+2; \\ 1+k+\Sigma\Sigma c_n(k-3), & \text{if } l_n+2<k\le k_n+3; \\ \Sigma\Sigma c_n(k-4), & \text{if } k_n+3<k\le 2^{2n}; \end{cases} \qquad \equiv \begin{cases} \Sigma\Sigma c_n(k-1), & \text{if } 0<k\le k_n+1; \\ 1+k+\Sigma c_n(k-2), & \text{if } k_n+1<k\le l_n+3; \\ 1+\Sigma\Sigma c_n(k-4), & \text{if } l_n+3<k\le 2^{2n}. \end{cases}$$

$$\Sigma\Sigma f_{2n}(k) \equiv [j_n \bmod 4 < 3]\lfloor k/2^{2n} \rfloor + \Sigma\Sigma f_{2n}(k \bmod 2^{2n}).$$

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 $\gamma = a_0b_0\ldots a_{n-1}b_{n-1}$ 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 $\beta = b_0 b_1 \ldots b_n$ in $f_{n+1}()$ when n is even, suppose that the n-bit string $(b_0 \oplus b_1) \ldots (b_{n-1} \oplus b_n)$ 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 $\lambda_1 \ldots \lambda_t$. 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 (60) 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 $\alpha = \lambda_1\beta = a_1 \ldots a_n$ 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 $j < k \leq n$.

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 $P_m(n) = Q(m,n) + O\bigl(Q(\sqrt{m}, n)\bigr)$, 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 $nP_m(n) - \sum_{j=1}^{n} j L_m(j) = m^{n+1}(1/((m-1)^2 n) + O(1/(mn^2)))$. 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 $\lambda_{k+1} \leq \beta 9^{|\alpha|}$, 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 $\beta \leq \lambda_k$; then $\lambda_k \leq \beta\alpha$, so β is a prefix of λ_k. Since β is a preprime, it is the $|\beta|$-extension of some prime $\beta' \leq \beta$. 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 $n - |\beta'| \geq n - |\beta|$ 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 $\lambda_{k+1} \leq (\beta\alpha)^{d-1}\beta 9^{|\alpha|}$, because the latter is prime. Otherwise λ_{k-1} ends with at least $(d-1)|\beta\alpha|$ 9s, and $\lambda_{k+1} \leq (\beta\alpha)^{d-1}9^{|\beta\alpha|}$, 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 $0 \leq a_n < 9$ (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 $1 \leq a \leq 9$, so we must not place 0 after $9^j 0^{n-j-1}$.

109. Suppose we want to locate the submatrix

$$\begin{pmatrix} (w_{n-1}\ldots w_1w_0)_2 & (x_{n-1}\ldots x_1x_0)_2 \\ (y_{n-1}\ldots y_1y_0)_2 & (z_{n-1}\ldots z_1z_0)_2 \end{pmatrix}.$$

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

$$\begin{array}{rrrrl} b_5 = w_2, & b_4 = x_2, & a_5\oplus b_5 = y_2, & a_4\oplus b_4 = z_2, & \text{if } a_0=0,\ b_0=0; \\ b_4 = w_2, & b'_5 = x_2, & a_4\oplus b_4 = y_2, & a_5\oplus b'_5 = z_2, & \text{if } a_0=0,\ b_0=1; \\ a_5\oplus b_5 = w_2, & a_4\oplus b_4 = x_2, & b_5 = y_2, & b_4 = z_2, & \text{if } a_0=1,\ b_0=0; \\ a_4\oplus b_4 = w_2, & a_5\oplus b'_5 = x_2, & b_4 = y_2, & b'_5 = z_2, & \text{if } a_0=1,\ b_0=1; \end{array}$$

here $b'_5 = b_5 \oplus b_4b_3b_2b_1$ takes account of carrying when j becomes $j+1$.

110. Let $a_0a_1\ldots a_{m^2-1}$ be an m-ary de Bruijn cycle, such as the first m^2 elements of (54). If m is odd, let $a_{ij}=a_j$ when i is even, $a_{ij}=a_{(j+(i-1)/2) \bmod m^2}$ 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

$$a_{ij}\equiv a'_{ij} \text{ (modulo } m') \qquad \text{and} \qquad a_{ij}\equiv a''_{ij} \text{ (modulo } m'')$$

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

$$0011\ 021331203223\ 04152435534251405445\ 0617263746577564\ldots07667\ 08\ldots$$

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 $a_{ij}=a_j$ if $i+j$ is even, $a_{ij}=a_i$ if $i+j$ is odd. For example, when $m=4$ we have

$$\left(\begin{array}{c} 0010021220302232\\ 0001020320212223\\ 0111031321312333\\ 1011121330313233\\ 0010021220302232\\ 0203000122232021\\ 0111031321312333\\ 1213101132333031\\ 0010021220302232\\ 2021222300010203\\ 0111031321312333\\ 3031323310111213\\ 0010021220302232\\ 2223202102030001\\ 0111031321312333\\ 3233303112131011 \end{array}\right) \text{(exercise 109);} \quad \left(\begin{array}{c} 0010001030203020\\ 0001020301000203\\ 0111011131213121\\ 1011121311101213\\ 0010001030203020\\ 2021222321202223\\ 0111011131213121\\ 3031323331303233\\ 0313031333233323\\ 1011121311101213\\ 0212021232223222\\ 0001020301000203\\ 0313031333233323\\ 2021222321202223\\ 0212021232223222\\ 3031323331303233 \end{array}\right) \text{(Tóth).}$$

111. (a) Let $d_j = j$ and $0 \le a_j < 3$ for $1 \le j \le 9$, $a_9 \ne 0$. 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.

INDEX AND GLOSSARY

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

65